The Impact of Petri Nets on System-of-Systems Engineering

KIRSTEN MHAIRI SINCLAIR

The Impact of Petri Nets on System-of-Systems Engineering

Kirsten Sinclair

Abstract

The successful engineering of a large-scale system-of-systems project towards deterministic behaviour depends on integrating autonomous components using international communications standards in accordance with dynamic requirements. To-date, their engineering has been unsuccessful: no combination of top-down and bottom-up engineering perspectives is adopted, and information exchange protocol and interfaces between components are not being precisely specified. Various approaches such as modelling, and architecture frameworks make positive contributions to system-of-systems specification but their successful implementation is still a problem.

One of the most popular modelling notations available for specifying systems, UML, is intuitive and graphical but also ambiguous and imprecise. Supplying a range of diagrams to represent a system under development, UML lacks simulation and exhaustive verification capability. This shortfall in UML has received little attention in the context of system-of-systems and there are two major research issues:

1. Where the dynamic, behavioural diagrams of UML can and cannot be used to model and analyse system-of-systems
2. Determining how Petri nets can be used to improve the specification and analysis of the dynamic model of a system-of-systems specified using UML

This thesis presents the strengths and weaknesses of Petri nets in relation to the specification of system-of-systems and shows how Petri net models can be used instead of conventional UML Activity Diagrams. The model of the system-of-systems can then be analysed and verified using Petri net theory. The Petri net formalism of behaviour is demonstrated using two case studies from the military domain. The first case study uses Petri nets to specify and analyse a close air support mission. This case study concludes by indicating the strengths, weaknesses, and shortfalls of the proposed formalism in system-of-systems specification. The second case study considers specification of a military exchange network parameters problem and the results are compared with the strengths and weaknesses identified in the first case study.

Finally, the results of the research are formulated in the form of a Petri net enhancement to UML (mapping existing activity diagram elements to Petri net elements) to meet the needs of system-of-systems specification, verification and validation.

The Impact of Petri Nets on System-of-Systems Engineering

Kirsten Sinclair

Ph.D. Thesis

Electronics Group

School of Engineering

University of Durham

2009

Contents

Figures

Tables

Declaration

The material presented in this thesis is the sole work of the author and has not been previously submitted for a degree at this or any other university.

Statement of Copyright

Acknowledgements

My sincerest thanks to Professor Keith Bennett, Mr Peter Baxendale, Dr John Hartas, and Mrs Elaine Dolan for all their support over the previous three years. Colleagues at SyntheSys Systems Engineers Ltd discussed much of the case study work with me and I am very grateful for their help and advice.

I would also like to thank my family and friends for proof reading and providing much encouragement throughout.

This work was part of a Knowledge Transfer Partnership programme funded by the Technology Strategy Board and SyntheSys Systems Engineers Ltd.

Chapter 1 Introduction

1.1 Context

The engineering of modern, dynamic, large-scale distributed systems so that their overall behaviour reflects stated requirements is an achievement much sought after in both private and public sectors. These complex systems, or systems-of-systems, are uniquely composed of multiple (mainly legacy) component parts integrated dynamically using standardised communications to deliver a particular application. Presently, this integration is difficult and unsuccessful [1, 3, 124], with verification and validation of the resulting system-of-systems only undertaken post-implementation (when it is often too late). Failure to meet the required outcomes impacts negatively on cost, safety, reliability, and worse case, human life.

Consequently, it is desirable for the problem and solution design specification of these large-scale systems-of-systems to offer a high degree of reassurance that the physical implementation achieved using the design will preserve the functional and non-functional requirements of the co-ordinator(s). This requires a suitable means of capturing, verifying and validating the design of the system-of-systems prior to its actual solution manifestation.

The design specification of a system can be produced using various methods including textual documentation, or modelling languages such as the de-facto Unified Modelling Language (UML) [11]. Complete, consistent, and correct design specification requires consideration of the use of UML in the context of large-scale system-of-systems problem and solution specification and where the Petri net formal notation can be used to enhance UML.

1.2 Area of Interest

The term ‘system-of-systems’ has been in use for at least a decade but as yet there is still no universal agreement on a definition. Definitions in the literature have often been flawed, failing to differentiate between a system-of-systems and a collection of large-scale systems, or treating a system-of-systems as similar to any other system. System-of-systems is defined for the thesis to show the concept is different to that of a traditional system.

In order to provide a definition of a system-of-systems, it is necessary to start by considering definitions for a system.

The concept of a system arose from the 1940’s when several disciplines and technologies were integrated to achieve goals which otherwise would not have been achievable. For example, integration of high and low altitude radars, ground and air communications links and human decision makers for the Battle of Britain can be viewed as an early complex system. In a complex system, the behaviour of its elements and how they act together to form the behaviour of the whole must be understood. Elements of a complex system can be composed of simple or complex systems. Bar-Yam [26] suggests that:

‘the complexity of a system is the amount of information needed in order to describe it. The complexity depends on the level of detail required in the description’.

ISO/IEC Standard 15288 [5] defines a system as a:

‘combination of interacting elements organised to achieve one or more stated purposes’

This definition is recursive and implies that a system defined in this way can apply to both the smallest subcomponent and the largest aggregation of systems. Another definition of a system provided by the International Council on Systems Engineering (INCOSE) [6] reinforces this:

‘a system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behaviour and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected’.

[6] emphasises that a system produces results unachievable by the components alone and is further illustrated by Rechtin [33] in this example:

‘imagine that your automobile was completely disassembled and laid out on your driveway. All the elements individually would be just as before, all in working order. But you would have no transportation. Transportation, the unique system function, only exists when all the elements are connected together and function as a whole’.

A system can be seen as an entity that is capable of interacting with its environment (i.e. everything other than the system) and can react differently over the progression of time to the same input activity. For example, in a timer-controlled heating system the sensed temperature depends on the current time. Together with the thermostat, the sensed temperature dictates whether or not the heating system is switched on. As a result, the system can react in a different way at different times to the same sensed temperature. Therefore, a system can be viewed as a collection of parts whose behaviour is dictated by the interfaces that define its boundary. System state can also influence its potential behaviour. It is this behaviour that the user of a system is concerned with, i.e. the activity at system interfaces.

An interface is defined as a point of interaction between a system and its environment. An interface can be an output (information is produced for the system environment), an input (information is taken from the environment) or a bi-directional interface. Communication among systems is carried out using messages, i.e. data structures formed for the purpose of communication among systems. A system’s behaviour is the sequence of send and receive operations and is normally characterised by send operations. Links between the interfaces of two or more interacting systems are known as connections and these are governed by a set of rules which are defined as

protocols. Each interface has a set of attributes associated with it which control the possible types of interaction. For example, information encoding, structure, meaning and timing of information exchanges control interaction at the interface. Interfaces must be compatible (directly or following adaptation) in order to integrate them.

Recursive decomposition can be performed on a system to obtain its interacting parts. These may also be systems that can be further decomposed. This process can be stopped when the details of a component system are of no relevance to the particular project. Another term, 'system of interest', also helps to specify the level of detail the discussion of a system is taking place at. ISO/IEC Standard 15288 [5] defines 'system of interest' as the system whose lifecycle is under consideration. Usually systems contain subsystems which are made up of components which in turn contain units. A unit is the smallest managed part. For example, a sensor system may comprise an information processing subsystem (controller) and a mechanical subsystem (sensor element). The controller records the sensed information and passes the information to the sensor interface in a message.

Over time, systems have grown to include many systems of increased complexity and descriptions like 'subsystem', 'component', and 'unit' become recursive. ISO/IEC Standard 15288 [5] uses the term 'system element' to make the concept of a system more flexible and describe the parts from which a system of interest is composed. Therefore, depending on the system of interest, system elements can be systems in their own right, subsystems, components or units.

The following definition from IEEE Standard 1220 [7] reinforces that a system also has a lifecycle and implied supporting infrastructure:

'a set or arrangement of elements [people, products (hardware and software) and processes (facilities, equipment, material and procedures)] that are related and whose behaviour satisfies operational needs and provides for the lifecycle sustainment of the Products'.

1.2.1 Approach for Developing Systems

The effort to understand the lifecycle of systems has been through systems engineering. Systems engineering as a discipline was born during the Second World War in order to cope with the increase in complexity of systems. It uses a process model to support a system's lifecycle. INCOSE [6] define systems engineering as:

'an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle'.

It is an interdisciplinary approach concerned with the design, architecting and integration of elements to form a system. Systems engineering aims to address the business and technical needs of all system owners by delivering a product which meets their needs. The process involved uses a management process to organise the technical effort and is usually comprised of seven tasks: state the problem; investigate alternatives; model the system; integrate; launch the system; assess performance; and

re-evaluate. The various technical stages are linked via models such as the ‘Waterfall Model’ or ‘Vee Model’.

In systems engineering efforts there are usually several systems involved. These are: the system that the user operates (‘end product’); the system that supports the end product (‘enabling system’); the system that is used to develop the end product (‘process system’); and systems for test, deployment, training and management. Each of these are designed and built using methods, toolsets and quality assessments. Systems engineering is a process that is comprised of activities that define the requirements for a system, transform these requirements into a system using development and deploy the system operationally. The systems engineer adopts a process oriented view of these activities and takes into account the needs of the customer, system implementation team and systems to be integrated.

1.2.2 Characteristics of a Traditional System

In summary, traditional deterministic systems exhibit the following characteristics:

1. Formal requirements engineering with traceability from requirements to design (formal is taken here to mean an established process, whereas the term 'formal method' relates to mathematically-based techniques for system specification).
2. Architecture and design well understood.
3. Application of standard processes to building them by a central authority.
4. Availability of staff trained in systems engineering.
5. Support toolsets (such as Integrated Development Environments) for their construction.
6. Evolution done by well understood process.
7. Application of models to link technical stages e.g. Vee Model.
8. Verification, validation and testing process, technologies and toolsets well understood and practised with available statistical results.
9. Metrics and optimisation of process well understood e.g. Capability Maturity Model Integration (CMMI) has several process models which can be used to refine an organisation’s systems engineering processes.
10. Well understood problem areas.
11. Autonomy is exhibited by the system as a whole rather than its parts.
12. Parts are engineered purposely for the system of interest and may not be useful in any other system.
13. Connectivity is engineered into the system and normally kept to a minimum between subsystems.
14. Desirable behaviour is designed for and undesirable behaviour reduced via testing. This includes provision of fault tolerance, essential for many real-time systems.

1.3 The System-of-Systems Concept

Historical applications of the system-of-systems term (mainly within defence sector applications) have existed for decades. Projects including communications satellites, space flight and nuclear-powered submarines are classified as systems-of-systems or complex systems with increased technological risk, design constraints and auditing.

1.3.1 Characteristics of Complex Systems

Expanding on the definition of complex systems provided earlier in section 1.2, Sheard [27] defines a complex system as:

'…systems that do not have a centralising authority and are not designed from a known specification, but instead involve disparate stakeholders creating systems that are functional for other purposes and are only brought together in the complex system because the individual "agents" of the system see such co-operation as being beneficial for them' .

Sheard [27] goes on to summarise a number of characteristics for complex systems:

1. Their structure and behaviour is not deducible or inferable from the structure and behaviour of the component parts.
2. Their elements adapt to the environment as they evolve.
3. They have a large number of autonomous, heterogeneous elements.
4. They display emergent macro-level behaviour from the actions and interactions of the individual agents.
5. They exhibit non-deterministic behaviour.
6. They include not only component systems but also the designers and users of the component systems.
7. They are not green field projects where the development begins at the same time.

Autonomous is taken here and in the remainder of this document to mean elements capable of independent action or decision-making. From the definitions and characteristics above for complex systems it would seem non-deterministic system-of-systems are complex systems. This is further reinforced by Keating et al [9]. They summarise the definitions for system-of-systems suggested by several researchers and state their view of a system-of-systems as being:

‘comprised of multiple autonomous embedded complex systems that can be diverse in technology, context, operation, geography and conceptual frame’.

However, it is not clear from this definition how a system-of-systems differs from a system. The Defence Industrial Strategy [21] uses this definition for a system-of-systems:

‘these contain systems which have purpose and are viable independent of the System-of-Systems, but which can when acting together perform functions unachievable by the individual systems acting alone. For instance, the future aircraft carrier, combining its aircraft carrier group with its own sensors, communications and command systems and weaponry and interacting with wider networks, represents a System-of-Systems’.

Caffal et al [8] define system-of-systems as:

‘an amalgamation of legacy systems and developing systems that provide an enhanced military capability greater than any of the individual systems within the system of systems’.

Gaudel et al [10] suggest this definition:

‘A system constructed from autonomous component systems, where autonomous means independence with respect to existence, operation and/or evolution’.

The key points from these definitions would appear to be the ability of the component systems to function independently of one particular system-of-systems and that system-of-systems are an integration of legacy and new technology systems.

Certain attributes of systems are particularly relevant to their integration into a system-of-systems and relate to autonomy, controllability and encapsulation. Autonomy is concerned with independence of a component system in relation to its existence, operation and evolution. It is important to discover whether the component system was purposely built for one particular system-of-systems (custom-off-the-shelf) or re-used (legacy component). Also, component systems involved in a system-of-systems can operate and evolve under independent management (controlled by service level agreements), under no contract or under the same control as the system-of-systems. Unless the component system has been specifically built for the system-of-systems, information on the following may not be readily available: construction and verification methods used by designers; assurance relating to dependability, security and safety; formal semantics of the services offered by each component system; and adaptability of the component systems. In combining the systems together, systems engineering practice recommends hiding complex connectivity detail between elements through encapsulation. In a system-of-systems, connectivity needs to be established between legacy and new systems. This implies that it may be necessary to expose internal element details to facilitate connectivity.

It is likely that each autonomous legacy system was developed according to its own rules and conventions concerning data structure, information exchange protocols and error handling. It is also likely that any legacy systems to be integrated will be incompatible at the connection level. In this case, the connection has to try and reconcile these conventions to enable communication (an additional requirement of the system-of-systems may be to tolerate such failures of component systems).

1.3.2 Characteristics of a System-of-Systems

A system-of-systems can be seen as having similar characteristics as those of a complex system. DeLaurentis et al [2] confirm that a system-of-systems is not merely ‘a simplistic box-inside-a-box approach’ (e.g. a power supply unit within an aircraft) but that there are distinguishing traits associated with them. The traits associated with a system-of-systems are described by Maier [4], Boardman et al [22] and DeLaurentis et al [2] as:

1. Heterogeneity: each component system is distinct with different important characteristics and can operate to different timescales.
2. Emergence: a system-of-systems exhibits capability that its component systems cannot achieve independently or as a subset. Dyson [12] suggests 'Emergent behaviour is that which cannot be predicted through analysis at any level simpler than that of the system as a whole. Emergent behaviour, by definition, is what's left after everything else has been explained'. Fisher [13] describes emergent behaviour as '..actions that cannot be localised to any single

component of the system..' and '..the unavoidable result of interactions among autonomous entities and thus will occur in systems of systems whether by accident or intention'. Fisher [13] labels the emergent products or services of these interactions between autonomous constituents as 'cumulative effects' where constituents are any automated or human participant.

3. Operational and Managerial Independence: component systems within a system-of-systems exhibit ‘autonomy’ in that they are able to function usefully alone and their behaviour may differ from that fulfilled by the system-of-systems. Component systems are usually acquired individually.
4. Evolutionary Behaviour: component systems can be added, amended or withdrawn over time.
5. Geographical Distribution: component systems are often geographically dispersed and are likely to exchange information (not physical mass or energy) via communication networks.
6. Inter-disciplinary: system-of-systems typically integrate a variety of engineered systems using many disciplines (engineering, game theory, uncertainty, mathematics, economics and management).
7. System of Networks: rules of interaction govern the connectivity between component systems and the topology can change over time.
8. Belonging: component systems opt to join based on a cost-benefit basis, belief in the overall system-of-systems goal, and to try and enhance their own goals.
9. Diversity in system-of-systems function: achieved through component system autonomy, belonging and open connectivity.

In addition, there are two other traits:

10. Funding: planning, incentive and budgeting systems are often not synchronised with the development of the system-of-systems [31].
11. Verification and validation: there is no method for verifying and validating the system-of-systems prior to implementation. Verification and validation normally takes place post-implementation when it is too late.

These traits are now examined in more detail below.

For operational and managerial independence, a system-of-systems is composed of components capable of independent action or decision-making (i.e. autonomous elements). If a system-of-systems is decomposed, each component could perform independently of the others and be useful in its own right (i.e. has ‘a life of its own’). In contrast to the earlier Rechtin [33] example of a car in section 1.2, in a system-of-systems the individual components would not remain laid out on the driveway for assembly. Able to operate when disassembled from the whole, the elements have their own purposes independent of each other and are acquired, integrated and managed separately. Examples fulfilling this characteristic are the internet and military joint operations where different agencies own different systems in each system-of-systems.

As well as autonomous elements, another important enabling concept in system-of-systems is communication between these elements. In traditional systems engineering, integration is a centrally controlled process used to combine elements into a system. With a system-of-systems, autonomous elements can only contribute to system-wide goals via co-operative interactions with other elements. The process used to combine

autonomous elements to form the system-of-systems is interoperation. Interoperation methods should take into account that system-of-systems do not have clearly defined boundaries; all outcome information is often unavailable; requirements are dynamic and imprecise; centralised control is unlikely; system-of-systems are expected to be resilient to unreliability in other elements and unexpected events external to the system; and the continuous adaptive, evolving and emergent behaviour of a system-of-systems is directly influenced by human elements.

Continuous evolutionary development in system-of-systems is also enabled by operational and managerial independence. A system-of-systems is not fully formed or complete and purposes are added or modified with accumulated experience. Autonomy of elements means each can be designed, implemented and evolved independently of the systems in which it will be integrated. These independently operated elements with evolving purpose and structure interact with other elements they have no control over. In traditional systems, evolution was rarely considered as an integral part of their development. Two forms of evolution must be considered within a system-of-systems: evolution of the elements and evolution of the system-of-systems as a whole.

In system-of-systems, evolution of independent elements greatly increases the complexity of their interactions with other elements as they are developed or upgraded on uncoordinated timescales. This evolution also includes elements associated with information exchange, i.e. the protocols and interfaces. Since there is no comprehensive capture of the interface and protocol requirements for a system-of-systems, there is no means of ensuring that its elements achieve adequate interactions or interoperability with other elements.

The internet demonstrates evolutionary behaviour. Computers and networks offering new services are frequently introduced or modified. The World Wide Web Consortium (W3C), and a collection of owners (developers, and users) collaborate via bottom-up discussion to produce the standards (Request for Comments or RFCs) by which the world wide web operates.

Evolution, and operational and managerial independence are also encouraged by geographic distribution of elements (but can also occur without it). Here, individual elements can be distributed over large geographic areas exchanging information and producing emergent behaviour as seen in the case of the internet.

The earlier descriptions of emergent behaviour in section 1.3.2 are unclear as to whether it can be predicted or analysed. Global properties can be service-based (e.g. performance, or safety) or product-based (e.g. electricity generation from a national power grid). An example of emergent behaviour is road congestion during rush hour where slow progress emerges as a global, service-based property of the road system. Here, the congestion can be reasonably predicted based on the number of vehicles involved but it may be extremely difficult to analyse and explain exactly how this global property was achieved. Other examples of emergence include: trees and their forest; a mobius strip where one-sidedness is the emergent property obtained by twisting and attaching the ends of a rectangular strip of paper; an orchestra using its components to produce a symphony; military network-centric operations, e.g. the

networking of sensors, decision-makers and platforms for shared situational awareness and improved decision-making.

Recent public sector approaches to procurement aim to move the focus away from specific individual systems meeting particular performance requirements to solutions emerging to meet broad sets of needs or capability (e.g. vision for Air Traffic Management or the US Army Future Force/UK Future Integrated Soldier Technology). Capability requires the co-operation of multiple constituent systems within a system-of-systems. It is anticipated that this approach will enable organisations like the military to prepare for the unknown as it is unlikely all required information and collaborations will be known in advance.

In the defence sector, systems to support military strategic vision are realised through defence acquisition initiatives such as the UK Ministry of Defence's (MoD) 'Smart Acquisition' initiative offering tools and processes within the associated Acquisition Operating Framework [14]. The goal of smart acquisition is to:

'…acquire defence capability faster, cheaper, better and more effectively integrated'.

Here, defence capability is defined by the MoD Acquisition Operating Framework [14] as:

'the ability to generate an operational outcome or effect in the context of defence planning, Capability is the enduring ability to generate a desired effect'.

This means military equipment should no longer be replaced on a like-for-like basis. Instead, the acquisition operating framework tries to ensure delivery of fully integrated defence solutions. When a gap in capability is identified between predicted capability requirements and those covered by ongoing projects and existing systems, the acquisition lifecycle process is invoked to explore, procure and manage capability solutions. Within the acquisition operating framework, systems are specified and procured through independent projects to meet their own set of user requirements within established time and budgets. Ideally, military strategic vision will then be realised by combining the effects of these independently acquired systems into the desired defence capability.

Given the independence of component systems and the aims of smart acquisition, there are still funding issues for systems-of-systems. Unlike traditional systems engineering projects, systems-of-systems (particularly in the defence sector) are differentiated by the huge number of legacy systems from which they will be composed. Defence system-of-systems' requirements tend to have a much shorter lifecycle than that of the entities developed to realise them [68, 14]. These legacy entities are the result of lengthy, expensive procurement processes. When it comes to integrating these legacy and new technology entities, funding has tended to be piecemeal [15] with a lack of co-ordination between the agencies involved and appreciation of the additional resources needed to integrate and realise the overall system-of-systems from these component systems. Consequently, projects such as Future Integrated Soldier Technology have been delayed and procurement budgets eroded even further [16].

The lack of co-ordination mentioned above is not helped by the fact procurement of component systems and subsequent trade-off analysis according to capability demands clear understanding of the particular defence problem being addressed. Smart acquisition uses systems engineering processes to help system through-life management and is characterised by the MoD Concept, Assessment, Development, Manufacture, In-Service and Disposal (CADMID) initiative. The objective of systems engineering is to steer practitioners towards demonstrating that the implemented system meets the requirements expressed by the customer. The 'Vee' Model [18] is often used within CADMID to decompose the system lifecycle from customer requirements to detailed system level requirements and then compose the elements back into an operational system. The 'Vee' Model refers to verification (has the system been built in the right way?) and validation (has the right system been built?) between the decomposition of the system levels on the left hand side of the model and the composed system on the right hand side of the model. In the workshop summarised by [31], one of the key issues related to requirements management for a system-of-systems was that:

'there is no method for validating and adjudicating interoperability requirements in the documentation process; interoperability requirements are not defined early or identified as a common development goal'.

Another key issue reported them as being:

'not clearly documented or configuration controlled/managed; they cannot be further allocated, derived, or met'.

A separate issue is the fact that where requirements are captured, it is usually in static, textual format which can be ambiguous, inaccurate, lengthy, incomplete, and difficult to comprehend. Consequently, within defence, it is normally tangible, operational analysis that is relied upon during the lifecycle of a system to decide between alternative solutions. Modelling and simulation for validation of intangible design seem to be in their infancy in terms of their potential to system-of-systems engineering [19].

From the definitions and discussion of a system it can be generalised that a system-of-systems is a system in the sense that both are made up of parts, relationships and an end result which is greater than the sum of its parts. However, using the characteristics in this section associated with a system-of-systems, it can be seen that it is possible to differentiate between a traditional system and a system-of-systems. Composition of a traditional deterministic system involves a greater degree of pre-meditation and control within the established systems engineering framework. Composing systems-of-systems from predominantly legacy components (and new technology systems) to fulfil a desired need is much more challenging.

In a system-of-systems, a large number of integrated autonomous components exhibit behaviour not necessarily present in any one of them and each is likely to be managed separately from the system-of-systems. Here, an element of uncertainty prevails as a set of component systems co-operate to form a system-of-systems with ability perceived to be far superior to that of a mere component system. These diverse component systems co-operate by forming their own (dynamic) connections between

interfaces within a communication infrastructure but there is no assurance in place that the required behaviour from the co-operation is actually achieved. There is also ambiguity in terms of the funding of such co-operations and resulting incentive for achieving successful co-operation amongst component systems.

1.3.3 Thesis Definition of System-of-Systems

The previous discussion leads to a suggested definition of a system-of-systems for the purposes of this thesis as:

'a large-scale system engineered for desirable behaviour from autonomous component systems that have existence, and purpose beyond that of one particular system. By forming connections using well-defined interfaces and protocols, these diverse, independently owned component systems create a series of stable states of deterministic system-of-systems behaviour. Typically, the component systems can be part of multiple systems-of-systems'.

1.3.4 Summary

The concept of system-of-systems now appears to be firmly recognised as a particular kind of modern, large-scale system and a consequence of advances in computing and communications technology. Using several characteristics unique to systems-of-systems (although it is not a requirement for a system of interest to have them all), many modern systems are classifiable as systems-of-systems rather than traditional systems.

It is highly unlikely that a system-of-systems will be designed and built completely anew. Typically, it will be assembled from shared, reusable component systems developed for multiple purposes.

The following conclusions can be drawn between a traditional system and a system-of-systems:

Feature	Traditional System	System-of-Systems
Elements	All known and visible.	Dynamic and often unknown at requirements, design, or build time.
Purpose	Known by system owner and elements.	Behaviour spectrum ranging from fully-engineered to fully emergent (and continuously evolving). Purpose can be determined co-operatively, and may be unknown by elements.
Control	Hierarchical structure and centrally-controlled by a system owner.	Element owners are unlikely to have control over usage of their element within the system-of-systems.
Requirements	Managed by system owner and tend to be detailed system specifications.	Inadequate requirements specification and ownership at system-of-systems problem and solution design level.

Feature	Traditional System	System-of-Systems
Ownership	All elements are managed by system owner.	Elements are managed independently and co-ordinated for the system-of-systems.
Boundaries	Clearly bounded.	Unbounded and may be part of larger system-of-systems.
Visibility	All structure can be seen and managed.	Structure likely to be beyond control and visibility.
Unification	Centrally controlled integration process.	Interoperation between elements.
Standards	Standard processes for development, and systems implement relatively stable standards.	No standard development process. System-of-systems implement relatively stable standards across a wider set of component systems. Standards are key to interoperability.
Interfaces and Protocols	Connectivity tends to be minimised and uses well-defined, relatively stable interfaces and protocols.	Key enabler of interaction between component systems. A large number of (mainly legacy) component systems implement relatively stable interfaces and protocols. Inadequate capture of interface and protocol requirements for system-of-systems.
Verification and Validation	Well understood processes for testing supported by technologies and toolsets to help ensure undesirable behaviour reduced.	Testing normally carried out post-implementation in ad-hoc, trial-based manner (at the systems-level). Inadequate verification and validation at the system-of-systems problem and design level.
Funding	Single source of funding.	Multiple sources of funding make it difficult to co-ordinate resources for development of system-of-systems.

Table 1.1 'Why Isn't This Just a Scaling Issue?' adapted from Smith [23] pp.15-17

In discussing systems that are composed of systems, different terms such as 'system-of-systems and its component systems' or 'system and its subsystems' could be used. Here, 'system-of-systems and its component systems' is used to describe systems in which the components are systems and 'system' is used to refer to systems in general.

Although systems-of-systems have been around for at least fifty years, it is clear from Table 1.1 that the main problems in their engineering surround requirements specification, interfaces and protocols specification, and verification and validation of the design specification. These are summarised as follows:

1. There is no adequate capture of their problem and solution design specification (particularly information exchange specification) at the system-of-systems level.
2. Assurance that the design will lead to desirable implemented behaviour (through verification and validation) is also lacking.

1.4 Discussion of Problem

To help meet the need to successfully engineer systems-of-systems towards deterministic behaviour, this thesis addresses the problem of their behaviour specification, together with verification and validation of this design specification in order to provide assurance that the physical implementation of the design will behave as expected.

Organisations must respond to continuous change, especially in information technology, if they are to compete, and thrive. Consequently their mission strategies seek to take advantage of new technology as quickly as possible. Ultimately, it is the flexibility offered by the integration of (existing and new technology) individual systems that will enable collection, processing and delivery of information needed to support organisational decision processes in a timely manner. Core to this industry vision is the successful engineering of large-scale systems-of-systems which behave as desired. Unfortunately, deficiency in interoperability between component systems is the persistent problem that has plagued such integration.

The engineering and implementation of systems-of-systems provide organisations with a means of responding quickly to changes in their operational environment. For example, in the defence sector, systems-of-systems such as the US Army Future Combat Systems program (part of the US Army Future Force capability initiative) [20] aims to exploit advances in communications technologies to integrate soldiers with ground and air platforms by 2014. Component systems include the communications network, soldier, and fourteen independent manned and unmanned combat systems. Benefits of the integration are expected to include: improvements regarding speed and accuracy of operations; the ability of individual component systems to be configured in support of strategic and tactical level activities; the ability to reach globally diverse, distributed sites; the potential to combine situational awareness, command and control, weapons, protection, recovery, and logistics systems; and the ability to connect joint services, support agencies, and coalition partners.

As well as new technologies, there will be further diverse, unpredictable changes in political environments. Consequently, continuous revision of strategies, doctrine, and operational procedures will be vital. Again, the emphasis will be on adapting solutions to meet emerging operational challenges. Producing the required capability relies on successful integration of suitable constituent systems. Systems-of-systems will need to be configured and reconfigured to varying timescales depending on the operational context and will rely on components which continue to evolve without consultation.

The successful composition of component systems continues to be elusive given past and present examples. The recent (May 2008) British Airways Heathrow Terminal 5 baggage-handling failure was reported to have resulted in financial losses for British Airways of £16m. The overall purpose of this system-of-systems was accurate baggage-tracking (and improvement of British Airways' lost baggage record) and it integrates security, network, barcode baggage-tracking, baggage-reconciliation, manual baggage handling, self-service check-in, flight data, flight bookings, and third-party baggage-reconciliation (existing and new) component systems. [3] summarises evidence submitted to the Transport Select Committee regarding the causes for the failure. Lack of testing of the implemented system-of-systems; errors in

the transmission of data between the baggage-handling and baggage-reconciliation systems; and errors in the transmission of flight data between BAA and a third-party contractor were reported to have played a significant part in the cancellation of five hundred flights and manual processing of approximately twenty-three thousand bags over five days.

The Theatre Battle Management Core System studied in [1] is an air command and control system-of-systems integrated to perform secure, automated air battle planning and management for the US Air Forces and land, and maritime allies. Component systems were mainly legacy, including the Joint Maritime Command Information System, and Contingency Theatre Automated Planning System, Wing Command and Control System, Integrated Imagery and Intelligence System, Airborne Warning and Control System, and Joint Surveillance Target Attack Radar System. Interfaces were listed to over twenty systems.

[1] reported the system-of-systems actual delivery was to be eighteen months following award of contract. Instead, it was delivered three years late with cost estimates at tens of millions of dollars. Key reasons for this delay were: governmental instruction for the contractor to prioritise improvement of the legacy Contingency Theatre Automated Planning System component system; immaturity of modern third-party software applications and their failure to operate over the legacy communications components; requirements creep (a requirements baseline was not established for Theatre Battle Management Core System by the government. Initially, the government expected the contractor to control the requirements baseline); and pressure from the user community for the Theatre Battle Management Core System meant the test planning process was short-circuited leading to test failure on two occasions (the contractor was originally given the role of orchestrating testing).

Wentz's testimony of Bosnia [32] provides further insight into the difficulties achieving an integrated communications and information system-of-systems (CIS) in the military domain:

'the challenge facing NATO and the nations was to build a long haul and regional CIS network out of a mixture of military and commercial equipment that would vary widely in age, standards, and technology and would be built very quickly once given the order to deploy. Putting the pieces of the puzzle together would most likely not result in a true system of systems. Furthermore, there would be a need to interface systems that had not been planned or designed for interfacing. The independent national systems would be tied together, not engineered as a single system. Given the uncertainty of the situation it would most likely be a case of integrating what you get, not necessarily what you need, and then making the best of it'.

One of the conclusions in Table 1.1 (section 1.3.4) was that systems-of-systems rely on this communication (usually realised by information technology) between constituent systems. The present means of integrating the systems involved in a military mission are tactical data links. These support message transfer within the system-of-systems. As discussed, these constituent systems are non-trivial systems in their own right i.e. they are not controllable, 'simple' systems built from basic parts using fixed interfaces. Their integration in a system-of-systems is achieved by a communication service between interfaces of the component systems. At an interface,

a component system's specification can be reduced to the functional and timing description of services required for the integration along with its quality of service. As indicated by [3, 32, 133, 134], as well as ensuring selection of constituent systems whose behaviour is likely to contribute to the overall purpose of the system-of-systems, their integration involves enabling the required level of communication between them.

This required level of communication is the foundation of the integration process and is termed 'interoperability'. Interoperability is now defined for the purposes of this thesis.

Environmental object data captured by sensors requires frequent interoperable exchanges of complex information between systems. Simpler web browser communication between a customer and a home banking system also demands interoperability. Due to different contextual interpretations of interoperability, many definitions exist for it, including these four from the IEEE [36]:

'the ability of two or more systems or elements to exchange information and to use the information that has been exchanged'.

'the capability for units of equipment to work together to do useful functions'.

'the capability, promoted but not guaranteed by joint conformance with a given set of standards, that enables heterogeneous equipment, generally built by various vendors, to work together in a network environment'.

'the ability of two or more systems or components to exchange information in a heterogeneous network and use that information'.

These IEEE definitions are incorporated into definitions of interoperability used by the US Department of Defence (DoD):

'the ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together' [17].

'The condition achieved among communications-electronics systems or items of communications-electronics systems equipment when information or services can be exchanged directly and satisfactorily between them and/or their users. The degree of interoperability should be defined when referring to specific cases. For the purposes of this instruction, the degree of interoperability will be determined by the accomplishment of the proposed Information Exchange Requirement (IER) fields' [34].

These definitions do not qualify what services are referred to in each case but the fourth definition below from the MoD's Integration Authority [38] refers to communication and information services and it is assumed that the same communication and information services are referred to by [17, 34].

'(1) Ability of information systems to communicate with each other and exchange

information. (2) Conditions, achieved in varying levels, when information systems and/or their components can exchange information directly and satisfactorily among them. (3) The ability to operate software and exchange information in a heterogeneous network (i.e., one large network made up of several different local area networks). (4) Systems or programs capable of exchanging information and operating together effectively' [35].

This definition of Communication/Information Systems/Services (CIS) interoperability was put forward by the MoD's Integration Authority:

'The ability of systems, units or forces to provide (communication / information) services to and accept (such) services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together [25]' [38].

Interoperability is defined for the purposes of this project as:

'the ability of a set of communicating entities to exchange specified data by electronic means and operate effectively using that data according to specified operational processes'.

Achieving interoperability between systems that originally did not interact or within new systems has posed a difficult challenge for the public and private sectors. Reasons for this include: at the start of a system-of-systems development project often little is known about interoperability requirements. Systems that will interoperate may not yet be conceived or constraints imposed by existing systems compromise approaches to achieving interoperability; maintaining compatibility with older systems sometimes conflicts with achieving interoperability between newer systems; lack of maintenance funding to cover upgrades or fixes for older systems; interoperability between systems is specified in transitive form. This implies that because system A is interoperable with system B and system B is interoperable with system C, then system A will be interoperable with system C; standards and models for architecture have been developed to ensure interoperability but contain ambiguities and inconsistencies. Used in isolation these standards are insufficient for achieving interoperability; the complexity of the systems being built mean a high number of contractors are required. Processes have not been established between contractors to ensure required levels of interoperability; ambiguity of terms used inter and intra-organisationally can be conflicting; and operational context addresses how a system is used and is described in military doctrine. For interoperability between multiple systems, doctrine also must be interoperable.

In order to pinpoint the aspect of interoperability this thesis addresses, it is useful to identify interoperability in terms of the Open Systems Interconnection (OSI) Seven Layer Model shown in Fig. 1.1:

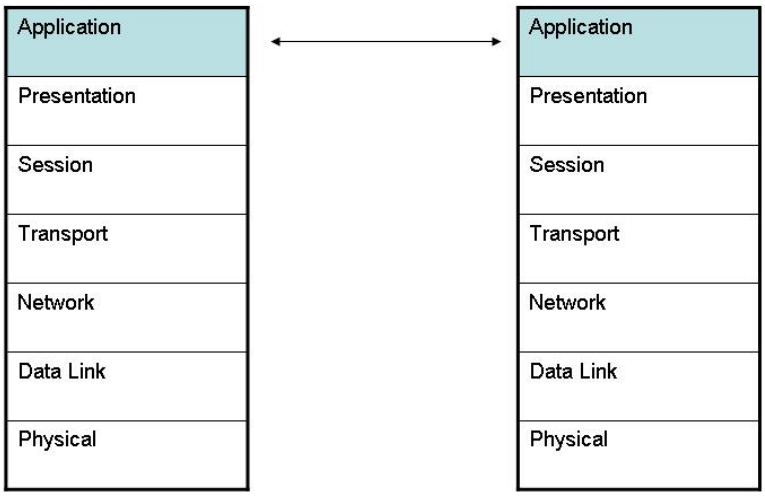


Fig. 1.1 'OSI Seven Layer Model and Interoperability'

In the OSI Model (Fig.1.1), physical exchange of data takes place at the physical layer, e.g. via telephone or radio links. Encryption, error correction and link management are example functions of the data link and network layers. Procedural interoperability takes place over the next upper four layers with the transport layer looking after correct message exchange and the session layer addressing message sequencing. The presentation layer deals with message formatting. At the application layer, messages are presented to the host and interpreted by application layer protocols such as Simple Mail Transport Protocol (SMTP), or File Transfer Protocol (FTP), or an application programming interface such as Winsock or Berkeley sockets. In each case, a well-defined, standard interface is provided detailing the information messages that can be sent and received to each application. The OSI model could also be extended with additional layers above the application layer such as a 'host operator application' layer representing the programs the host operator can interact with through a human-computer interface (and the information messages associated with the host operator application), or a 'human-to-human' layer, representing interoperability at the host operator level (i.e. operator interpretation of the host operator application they are interacting with).

This thesis is concerned with messages constructed and exchanged to satisfy associated system-of-systems information exchange requirements at a 'host operator application' level.

Fratricide of friendly soldiers during operations such as Desert Storm to liberate Kuwait (1991) highlighted the ultimate penalty of the military's inability to assure interoperability. In response, the military moved towards a standards-based approach (with mandatory compliance) to development of systems-of-systems. However, building military systems-of-systems has to take into account a huge number of legacy systems from which systems-of-systems will be composed. The majority of these existing systems do not comply with the current version of standards. This is due to the rate of technology change or fact their development preceded the introduction of a common set of standards. Each legacy system has been developed independently according to different versions of military rules and standards covering data representation and protocols. To make matters worse, these text-based standards are often ambiguous, open to interpretation by system designers, and can be deviated from. Unsurprisingly, legacy systems to be integrated often fail upon connection or compromise the stability of existing services offered by other component systems.

These legacy systems cannot simply be discarded as many have taken as long as fifteen years to acquire at significant cost. Vast amounts of data on reliability exist in the software that re-writing would eradicate. In addition, defence budget cuts and the lengthy timescale required for procurement ensure the longevity of these existing systems (typically up to thirty years). Legacy systems continue to present an integration challenge to both the commercial and defence sectors. However, it is the attributes of high quantity, long lifecycle, and long acquisition cycle belonging to military legacy systems that make their integration a particularly unique problem for defence.

To tackle this problem, there needs to be a comprehensive system-of-systems description available to all designers by which interfaces can be determined. Without such a description and framework to achieve, designs will fail to achieve interoperability. As part of their common standards approach, both defence and commercial sectors evolved architecture frameworks such as the Zachman Framework [28], Department of Defence Architectural Framework (DoDAF) [29], and Ministry of Defence Architecture Framework (MoDAF) [30] to adequately describe systems-of-systems.

An architectural framework such as that standardised by ISO/IEC 42010 [37] aims to provide a basic framework or checklist for describing the content of an architecture, taking into account the environment of the system of interest. IEEE 1471 upon which [37] is based, defines an architecture as:

'the fundamental organisation of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution'.

Architectural descriptions are structured to meet the needs of the system owners (e.g. users, developers, component vendors, maintainers) using multiple views of the system, with each view covering an identified set of system concerns (e.g. reliability, functionality, security, data integrity, usability). In order to comply with ISO/IEC 42010 [37], an architectural description has to identify the owners of the system and their concerns; define viewpoints to address those concerns; define views of the architecture satisfying those viewpoints; and document rationale for decisions made in

the architectural description. ISO/IEC 42010 [37] defines a framework for an architectural description but does not recommend a process on how to produce one. Usually an organisation undertaking architecture description does so within the context of a well-defined engineering process. The intended consequence of using architecture frameworks is more understandable architectural descriptions, leading to improved architectures and system products.

Design and construction of a new building is a useful parallel to architecture frameworks drawn from the civil engineering discipline. In designing a multi-storey building, an architect will elicit customer requirements and translate them into various views (blueprints) of the proposed structure. A civil engineer will then develop structural drawings and plans to build the new structure. The civil engineer builds the structure using the framework from the architect, i.e. the set of blueprints for the proposed building which has different views covering the physical construction of the building; detailed construction of each floor; heating and air-conditioning systems; and plumbing, electrical wiring, and communications. These blueprints provide the concept of the proposed building to help all owners understand the design being proposed.

Building such a complex structure without a visual framework would result in disaster. Here, the set of blueprints present many views of the new building, each of which is important to its overall construction. Compare this with a criminal trial in which the prosecutor combines physical evidence, motive, timing of events, and defendant accessibility to the scene of the crime in order to present a visual picture to the jury. Failure of the prosecutor to provide this integration of views on the crime means the jury may not be able to visualise a realistic picture of it. Relaying the concepts of a problem to be solved using multiple viewpoints are key to design of complex systems. Here, presenting the problem concept is essential to shared understanding of the problem. Multiple views enrich the conceptual view of the problem and the potential solutions to it.

Architecture frameworks provide a means to visually describe architecture products. They do not prescribe how to do this with a process. Even the boundaries of the traditional systems engineering process are stretched by systems-of-systems integration problems [9, 24]. The reality is that most large-scale engineering projects which follow traditional systems engineering practices are less successful [24]. It is widely assumed that for system-of-systems applications, new technology will be used; the new technology is based on a clear understanding of the principles that govern the system-of-systems; project objectives and specifications are clearly understood; and a design will be implemented based upon these specifications so that the application will be achieved.

In reality, it is highly unlikely that only new technology will be used for systems-of-systems. In addition, there may not be a clear understanding of the governing factors of the system-of-systems due to their complex characteristics. Of equal importance are the human, organisational, policy and political environments that influence feasible solutions [9]. While the overall purpose of the system-of-systems mission might be clear in high-level, abstract form, the specific objectives are most likely poorly defined and ambiguous [9, 24]. There can be no conception or a priori planning of how a system, and systems-of-systems will evolve in the future.

Traditional systems engineering fails to take into account the ambiguity and requirements shift present in dynamic systems-of-systems environments [9]. It tends to place human, organisational, policy, and political environments in the background during system problem analysis and resolution. Systems-of-systems integration means addressing these environments as key constraints during these phases [9]. Traditional systems engineering succeeds in realising complete system solutions through iterative development processes where as design of systems-of-systems often means deployment of a partial solution followed by iterative development again. This goes against traditional systems engineering which usually aims to complete design with implementation of the system [9].

System-of-system designers either lack such an architecture framework showing how system components should interface, interact, and fulfil overall system requirements or a suitable process based on systems engineering principles for building one. The architecture framework should help define the system-of-systems in terms of component systems and the interactions between these components. Also, the resulting architectural description can serve to document the rationale for design decisions, helping to audit requirements and their realisation. Although an architectural description will not guarantee that a system meets its requirements, a poorly designed architecture makes it nearly impossible for designers to develop a system that meets its requirements.

This is further supported by [31] where issues hindering system-of-systems interoperability were summarised as: unclear requirements documentation and poor configuration management; no method for validating and governing interoperability requirements; interoperability requirements are not defined early or identified as a common development goal; no path leading to a view (architecture) that can be used for a statement of specification for a material developer or test criteria by the developer; no direct link from requirements to end product; no tools to adequately model interoperability; the need to wait until post-integration to check whether interoperability has been achieved (on a trial and error basis); lack of application of systems engineering principles to capability development and gap analysis (multiple uncoordinated organisations work on solutions to solve similar problems); system-of-systems 'lessons learned' are not managed so the benefits of experience and impact of 'requirements creep' are not considered; and finally, inter-agency system-of-systems requirements are not clear, interoperability is not guaranteed and testing results are questionable.

New layers of functionality are often added to implementations initiated from poorly designed system architecture views with no clear insight into the overall organisation of the system-of-systems. Where there are detailed system specifications, these tend to address the 'leaf' level of the system-of-systems requirements decomposition tree, i.e. at the component system level and not the system-of-systems context level. Using these system-centric or 'stove-piped' specifications in isolation means that systems engineers struggle to deliver interoperable, effective capability for defence or industry.

As well as their lack of a standard process in terms of system-of-systems specification, another drawback in the way architecture frameworks have been used to

date is that they are a static, textual documentation model of a system open to interpretation by system owners, designers, and developers.

[39] describes a model as:

'..the essential nature of a process or thing. They are not the thing itself. Models are validated only when they have been verified by observation and measurement under controlled conditions'.

[39] argues that modelling can help deploy better quality systems in less time and for less money by:

1. Executing the model of the system specification in order to observe system behaviour.
2. Transforming model languages into different modelling languages (automatically) in support of the multiple disciplines involved in the lifecycle of a system.
3. Enhancing systems engineering with unambiguous, executable modelling.

Use of modelling aims to improve the design of a system in terms of its length; completeness, correctness, and consistency of specification; preservation of the design (i.e. a form of documentation) for subsequent system lifecycle stages; and unambiguous specification.

The adequacy of a systems engineering modelling language depends on its ability to enable a complete description of the contextual real-world domain using unambiguous language constructs. Generally, a good modelling technique should be:

1. Precise in describing both static and dynamic system properties.
2. Standardised and open to promote acceptance and popularity.
3. Precise in identification of requirements allowing owners and designers to reach agreement about what should be accomplished. A graphical concrete syntax can help promote this shared understanding.
4. Applicable to different styles of modelling (state-based, event-based, data-based).
5. Based on formal techniques and suitable for automation (execution or simulation).
6. Intuitive to use with: graphical interface supporting execution of design models; conformance to known standards and techniques; and compatible with other toolsets implementing the same technique.

A model can be:

1. Informal: providing only descriptions of functionalities (e.g. UML).
2. Formal: all elements can be described mathematically and support simulation, verification (syntax, semantics, and model structure and logic may be tested automatically), and transformations (mapping of a model to a lower abstraction level for design, development or verification purposes).

From a system-of-systems perspective, a modelling technique should offer:

1. Abstraction: support different views of the system-of-systems architecture capturing characteristics appropriate to the abstraction level without imposing any particular solution upon designers and developers.
2. Modularisation: define and organise parts of the system-of-systems.
3. Data typing: model concepts of the domain as closely as possible.
4. Adequate toolset support.
5. Timing: achieve performance predictions before the system-of-systems is physically implemented in conjunction with simulation. Each system-of-systems application provides different capabilities and varying degrees of criticality that need to be analysed using timed simulation.
6. Verification and validation: check model correctness in terms of syntax, semantics, structure (absence of deadlocks, livelocks, and correct termination), and logic through simulation and calculation of state space graph. In critical applications, correctness of specifications is vital to achieve before their physical implementation.
7. Precision in specification of requirements: a graphical concrete modelling language syntax can help promote shared understanding between technical and non-technical audiences.
8. Scalability, concurrency, state, information, and event-based specification.

While modelling languages can be used in their own right to specify systems [135], text-based products of architectural frameworks have been translated into different modelling languages for simulation purposes [40-42]. There have also been attempts to use UML as the primary means of describing architecture framework products [44, 76] so that a proprietary UML modelling toolset such as IBM Telelogic Rhapsody can be used to simulate the model, or the UML can be translated to a formal modelling language (Petri nets) for subsequent execution [66]. These attempts focused on providing process guidance and improving architecture frameworks for system-of-systems engineering. They do not consider the actual specification capability UML offers from a system-of-systems modelling perspective and where the language can be complemented and improved in terms of the system-of-systems engineering problems related to specification and analysis.

UML is intuitive, multi-purpose, has a graphical concrete syntax, and does not prescribe a process. It is also ambiguous and imprecise, and viewed as a system of modelling languages, each with its own particular focus to represent a system under development. Each diagram, represented by its own language, can be used in a number of situations, e.g. class diagrams can be used at analysis, design, and architecture stages of the lifecycle.

Structured using a meta-model, due to its semi-formal nature, UML lacks simulation and exhaustive verification capability. This shortfall in UML has received little attention in the context of system-of-systems. Petri nets were selected as a potential means of addressing the industrial need of assuring that system-of-systems implementations meet original co-ordinator requirements through adequate capture, verification and validation of these requirements in the system-of-systems design. Although Petri nets have been used in [40] to offer simulation and analysis of architecture framework products, the work did not determine how Petri nets can be used to improve specification and analysis of a dynamic model of a system-of-systems specified using UML.

1.4.1 Overall Problem Statement

This thesis addresses the system-of-systems level design specification and analysis problem. Specifically:

1. There is no complete, correct, and consistent capture of the problem and solution design specification (particularly information exchange) at the system-of-systems level.
2. There is inadequate verification and validation of the design specification providing low levels of assurance that the design will lead to desirable implemented behaviour.

1.4.2 Research Issues

Two major research issues can be identified within the overall system-of-systems level design specification and analysis problem:

1. Clarifying where the dynamic, behavioural diagrams of UML can and cannot be used to model and analyse system-of-systems.
2. Determining how Petri nets can be used to improve the specification and analysis of the dynamic model of a system-of-systems specified using UML.

This thesis presents the strengths and weaknesses of Petri nets in relation to the specification of system-of-systems and shows how Petri net models can be used instead of conventional UML activity diagrams. The design specification model of the system-of-systems can then be analysed and verified using Petri net theory.

1.4.3 Problem Boundaries

This thesis focuses on the specification and analysis of system-of-systems engineered towards a common purpose.

1.5 Research Aims and Criteria for Success

The aims of this research and criteria for success cover the problem and solution space. Petri nets and conventional UML behavioural diagrams are examined in relation to the specification of system-of-systems to show how Petri nets can help the system-of-systems specification and analysis problem. Case studies are then conducted to demonstrate the viability of the use of Petri nets in the specification and analysis of system-of-systems. The criteria for success are:

1. To address the first main research issue, indicate the strengths and weaknesses of the behavioural diagrams of UML regarding the specification, and verification and validation of systems-of-systems.
2. To address the second main research issue, determine the strengths and weaknesses of Petri nets regarding the greater formalism of dynamic behaviour in systems-of-systems, i.e. their specification, and verification and validation. This should cover Petri nets' ease of use; comprehensibility; scalability; state, data, and event-based modelling capability; concurrency modelling capability; and verification and validation capability. The role of Petri nets as a means of engaging stakeholders should also be examined.

3. Show how Petri nets can be used instead of UML activity diagrams to address the overall problem of system-of-systems specification and analysis.
4. Demonstrate and evaluate the feasibility of the Petri net solution to the overall problem of system-of-systems specification and analysis using a case study approach.

Chapter 7 presents a discussion on the success of this research in relation to the above criteria.

1.6 Evaluation Criteria

The main objective of this work is to define how Petri nets can improve the specification and analysis of systems-of-systems using a case study approach. Chapters 5 and 6 present evaluations of the Petri net technique based on the following criteria:

1. Design Quality (scalability and representational ability).
2. Functional Correctness.
3. Toolset Issues.

1.7 Contribution

The main contribution of this work is greater formalism of dynamic system-of-systems behaviour specification using Petri nets. Two research issues within the main problem of system-of-systems specification, verification and validation are addressed:

1. Clarifying where the dynamic, behavioural diagrams of UML can and cannot be used to model and analyse system-of-systems.
2. Determining how Petri nets can be used to improve the specification and analysis of the dynamic model of a system-of-systems specified using UML.

1.8 Thesis Structure

This thesis consists of seven chapters.

Chapter 1 introduces the context for the research, discusses the main problem to be solved, and sets out the aims of the research and criteria for success.

Chapter 2 introduces the Petri net technique, including why they should be useful in the specification and analysis of systems-of-systems.

Chapter 3 discusses the case study research method used to demonstrate and evaluate the strengths and weaknesses of Petri nets in chapters 5 and 6.

Chapter 4 describes the strengths and weaknesses of Petri nets in relation to the specification of systems-of-systems and indicates how Petri net models can be used instead of conventional UML behavioural diagrams to analyse and verify the system-of-systems using Petri net theory.

Chapter 5 executes the case study design described in chapter 3 for the first study in the case study research approach used by this thesis.

Chapter 6 executes the case study design for a second time in order to investigate the results that needed further clarification from the first study and replicate the results obtained from the first study in the second, demonstrating experimental reliability and triangulation.

Chapter 7 concludes the thesis by summarising the main problems associated with the engineering of systems-of-systems and the solution achieved by the thesis. The success of the Petri net formalism of system-of-systems behaviour specification is discussed in terms of the criteria presented in section 1.5 and further research opportunities are suggested.

The Appendices contain an initial investigation into Petri nets, a description of the Petri net enhancement to UML proposed by this thesis, and the Petri net toolset selection exercise. They are followed by a list of references.

Chapter 2 Petri Net Specification Framework

2.1 Introduction

Chapter 2 provides an introduction to the Petri net formalism. An indication of why it should be useful in the specification and analysis of system-of-systems is also discussed. This introduction serves as the context for the demonstration and evaluation of their strengths and weaknesses in relation to system-of-systems specification in the case studies of chapters 5 and 6.

2.2 Petri Nets

Seen as a generalisation of state machines, Petri nets have been around for almost fifty years (developed by Carl Adam Petri in his doctoral thesis [58]). They can be used graphically and mathematically to communicate between technical and non-technical audiences and construct behavioural models of process-oriented systems. In [45], Van der Aalst suggests the following as main reasons for using Petri nets as a process modelling technique:

1. Their graphical nature has underlying formal semantics.
2. They are state-based and event-based.
3. A range of analysis techniques is available to examine properties of the modelled system expressed as a Petri net.

The analysis techniques referred to here are static (reachability tree and matrix equation representation are the two methods used to verify a number of useful Petri net properties such as reachability, liveness, boundedness, and home state) and dynamic (execution or simulation of the Petri net is used to verify its behaviour and validate it in terms of performance if timing is introduced into the net).

To improve the usefulness of classic Petri nets, Petri nets have been extended over time to incorporate hierarchy (for structuring models), colour (for modelling attributes) [63] and time (for performance analysis). In classic form, their basic concept is that of an underlying directed, bipartite graph with a starting state called the 'initial marking'. The underlying graph is directed and weighted and consists of two node types, 'places' and 'transitions'. These nodes are connected by directed, weighted arcs known as 'input arcs' and 'output arcs'. An input arc goes from a place to a transition and the set of places with input arcs going to a particular transition are called the transition's input places. An output arc goes from a transition to a place and the set of places with output arcs from a given transition are called the transition's output places. Arcs can only go from a place to a transition or vice versa. Places and transitions are graphically represented by circles and rectangles respectively. Usage of Petri nets normally interprets transitions as events or activities and places as triggers or results of these events [48, 65]. Murata suggests some typical application of transitions and their input and output places in [57].

A 'marking' assigns a non-negative integer 'x' to each place. This means the place is mapped to x tokens (shown as identical black dots). Tokens are dynamic objects

whose movement between places is controlled by the Petri net's transitions. Places can have a finite capacity restricting the maximum number of tokens they can hold. Transitions have a certain number of input and output places that represent pre and post conditions for the enabling or 'firing' of a transition. The state of a Petri net is determined by the distribution of tokens over the places. This distribution can be used to define situations such as satisfied conditions and resource availability. Reachable state refers to a state reachable from the current state by firing a sequence of enabled transitions. Dead state is a state where no transition is enabled. By setting an initial marking of a state and adhering to the following firing rules it is possible to model and execute (or simulate) processes modelled using nets:

1. Transitions are enabled if each of their input places is marked with the same number of tokens as indicated by the weight of the arc leading to the transition.
2. Enabled transitions may or may not fire.
3. Firing of an enabled transition is atomic and consumes a number of tokens (dictated by its corresponding input arc weight) from each input place and adds a number of tokens to each output place (dictated by its corresponding output arc weight).

The order in which Petri net transitions fire is known as a firing sequence. Depending on the marking, a Petri net can have a number of different firing sequences that occurs if more than one transition is enabled during the firing sequence. A Petri net can also have a number of defined properties, some of which are outlined below. Other properties can be found in [57].

1. Conflict (or confusion): if more than one transition is enabled simultaneously and the firing of one of these transitions will disable the remaining enabled transitions then a conflict is said to exist between transitions for a certain marking. In such situations, the transition that fires is dictated by firing rules. This property can be used to show the effect of resource-sharing on system performance. A number of strategies can be applied to resolve conflict [67]. These include using timed transitions (the timed transition with the shortest time fires) and transition weights (the immediate transition to fire is determined probabilistically using the weights).
2. Deadlock: when no transitions can fire and the execution of the Petri net is halted, deadlock is said to occur. A Petri net is known as 'live' for a particular initial marking if it is deadlock-free.
3. Reachability: given a marking M_{i+1}, this marking is said to be immediately reachable from a marking M_i if a transition enabled by M_i fires to give M_{i+1}. A marking M_{i+n} is said to be reachable from M_i if a transition firing sequence exists such that after firing all of these transitions the resultant marking is M_{i+n}. A reachability tree or graph describes the possible markings of a Petri net starting from the initial marking (its root). Below this marking each possible immediately reachable marking is listed together with directed arcs labelled with the corresponding transition required to reach the marking. This process is repeated for these markings. If a generated marking is the same as one which appears earlier in the tree it is connected to this via an arc labelled with the corresponding transition. For certain Petri nets this process can continue indefinitely and necessitates formation of a coverability tree. Here, any set of markings which differ only by the number of tokens found in unbounded places are represented by one marking. A symbol is placed in the unbounded places flagging up that the number of tokens in that place is unbounded. Reachability or coverability trees can be used to determine safeness, boundedness and reachability of a Petri net.

4. Boundedness: if the number of tokens in a place never exceeds x then the place is said to be x-bounded. When all places in a Petri net are bounded it is called a bounded Petri net. This maximum number of tokens allowed on a place can be used to specify the maximum length of a queue. Unbounded places can cause bottlenecks. Petri nets where all places are one-bounded are known as safe and can be used to model computer systems as the state of each place can be represented by a one or a zero.

2.2.1 Petri Net Example

The following Petri net serves as an introduction and represents the process of a chemical reaction involving two units of hydrogen and one of oxygen. Net places serve as storage for tokens and capture the state of the reaction. The transition represents an event or activity taking place, in this case the reaction. Each input place in Fig. 2.1(a) has three and two tokens respectively. As the input arc associated with place hydrogen has a weighting of two (specifying that two units of hydrogen are required for the reaction) and the input arc associated with place oxygen has a weighting of one (specifying one unit of oxygen is needed), the transition is enabled. After firing, the marking changes to the one in Fig. 2.1(b) and the transition is no longer enabled:

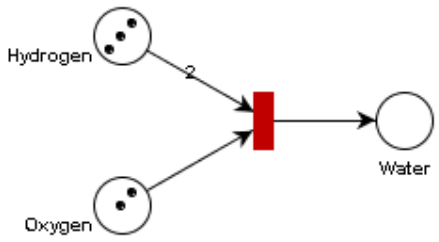


Fig. 2.1(a) 'Classic Petri net of chemical reaction with an enabled transition' adapted from [57], p543

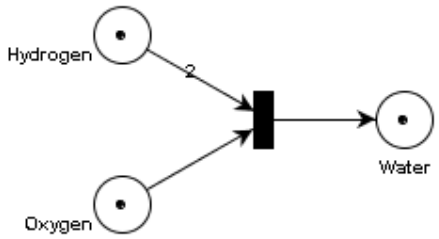


Fig. 2.1(b) 'Classic Petri net following transition firing' adapted from [57], p543

Fig. 2.1 clearly shows the state change and corresponding executed activity via the removal and addition of tokens and firing of transition modelling elements. The current process state is given by the placement of the tokens and the arcs indicate that there is a relation (function) between a place and a transition.

2.2.2 High-level (Coloured) Petri Nets

Coloured Petri nets enhance classic Petri nets [43] with data typing abstraction and manipulation capability similar to that of a high-level programming language. Here, tokens have colour (values) to combat complexity found in classic, uncoloured Petri nets attempting to model real-life systems. Uncoloured nets can be compared to assembly language in that tokens are essentially markers in the net, relying upon the labelling of places to express the state of a system and requiring multiple net elements to do so. Coloured tokens enable real-life objects and their attributes to be represented in the model through the introduction of data typing abstraction.

Similar to classic nets, coloured Petri nets have a fully formal, mathematical underpinning that provides a basis for analysis of model properties. Well-known high-level nets using colour are Predicate/Transition nets [46] and Coloured Petri nets [63, 80]. Visual representations of coloured Petri net models can be developed and analysed using graphical tools such as CPN Tools [47].

Aspects of a system are modelled using a small set of elements in coloured nets, the rules of which are described informally as: occurrences of activities or transitions depend on the data type and token colour (value) associated with the input place(s) of the transition and the enabling condition(s) specified by the input arc expression(s) of the transition. Equally, if the transition is enabled and occurs, tokens are added to the output places of the transition with values determined by the output arc expression(s) of the transition and data type associated with the output place(s). The formal mathematical definition of this informal description is defined in [63, 80].

2.2.3 Timed Petri Nets

To enable enhanced specification (scheduling) and performance evaluation of nets, time delays have been associated with transitions [49] and/or place net elements [50]. There are many extensions to Petri nets related to time, for example [80, 83].

Petri nets are deterministic timed nets if the delay is known, or stochastic timed nets if the delays are random, or deterministic and stochastic timed nets if a combination of fixed and random delays are present. In stochastic nets, firing time is associated with each transition indicating the delay from when the transition is enabled until it fires. Usually, the transition with the minimum remaining firing time affects the next marking of the net. Following this marking update, each newly enabled timed transition obtains a delay from the delay distribution and each timed transition enabled in the previous marking (and still enabled in the current marking), keeps its remaining delay. Transitions disabled in the current marking lose their remaining delay. Common stochastic Petri net models are by [51] and [49].

Deterministic and stochastic nets contain immediate transitions (when enabled, fire without delay), stochastic transitions (when enabled, fire after some delay sampled from a distribution), and deterministic transitions (when enabled, fire after a constant delay). Enabled immediate transitions have firing priority over enabled timed transitions. Multiple enabled immediate transitions should be specified with firing probabilities to resolve firing conflict.

Coloured Petri nets can be created with or without timing. Jensen [80] extended coloured Petri nets with timed coloured Petri nets. With these nets a global clock is introduced for the net model. The state of a timed coloured Petri net consists of a marking and the global clock time. Timed coloured Petri nets can contain both timed and un-timed coloured tokens. Timestamps are controlled by initial marking, transition or output arc expressions where discrete and probability distributions can be used to define the time taken for a transition to fire. Timestamps allocated to the tokens must be less than or equal to the current model time in order to be removed. In this way, timed transitions represent the time taken by the system to perform a given task. Un-timed enabled transitions fire in zero time.

2.2.4 Hierarchical Petri Nets

Hierarchical nets were developed to allow models of large-scale systems to be created using both top-down and bottom-up approaches. Detail at a certain level of abstraction can be hidden or exposed as needed in a similar way to subroutines in programming. This abstraction mechanism makes model development and modification easier. Jensen [63] defines hierarchy implementation for coloured nets.

2.2.5 Analysis

Existing analysis methods for timed, coloured Petri nets are simulation; reachability analysis; and Markovian analysis.

Simulation helps to predict the behaviour of the modelled system but it is not possible to exhaustively check a system has a desired set of properties by this method. As well

as conducting extended simulation runs to test assumptions and performance, simulation can be visualised (animated), helping communicate behaviour to technical and non-technical audiences.

Reachability analysis builds a reachability graph (reachability tree or occurrence graph) with nodes representing the possible system states and arcs representing each possible state change. This method is an exhaustive way of checking properties of the modelled system. A disadvantage of this method of analysis is the fact the reachability graph can become infinitely large and various largeness avoidance and reduction measures have been considered [52, 91-93]. For stochastic timed coloured nets sampling from an exponential probability distribution, the net can be translated into a continuous time Markov chain for the purposes of performance statistics generation. Normally simulation-only analysis is reserved for timed coloured nets.

2.3 Petri Nets and the System-of-Systems Problem Areas

As discussed in chapter 1, the main problems for system-of-systems engineering surround requirements specification, interfaces and protocols specification, and verification and validation of the design specification. These are summarised as follows:

1. There is no adequate capture of problem and solution design specification (particularly information exchange specification) at the system-of-systems level.
2. Assurance that the design will lead to desirable implemented behaviour (through verification and validation) is also lacking.

Currently, in terms of specification, chapter 1 highlighted that modelling using architecture frameworks (e.g. Zachman, DoDAF, and MoDAF) and graphical modelling languages (UML, IDEF, Petri nets) had been implemented to help improve large-scale, system-of-systems specification. Adoption and maturity of these initiatives in system-of-systems specification are still relatively low, lacking guidance and used for specification at the systems (rather than the system-of-systems) level. Chapter 1 also indicated that although Petri nets had been used in conjunction with architecture framework products to provide simulation and analysis capability, no work has directly investigated their strengths and weaknesses in terms of system-of-systems specification and analysis. In addition, based on these strengths and weaknesses, no work has considered how they can improve a system-of-systems specification captured in a UML dynamic model.

A number of desirable features for a modelling language used in the specification of systems-of-systems were suggested in chapter 1. Each of these is now discussed in relation to the Petri net formalism and reference is made to a simple example (a telephone system) in Appendices A-D to further illustrate the potential applicability of Petri nets with respect to that feature.

1. Abstraction
The ability to support a range of views of the system-of-systems architecture at different levels of detail without imposing any particular solution upon designers and developers.

Based on the initial familiarisation work with Petri nets (Appendix A), it was clear that use of classic Petri nets resulted in flat, large, complex nets even with a narrowly scoped system such as the telephone system. The nets produced represented the telephone system at a high-level of abstraction, i.e. that of an operational process. Classic nets provided no formal means of linking activities represented by transition net elements in greater detail in the same model. To achieve this, a separate net model would need to be constructed but there would be no formal connection.

Instead, classic nets were abandoned in favour of high-level, hierarchical Petri nets, specifically coloured Petri nets with hierarchy [63]. Two language constructs can be used to enable hierarchy within one net model, substitution transitions and fusion places.

Hierarchy can structure complex Petri nets in a similar way to hierarchy within data flow diagrams and subroutines in programming. The challenge using hierarchy is deciding upon an appropriate abstraction level and viewpoint for the model. With substitution transitions (Appendix A, section A.3.2), a net at a certain level of abstraction (parent net) can have some or all of its transitions described in a greater level of detail by subnets (top-down decomposition). These subnets can be composed of places, transitions and other subnets. Also, hierarchy can be facilitated by linking existing lower-level subnets to transitions within parent nets (bottom-up development). The parent net aims to provide a coherent overview of the modelled process and indicate clearly that more detailed descriptions of its main activities are available on subnets. The toolset selected for use in the thesis, CPN Tools, also allows instantiation of a subnet in that once it has been defined, the same subnet can be re-used by different transitions in the model (each with independent input and output values, similar to parameterised procedure call in programming). The toolset evaluation exercise is presented in Appendix F.

As mentioned, hierarchical nets can also be constructed in a bottom-up fashion. For large-scale system-of-systems, the concept of subnets as components is extremely useful both in terms of reuse of existing nets and as a means to explore variations in the design of components. Existing, amended or brand new nets relating to individual components of the system-of-systems could be substituted into and out of the composition when considering different application scenarios or designs of components. Substitution transitions make use of input and output socket places to and from the decomposed transitions on the parent net. These sockets have corresponding port places on the resultant subnet describing the decomposition. The colours (types) of these socket and port places can be used to specify the types of the information used and produced by the decomposed activity. In this way, sockets and ports can be viewed as a means of explicitly specifying required and provided interfaces to the decomposed transition.

Fusion places (Appendix B, section B.1.3) represent one conceptual place element so that when a token is added (removed) at one of the places in the fusion set, an identical token will be added (removed) at the other places. Fusion places can also be used to represent abstraction but there is no explicitly associated net page at a higher-level of abstraction (i.e. parent net). Each subnet on a page is independent and passes information to another subnet page using a set of places (fusion places). To mimic the abstraction depicted by the parent net of the substitution transition (port and socket)

hierarchical approach, a net can use the fusion places to pass (receive) information to (from) a subnet but the only way of associating the net at the higher abstraction level with the subnet at the lower abstraction level is via the labelling of the fusion places. Port and socket hierarchy makes the association with the lower abstraction level in a more explicit way.

An advantage of fusion places in abstraction is the ability to share the same information between multiple processes. With hierarchy and port and socket places, if information needs to be passed from an interface to more than one component at a point in time, sufficient copies of the token need be deposited on the interface place for consumers to remove. Hierarchy's main advantage is explicit abstraction. Use of either abstraction technique depends on the modelling context. Some examples of work where abstraction has been used include [77, 87].

2. Modularisation
The ability to define and organise parts of the system-of-systems specification.

Again, similar to abstraction, two language constructs can be used to enable formal modularisation within one net model, substitution transitions and fusion places. Also discussed in Appendix B, sections B.1.3-B.1.4, is the work of Petrucci et al [91-93] on modularity using fused transitions as a means to separate a flat net into modules but this thesis focuses on constructing non-flat, modular nets. Modules or component systems within systems-of-systems are likely to exhibit the characteristic of strong cohesion and loose coupling. Here, component systems are modular in the sense that they use communication interfaces (and protocols) to integrate and share information. As demonstrated in Appendices A-D, substitution transitions were the primary method of capturing component systems of the telephone process and their interfaces.

3. Data typing
The ability to model concepts of the domain as closely as possible.

Based on the initial familiarisation work with Petri nets (Appendix A), it was clear that use of classic Petri nets resulted in flat, large, complex nets even with a narrowly scoped system such as the telephone system. The tokens of classic nets are indistinguishable from one another in the sense that the places they are associated with have no data typing. Tokens are simply markers. This means the only way of capturing information represented by tokens in a classic net is through the labelling of their associated places, resulting in significantly increased numbers of net place elements and large, complex nets.

Coloured Petri net tokens are associated with place elements that do have a data type (colourset in CPN Tools) meaning the token can take on values (colours) specified by the data type defined for its associated place. In CPN Tools, a variety of data types are implemented within the toolset including simple (e.g. boolean, enumerated, string, integer), compound (e.g. product, record, list), and timed. Consequently, it is possible to use colourseets to help reduce net size and capture the domain being modelled. This is demonstrated by use of coloured Petri nets the telephone example in Appendices A-D.

Use of data typing is also essential for the specification of information to be exchanged at the interfaces of component systems in system-of-systems.

4. Adequate toolset support
Availability of a Petri net editing, simulation, and analysis environment adequate for the system-of-systems modelling requirements of the organisation.

Obtaining a comprehensive Petri net toolset can be achieved in three ways: developing the Petri net toolset in-house (this ensures all personal requirements are met but a disadvantage includes the time and effort involved. This effort can be short-circuited if there are suitable extensible frameworks available on which to build); compiling a toolset from existing Petri net graphical editing and analysis tools (again, a disadvantage is the time and effort involved in integrating the tools); or identifying a suitable existing integrated Petri net toolset and adapting it accordingly (this relies on the toolset being open and well supported in terms of documentation). Appendix F details the toolset evaluation and selection exercise conducted for the purposes of this thesis. For the purposes of modelling system-of-systems, key features of a toolset are its navigability of large nets; ability to execute nets; ability to analyse nets (state space graph calculation and temporal logic queries); re-use of existing nets (instantiation, configuration management); support for high-level, timed, hierarchical nets; and error reporting.

5. Timing
The ability to achieve both enhanced specification and performance predictions from the design before the system-of-systems is physically implemented. Each system-of-systems application will provide different behaviour and varying degrees of criticality that need to be analysed using timing (often in conjunction with simulation of the model).

To capture the efficiency or performance of a system and facilitate validation of its design, time-dependent actions such as timeouts, processing delays or deadlines are essential. As well as efficiency specification, time-dependent actions also enhance a system behaviour specification in terms of correctness. Activity ordering alone is insufficient to capture overall system behaviour precisely. Tokens representing information in large-scale systems will be processed according to the time they entered the system, time involved in their consumption and generation, and involvement in delays and transfer failures. Timing will be needed to specify the ordering multiple tokens receive (scheduling) over and above any activity sequence they experience. Timing information may need to be approximate, exact or both depending on the stage of development of the system. Classic Petri nets only include a basic concept of time in that actions (transitions) follow a particular execution order from an initial marking.

As indicated previously in section 2.2.3, Petri nets have been extended to incorporate the concept of time via their places, transitions, tokens, arcs or a combination of these. This thesis uses timed coloured Petri nets. CPN Tools implements timed coloured Petri nets and supports deterministic and stochastic model behaviour via discrete and continuous function provision associated with token type. As this thesis is concerned with large-scale, discrete event system-of-systems where their behaviour is (ideally) deterministic and terminating, use of CPN Tools was maintained. Continuous

specification will be required in physical monitoring at a lower level of (component system) detail.

Several research initiatives have been undertaken using timed Petri nets. These include: Christensen et al in [82] make use of timed coloured Petri nets to optimise the performance and capacity of a web server; Van der Aalst et al in [83] use interval timed coloured Petri nets to study rail time-tabling; Bulitko et al in [84] use time interval Petri nets to analyse real-time damage limitation on ships; Van der Vorst et al and Makajic-Nikolic et al use timed coloured Petri nets to examine supply chains [86, 87]; Dahl et al consider interval timed coloured Petri nets in penetration testing [85]; Kwantes uses timed coloured Petri nets to analyse a banking clearing process [88]; and Schomig et al use stochastic Petri nets to model business processes in [89].

All the approaches [82-89] are useful in providing guidance on development of performance models and contributing to parts of the validation of system-of-systems. Their approaches deal with continuous management, proactive and retrospective analysis of physical products but do not take into account the unique characteristics of system-of-systems. For development of system-of-system performance models, the owners in the process need to be considered, as well as the concurrent and co-operative nature of the provided and consumed functions realised by the processes and their components. In system-of-systems, intangible behaviour is realised using tangible resources in different environmental locations. The perceived quality of service of these intangible functions arises from the efficiency of the processes. Insight into the assessment of different ways of realising these intangible functions is needed at an early development stage as well as throughout the system-of-systems development stages. For example, an assessment model should help to answer whether an optimal combination of activities that leads to a reduced service response time exists.

At analysis, design and architecture stages of development, no physical components have been decided upon to realise the activities and processes specified. Of additional interest in system-of-systems design specification is using knowledge of (legacy or planned) physical assets to help optimise engineering of an operational process via analysis-of-alternative scenarios. As well as incorporating timing statistics, Salimifard et al [94] report on using nets to allocate physical resources and costs to activity execution.

Appendix C, section C.1, adapts the work in [94] to the telephone system and demonstrates the use of timed coloured Petri nets in its specification and performance analysis.

6. Verification and validation.
The ability to check model correctness and completeness in terms of syntax, semantics, structure (absence of deadlocks, livelocks, and correct termination), and logic through simulation (dynamic analysis) and calculation of state space graph (static analysis). In critical system-of-systems applications, correctness and completeness of specifications is vital to achieve before their physical implementation.

Due to their formal syntax and semantics, models produced using the Petri net language can be executed (simulated). Here, an execution algorithm is used to validate the behaviour of a system. Simulation can be used to detect undesirable behaviour, and incorrect or omitted logic but it is not an exhaustive means of checking correctness of the model. The toolset CPN Tools provides different simulation modes ranging from completely manual (interactive) to fully automatic. Interactive simulation is comparable to single step debugging a program and useful in initial investigations into model behaviour.

Simulation can also be configured to run repeatedly without graphical feedback and to bind variable values automatically, generating configurable analysis reports based on each automatic run. Runs can be initialised with a different selection of parameters per run. For example, simulation runs could be set up for the modelled system where input is initiated until a certain number is reached; different random distributions can be selected from per run; the number of resources available to a consumer in the net can be amended; further data collection points can be implemented to examine values across the net.

Normally used in conjunction with timing in the net, simulation can conduct performance analysis of the specified system [82]. With CPN Tools, timing delays can be introduced at various points in a model using exponential distributions or deterministic ranges to represent arrival times and delays between each activity. The net toolset permits extraction of data from certain places or transitions during simulation of the operational process. In system-of-systems specification, this information could be used to calculate timing delays for processes associated with particular components, individual or groups of activities, and the process as a whole.

Based on the statistics calculated from the model, ways can be considered as to how to improve efficiency. For example, additional resources could be added to the model and the model re-simulated to check its effect.

Appendix C, section C.1, illustrates the use of simulation in performance analysis of the telephone system and all Appendices (A-D) include its use in the initial validation of the constructed nets.

As dynamic analysis via simulation cannot guarantee that all possible execution paths of the process have been covered, static analysis of Petri nets is used to provide a more exhaustive, deeper level of verification over and above simulation alone. Static analysis using reachability tree or state space analysis was conducted to check for standard structural Petri net properties such as reachability, boundedness, home, liveness and fairness. It is also possible to use temporal logic to inspect the markings across the generated state space graph for further checking of expected model behaviour (CPN Tools implements ASK_CTL and it is possible for the modeller to construct non-standard property queries using this branching temporal logic). Using temporal logic, it is possible to combine pre-defined queries or construct new modeller-defined queries and undertake further checks related to model properties such as reachability and liveness. For example, the modeller may want to verify whether the designated dead markings are valid, i.e. the values of tokens on places involved in dead markings are as expected; or after reaching a certain place state another place state of interest can be reached; or after reaching a certain place another

place state of interest cannot be reached. Static analysis for the telephone system is discussed in Appendices B-D.

A known weakness of Petri nets is the complexity problem [57]. Even small sized process representations can have infinite reachable states (the state explosion problem). To alleviate this problem, methods are used to try and reduce the state space graph by focusing on its form (largeness avoidance) or a subset (largeness reduction). Largeness avoidance techniques are investigated for the telephone system in Appendices B-D.

Of further note is static analysis of an un-timed net amended for enhanced specification and performance analysis through the addition of timing. Based on the work in Appendix C with timed coloured Petri nets using the net toolset, CPN Tools, and recommendations from Jensen [90], static analysis of timed nets requires careful management. According to Jensen [90, 100], non-determinism in nets means that a marking cannot be uniquely determined following an enabled transition's execution. For non-deterministic nets, the same state space cannot be generated twice due to this unpredictable behaviour. [90] suggests evaluation of a non-deterministic net to check whether it can be made into a deterministic one. Although these measures can apply to untimed as well as timed nets, timed nets are at much greater risk of experiencing the state space explosion problem than their equivalent untimed net during static analysis as state space graph calculation considers all potential times that can be associated with tokens as well as their colour (types).

Both analyses possible with the Petri net formalism are investigated in detail in Appendices A-D using the telephone system specification.

7. Precision in specification of requirements.
A graphical concrete modelling language syntax can help promote shared understanding between technical and non-technical audiences. A formal notation has well-defined concrete and abstract syntax as well as static and dynamic semantics contributing to unambiguous, consistent and correct system specification and the ability to execute and analyse the model described by the notation.

As well as having a graphical concrete syntax, the abstract syntax, and static and dynamic semantics of timed high-level nets with hierarchy can be described mathematically. In terms of their graphical notation, nets offer a small range of elements from which to construct models of systems. Unlike UML, there is no system of modelling languages. High-level Petri nets such as coloured Petri nets combine classic Petri nets with the strengths of a high-level programming language, providing data typing and manipulation capability. This helps to facilitate more compact specifications as the concepts of real-life systems can be described using data typing rather than additional net elements. Specification of real-life large-scale applications becomes feasible using these high-level nets. With CPN Tools, syntax is enforced by the syntax checker in its editor and Petri net semantics are enforced by its simulator and state space analysis tool.

Ambiguity is introduced via domain concept labelling of net elements.

The specification of the telephone system using Petri net concrete syntax is demonstrated in Appendices A-D.

8. Scalability, concurrency, state, information, and event-based specification.

It is anticipated that the engineering of large-scale systems-of-systems will produce stable, desirable behaviour states. These states will be the culmination of co-operative, concurrent interaction between component systems, triggered by particular executions of event sequences consuming and producing certain types of information. A modelling language should be able to offer a means of representing this behaviour. In addition, given the number of component systems likely to be involved in a system-of-systems, the modelling language has to facilitate scalability.

As discussed previously in section 2.2, Petri nets have been used to model concurrent, state and event-based systems. High-level nets offer a means of specifying the information exchanged within the system-of-systems. In terms of scalability, from the work in Appendices A-D with the telephone system, it was not clear what the enlargement limits are for Petri nets before their use becomes impractical. This aspect is explored further in the case studies of Chapters 5 and 6. The work in Appendices B-D did show that nets do not have to be large to encounter the state space explosion problem during static analysis and discussed ways to alleviate this problem.

Table 2.1 provides an indication of the potential usefulness of Petri nets in terms of the specification and analysis of system-of-systems:

Systems-of-Systems Modelling Language Feature	**Petri Net Formalism**
Abstraction	Yes, hierarchical nets.
Modularisation	Yes, hierarchical nets.
Data Typing	Yes, coloured nets.
Toolset Support	Toolset evaluated and implemented as per requirements of organisation.
Timing	Yes, timed nets.
Verification & Validation	Yes, simulation and state space analyses.
Precision	Yes, formal basis.
Scalability	To be determined.
State and event-based	Yes, coloured nets allow enhanced specification of transition-enabling rules.
Information-based	Yes, coloured nets.

Table 2.1 'Indication of Petri net suitability for system-of-systems specification'

Chapter 3 Research Method

3.1 Introduction

Chapter 3 discusses the case study research method used to demonstrate and evaluate the strengths and weaknesses of Petri nets in chapters 5 and 6. Rationale for the use of case studies and justification of the results obtained from their use is presented.

3.2 Characteristics of Case Study Methods

For a large variety of challenging, real-life context problems, the deliberate control exercised in experimental research investigations is often not a feasible option to help understand them. Instead, a part of the real-life context where the phenomena occurs needs to be considered.

Adoption of in-depth case study research in systems engineering has gradually increased over the last decade from its dominant use in the social science domain. Although there is still relatively sparse guidance for researchers undertaking case studies and concerns as to their suitability as a research method, case studies do offer the opportunity to study contemporary objects in real-life situations and enable greater understanding of them. Case study approaches in systems engineering have been discussed by [53, 55, 64, 96, 101, 108] and their use in systems engineering to-date tends to be either in large, project-based studies [102] or in a small set of well-known examples.

Flyvbjerg [54] highlights that some definitions of the term 'case study' reinforce scepticism of its suitability as a research method:

'The detailed examination of a single example of a class of phenomena, a case study cannot provide reliable information about the broader class, but it may be useful in the preliminary stages of an investigation since it provides hypotheses, which may be tested systematically with a larger number of cases' [103].

Flyvbjerg [54] goes on to argue against the main perceived criticisms of the case study approach, namely:

1. Inability to generalise from a single case.
2. Most useful for generating hypotheses rather than their testing and subsequent theory building.
3. Theoretical knowledge is more valuable than practical knowledge.
4. Case studies can be biased in verification.
5. Case studies can be difficult to summarise.

[54] believes knowledge gained through practical means (rather than purely statistical) has much potential merit and development of guidelines can alleviate the criticism previously levelled at case studies.

Yin [104] defines a case study as empirical enquiry that:

'Investigates a contemporary phenomenon within its real-life context, especially when boundaries between phenomenon and context are not clearly evident, and in which multiple sources of evidence are generally used'.

The second point of this definition relating to boundaries is highly relevant to system-of-systems implying the potential usefulness of case studies to their engineering. Yin [104] accompanies the definition with the following characteristics of a case study:

'copes with the technically distinctive situation in which there will be many more variables than data points, and as one result relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result benefits from the prior development of theoretical propositions to guide data collection and analysis'.

Triangulation is taken here to mean different views of the evidence and Yin [104] infers from the characteristics above that the results of a case study will rely on different pieces of evidence. The characteristics also imply that a case study is appropriate when the subject being studied cannot easily be isolated from the real-life context (as is the case in controlled experiments) and there is interest in the relationships across a number of factors. Here, scientific methods are usually inappropriate for dealing with unstructured problems and this is where research design can be used for adapting research methods to do so.

Sjoberg et al [55] supply four purposes for research of which three can be applied to the case study method used to demonstrate the strengths and weaknesses of Petri nets in system-of-systems engineering. The three purposes are: exploratory ('finding out what is happening, seeking new insights and generating ideas and hypotheses for new research'); descriptive ('portraying a situation or phenomenon'); and improving ('trying to improve a certain aspect of the studied phenomenon').

[108] goes on to summarise the primary characteristics of the case study as a methodology that adds to existing knowledge through previously established theory or new theory; uses a chain of evidence, qualitative or quantitative, from multiple sources in a planned manner; and is flexible in the sense it deals with dynamic characteristics of domains such as systems engineering.

This thesis aims to evaluate the benefits of using Petri nets in systems-of-systems specification through a case study performed in an office environment. The case study approach's main strength is its realism: a researcher undertaking all net construction with an integrated Petri net toolset, and verification and validation tasks using two real-life, non-trivial system-of-systems from one application domain. Iterative guidelines and processes suggested by [55, 64, 96, 101, 108] have been followed in this thesis to plan and execute the case study in order to ensure a systematic approach to the studies. These are discussed in section 3.3 below.

3.3 Thesis Case Study Approach

Both Runeson et al and Kitchenham et al [64, 96, 101, 108] include the following steps in designing a case study:

Step 1. Define case study objective, its related research questions and plan for case study.

In the case of this thesis, the objective or quantifiable requirements specification [64, 96, 101] of the case studies in chapters 5 and 6 was to evaluate the strengths and weaknesses of Petri nets in terms of the system-of-systems problems identified in chapter 1. Specifically, the benefits of Petri nets regarding functional specification correctness, and the quality of the design were to be considered. This objective was viewed as exploratory, descriptive, and improving. In terms of a baseline with which to compare Petri nets, currently textual, or graphical (static) specifications exist to describe system-of-systems functionality. This baseline is used to determine if the design specifications captured by Petri nets help to improve functional specification correctness and design quality.

The thesis case study considers:
1. Do Petri nets improve the functional correctness of the system-of-systems design specification?
2. Do Petri nets increase the quality of the design specification?
3. What are the shortcomings of the state-of-the-art Petri net tool and how can it be improved?

The research questions relate to the criteria for success defined for both case studies in chapters 5 and 6.

The experimental subjects are the Knowledge Transfer Partnership project team members. The results were produced by one member of the project team who used a simple example specification to gain experience using the selected Petri net integrated development environment, CPN Tools. The evaluation exercise undertaken for selecting CPN Tools is presented in Appendix F. All members of the project team were involved in the verification of the results.

The case studies have one independent variable, the application of Petri nets to system-of-systems specification and analysis. The Petri net technique is regarded as the treatment, i.e. the technique being evaluated. The control treatment is the technique currently used but as this thesis is concerned with evaluating Petri nets in comparison to existing methods used in system-of-systems design, this does not have to be specified [96]. It is anticipated that functional correctness and design quality are the response (or dependent) variables [96, 101] expected to change from the application of the treatment. Functional correctness is expressed in terms of number of errors detected by simulation; and number of errors detected by static analysis. Design quality is expressed in terms of comprehensibility (e.g. use of hierarchy, annotation, timing), and scalability. Time spent editing models will depend on the Petri net toolset used and experience of the practitioner. It is not included for these reasons. The alternative hypotheses state that use of Petri nets will improve both response variables and these will be checked at each iteration of model design.

In terms of the cases or 'experimental objects' [96, 101] that the treatment is applied to, defence sector close air support (CAS) and exchange network parameter (XNP) problems are used. Both are systems-of-systems, i.e. they exhibit the systems-of-systems characteristics listed in chapter 1, in particular: a dominance of legacy component systems; multiple interfaces between component systems; and autonomy of component systems.

Close air support deals with the neutralisation of a threat detected by military personnel. An information message exchange takes place between the actors involved, culminating in the threat being neutralised or another attempt at neutralisation being initiated. Its component systems include air, ground, and sea-based military platforms, soldiers, information communications (tactical data links), and operation support centres.

Exchange network parameters aims to automate the change of network parameters and address from one subnetwork to another. Again, an information message exchange takes place between the actors involved, culminating in a network address being allocated or rejected. Its component systems can include air, ground, and sea-based military platforms, soldiers, and information communications (tactical data links). In these cases, two system-of-systems from one application domain are compared by applying the same treatments to both. Yin [104] defines this as an embedded case study, i.e. there are two units of analysis where the context is system-of-systems application domains in general:

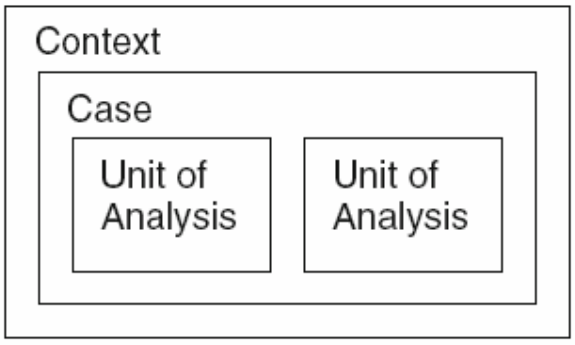


Fig. 3.1 'Embedded case study' adapted from [108], p139

Each case is expected to be a 'typical' representation of a system-of-systems, selected to predict similar results in relation to the response variables. The case study was conducted in two phases with close air support as the first phase and exchange network parameters as the second phase.

In close air support, military standards documents [105, 106] provided details of the close air support mission process. Workshops were also undertaken with subject matter experts from the Knowledge Transfer Partnership host company to verify the operational processes, i.e. the activities, activity sequences, actors, performance parameters, and information exchange identified from the standards documentation. This operational process information was then used to construct models of the specification using Petri nets at a system-of-systems level of engineering.

Specifically, models were to be produced at system-of-systems analysis, and design and architecture levels of abstraction for the entire close air support operational process. The modelling process is detailed as follows and shown in Fig. 3.2:

1. Identify relevant textual, graphical documentation relevant to the process describing the system-of-systems.
2. Extract actors, information, activities, activity sequence, performance parameters from documentation.
3. Verify information with subject matter experts.
4. Use step 3 as input to Petri net model at system-of-systems analysis level of abstraction
5. Verify with subject matter experts.
6. Use step 5 as input to Petri net model at system-of-systems design and architecture levels of abstraction.
7. Verify with subject matter experts.

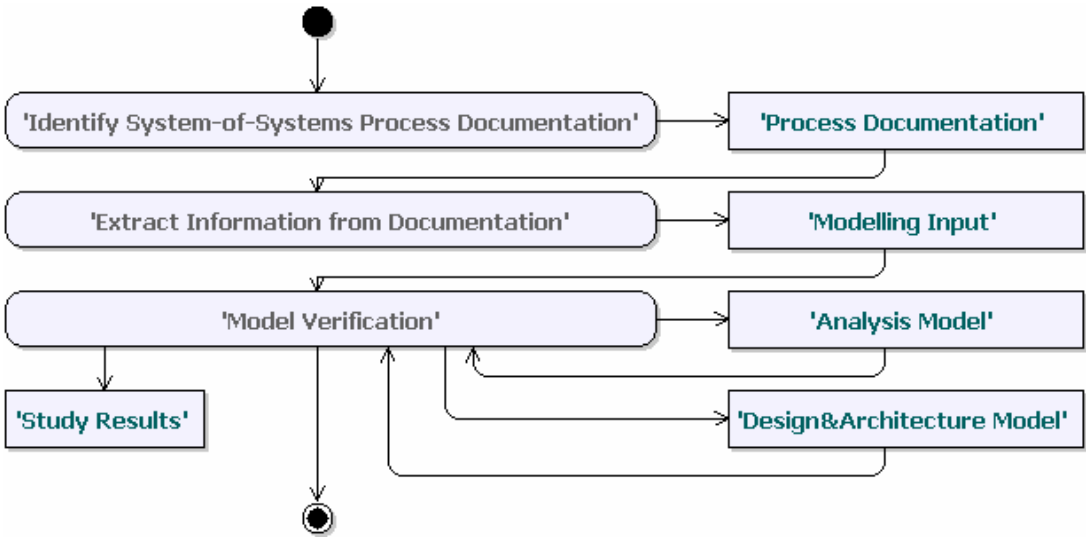


Fig. 3.2 'UML activity diagram of case study modelling process'

At each iteration of model design, response variables would be checked and results noted, i.e. scalability, comprehensibility, and number of errors detected by executing the net and calculating the reachability graph of the net.

In the second case study, exchange network parameters, a similar process to that used for close air support was followed.

Runeson et al [108] suggest a checklist to ensure adequate case study design and this was used for the two studies in the thesis:

Case Study Design Requirement	**Addressed**
Definition of a case and units of analysis	Yes.
Definition of objective, associated research questions, and hypotheses	Yes.
Theoretical basis	Yes, existing Petri net literature.
Clear cause-effect relationships	Yes.
Evidence triangulation	Yes, two cases.

Case Study Design Requirement	Addressed
Rationale for subjects, roles, viewpoints	Yes.
Relevance of case(s) to address research questions	Yes.
Integrity of involved individuals and organisations	Yes.

Table 3.1 'Checklist for case study design' adapted from [108], p144

In terms of planning for data collection, the case study plan indicates which methods of collection to use i.e. direct, indirect, or independent [107]. This thesis uses direct and independent sources of evidence collection, details of which are provided in step 2.

Step 2. Define case study procedures for data collection.

Runeson et al [108] recommends maintaining a case study design document and guidance for execution. The case study protocol for the thesis adapts the tabular template provided in [108]:

Section	Content
General	Objective: evaluate the strengths and weaknesses of Petri nets. Why: address the system-of-systems problems identified in chapter 1. How: use of two comparative, typical, exploratory cases. Consider their functional specification correctness, and the quality of the design.
Procedures	Contacts: Organisations: Host company partner, University. Equipment: Petri Net Integrated Development Environment, CPN Tools. Subjects: Knowledge Transfer Partnership project team. Process: Construct Petri net models at system-of-systems analysis, and design and architecture levels of abstraction noting effects on comprehensibility and scalability and performing simulation and reachability analysis at each iteration of model design.
Research Instrument	CPN Tools analysis reports, research notes and screenshots.
Data analysis guidelines	Detection of errors through reachability and dynamic analyses at each iteration of model design. Examination of scalability, and comprehensibility at each iteration of model design.

Table 3.2 'Case study protocol for thesis' adapted from [108], p142

Runeson et al [108] point out the importance of ethical considerations, particularly confidential information. In the thesis, confidentiality was handled through the signing of a Non-Disclosure Agreement between the Knowledge Transfer Partnership partners.

Step 3. Collect evidence.

For each case, the research instruments were identified prior to starting in order to clarify how the evidence would be collected from the study. The data sources used were:

1. Simulation screenshots illustrating problems detected using the ability to execute the net.
2. Copy of net source code at each iteration of design.
3. Copy of standard net analysis report produced by CPN Tools at each iteration of design.
4. Net screenshots illustrating comprehensibility or scalability.
5. Project team notes taken based on usage of nets at each iteration of design.

For the thesis, triangulation was used to draw conclusions from the two cases in order to avoid interpretation from one source of data only. For these case studies, data collection was first degree [107], i.e. the project team had direct control over when, what, and how data was collected and it could be collected in real-time.

[108] suggests two checklists to ensure adequate preparation and conduct of data collection and these were used for the two case studies in the thesis:

Data Collection Requirement	**Addressed**
Definition of a case study protocol	Yes.
Multiple data sources (triangulation)	Yes, two cases.
Definition of measurement instruments	Yes.
Can objective of case study be met	Yes.
Adequate confidentiality and case study sign-off	Yes.
Evidence Collection Requirement	
Case study protocol used to collect evidence	Yes.
Adequate implementation of treatment	Yes.
Adequate recording of evidence for further analysis	Yes.
Adequate processing of confidential evidence	Yes.
Traceability of evidence collection procedures	Yes.
Adequate evidence to meet research questions	Yes.

Table 3.3 'Checklist for data collection' adapted from [108], pp.149-150

Step 4. Analyse collected data.

The analysis step of a case study deals with the processing of the quantitative and qualitative data collected from the case study. Qualitative data analysis can be used in two forms of analysis, hypothesis generating (exploratory case studies) and hypothesis confirmation (explanatory case studies). The cases in this thesis use hypothesis confirmation analysis techniques to confirm that a hypothesis is true. Both cases used quantitative and qualitative data to test the hypotheses. The qualitative data may also be able to indicate the reason for the errors detected quantitatively. Qualitative analysis uses a chain of evidence to form conclusions based on the data, i.e. sufficient information from each part of a case and each decision made by the project team must be presented.

Notes were recorded by the project team immediately upon completion of design iterations for each model. These commented on toolset problems faced, for example, in net creation, and editing as well as results of simulation and reachability analysis and reasons for these results. Triangulation can be undertaken per case helping to confirm the hypotheses using tabulation to present an overview of the results of each case study.

Step 5. Report.

The case study report should present the conclusions of the study and give an indication of the quality of the study without compromising confidentiality where this is of concern. It is also important to relate a history of the study or chain of evidence, linking items of evidence to conclusions drawn. This thesis adopts the linear-analytic [104] report structure. Results and conclusions based on the case study design discussed in this chapter are presented in chapters 5 and 6.

According to Kitchenham et al [64] the five steps above aid in forming conclusions from the case study and relate to the four research design quality criteria defined by Yin that enforce the concepts of reliability and validity:

1. Construct validity: establishment of correct operational measures for the studied concept.
2. Internal validity: awareness of the range of causal relationships between independent and dependent variables during examination. To be internally valid an experiment must be designed so that conditions other than the independent variable are ruled out as potential causes of the behaviour change.
3. External validity: establishment of whether findings of a study can be generalised beyond the current study.
4. Experimental reliability: demonstration that the operations of a study can be subsequently repeated with similar results.

Case study validity depends upon the systematic process underlying its design and execution to reach trusted, unbiased results. In the studies of chapters 5 and 6, the main advantage of the case study approach was its external validity where specification of real-life systems-of-systems was undertaken over an extended period of time using an industrial Petri net toolset. A minimal degree of control was exercised as a means of maintaining external and internal validity. Petri net strengths and weaknesses in the specification and analysis of systems-of-systems require consideration of the correctness of functional specifications and improved design quality. To have construct validity the study has to examine these dependent variables. Experimental reliability is demonstrated by the repetition of operations used in the first phase of the case study in the second phase.

Many systems engineering case studies rely on large industrial studies where there is much less control. Instead, this thesis conducts a case study with a greater degree of control in an office environment, maintaining a real-life context to examine the strengths and weaknesses of Petri nets on specification of systems-of-systems.

Chapter 4 Petri Net Strengths and Weaknesses in relation to System-of-Systems

4.1 Introduction

The success of a deterministic system-of-systems engineering project depends on integrating autonomous components using international communications standards in accordance with the owner's specification. Discussed in chapter 1, a main characteristic of large-scale, system-of-systems is that they are distributed, relying upon a communications infrastructure to achieve global behaviour. Large numbers of distributed component systems contribute huge complexity in terms of concurrency with one another, exchanged information, and dynamics. Based on the individual behaviours of components, the goal of system-of-systems engineering is to design desirable behaviour into a global system-of-systems and design undesirable behaviour from it.

Current state-of-the-art approaches such as architecture frameworks and enterprise architecture provide a means of capturing an organisation from different viewpoints at different levels of abstraction. Modelling promotes understanding between the communities involved in system-of-systems development (business analysts, designers and original equipment manufacturers) and offers the opportunity to create and reuse reference models.

All these approaches can make positive contributions to system-of-systems engineering but at the moment, successful implementation of system-of-systems is still a problem. Architectures require commitment at senior management level to drive their adoption. This justification is difficult based on their current level of maturity and lack of a standard system-of-systems implementation process.

Modelling also suffers from the same lack of process surrounding what should be modelled in system-of-systems design. One of the most popular modelling notations available for specifying systems, UML, is intuitive and graphical but also ambiguous and imprecise. Supplying a range of diagrams to represent a system under development, UML lacks simulation and exhaustive verification capability. This shortfall in UML has received little attention in the context of system-of-systems and there are two major research issues:

1. Where the dynamic diagrams of UML can and cannot be used to model and analyse system-of-systems.
2. Determining how Petri nets can be used to improve the specification and analysis of the dynamic model of a system-of-systems specified using UML.

Chapter 4 discusses the benefits and shortfalls of Petri nets in relation to the specification of systems-of-systems and shows how Petri net models can be used instead of conventional UML behavioural diagrams to analyse and verify the system-of-systems using Petri net theory. The chapter concludes by introducing the first case study used to demonstrate use of Petri nets in system-of-systems specification.

The chapter begins by reviewing the behavioural diagrams of UML.

4.2 UML Behavioural Diagrams (Dynamic Model)

The development of a system-of-systems relies on specification of its dynamic structure. The mutually connected component systems of the system-of-systems and the relationships between them represent its overall behaviour. The state of the system-of-systems is defined by the states of its component systems. The overall behaviour is a result of concurrent services performed by each component system causing them to change their state.

The UML 2.x modelling notation offers modelling concepts from static and dynamic model viewpoints and includes use case, class, state machine, communication, and activity diagrams for these purposes. The static model represents the structure of a system through classes, objects, and their relationships. The dynamic model represents behaviour of a system and uses state machine, activity, sequence, and communication diagrams to do so. Given the dynamic nature of system-of-systems, the primary concern is capturing these critical dynamic aspects. The main behavioural diagrams of UML are now considered more closely in relation to the desirable features for a modelling language used in the specification of systems-of-systems suggested in chapter 1.

4.2.1 Sequence

A sequence diagram, based on the Message Sequence Chart formalism, describes the communication and functions used by objects together with the order of message flow. They are normally used to reach a better understanding of a scenario. Its concrete syntax includes lifelines represented by a vertical line indicating the progress of time and message exchanges to (from) the lifeline; with asynchronous or synchronous messages represented by arrows between lifelines. The sequence diagram is now considered in terms of the desirable system-of-systems specification features.

1. Abstraction: InteractionUse references can be used to hide detail of interactions contained in separate diagrams. Lifeline classifiers can also be decomposed in separate diagrams showing detail of component classifiers and their message exchange.
2. Modularisation: lifeline classifiers are a useful way to organise parts of the system-of-systems. Component systems within a system-of-systems are modular in the sense that they use communication interfaces (and protocols) to integrate and share information. Messages (defined within class diagrams) and lifelines are the primary method of capturing component systems and their interfaces in sequence diagrams.
3. Data typing: supported by messages defined within class diagrams.
4. Adequate toolset support: wide variety of UML CASE toolsets.
5. Timing: basic support for the specification of message ordering via DurationObservation, DurationConstraint, TimeConstraint, TimeObservation, and GeneralOrdering nodes. The UML Profile for Schedulability, Performance, and Time can also be used to introduce timing and probabilistic information.
6. Verification and validation: no simulation or model-checking in native form.
7. Precision in specification of requirements: no mathematical foundation.

8. Scalability: sequence diagrams tend to get very large quickly unless suitable abstraction is used; they can be used to represent concurrent, asynchronous, synchronous, event, and information-based message exchange (in conjunction with the UML static model) between components; they are not intended to capture states passed along lifelines, resource contention, non-deterministic behaviour, data manipulation, or process-based flow; sequence diagrams are intended to capture and explore single-cycle, scenario-based courses of action.

4.2.2 State Machine

A state machine diagram, based on Finite State Machines, describes the detailed behaviour of a part of a system by showing component states (attribute configurations) and the events responsible for state change. State machine models of component behaviour and protocol behaviour can be developed. The state machine diagram is now considered in terms of the desirable system-of-systems specification features.

1. Abstraction: Composite state can be used to hide detail of a state in a separate diagram. Submachine state can also be decomposed in separate diagrams showing detail of a state and associated transitions.
2. Modularisation: Submachine state is a useful way to organise parts of the system-of-systems. Component systems within a system-of-systems are modular in the sense that they use communication interfaces (and protocols) to integrate and share information. Transition triggers and actions (depicted graphically by signals with parameters defined within class diagrams) and Submachine state are the primary method of capturing component systems and their interfaces in state machine diagrams.
3. Data typing: supported by parameters defined within class diagrams.
4. Adequate toolset support: wide variety of UML CASE toolsets.
5. Timing: basic support for the specification of message ordering via TimeEvent nodes. The UML Profile for Schedulability, Performance, and Time can also be used to introduce timing and probabilistic information.
6. Verification and validation: no simulation or model-checking in native form.
7. Precision in specification of requirements: no mathematical foundation.
8. Scalability: state machine diagrams tend to get very large quickly unless suitable abstraction is used; they can be used to represent concurrent, state, event, and information-based message exchange (in conjunction with the UML static model); they are not intended to capture whole system behaviour, asynchronous or synchronous message exchange, resource contention, or process-based flow between components; state machine diagrams are intended to capture and explore the state-dependent behaviour of a complex component.

4.2.3 Activity

Activity diagrams were originally derived from Petri nets and flow charts. They model behaviour by organising it into units and describing the control and data flow between these units and their distribution across a system. However, in UML 1.x, the activity diagram was an adaptation of a state machine diagram (ActivityGraph was a subclass of StateMachine in the metamodel) but its semantics were redefined in UML 2.0 to 'use a Petri-like semantics instead of state machines' [11]. It is used to capture the ordering of activities. In terms of concrete syntax, ovals describe Action nodes representing a single step in an activity, and rectangles describe ObjectNode nodes for

object flow capture. The diagram starts in an initial state (black circle) and ends in an end state (black inner circle). Decision points and forks are described by a diamond and bar respectively as shown in Fig. 4.1.

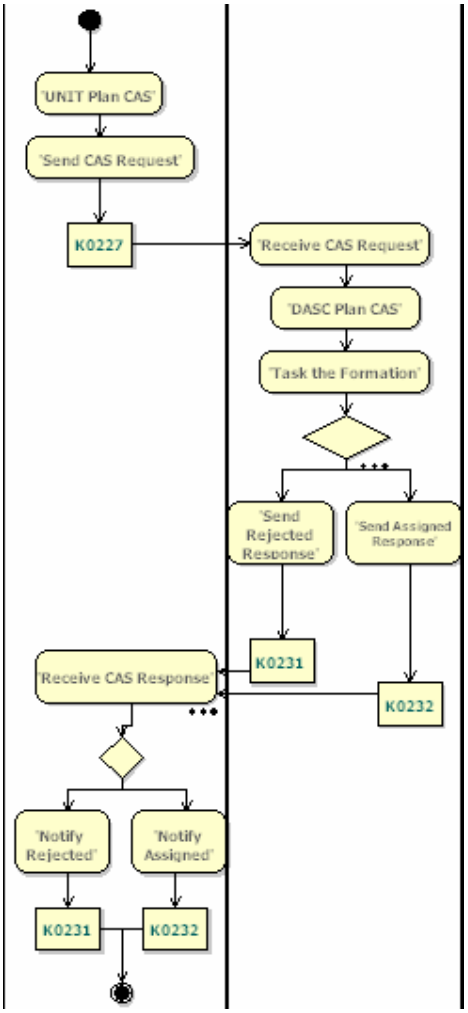


Fig. 4.1 'Example activity diagram'

The activity diagram is now considered in terms of the desirable system-of-systems specification features.

1. Abstraction: Activity containment elements can be used to hide detail of an activity in separate diagrams.
2. Modularisation: ActivityPartition containment elements are a useful way to organise parts of the system-of-systems. Component systems within a system-of-systems are modular in the sense that they use communication interfaces (and protocols) to integrate and share information. ObjectNodes and Pins (defined within class diagrams) and Activity containment elements are the primary method of capturing component systems and their interfaces in activity diagrams.
3. Data typing: supported by parameters defined within class diagrams.
4. Adequate toolset support: wide variety of UML CASE toolsets.

5. Timing: basic support for the specification of message ordering via AcceptTimeEventAction nodes. The UML Profile for Schedulability, Performance, and Time can also be used to introduce timing and probabilistic information.
6. Verification and validation: no simulation or model-checking in native form.
7. Precision in specification of requirements: no mathematical foundation.
8. Scalability: activity diagrams have increased scalability over sequence and state machine diagrams (particularly if suitable abstraction is used); they can be used to represent concurrent, asynchronous or synchronous, state, event, process and information-based message exchange (in conjunction with the UML static model) between components; they are unable to capture resource contention; activity diagrams are intended to capture and explore the control and conditions for co-ordinating whole system behaviour and lower-level component behaviour.

Derived from previously existing modelling languages used in the domain of software engineering, UML is not an executable specification language. Its usage tends to be for static examination or code generation. Even though it has a concrete, extensible, and intuitive graphical notation, the semi-formal nature of these diagrams prevents the evaluation of completeness, consistency, and correctness in system specification. The UML dynamic model provides weak support for simulation and verification. As there is no full formal syntax and semantics, models produced using UML behavioural diagrams cannot be executed (simulated). Here, an execution algorithm would be used to validate the behaviour of a system. Simulation can be used to detect undesirable behaviour, and incorrect or omitted logic. With no executable model, behaviour of different system designs cannot be checked prior to their implementation. Although commercial UML toolsets such as IBM Rational Tau and Rhapsody suites [109, 110] offer execution of UML behavioural diagrams, their execution algorithms are proprietary and based on semantic decisions taken by the toolset vendor.

Simulation alone cannot guarantee that all possible execution paths of a modelled process have been covered, and reachability graph calculation is used to provide a more exhaustive, deeper level of verification with formal languages. Standard structural properties such as reachability, boundedness, home, liveness and fairness can be automatically checked for. Again, due to their semi-formal nature, reachability graphs cannot be calculated for UML behavioural diagrams unless semantic decisions are taken by toolset vendors.

Attempts are being made to formalise UML but a formal notation requires well-defined concrete and abstract syntax as well as static and dynamic semantics. Approaches so far have tended to focus on the static semantics of UML but do not address dynamic semantics. These dynamic semantics are needed in order to execute the behavioural diagrams of UML. A pre-requisite of being able to formalise the dynamic semantics is the static model. The diagrams of the static model describe the objects of the system and their interfaces, the dynamic model describes how these objects and their interfaces are used to exchange information. Although the dynamic model of UML is focused on, the pre-definition of the classes the behavioural diagrams use is assumed.

4.3 UML Behavioural Diagrams and System-of-Systems Specification

In the context of the system-of-systems problems:

1. Verification and validation of the specification at the analysis, design and architecture phases. In critical system-of-systems applications, the ability to check design functional correctness and completeness in terms of syntax, structure, and logic is vital before their physical implementation.

2. Specification of the information exchange protocol and interfaces of the large numbers of component systems involved.

In terms of the first system-of-systems problem, the UML dynamic model is static in the sense that the behaviour it describes is not executable. Verification and validation of model correctness and adequacy in terms of requirements can only be undertaken via manual inspection or testing following implementation of a prototype or the actual system. As indicated, some commercial UML toolsets have made it possible to execute state machine and activity diagrams based on proprietary algorithms. However, these toolsets offer simulation rather than exhaustive model-checking functionality achievable using formal methods. Model-checking is where a finite state model of a concurrent system described using some formalism is checked for certain properties represented by a temporal logic formula. The model-checking method of verification is automatic, performing an exhaustive search of the calculated state space graph to check whether a property is true or not. Clarke et al provide an overview of the benefits of model-checking in [111].

Regarding the second system-of-systems problem, all three behavioural diagrams could be used to specify aspects of information exchange protocol. However, given the large number of component systems involved in system-of-systems and their concurrent nature, state machine and sequence diagrams have very limited application. Structure diagrams would be needed to define and describe interfaces and attributes.

In addition, specification of non-functional properties (such as timing information and quality of service) requires UML extensibility mechanisms to be used such as the Profile for Schedulability, Performance, and Time [112] (to be replaced by Profile for Modelling and Analysis of Real-time and Embedded Systems [113]) and Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms [114]. Again, the UML dynamic model using any of these profiles is static in the sense that the behaviour it describes is not executable. [115, 116] describe work translating UML behavioural diagrams to Petri nets for performance analysis purposes.

4.3.1 UML Behavioural Diagrams Strengths and Weaknesses in terms of System-of-Systems

In general terms, UML is a standardised, intuitive graphical notation for specification communication and documentation purposes, with widespread toolset support and adoption. UML diagrams used for a system specification are not independent of one

another, and creation of a dynamic model relies on the pre-existence of a static structure model. In terms of system-of-systems, UML has no development process and best practice is required in terms of which diagrams to develop and their order of development. Table 4.1 summarises the strengths and weaknesses of the three behavioural diagrams in the specification of systems-of-systems:

UML Behavioural Diagram/	**State Machine**	**Sequence**	**Activity**
Specification Feature			
Abstraction	Yes.	Yes.	Yes.
Modularisation	Yes.	Yes.	Yes.
Data typing	Yes (via class diagram).	Yes (via class diagram).	Yes (via class diagram).
Toolset support	Yes.	Yes.	Yes.
Formal dynamic semantics	No.	No.	No.
Verification and validation	Static.	Static.	Static.
Process-based	No.	No.	Yes.
Scalability	Low.	Low.	Improved over state machine and sequence diagrams for system-of-systems specification.
Timing	Yes (via profile).	Yes (via profile).	Yes (via profile).

Table 4.1 'Strengths and weaknesses of UML behavioural diagrams for system-of-systems'

From section 4.2, the only UML behavioural diagrams that are candidates for specifying concurrent, state, event-based, individual behaviour of a component and combined component behaviour are state machine and activity. From Table 4.1, of these two, state machine diagrams do not exhibit scalability and are not intended to capture process-based control and data flow behaviour. Sequence (and communication) diagrams can only describe specific scenarios (single cycles) and not the general behaviour of the system.

The main weakness of all the UML behavioural diagrams is their lack of formal dynamic semantics. It is not possible to execute or perform exhaustive checking of models built using the present syntax and semantics of these three types of UML diagram, i.e. no simulation, or performance analysis, or reachability graph calculation. These functions are essential in helping the production of correct specifications for large-scale system-of-systems. The behavioural diagrams discussed above have all been translated by existing work [74, 75, 116, 117] to the Petri net formalism to take advantage of its formal semantics.

The behavioural diagrams have also been translated to other formal approaches such as temporal logic, labelled transition systems, event calculus, and process algebras. Although expressive in specifying concurrent system behaviour and requirements for subsequent model-checking, these techniques involve text-based specification that can quickly become difficult to comprehend. Intuitiveness is often sacrificed for expressiveness of the notation. Formal properties can be very difficult to specify for modellers without solid mathematical backgrounds. Petri nets on the other hand offer

a graphical means of specification with a long history of research providing manual and automated verification. Non-UML, formal state machine approaches can also offer a graphical means of specification and verification but concurrency and abstraction are not normally supported. Their precise low-level focus can lead to the state-space explosion problem modelling large-scale systems.

Petri nets appeared to offer an established, intuitive, process-based approach to system-of-systems specification although none of the translation work focused on the benefits of Petri net use in their specification. For this thesis, UML 2.x activity diagrams are concentrated on for two reasons: first, the standard's [11] claim that they "use a Petri-like semantics" and secondly, based on the strengths and weaknesses identified in this section, they appear to be most applicable for use in system-of-systems specification. The thesis puts forward the idea of the Petri net formalism as a complementary enhancement to UML activity diagrams and evaluates its strengths and weaknesses using a case study approach.

4.4 Petri Nets Strengths and Weaknesses in terms of System-of-Systems

Introduced in chapter 2, Petri nets are a graphical and mathematical technique for describing concurrent, asynchronous, distributed, non-deterministic and stochastic information processing systems [56]. To-date their applications include workflow pattern development for the analysis of process-aware information systems [59], military command and control systems [60, 61], task planning research [62], computer circuits [43] and manufacturing. No work has looked at how Petri nets can help in the context of large-scale system-of-systems and the problems described in chapter 2, particularly verification and validation of the specification at the analysis, design and architecture phases; and specification of the information exchange protocol and interfaces of the large numbers of component systems involved. Most or all of these component systems will already exist but an equivalent Petri net model for them will not.

In order to find out exactly how Petri nets can be used to help address these system-of-systems problems, a well understood problem (a telephone system) was considered before exploring use of nets in creating system-of-systems models. An example specification involving a telephone system enabled experience to be gained developing nets and check their behaviour against a familiar result set. Based on addressing the system-of-systems problems highlighted above, criteria for success (Table 4.2) were defined at the start of the exercise.

Goal 1: Gain knowledge and experience in development of Petri net models (including chosen toolset). Metrics: Use Petri nets to specify the telephone example using chosen toolset noting details of the construction process (time involved, ease of mode creation); net readability and understandability; and features of the toolset.
Goal 2: Precisely specify the telephone example. Use Petri nets to capture the operational processes, components and information exchange involved in the telephone system at analysis, design and architecture phases. Metrics: Check if Petri net elements can describe operational processes, components and protocols of the telephone system. Note syntactical, semantic, and feature support provided by the chosen toolset.
Goal 3: Determine the scalability of Petri nets. Metrics: Explore the use of abstraction in Petri nets to check if nets can be used to create models of large-scale systems.
Goal 4: Determine how Petri nets can be used to verify and validate the telephone example specifications. Use Petri nets and the selected toolset to explore verification and validation of the telephone specifications at the analysis, design and architecture phases. Metrics: Employ static (state space) analysis of nets to check for well-known properties in models. Employ dynamic analysis of nets to explore behaviour and efficiency. Note when static and dynamic analyses should be used and potential for errors to go undetected i.e. existence of verification and validation spectrum.

Table 4.2 'Criteria for success for the specification of the telephone system using Petri nets'

Appendix A details the specification, initial verification, and use of hierarchy in the telephone process using classic and coloured Petri nets. The analyses possible with nets were explored and recorded in Appendix B, together with an introduction to techniques for reducing the complexity problem normally associated with nets. In Appendix C, the addition of timing to nets was considered in order to validate the telephone process. Again, techniques for reducing the complexity problem in nets were reviewed for timed coloured nets. In Appendix D, the techniques recorded in Appendix A-C are applied at design and architectural levels of abstraction for the telephone process.

Conclusions from the work in Appendices A-D using Petri nets to specify a telephone process and the possible implications for specification of systems-of-systems are presented in section 4.4.1.

4.4.1 Conclusions from Specification of the Telephone System using Petri Nets

Referring back to the criteria for success of the telephone exercise defined at the beginning of section 4.4, each goal in Table 4.2 is considered in Table 4.3 below.

Goal 1: Gain knowledge and experience in development of Petri net models (including chosen toolset).	CPN Tools was the toolset used in Appendices A-D to explore creation, verification and validation of high-level net models of the telephone process. For this well-known problem, the toolset proved comprehensive with many modelling aides available in one software package. From an industrial and large-scale system modelling perspective, it would benefit from enhancements such as: an improved model-checking report; improved (visual) guide to errors and problems detected in the model; a customised query builder; a graphical front-end to the simulator; net management facilities such as version (audit) control and comparison between nets, re-use of nets, layout of declarations, synchronisation of nets, layout and presentation of multiple folders for editing and simulation, and a facility to manage large hierarchies of nets.
Goal 2: Precisely specify the telephone example. Use Petri nets to capture the operational processes, components and information exchange involved in the telephone system at analysis, design and architecture phases.	Nets provide unambiguous specification via their mathematically-based abstract and concrete syntax and semantics. The use of Coloured Petri nets allows colours (types) to be associated with places, significantly reducing the number of elements required to represent information in a net. Timing in nets allows the modeller to associate the concept of time to activities in the net enabling capture of activity duration, timeouts, and ordering of tokens themselves (in addition to basic control sequence order) and is essential to the modelling precision of large-scale system-of-systems. Labelling and annotation within nets can be open to interpretation. The telephone example illustrated information exchange at conceptual, and design and architecture levels specifying the type of information, control order sequence, transactions, message order sequence, timing of exchange (via timeouts, parameters) and potential failure states (e.g. deadlock through underlying communications failure). To help achieve the goal of precise specification, identification of an adequate net toolset supporting high-level nets, organisation-wide net management and construction method (including a suitable hierarchy), practitioner training, and domain user involvement are vital. Enhancements in

	terms of the existing limited Petri net graphical notation are likely to be required to support the domain being modelled and for readability and comprehension purposes.
Goal 3: Determine the scalability of Petri nets.	Use of high-level nets and hierarchy (at model level and within models themselves) can help facilitate the construction of models of large-scale systems through levels of abstraction and use of colour (types) to reduce the number of elements required to represent information in a net. Based on the simple concept of the telephone process, for models of large-scale system-of-systems, it is highly likely that nets will have to be divided for development purposes. Hierarchies at model level and within models will help to facilitate this division in conjunction with a net management method. In addition, use of an adequate net toolset can also govern the overall size of a model through options provided for re-use of parts of a net and how parts of large models are presented to the modeller. At this stage, it has not yet been determined exactly how well nets can scale in the modelling of system-of-systems.
Goal 4: Determine how Petri nets can be used to verify and validate the telephone example specifications. Use Petri nets and the selected toolset to explore verification and validation of the telephone specifications at the analysis, design and architecture phases and potential for errors to go undetected i.e. existence of verification and validation spectrum.	Dynamic and static analyses of high-level nets were investigated for the telephone example. Both provide highly beneficial behavioural and structural checking for the modeller. Dynamic model execution (simulation) enables verification of behaviour. Not only can the modeller use the execution sequence to check the design of the net elements, simulation can also be used to elicit knowledge from subject matter experts in order to verify the model. Validation in terms of efficiency (performance) analysis of the model can be undertaken if timing is introduced into the net. Modelled components can also be associated with information based on real-life resources in analysis-of-alternatives scenarios to further validate the design of the model. Static analysis or model-checking provides a means of exhaustively verifying a model based on familiar Petri net properties such as deadlock. In addition, depending on the toolset employed, branching temporal logic standard and customised queries can be made on the resulting state space graph. Static analysis encourages the modeller to discover why a model behaves in a certain manner or why it does not. The main weakness of static analysis is that it is susceptible to state space explosion even with relatively compact nets

	defining a narrow problem scope. Largeness avoidance techniques were employed together with a suitable hierarchy to illustrate how they can be used to combat state space explosion and help improve understanding and specification of nets. Again, hierarchy and abstraction play a key role in facilitating verification and validation.

Table 4.3 'Results from the specification of the telephone system using Petri nets'

It became clear from modelling the telephone system that as well as the requirement to implement timed, coloured Petri nets with hierarchy, use of hierarchy and an adequate modelling framework will be vital to the success of using nets to capture large-scale systems.

The potential strengths and weaknesses of Petri nets in relation to modelling a system-of-systems were also indicated. A summary of what was learned in this section is presented in Tables 4.4 and 4.5 below. Each point is related to system-of-systems and the problems defined in chapter 2. In section 4.5, this experience is used to highlight where Petri nets are likely to be of benefit enhancing UML activity diagrams in the more complex case studies to follow.

Petri Net Strengths	**Comments**	**Relevance to System-of-Systems**
Mathematical Description	Precision in both syntax and semantics.	Specification can be described mathematically (as well as graphically).
Graphical Notation	Places, directed arcs, transitions, textual annotation and underlying interpretation rules.	Point-of-reference model for those involved with system-of-systems lifecycle.
State and Event-based	Able to capture rules-based information state, conditions and activities.	Characteristic of system-of-systems function.
Application Flexibility	Nets can be used to specify rules-based applications from different domains.	Anticipated that nets can specify system-of-systems applications.
Concurrency	Able to capture activities occurring in parallel.	Characteristic of system-of-systems function.
Single View	The set of net elements (and their underlying interpretation rules) is consistent regardless of the abstraction level or viewpoint of the model.	Specification information provided by one set of net elements.
History of State	Markings can be analysed along execution paths.	Verification of reachability and correct deadlock states.
Specification of Behaviour at Analysis, Design and Architecture Stages	Capture of operational, transactional, and physical components and control sequence, roles, states, activities involved.	Operational processes, transactions, and physical components key to realisation of overall system-of-systems function.
Specification of Information Exchange Protocol	Capture of roles, information involved, timing, failure states and sequencing.	Information exchange protocol key to realisation of overall system-of-systems function.

Petri Net Strengths	Comments	Relevance to System-of-Systems
Specification of Interfaces	Capture of operations provided by and required by component systems together with parameter information.	Component system interfaces key to realisation of overall system-of-systems function.
Hierarchy	Ability to facilitate scalability by representation of different levels of abstraction within net (capture of top-down and bottom-up approaches).	Hierarchy key to scalability, readability and analysis of system-of-systems function and efficiency specification using nets.
Dynamic Analysis (verification of specification)	Ability to execute net and verify its logic and behaviour interactively or automatically.	Verification of function and information exchange protocol of system-of-systems.
Static Analysis (verification of specification)	Ability to calculate state space graph of net and perform standard (and non-standard) analysis on the state space.	Deeper verification of function and information exchange protocol of system-of-systems.
Timing (enhancement and validation of specification in conjunction with dynamic analysis)	Capture correctness, efficiency or performance of system via introduction of concept of time in nets.	Validation of system-of-systems function specified by net in terms of prioritisation and performance.

Table 4.4 'Petri net strengths and their relationship to system-of-systems development'

Petri Net Weaknesses	Comments	Relevance to System-of-Systems
State Space Explosion	Depending on the logical structure of untimed and timed nets, state space graph calculation may have infinite reachable states.	In order to benefit system-of-systems specification, static analysis of the net is vital. Ways to reduce the state space while preserving system function need to be considered.
Scalability	Nets can quickly become large and complex (need to manage through use of hierarchy, discretisation and coloured nets).	System-of-systems are large-scale, complex systems. In order to benefit their specification, there needs to be a way to control scalability.
Readability and Comprehensibility	To non-practitioners, nets can be difficult to interpret, relate to a domain, and comprehend (need to manage through scalability and best practice).	In order to be used as primary point-of-reference to those involved in system-of-systems development, nets need to be understood by skilled and unskilled practitioners.
Specification of Behaviour at Analysis, Design and Architecture Stages	Implicit in nets' native form.	Needs to be managed through net enhancements.
Specification of Information Exchange Protocol	Implicit in nets' native form.	Needs to be managed through net enhancements.
Specification of Interfaces	Implicit in nets' native form.	Needs to be managed through net enhancements.
Lack of Method	Nets are a notation and not a method.	In order to benefit system-of-systems specification, a suitable method governing Petri net construction and management will be vital.

Petri Net Weaknesses	Comments	Relevance to System-of-Systems
Application Flexibility	Small set of basic graphical net elements makes static architecture and structure descriptions difficult. There is often more than one way to describe behaviour using nets.	In order to benefit system-of-systems specification, finding the abstraction levels and enhancements useful at different stages in their development will be vital.
Perceived Learning Curve	Training in Petri nets and the adopted toolset is essential to maximise benefit.	Skilled practitioners within an organisation adopting Petri nets for system-of-systems development will be vital.
Adequacy of Toolsets	An organisation needs to acquire a suitable Petri net development framework according to its objectives for Petri net usage.	In order to benefit system-of-systems specification, an organisation needs to identify an appropriate Petri net development framework.
Management of Nets	An organisation needs to manage and re-use developed nets according to best practice.	In order to benefit system-of-systems specification, a suitable method governing Petri net construction and management will be vital.
Change Auditing of Nets	Lack of change auditing within nets can make it difficult to track changes in versions of nets and compare one net to another.	When dealing with large nets associated with system-of-systems specification, change tracking will be vital. Unless the net toolset offers versioning and auditing, this will have to be governed by a management method but could become tedious.
Interoperability of Constructed Nets	Attempts are being made to address exchange of Petri nets between toolsets (XML-based Petri Net Markup Language, PNML).	May or may not be an issue to an organisation depending on net toolset adopted.
Concept Semantics within Net	Net element labelling convention may affect precision of model.	In order to benefit system-of-systems specification, a suitable method governing Petri net construction and management will be vital.

Table 4.5 'Petri net weaknesses and their relationship to system-of-systems development'

4.5 How Petri Nets can be used instead of UML Activity Diagrams

Petri nets are a formal notation with a concrete graphical syntax used to specify concurrency, state, events, and execution order in a system. Petri nets main advantage over UML activity diagrams is their fully formal syntax and semantics which can be described mathematically. It is possible to execute (simulate) a Petri net model and undertake deeper verification via calculation of its reachability graph prior to implementation of the system. Timed coloured Petri nets can also be used to undertake performance analysis of the specified system using the executable model.

There is existing work dealing with conversion of UML sequence, state machines and activity diagrams to Petri nets [74, 75, 116, 117]. In particular, [118, 119] considers the UML 2.x specification for activity diagrams and compares it to the Petri net formalism, determining that it is possible to map between activity diagrams and nets. According to [118], activity nodes in activity diagrams are mapped as follows: action nodes become net transitions, control nodes become net places or small net fragments, and object nodes become net places. Activity edges (including object flows) in activity diagrams become net arcs, possibly with extra places or transitions. Fig. 4.2 from [118] shows this mapping.

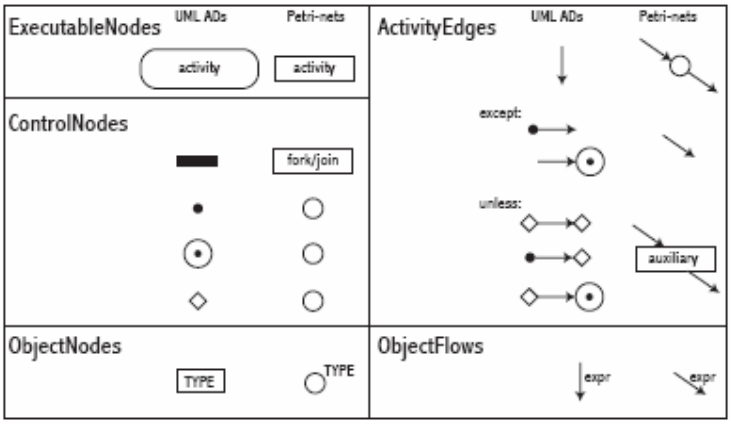


Fig. 4.2 'The intuition of the semantic mapping for control and data flow of Activities' [118], p8

Storrle [118] was not concerned with the potential benefits use of the Petri net formalism can bring to the specification and analysis of system-of-systems over and above that possible with UML activity diagrams. To further investigate the conclusions reached from the introductory work using Petri nets in section 4.4, a case study research technique was adopted for its flexible approach to studying contemporary objects in real-life situations, enabling greater understanding of them. In the case of systems-of-systems, the subject being studied cannot be easily isolated from its real-life context and there is a requirement to investigate the relationships across a number of variables. A controlled experimental environment would be inappropriate. Instead, using the case study designed in chapter 3, the first of two case studies is introduced.

The first case study from the military domain uses the Petri net formalism to specify and analyse a close air support mission. As discussed in chapter 3, the exact Petri net contributions to be explored are:

1. Do Petri nets improve the functional correctness of the system-of-systems design specification in terms of detection and reduction of the number of errors?
2. Do Petri nets increase the quality of the design specification in terms of comprehensibility and scalability?

Section 4.4 used a familiar problem to gain experience using Petri nets. In order to qualify these conclusions further, and unlike previous work to date [40, 52, 76, 99] recommending use of nets in the development of large-scale systems, nets are applied

to the modelling of a large-scale system-of-systems problem. As this is a Knowledge Transfer Partnership funded project between the University of Durham and a specialist systems engineering company partner within the defence domain (SyntheSys Systems Engineers Ltd), the initial case study focuses on the system-of-systems problem of a close air support mission.

Close air support seeks full co-operation between the actors involved (people, physical assets, and communications networks) to exploit detection, and command and control functions within the assembled ad-hoc network and support infrastructure beyond. The assets involved are normally expected to perform several functions using standard communications infrastructure(s) within a challenging, time, life, and mission-critical operating environment. Due to the large quantity of component systems involved, close air support exhibits dynamic concurrency and suffers from information exchange interoperability problems. Components such as aircraft and ships are likely to be legacy systems (with a typical in-service lifecycle of fifteen years), composed in turn of a number of component systems (e.g. communications, weapons, and sensor systems) each running bespoke software (typically monolithic, millions of lines of code, written in languages such as ADA, and poorly documented). These legacy issues are further compounded by environmental, political and technological constraints as well as performance metrics and doctrine. All these constraints are subject to considerable change over the lifecycles of the existing and planned assets useful in close air support. These effects of such changes need to be reflected in the close air support problem specification in order to identify shortfalls in expected overall behaviour from the behaviour of components.

Presently there is no top-down engineering approach to identify and specify the information that should be exchanged (including quality of service parameters such as time to perform the exchange) between the actors involved and no integration with specifications of legacy and planned actors (bottom-up engineering approach). This includes the order of information exchanged inside a component system and the order of information exchange between component systems. In a defence operational environment, there are several underlying communications infrastructures that can be used to enable communications between the actors. All these underlying communications infrastructures have their own timing, protocols, and information sets to enable the information exchange. A hierarchy of defence standards is used to specify information exchange for these different communications infrastructures but these are text-based, open to interpretation, erroneous, with a number of different versions. In addition, design specifications for component systems can opt to implement subsets of the standards, further complicating and hindering interoperability.

Once these independent assets have been integrated, their combined close air support system-of-systems behaviour is unlikely to be observable universally. There needs to be some assurance that the system-of-systems design will behave and perform as expected operationally. Discussed in chapter 1, traditional systems engineering approaches inadequately address this design complexity. Textual documents and ad-hoc diagrams are commonly employed at the specification phase, and the need to employ a flexible design foundation for system-of-systems evolution is not realised. By employing models as primary design reference points, the aim is to provide re-usable, evolvable design foundations. By developing models based on a formal

technique such as Petri nets, a further aim is to enable the enhanced verification of system-of-systems properties such as deadlock and event occurrences and show the technique can be used in both a top-down and bottom-up engineering manner.

Before moving to the study itself, it is fair to state that close air support exhibits a number of characteristics which make it interesting outside of its military context and applicable (typical) to other domains, and these are the reason it has been selected as a study. These are summarised in Table 4.6.

Study Characteristic	**Close Air Support**
Legacy components	Yes: majority (fifteen year lifecycles).
Ten or more interfaces between components	Yes.
Textual specification documentation (static)	Yes: erroneous, ambiguous, versioned standards for information exchange; inadequate capture of system-of-systems requirements; inadequately documented code implementations on components.
Bespoke component software	Yes: millions of lines of code; source code may not even be present.
Dynamic constraints	Yes: environmental, political, performance, and technical.
Integrated engineering approach	No: bottom-up engineering rather than combined top-down/bottom-up approach.

Table 4.6 'Characteristics of the close air support study'

Chapter 5 Case Study (Close Air Support)

5.1 Introduction

Chapter 5 executes the case study design described in chapter 3 for the first study in the case study research approach used by this thesis. The military mission of close air support is used to demonstrate the potential benefits and shortfalls use of Petri nets can bring to the specification and analysis of large-scale systems-of-systems.

Similar to chapter 4, and using the case study objective and research questions identified in chapter 3, the case study exercise begins by defining the criteria for success by which the Petri net approach is measured. The objective of the case study and research questions from chapter 3 are detailed below.

Case Study Objective: evaluate the strengths and weaknesses of Petri nets in terms of the system-of-systems problems identified in chapter 1.

Research Questions:
1. Do Petri nets improve the functional correctness of the system-of-systems design specification?
2. Do Petri nets increase the quality of the design specification?
3. What are the shortcomings of the state-of-the-art Petri net tool and how can it be improved?

Table 5.1 lists the criteria for success derived from the research questions above.

Goal 1 (research questions 2 and 3): Precisely specify the close air support example. Use Petri nets to capture the operational processes, components and information exchange involved in the system-of-systems completely, concisely and correctly at analysis, design, and architecture phases. Metrics: Check if Petri net elements can describe operational processes, components, information to be exchanged, information interfaces, and information exchange protocols of the close air support system-of-systems. Note syntactical, semantic, and feature support of selected toolset.
Goal 2 (research question 2): Determine the scalability of the close air support system-of-systems model implemented using Petri nets. Metrics: Explore the use of hierarchy within Petri nets to check if they can be used to create a scalable specification model of close air support.
Goal 3 (research question 1): Confirm if the same Petri net verification and validation techniques used in the telephone exercise are effective in the close air support system-of-systems specification models. Use Petri nets and the selected toolset to explore verification and validation of the close air support specifications at the analysis, design and architecture phases. Metrics: Employ static (state space) analysis of nets to check for well-known and

user-defined properties in models. Employ dynamic analysis (simulation) of nets to explore correctness of behaviour and efficiency of specifications. Functional correctness is expressed in terms of number of errors detected by simulation; and number of errors detected by static analysis. Investigate the application of largeness avoidance techniques.

Table 5.1 'Criteria for success for the specification of close air support using Petri nets'

Referring back to chapter 3, the case study plan was executed for close air support. The process outlined in that chapter relating to construction of the Petri net models was followed in order to check the desired response variables relating to the criteria for success. This process is described in the next section.

5.2 Specification of Close Air Support using Coloured Petri Nets

5.2.1 Description of Close Air Support

From military doctrine [105], close air support (CAS) is defined as:

'..air action by fixed- and rotary-wing aircraft against hostile targets that are in close proximity to friendly forces and that require detailed integration of each air mission with the fire and movement of those forces'.

Unlike the familiar concept of the telephone example, the problem of close air support required further investigation. From doctrine [105, 106], Fig. 5.1, and subject matter experts, the case study began by trying to understand the concept of close air support. The problem was summarised textually as follows:

A commander nominates a target for destruction once it has been detected by a support unit. The support unit then assembles the necessary information including the target details into a request for close air support which it sends through the command and control network to an air support centre. The air support centre then checks for and allocates available strike aircraft to the mission and confirms acceptance or rejection of the close air support request.

The allocated aircraft check-in with the close air support co-ordinator (the forward air controller), who provides a brief on the mission. Once all assigned aircraft are aware of the mission, they proceed towards the target co-ordinates supplied by the forward air controller until they reach a pre-determined distance from the target. When instructed to do so, the aircraft leave this pre-determined point and release weapons. Depending upon battle damage assessment, the mission completes or another attempt is made on the target.

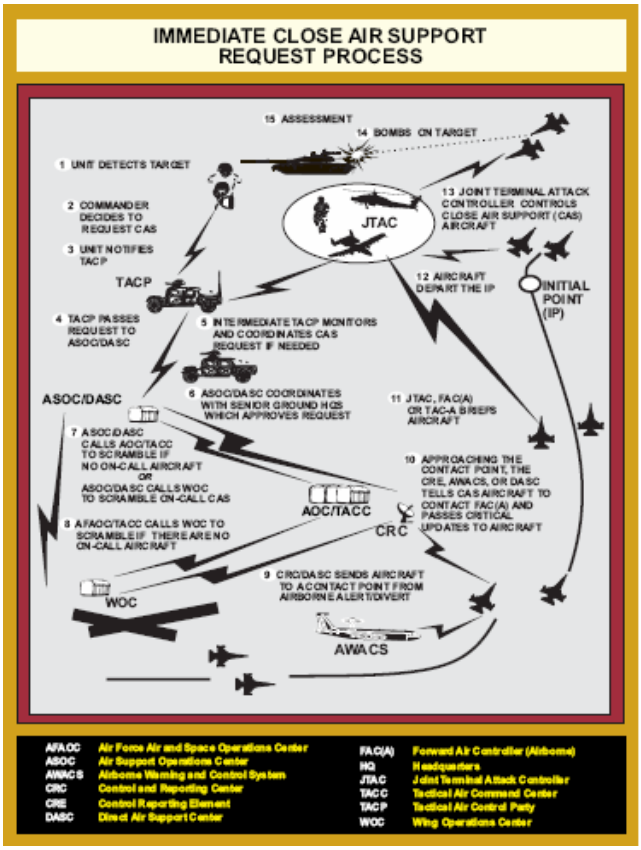


Fig. 5.1 'Immediate close air support request process' from [105], pIII-29 (Figure III-8)

5.2.2 Petri Net Construction Method

Similar to the approach adopted for the telephone example in Appendices A-D, the modeller was keen to specify the close air support problem space in a way that promoted the need for a flexible, adaptable solution without being prescriptive. A multi-viewpoint modelling approach similar to the approach used in UML modelling (with its functional, static and dynamic views) and [70, 71, 76] was adopted where the specification of close air support was considered from analysis, design, and architecture abstraction levels.

Close air support was functionally decomposed into a series of functions and sub-functions (activities). These functions were then used with the concept summary above to suggest operational processes. This function-based approach is similar to the principle behind service-oriented architecture. The operational processes outline a particular timed ordering of activities and information exchange to be performed by roles so that the function (or service) may be realised. Optimised where possible in terms of resources, the operational process level helps to specify the overall function of the close air support system-of-systems to domain users and developers, and drive its lower-level design and implementation. At the analysis stage, functions (together

with the information and information exchange protocol used by these functions) rather than the physical components able to meet these functions are specified.

Jensen's guidelines for construction of coloured Petri net models [63] were considered in conjunction with approaches made by [72-76] to transform UML or object-oriented system models to high level Petri nets. To begin with, the net was created at the operational process (conceptual or analysis) model level of abstraction. Functions (activities) from the processes were mapped to net transition elements and information exchanged was assigned to net place elements following the control sequence presented within [105, 106] and guidance from domain experts. Colours (types) were defined in the toolset according to the information exchanged at each place. Compound or structured type definitions were used to specify the information exchanged. Roles (owners of the identified functions and sub-functions) were indicated by text labels on the page allocated to the net within the toolset. Initially, a net was created for the entire process (from the point-of-view of a close air support mission planner). It was clear almost immediately that due to low range of net elements, symbol expressiveness and amount of symbols needed, the net quickly became complicated in terms of readability. This re-emphasised one of the preliminary conclusions from chapter 4, i.e. that hierarchy would be essential in achieving any kind of scalability for large-scale system-of-systems design with Petri nets.

In response, one of the functions of the system-of-systems, i.e. the part of the close air support system-of-systems problem italicised above was focused on. This describes the problem of requesting close air support from an air support centre. The roles, control sequences, processes and information exchanged are presented in Fig. 5.2.

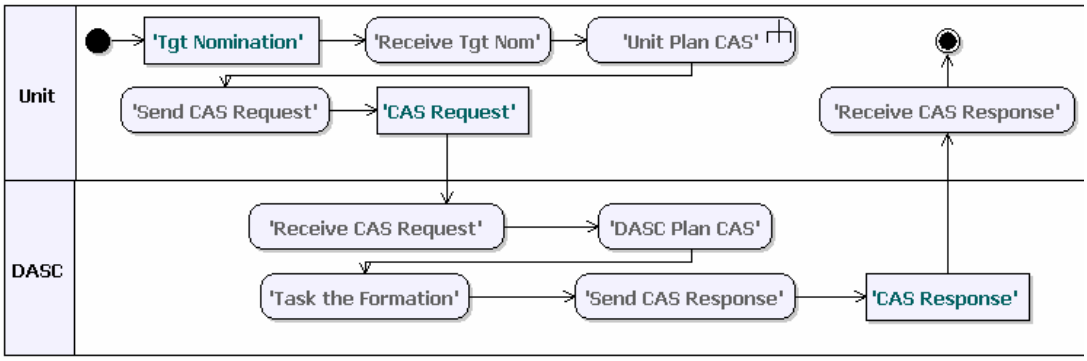


Fig. 5.2 'Request close air support UML activity diagram'

From Fig. 5.2 it can be seen that two roles, 'Unit' and 'DASC' (Direct Air Support Centre), have operational processes assigned to them. These operational processes are presented as a sequential series of activities. The detail of groups of these activities can be abstracted at a higher level of abstraction under parent activities such as 'Task the Formation' or 'Plan CAS' (shown using notation in Fig. 5.2). Fig. 5.2 also shows that two pieces of information are exchanged between the roles, 'CAS Request' and 'CAS Response'.

A coloured Petri net model of the operational processes represented in Fig. 5.2 was then attempted. Fig. 5.2's function (or service) level of granularity and naming was

based on existing underlying communications infrastructure information sets (tactical data link transactions and the associated function performed by assets). This information was used to help manage the construction of the net by developing a hierarchy of functions to promote readability and scalability of the net. The function of 'request close air support assignment' was decomposed into three main sub-functions: 'request close air support', 'assign close air support request' and 'receive close air support response'. Each of these sub-functions are realised using the processes identified from existing doctrine and domain experts. These processes were examined in turn in order to establish:

1. Executed activities together with their pre and post information states (i.e. input and output interfaces).
2. Control of activity execution.
3. Suggested roles (owners) associated with the control of activities.
4. Naming of activities, information and roles.
5. Net symbols that should be used on the net at the highest abstraction level (primary parent net) and associated sub-pages to accurately represent the hierarchy within the model.

The control flow of the three main sub-functions is shown in Fig. 5.2 as request followed by assign followed by receive.

At the highest abstraction level in the hierarchy, the pre-condition of the request function executing is incoming external trigger information in the form of a message from a commander, containing target details. Using [105], suggested fields for this target nomination information are: originator; target type; target longitude; target latitude; target elevation; ordnance required; and required result.

When close air support assignment is deemed to be required, the post-condition of request executing is outgoing information, containing close air support request details. This information is sent to the role dealing with assignment of firepower to the mission (DASC). Information content can be derived from [105] and existing tactical data link message content [120]. Suggested fields for this close air support request information are: request number; mission indicator; mission priority; target number; target latitude; target longitude; target elevation; and target type.

Considering the request function's process in more detail, the target information is used in the activity of planning of close air support. This planning activity can be decomposed further into sub-activities relating to deciding whether or not to pursue a close air support assignment. For the purposes of this case study, further activity decomposition is omitted and the result of the planning activity is that close air support assignment is requested by the activity of sending close air support request information.

At the highest level of abstraction in the hierarchy, according to [105, 106] and Fig. 5.2, the pre-condition of the assign function executing is incoming trigger information from Unit containing the close air support request information specified in this section.

The post-condition of assign executing is outgoing information containing close air support response details. This information is sent to the role which requested close air support (Unit). Its content can be derived from [105] and existing tactical data link message content [120]. Suggested fields for denial response information are: mission indicator; fire plan name; request number; target number; target latitude; target longitude; and reason denied. For acceptance response information, suggested fields are: mission indicator; mission number; request number; number of aircraft; aircraft type; and aircraft sign. Currently, doctrine describes two separate tactical data link response messages. Design optimisation of the assign function could consolidate the response information.

Considering the assign function's process in more detail, the close air support request information is used in the activities of planning the close air support assignment and tasking the formation (if it has been possible to identify suitable firepower). Both the planning and tasking activities could be decomposed further into sub-activities. For the purposes of this case study, further activity decomposition is omitted and the close air support request information is used directly by the tasking activity. The result of the tasking activity is that close air support response information is sent.

At the highest level of abstraction, the pre-condition of the receive function executing is incoming trigger information from DASC, containing the close air response information specified above.

According to [105] and domain experts there are two possible mutually exclusive post-conditions of receive executing. One is outgoing external information containing notification that the close air support request was unsuccessful. The other is an end state for this part of the overall close air support function indicating that the request for close air support was successful.

Considering the receive function's process in more detail, the close air support response information is used to notify the commander that the close air support request was unsuccessful or to proceed to the next stage of close air support. For the purposes of this case study, two activities are used to reflect this. The result of the send close air support rejected activity is that close air support denied information is sent. The result of the close air support assigned activity is that information to proceed to next stage is conveyed.

This information was then manually mapped to Petri net constructs. As before, the net was constructed from a close air support planner's point-of-view. Conditions were allocated to places and colours (types) were defined to represent the fields of messages suggested above. Naming convention followed Fig. 5.2's as closely as possible. For the net relating to the 'request close air support assignment' function, the three sub-functions (request, assign, and response) were represented as abstracted transitions ('Unit_CAS_req_service', 'DASC_CAS_assign_service', and 'Unit_CAS_resp_service' respectively) with more detail for each presented on subnets. Each abstracted transition has associated input and output sockets (interfaces) and together form the highest level of abstraction in the hierarchy (parent net). The operational processes that realise each of these sub-functions are expressed on additional net pages (subnets) together with their associated input and output ports (interfaces). A successful or unsuccessful close air support outcome from the assign

function was selected at random within the toolset. The derived net's highest level of abstraction is shown in Fig. 5.3.

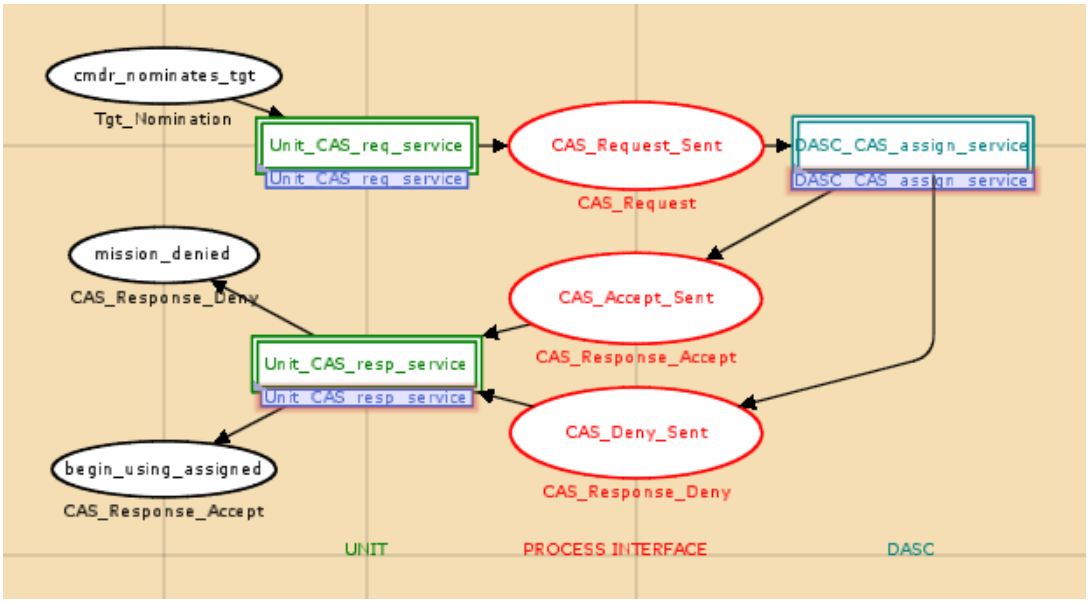


Fig. 5.3 'Request close air support assignment parent net'

Since Petri nets are generic in the sense they can be used to capture relationships between states and events of systems regardless of application domain, close air support system-of-systems concepts such as function, process, information interfaces between processes, and process execution owner are captured implicitly by the notation. Use of suitable place typing, net hierarchy construction via pages, labelling and colouring in the tool's editor helps to promote domain information and logic such as decision points more clearly in the net.

Fig. 5.3 also illustrates that hierarchy implemented in the net using substitution transitions and port and socket places allows tracking of state in a large-scale system-of-systems model. Given that one function of the close air support system-of-systems is captured, three subnets within the toolset editor were used to specify the 'request close air support assignment' function's decomposed processes at a lower level of abstraction. Without hierarchy, not only would it be difficult to have an accurate appreciation of the processes' localised states, it would be difficult to determine the overall state of the system-of-systems (and whether or not the system-of-systems net representation terminated properly and process states were as expected).

As well as allowing top-down and bottom-up engineering approaches through its hierarchy support, the toolset also permits re-use of subnets more than once, i.e. it facilitates instantiation. These instances of a subnet are independent in the sense their markings or state are independent (further illustrated in section 5.5 at design and architecture levels).

Unfortunately, it can be inferred from the specification modelled in Fig. 5.3 that request and receive are performed by the same role (Unit), and assign is performed by a different role (DASC). This can be viewed as prescriptive: not only does it suggest

that two distinct assets are needed to realise these three sub-functions, it suggests the likely asset candidates via the naming used. Instead, design optimisation of the 'request close air support assignment' function should be promoted, e.g. one asset may be able to perform all three sub-functions in less time. Attention to this fact may be drawn through textual annotation of the constructed net. In addition, the role names used could also be viewed as prescriptive (these were identified from existing doctrine) and may benefit from re-naming. Alternatively, a higher level of abstraction could be added to abstract the three sub-functions to one main 'request close air support assignment' function.

Based on this prescriptive view, it could be argued that since the request and receive sub-functions are highly likely to be undertaken by a 'Requester' role (avoiding the more prescriptive 'Unit' role description), both could be represented by a more general 'Request_CAS_Service' transition with no negative impact to specification (given that the interface information remains the same). These amendments are shown in Figs. 5.4-5.5.

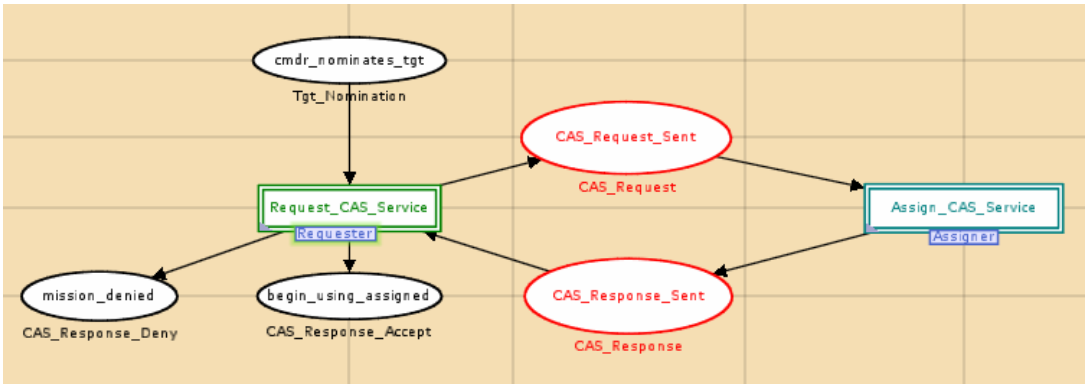


Fig. 5.4 'Revised parent net'

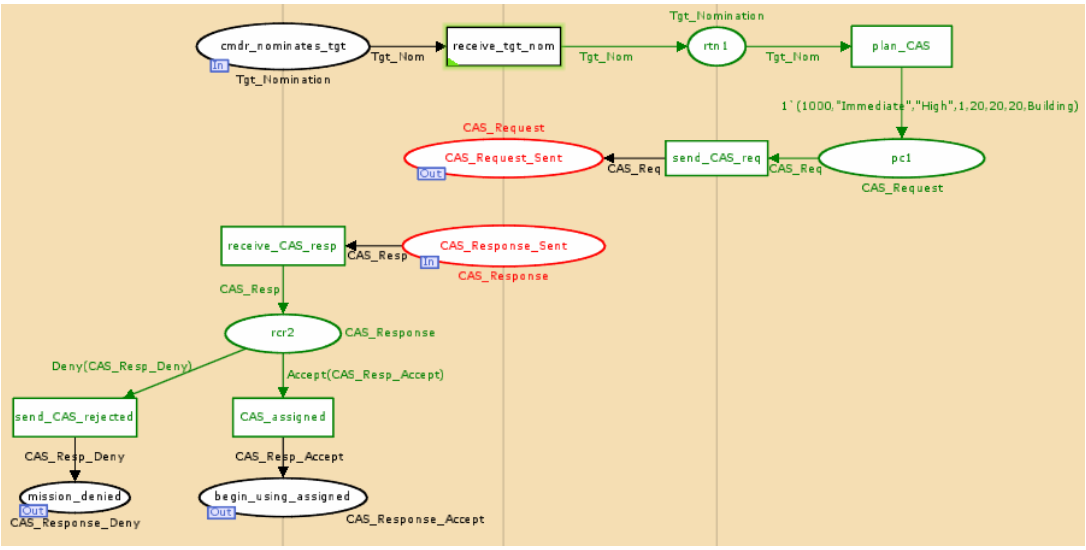


Fig. 5.5 'Revised Requester subnet showing the combined request and response functions'

From this section and design iteration of the model, it can be seen that it is possible to use coloured Petri nets and hierarchy to specify one sub-function of close air support (request close air support assignment) at an analysis level of abstraction, i.e. the operational process level. From Figs. 5.3-5.5, in terms of design quality and readability, net places capture process state in terms of the information input and output to net transitions (activities); net colours (types) define the information used by activities; net arc inscriptions govern the information required and produced by activities; and net arcs dictate the control flow of execution i.e. the order of activity execution and information exchange.

5.3 Analyses with Petri Nets and further Specification

5.3.1 Dynamic Analysis (Simulation)

To investigate functional correctness, simulation of the derived net in Fig. 5.4 was used to provide confidence in the correctness of the behaviour and logic specified by it. Using the toolset it was possible to interactively step through enabled transitions in the net from a given initial marking until there were no more enabled transitions. In doing so, a problem was revealed within the 'task_the_formation' subnet (shown in Fig. 5.6) detailing the activity undertaken by the close air support assigner actor. Simulation revealed that when the subnet's 'negotiate tasking' activity executes, two tokens are produced for output places 'rcr1' and 'ac_tasked' respectively, resulting in an infinite net. The intended behaviour should be to test for a successful outcome from the 'plan_CAS' activity. If this has not been achieved, the subnet should specify that the planning process is repeated, otherwise the tasking process continues.

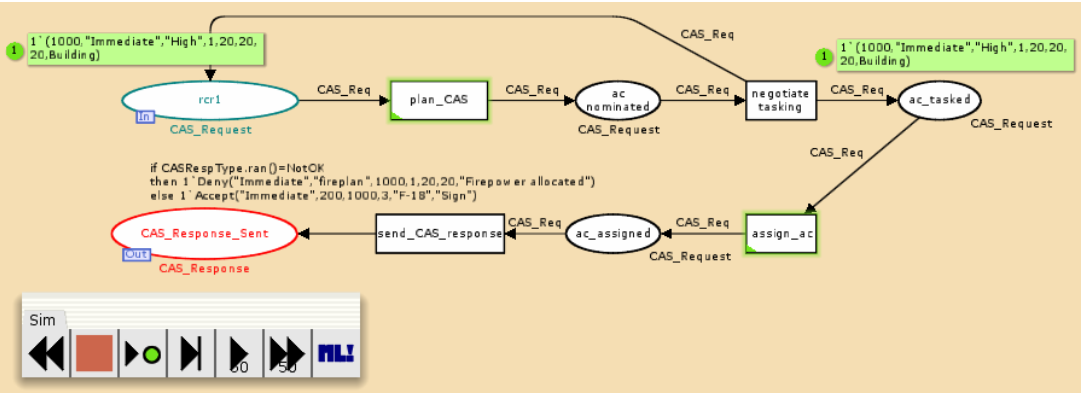
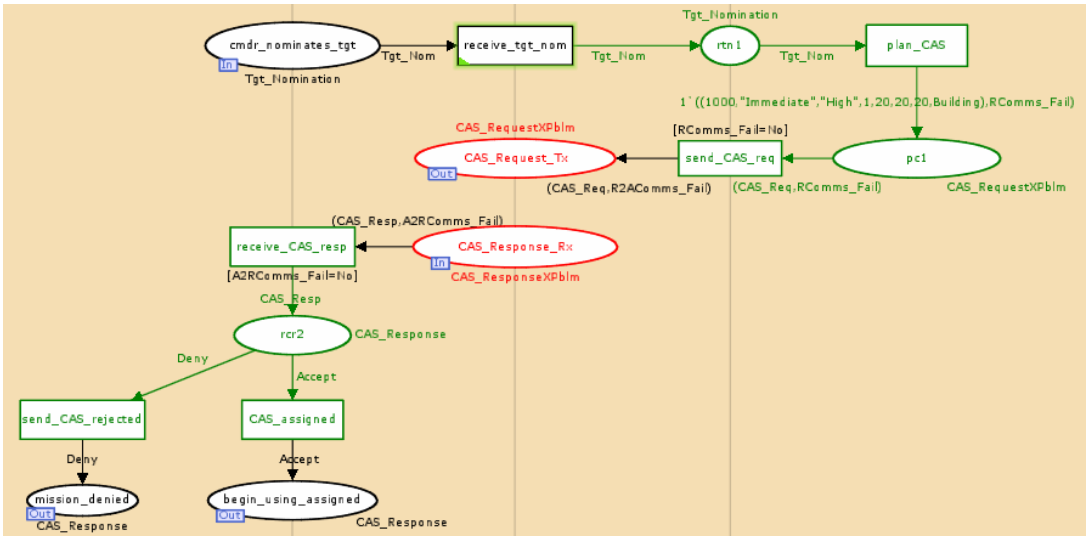


Fig. 5.6 'Infinite net problem detected by executing net'

Once this problem was corrected by defining an enumerated type and the relevant arc inscriptions, basic interactive simulation indicated that the hierarchy of processes appeared to produce the desired behaviour using one and three target nominations. For the three target nomination tokens, simulation showed that their order within the control sequence execution was non-deterministic and tokens could overtake one other. This can be addressed by the introduction of queuing places (timing can also impose order on tokens but will normally be used to reflect deterministic and

stochastic activity durations and cannot guarantee tokens will not overtake one another).

Potential deadlock situations within the close air support request assignment function were also highlighted by stepping through activity executions. These deadlocks can occur if either the assign or the response sub-functions fail to receive a close air support request or response to the original close air support request. As it stands, the model does not explicitly specify these situations as undesirable behaviour. Based on the work in [77-79, 100], the nets in Figs. 5.7-5.9 were developed to specify the possibility of communications failure (both on the external communications infrastructure and infrastructure internal to a system component) and to specify tracking of the state of the information exchange.




```
colset Problem = with Yes | No;
colset CAS_Request = product ReqNum*MissionInd*MissionPriority*TgtNum*TgtLat*TgtLong*TgtElev*TgtType;
colset CAS_RequestXPblm = product CAS_Request*Problem;
```

Fig. 5.7 'Net and colour (type) definitions specifying the potential failure points within the activities undertaken by the Requester role'

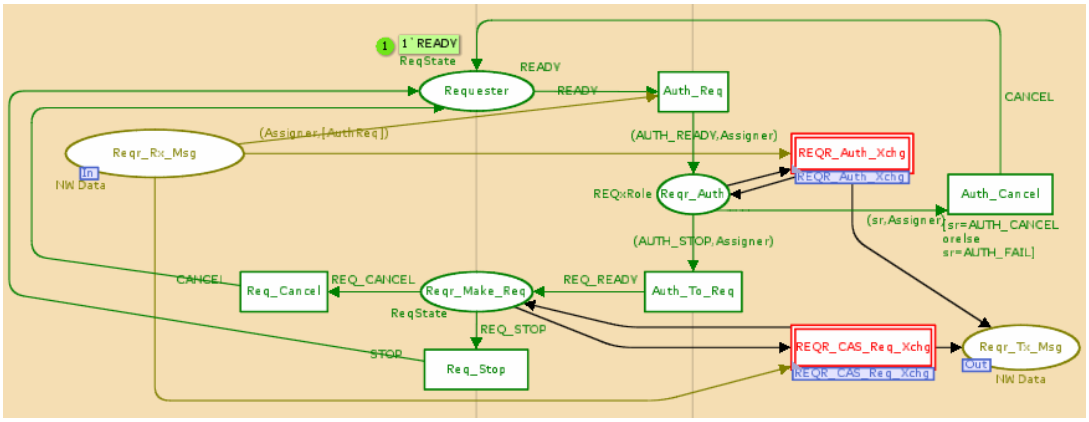


Fig. 5.8 'CAS_Request_Service net showing two information exchange transactions (transitions in red) undertaken by the Requester role'

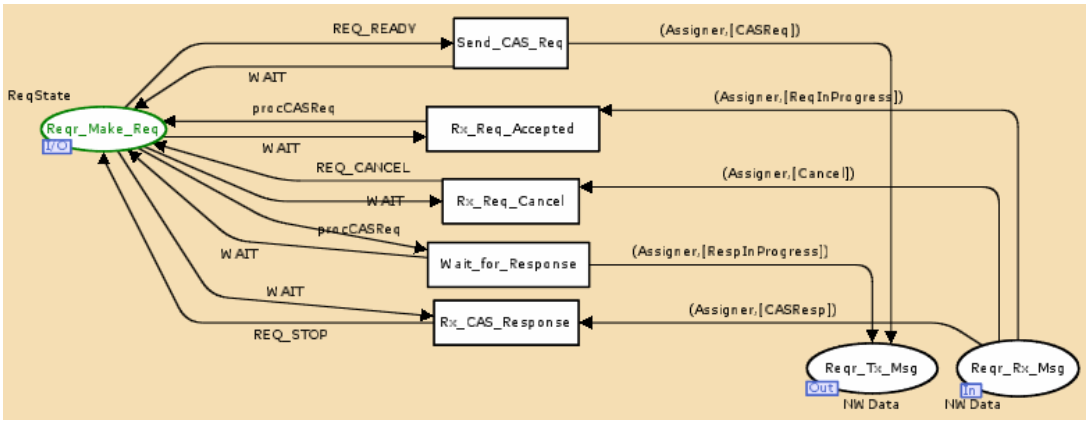


Fig. 5.9 'Net detailing state of the close air support request-response information exchange transaction (REQR_CAS_Req_Xchg) undertaken by the Requester role'

The net in Fig. 5.7 uses compound type definitions to introduce potential points of failure to activities dealing with the transmission or receipt of information between actors. At the moment, failure is boolean but more accurate probability of failure can be specified within the net.

'Requester' (CAS_Request_Service) and 'REQR_CAS_Req_Xchg' (Figs. 5.8-5.9) taken together with corresponding 'Assigner' (CAS_Assign_Service) and 'ASGR_CAS_Req_Xchg' subnets further specify the state of the information exchange protocol relating to the close air support request and close air support request assignment services introduced in the operational process of Fig. 5.4. The net specifies how each role could track its state in terms of what information is sent to and what information is expected from its partner role(s) on the network so that if a communications failure occurs, suitable recovery can be designed for. It should be noted that with the addition of timing to the net, timeouts could be specified in the net of Fig. 5.7. Again, this specification information is in addition to the operational process and aims to be non-prescriptive in terms of how tracking is to be implemented.

5.3.2 Static Analysis (Reachability Graph Analysis)

The nets in Figs. 5.4-5.9 are the results of being able to execute the operational process net and identifying its shortcomings. Although simulation is very much part of an iterative net development process, interactive simulation of large nets can be extremely time-consuming and does not provide an exhaustive means of net verification. Reachability graph or state space analysis (described in chapter 2) is used to complement simulation and provide this deeper level of verification.

CPN Tools' standard analysis of the state space (calculated for each of the nets created in Figs. 5.4-5.9 and described further in Appendix B) is shown in Table 5.2 below.

State Space
Nodes: 22951
Arcs: 78716
Secs: 600
Status: Partial

Scc Graph
Nodes: 22951
Arcs: 78716
Secs: 6

Boundedness Properties

Best Integer Bounds

	Upper	Lower
Assigner'rcr1 1	1	0
Overview'CAS_Request_Sent 1	1	0
Overview'CAS_Response_Sent 1	6	0
Overview'begin_using_assigned 1	4	0
Overview'cmdr_nominates_tgt 1	1	0
Overview'mission_denied 1	4	0
Requester'ReqID 1	1	1
Requester'pc1 1	1	0
Requester'rcr2 1	5	0
Requester'rtn1 1	1	0
task_the_formation'ac_assigned 1	8	0
task_the_formation'ac_nominated 1	1	0
task_the_formation'ac_tasked 1	13	0
task_the_formation'missionID 1	1	1

Table 5.2 'State space standard report'

The state space standard report of Table 5.2 calculated for an initial marking of one token for the net in Fig. 5.4 highlights there is a problem with the net for two reasons. First of all, in the given state space calculation limit of six hundred seconds, the state space explosion problem was encountered, resulting in the calculation of a partial reachability graph. Secondly, on inspection of the standard state space report produced by CPN Tools for the partial reachability graph, the 'Boundedness Properties' section reveals the presence of multiple tokens on places (rather than the expected one token) indicating a looping problem and helping pinpoint exactly where in the net it is located, i.e. the 'task_the_formation' subnet. If this infinite net had not been detected using simulation, static analysis would have alerted the modeller to its presence.

Once the infinite loop was removed from the net, its full reachability graph was re-calculated in less than one second with fourteen nodes, thirteen arcs and the expected two dead markings (indicating a successful or unsuccessful request outcome). This exercise was then repeated using an initial marking of two nomination tokens for Fig. 5.4 and a summary of the results are presented in Table 5.3.

STATE SPACE GRAPH	Net of Fig. 5.4	Amended Net (added queuing place and request input control)
Initial marking	1`[(Commander,Building,20,20,20,"1000",NEW,Destroy),(Commander,Vehicle,15,25,15,"1800",HSM,Interdict)]	1`[(Commander,Building,20,20,20,"1000",NEW,Destroy),(Commander,Vehicle,15,25,15,"1800",HSM,Interdict)]
Nodes and arcs	376 nodes, 676 arcs.	40 nodes, 39 arcs.
Generation time	1 sec.	0 secs.
Terminal markings	12	4
Token Overtaking	Present.	Controlled.

Table 5.3 'State space standard report summary'

Table 5.3 shows that the reachability graph calculated for the net of Fig. 5.4 has considerably increased in size using an initial marking of two tokens. Inspection of its upper multi-set bounds and twelve dead markings revealed that this increase was due to the possibility for the two tokens to overtake one another. Overtaking was controlled by adding a queuing place to the source place of the net of Fig. 5.4 and releasing a new token for input to the net once a previous one was processed (facilitated using an additional 'Next Req' place linked to the final two transitions in the Requester subnet, shown in Fig. 5.10). The reachability graph was re-calculated for the amended net and the results are also presented in Table 5.3.

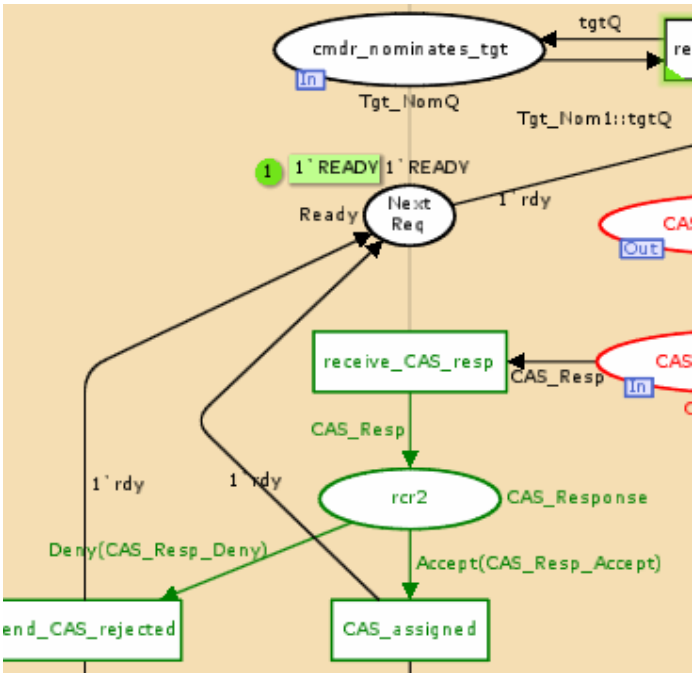


Fig. 5.10 'Amended net to control input nominations'

It can be seen that the reachability graph has reduced by approximately one tenth in terms of the numbers of nodes in the graph, and one third in terms of the dead

markings. It was calculated in less than one second (CPN Tools standard report does not report more granular timing information). Again, use of simulation alone may not necessarily have alerted the modeller to the issue (particularly if only a part of the net is being executed or the net is large) and impact of token overtaking when model behaviour is being checked with multiple close air support requests. Token overtaking leads to uncertainty in variable bindings unless ordering is imposed. It should be noted that the steps taken to control overtaking in the net of Fig. 5.4 may not be the most efficient for the request-response process (for example, it may be more efficient for two nominations to be considered together or for a new request to be processed by the assigner as soon as the previous one has been dealt with). The results show static analysis has been able to alert the modeller to inefficiency in the overall net.

```
Requester'cmdr_nominates_tgt 1: empty
Requester'rtn 1 1: empty
Requester'pc1 1: empty
Requester'CAS_Request_Tx 1: empty
Requester'rcr2 1: empty
Requester'mission_denied 1: empty
Requester'begin_using_assigned 1: empty
Requester'CAS_Response_Rx 1: empty
Requester'ReqID 1: 1`1001
Overview'cmdr_nominates_tgt 1: empty
Overview'CAS_Request_Sent 1: empty
Overview'CAS_Response_Sent 1: empty
Overview'begin_using_assigned 1: empty
Overview'mission_denied 1: empty
Assigner'CAS_Request_Rx 1: empty
Assigner'rcr1 1: 1`((1000,Immediate,High,1,20,20,20,Building),Yes)
Assigner'CAS_Response_Tx 1: empty
```

Fig. 5.11 'One of the five dead markings showing communications failure within the assigner'

Calculating the reachability graph for the net specifying communications failure with an initial marking of one nomination produces seven dead markings. Two of these relate to the desirable behaviour outcomes of a successful and rejected request close air support assignment and no communications failure. The other five markings relate to the five potential communication failure points in the model: one within requester; one between requester and assigner; one within assigner (shown in Fig. 5.11); and one between assigner and requester (which can potentially fail for a successful and rejected response). In this way, static analysis has verified that the model behaves as expected, i.e. where a communications failure could take place and deadlock the request close air support assignment process, static analysis confirms this is specified correctly in the model.

```
Liveness Properties
----------------------------------------

Dead Markings
  [39,48]

Dead Transition Instances
  Assigner'Auth_Cancel 1
  Requester'Auth_Cancel 1
```

Fig. 5.12 'Liveness properties for the information exchange transactions net'

When the reachability graph was calculated for the net specifying information exchange transactions, two dead markings were obtained along with two dead transition instances, Assigner'Auth_Cancel and Requester'Auth_Cancel (Fig. 5.12). The two dead markings correctly indicated possible terminal states resulting from the 'REQR_CAS_Req_Xchg' information exchange on places 'Assigner' (within the assigner subnet) and 'Requester' (within the requester subnet) of 'CANCEL' and 'STOP'. The presence of the two dead transitions (Assigner'Auth_Cancel and Requester'Auth_Cancel) indicated that the 'CANCEL' state on the 'Assigner' and 'Requester' places was not reached as a result of cancellation within the 'REQR_Auth_Xchg' information exchange. This meant that the logic within this information exchange was either incorrect or missing. Upon inspection of this information exchange, the logic had indeed been omitted within the 'REQR_Auth_Xchg' subnet. This is information that would not necessarily have been detected immediately by executing the net due to its otherwise desirable behavioural outcomes.

```
fun requestEndState n = (Mark.Requester'Requester 1 n = 1`STOP);
val requestEnd = POS(EV(NF("err",requestEndState)));
fun responseState n = (Mark.REQR_CAS_Req_Xchg'Reqr_Rx_Msg 1 n = 1`(Assigner,[CASResp]));
val noResp = NOT(POS(EV(NF("err",responseState))));
val properRequestDelivery = EXIST_NEXT(AND(noResp,requestEnd));
eval_node properRequestDelivery InitNode;
val requestEndState = fn : Node -> bool
val requestEnd = EXIST_UNTIL (TT,FORALL_UNTIL (TT,NF ("err",fn))) : A
val responseState = fn : Node -> bool
val noResp = NOT (EXIST_UNTIL (TT,FORALL_UNTIL (TT,NF ("err",fn)))) : A
val properRequestDelivery =
  MODAL
    (NOT
      (OR
        (NOT TT,
         NOT
          (MODAL
            (NOT
              (OR
                (NOT
                  (NOT
                    (EXIST_UNTIL
                      (TT,FORALL_UNTIL (TT,NF ("err",fn))))),
                 NOT
                  (EXIST_UNTIL
                    (TT,FORALL_UNTIL (TT,NF ("err",fn))))))))))) : A
val it = false : bool
```

Fig. 5.13 'Non-standard logic query confirming correct behaviour of information exchange protocol'

From this section, it was shown that it is possible to use two forms of analyses on coloured Petri nets to verify functional correctness of the model they represent. These were execution of the net (simulation) and analysis of the net's reachability graph (static analysis or model-checking). As well as producing a standard analysis report from the calculated reachability graph, standard and non-standard temporal logic queries can also be applied to it. Fig. 5.13 shows a non-standard logic query constructed to check when a request for close air support is sent, it is not possible to receive a response without the information exchange protocol ending in a 'STOP' state. Following these analyses, models can be amended and enhanced, improving design quality. In the case of close air support, further specification included: potential failure states (i.e. underlying communications failure); tracking of information exchange (transaction) state; and a means of controlling ordering of

multiple tokens in the model. As part of the iterative development process, simulation and static analysis were used again to verify these specifications.

5.4 Addition of Timing to Petri Net Model

As demonstrated by the telephone example (Appendix C), time-dependent actions such as timeouts, processing delays or deadlines are essential to capture the efficiency or performance of a system and facilitate validation of its design. As well as efficiency specification, time-dependent actions also enhance a system's behaviour specification in terms of correctness. Activity ordering alone is insufficient to capture overall system behaviour precisely. Tokens representing information in larger-scale systems will be processed according to the time they entered the system, time involved in their consumption and generation, and involvement in delays and transfer failures. Timing will be needed to specify the ordering multiple tokens receive over and above any activity sequence they experience.

Currently, close air support has been specified at an operational process (analysis) model level of abstraction and used as the first stage in large-scale, system-of-systems development. Typically, this viewpoint is useful for gaining a shared understanding of the problem concept and the intended technical and non-technical audience would include analysts, developers and domain users. The introduction of timing information to close air support at this abstraction level would help enable domain users and developers to decide whether the modelled concept was efficient and adequate for input into the design stage. Assessing performance would involve checking if the modelled processes reached desirable behaviour states (including recovery from undesirable states) within realistic time and resource estimates. Improving the efficiency of the process means looking for new or different ways to realise desirable behaviour within defined time, cost and quality parameters.

To examine alternative options for the process, it was necessary to determine the time, cost and quality performance indicators for the request close air support assignment process and implement these in the model. Examples of these indicators include request fulfilment time, communications resource usage (and related costs), and request fulfilment time within a certain time limit. The natural inclination would be to minimise the first two and maximise the last one but all three need to be taken in context with the strategy of the actors involved. In the case of system-of-systems, it is essential to understand the economic and operating environment for system-of-systems services, and which (if any), of the performance indicators carries more weight than the others.

In the close air support example, as the system-of-systems was specified originally from the mission planner's point-of-view, it is assumed that they are concerned with striking a trade-off between quality and cost parameters for the mission. This may mean the assigner would be interested in maximising request and resource allocation without necessarily maximising request allocations on first attempt for its customers (requesters).

Dynamic analysis (simulation) is used in conjunction with timing in the net. Timing delays were introduced at various intermediate places within requester and assigner processes using both stochastic and deterministic distributions to represent random

request placement and delays between each activity in the request close air support assignment process. A record declaration was used for each request in order to store the model time at which the 'send_CAS_req' activity executes (Fig. 5.14). This was viewed as the start of the attempt by the underlying communications infrastructure to connect the requester with the assigner. Again, a time delay was introduced here to the record token to represent the delay of the underlying communications infrastructure. The toolset data collection functionality was used to compare the model time following execution of the requester's 'receive_CAS_resp' transition with the start time of the transmission of the request. The result was viewed as the request fulfilment duration (Table 5.4). A timed net facilitated the specification of a timeout (Fig. 5.14) following execution of the 'send_CAS_req' transition. If for whatever reason a desired response to the sent request is not received within a certain time limit, the net terminates with a 'Timeout' problem result.

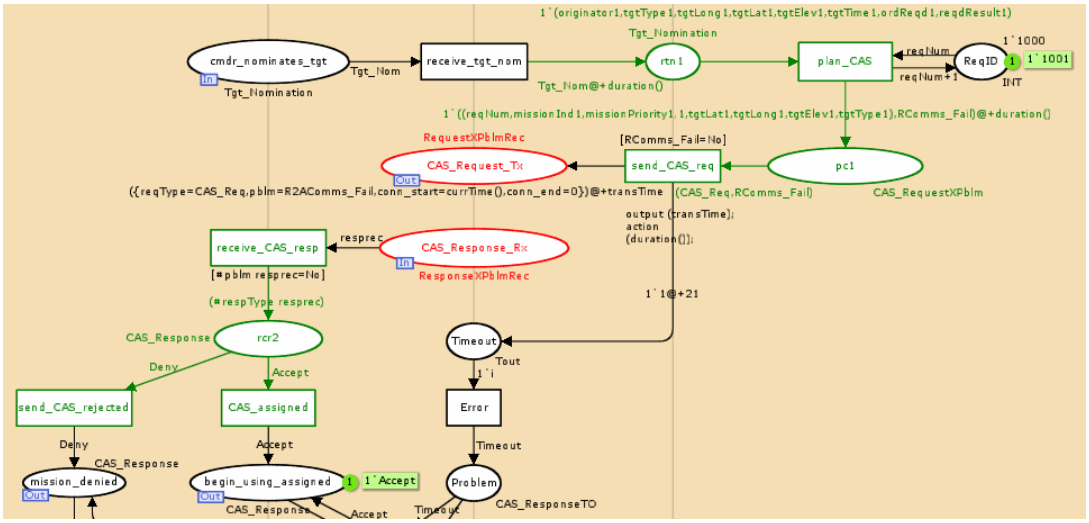


Fig. 5.14 'Requester subnet in performance analysis net of assign close air support request process'

It should be noted that it is also possible to specify an un-timed timeout mechanism in the net. This form of timeout may need to be specified at more than one location in the model depending on the potential failure areas.

Request fulfillment i.e. accepted or denied	#data counter step time
=================================	4 1 6 21
{respType=Accept,pblm=No,conn_start=17	

Table 5.4 'Toolset data collection functionality capturing request fulfilment duration'

Also mentioned in Appendix C, section C.1 for the telephone example was using knowledge of (legacy or planned) physical assets to help optimise engineering of the operational process level via analysis-of-alternative scenarios. Salimifard et al [94] report on using nets to allocate physical resources and costs to activity execution. This work is highly relevant to the development of large-scale system-of-systems, facilitating analysis-of-alternatives in operational process engineering and improved design quality. The adaptation for close air support is shown in Fig. 5.15.

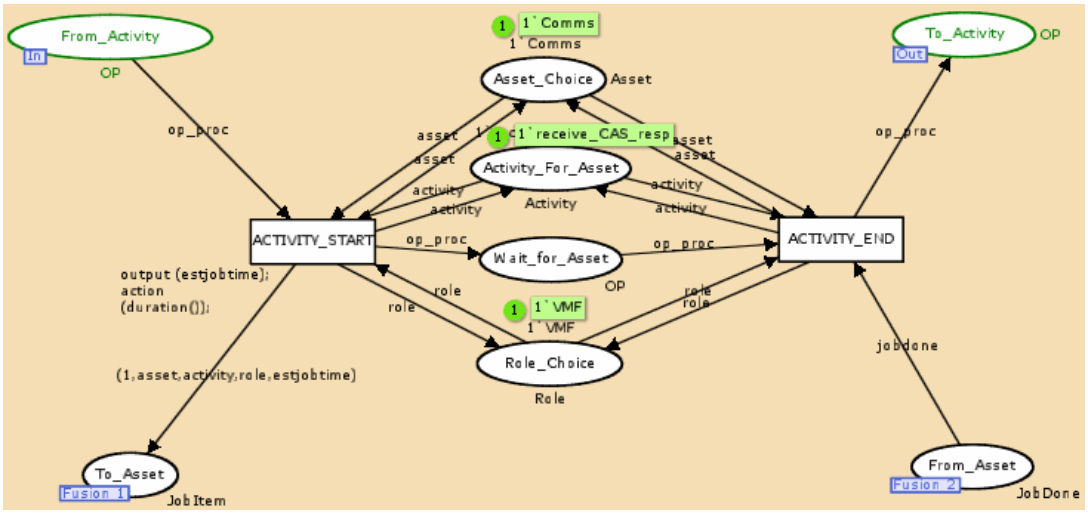


Fig. 5.15 'Activity subnet detailing physical asset and role to perform receive_CAS_resp activity'

Timing and cost information can be extracted from the overall net per activity, per component process, and for the process overall based on the physical assets used to realise the component activities and processes. An example data collection report is shown in Fig. 5.16 relating to the 'ACTIVITY_END' transition of Fig. 5.15 and its associated bindings. This analysis-of-alternatives net can be used in conjunction with the performance analysis net of Fig. 5.14 to adjust timing duration ranges.

```
#data counter step time
200 1 17 2
```

Fig. 5.16 'CPN Tools data collection log file capturing cost information based on resources allocated to perform communication activities'

It should be noted that although simulation was primarily used in this section to validate the models, it is also possible to conduct static analysis (as per recommendations from the telephone example, in Appendix C) to verify their correctness.

From this section, it has been shown that it is possible to use timing in coloured Petri nets to enhance correctness and conduct performance and analysis-of-alternative analyses. Use of timed colours (types), and suitable inscriptions on output arcs (in conjunction with stochastic or deterministic functions) help to specify duration of activities (and execution control flow) such as information exchange, task processing, arrival of requests, and timeout error recovery in the event of a communications failure.

5.5 Design and Architectural Levels of Abstraction for Close Air Support

Sections 5.2-5.4 have focused on using Petri nets to specify the close air support process at an analysis level of abstraction. Hierarchy and timing additions have been looked at to further enhance a specification in terms of scalability, understandability, readability, and correctness. As demonstrated in the telephone system example in Appendices B-C, hierarchy can also enable investigation into whether the abstraction design used in the net could be used to help alleviate the state space explosion problem during model-checking. Both model-checking and simulation were employed iteratively in verification and validation of the constructed analysis level net. Before it can be decided whether the criteria for success in relation to the close air support case study have been met and conclude chapter 5, Petri nets are checked to see if they can address the problem of being able to specify close air support at design and architecture levels of abstraction. This is the objective of this section.

5.5.1 The Design Level

The purpose of the design level of abstraction is the lead into the specification of a solution to the problem described by the analysis level. Again, a functional decomposition approach was used. This time it was used in conjunction with the parent net developed for the analysis level to think about how this net's main activities (e.g. 'Request Close Air Support' and 'Assign Close Air Support') would eventually be realised by physical implementations. To keep the design flexible, two components, 'Make_CAS_Request Component' and 'Assign_CAS Component', were used to depict the solutions that would realise each of the main activities. These are shown in Fig. 5.17.

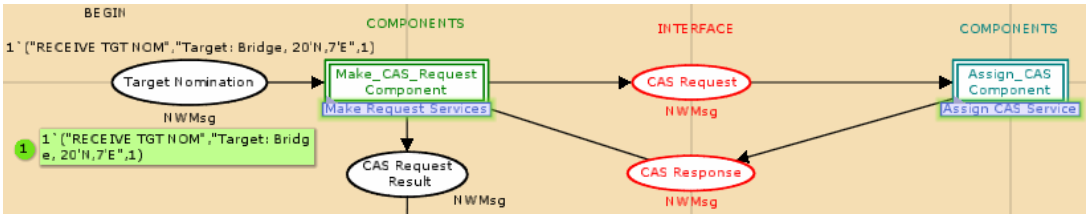


Fig. 5.17 'Parent net of design level'

It can be seen that Fig. 5.17 closely resembles the parent net of the analysis level except for the new place colours (types). The next level of design decomposition for the two components aimed to capture the functional service(s) each would be expected to realise. Again, the work developing the analysis level net helped suggest functional services for the design level by thinking about the purpose of the processes used to realise the main activities. 'Make_CAS_Request Component' would be responsible for providing close air support request setup and response services. 'Assign_CAS Component' would be responsible for providing an incoming close air support request processing service. These services are shown at the next lower abstraction level providing greater detail in Fig. 5.18.

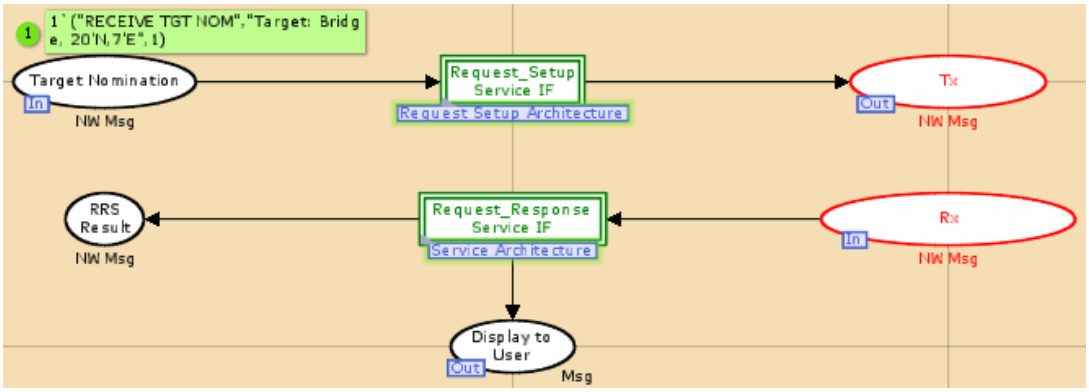


Fig. 5.18 'Services of Make_CAS_Request Component'

Having introduced the components and functional services at the design level, the next lower abstraction level providing greater detail, i.e. detailed design or architecture was focused on. Rather than develop a separate model at this stage, as the architecture level appeared to naturally manifest the next lower abstraction level of the design level, the design level model was further decomposed to capture the architecture level.

5.5.2 The Architecture Level

The purpose of the architecture level is detailed design of the services identified at the design level and flexible capture of the components required to realise these individual services. Constituent components were considered for each functional service resulting in the identification of a common component pattern for the three services associated with close air support request-response. The common components consisted of a user interface, transmit and receive (network) interfaces, and a controller interface to co-ordinate the sequencing of activities to and from the other two common components. The common component architecture is shown for the 'Request_Response Service' in Fig. 5.19.

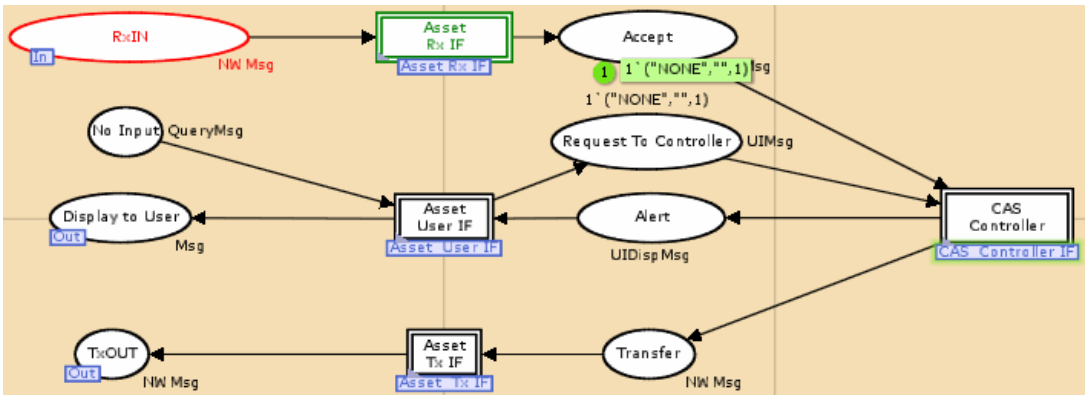


Fig. 5.19 'Service architecture'

From Fig. 5.19, it can be seen that net places are used to capture the input and output information for the user interface, network and controller common components.

Colour (type) definitions were lifted for re-use from the telephone example net (Appendix D) and adapted accordingly (definition labels reflect the nature of the interface, e.g. 'UIDispMsg' aims to reflect that the place is an input interface to the user interface component and is intended to be processed by the display function within this component). The intention with this labelling convention was improved net clarity and comprehension. Place types were based on character strings rather than enumerated types for flexibility reasons. The tuples in the type were populated with the functions implemented by each common control component and the associated parameters via logic on transition output arcs. Logic on transition output arcs within each of the common components was amended as necessary. As an example, consider the network transmit common component in Fig. 5.20.

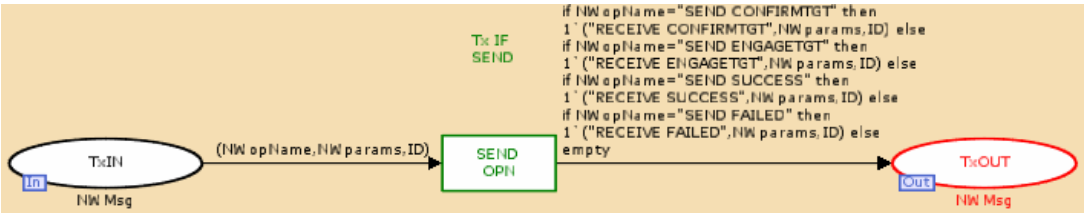


Fig. 5.20 'Transmit common component subnet'

Fig. 5.20 shows the subnet of the transmit common component. Its transition is labelled as 'SEND OPN' to reflect the function the component provides to the controller component. On the transition output arc (within the controller component subnet) to the input interface place ('TxIN') of the transmit component, there is logic to output a token with 'OpName' (a tuple within 'NWMsg' compound type) populated with required functions such as 'SEND REQUEST' or 'SEND ENGAGETGT'. In this way, the net specifies use of the transmit component's 'SEND OPN' function by the controller component more explicitly. The 'Params' tuple within 'NWMsg' is populated with values relevant to the function of the message described by 'OpName'. 'ID' is populated to differentiate between initiated requests. The other two common components, user interface and controller, are designed to reflect the same interface principles as those discussed above for the network components.

Considering the original parent net of the design level in Fig. 5.17, the specification of close air support at this level was extremely concise. When the architecture level of Figs. 5.19-5.20 was reached and the next lower abstraction level providing greater detail of the common component interfaces was completed, the modeller was extremely conscious of the requirement to manage the levels of abstraction. The toolset can present each level of abstraction as a separate page within a folder (or binder). These pages can be selected between using their tabs. By the common component interface level of abstraction for the request and assignment services, seventeen pages and tabs were present and it was tedious work identifying and selecting relevant pages. At this stage, the experience gained with the telephone example (Appendices A-D) was used to rationalise the model where possible, making use of the toolset's features and those of hierarchical coloured Petri nets. The main source of rationalisation was the common component interface nets.

5.5.3 Verification of the Design and Architecture Levels and further Specification

At this stage, simulation was employed to check the structure and logic of the model and was able to detect incorrect logic on transition output arcs. Errors included: missing or incorrect predicates (highlighted by incorrect or missing display notifications for the common user interface component or incorrect information messages for the network component); missing initial values on input places required by common component interfaces; and an unexpected disabled transition due to the same variable being used to bind values on more than one of its input arcs.

The necessary corrections were made and static analysis based on one initiated request was performed. No further errors were picked up by model-checking so the modeller proceeded to adapt the model for a close air support mission in its entirety.

The same process as used in sections 5.5.1-5.5.2 was followed for the other sub-functions composing a close air support mission, namely perform close air support briefing; perform close air support depart from initial point; confirm close air support target; and authorise close air support weapons delivery. Bearing in mind the experience gained developing the model of the telephone system (Appendices A-D) and one sub-function (request close air support assignment), the modeller continued to look for the most generic means of adding new components realising the remaining sub-functions of close air support to the system-of-systems model. Re-using the common component interface nets across the eight new components realising the additional four sub-functions resulted in an overall model of eighty-eight subnets. The toolset became increasingly difficult to work with in terms of organising folders according to sub-function and navigating subnets and associated colour (type) definitions. Without instantiation functionality, the construction process would have been even more tedious. Syntax checking is undertaken by the toolset on model opening and following creation of each new element of a net for the whole model and both duration and performance were adversely affected by the size of the close air support model.

Simulation was conducted incrementally, i.e. following the addition of each pair of components associated with each sub-function. Interactive simulation could only realistically be conducted per sub-function. An initial marking was set up for each pair of components and the model of the pair executed manually. Static analysis was attempted cumulatively following the addition of each pair of components. It was noted that with an imposed calculation time limit of eighty minutes, a full state space graph could only be calculated for three sub-functions, i.e. six components, fifty-three subnets, approximately one hundred and twenty places (Table 5.5).

STATE SPACE GRAPH	**Net of three sub-functions**	**Net of complete close air support (five sub-functions)**
Initial marking	1`("RECEIVE TGT NOM","Target: Bridge, 20'N,7'E",1)	1`("RECEIVE TGT NOM","Target: Bridge, 20'N,7'E",1)
Nodes and arcs	40128 nodes, 220064 arcs.	Explosion problem.

Generation time	688 secs.	4800 secs (limit set).
Terminal markings	2	N/A

Table 5.5 'State space graph calculation at design and architecture level'

In order to employ the benefits of static analysis on the full system-of-systems model of the close air support mission, the largeness avoidance techniques discussed in Appendices B-D were applied. In this case study, given that a full state space for three sub-functions could be calculated, the model was maintained by capturing one sub-function pair in detail and abstracting out the detail of one component in each of the four remaining sub-functions (Fig. 5.21). This process was then reversed, i.e. for the four sub-function pairs with one component originally abstracted, the detail was abstracted from the partner component. The results are shown in Table 5.6.

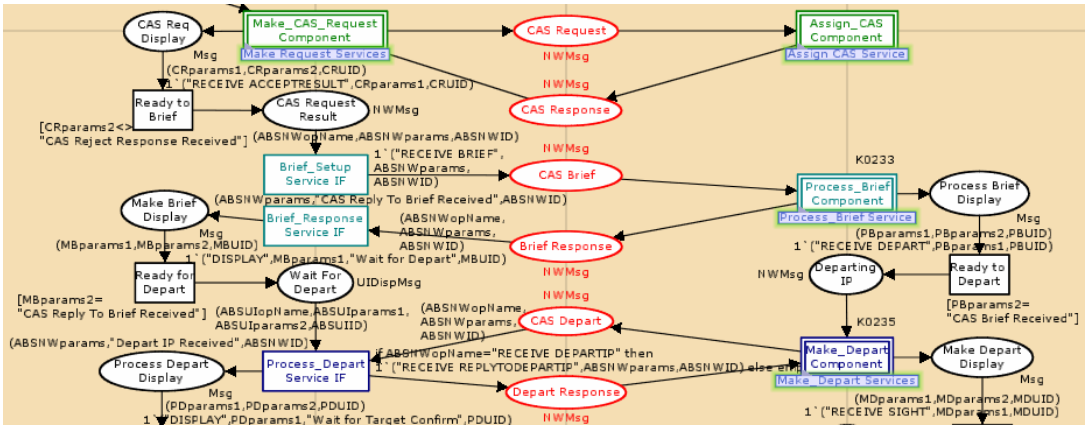


Fig. 5.21 'Abstraction largeness avoidance technique applied to one component in sub-functions (apart from Make Request and Assign CAS Services)'

STATE SPACE GRAPH	**Net of complete close air support (five sub-functions)**	**Net with one component in four out of five sub-functions abstracted**
Initial marking	1`("RECEIVE TGT NOM","Target: Bridge, 20'N,7'E",1)	1`("RECEIVE TGT NOM","Target: Bridge, 20'N,7'E",1)
Nodes and arcs	Explosion problem.	28544 nodes, 148800 arcs.
Generation time	4800 secs (limit set).	232 secs.
Terminal markings	N/A	3

Table 5.6 'State space standard report for full and abstracted net'

It should be noted the compositional largeness avoidance technique could also be applied to close air support, i.e. each sub-function could be analysed independently of the others (see Appendices B-D, sections B.1.5, C.1.3, and D.1.5 for details of this technique used with the telephone example).

From this section, it has been shown that it is possible to use coloured Petri nets and hierarchy to specify all five sub-functions of close air support (request close air support assignment) at design and architecture levels of abstraction, i.e. the solution specification level. From Figs. 5.17-5.21 above, net places capture state in terms of the information input and output to net transitions (activities); net colours (types) define the structure of the information used by activities (operations and parameters); net arc inscriptions govern the information required and produced by activities (including operations needed between components); net arcs dictate the control flow of execution, i.e. the order of activity execution and information exchange; toolset hierarchy facilitates levels of abstraction within the model (at the design level component and services, at the architecture level common components realising design level services) and offers instantiation for re-use of existing subnets; toolset colour palette and annotation improves readability of nets; and finally, dynamic and static analyses permit verification and validation of the models.

In terms of scalability, a model of three close air support sub-functions with fifty-three subnets and approximately one hundred and twenty places (and being kept as generic as possible employing instantiation) permitted calculation of a full state space graph. Beyond this, abstraction largeness avoidance techniques had to be applied. Scalability of the models within the toolset itself was the main issue. Manageability and navigability of the model within the toolset were considerably compromised. Without use of instantiation, model creation would have been extremely hampered. As it was, syntax and semantic checking and editing response times were degraded and the creation process time intensive and difficult.

5.6 Evaluation of Close Air Support Study

The design objective of the close air support study was derived from chapter 1's second criteria for success. This stated that the strengths and weaknesses of Petri nets regarding the greater formalism of dynamic behaviour in systems-of-systems and the role of Petri nets as a means of engaging stakeholders were to be determined. Following the case study design from chapter 3, nets were used to specify, verify and validate a close air support mission. As dictated by the study plan, data (evidence) was collected at design iterations of each model using screenshots of the model and simulations, standard reports from CPN Tools, model source code from CPN Tools, and project team notes. This evidence was presented in the report of the study in this chapter and discussed further in this section.

5.6.1 Quantitative Results

These related to the third criteria for success and the first research question, i.e. do Petri nets improve the functional correctness of the system-of-systems design specification? The functional correctness response variables were expressed in terms of the number of errors detected by simulation, and number of errors detected by static analysis. This data was captured from the CPN Tools integrated development environment at each iteration of model design through simulation (and screenshots, note-taking) or reachability graph calculation (and CPN Tools standard analysis report, screenshots, note-taking). An overview of the results is shown in Table 5.7.

Criteria for Success Goal 3 / System-of-System Engineering Level	Simulation Number of Errors Detected	Static Analysis Number of Errors Detected
Analysis	3	3
Design and Architectural	3	0

Table 5.7 'Quantitative results from close air support study'

From Table 5.7 it can be seen that use of simulation detected six functional errors in the specification of close air support. With textual or UML-based specification, given the nature of the errors detected through simulation of the Petri net model, it would have been extremely difficult to detect the same errors using both these means of static specification.

Use of static analysis detected three functional errors in the specification of close air support. As well as the two errors detected using simulation, static analysis was able to highlight missing logic within a model and localise the part of the net where the omission was made. From examination of the errors detected by static analysis at the system-of-systems engineering analysis level, it can be seen that if the same errors were not detected using simulation, they would have been detected through calculation of the reachability graph of the net. In this way, static analysis offers a means of exhaustively checking a net on behalf of the modeller but the modeller needs to know how to interpret the analysis report in order to isolate the detected errors and this is a time-intensive process. The modeller also needs to be aware of the state space explosion problem and means of largeness avoidance (previously discussed in section 5.5.3 and Appendix B).

Given CPN Tools in its standard form, simulation (or the Petri net 'token game') is a more intuitive means of stepping through the behaviour of a model (depending on its size) to check its functional correctness. It was used as an initial means of model verification prior to conducting static analysis so that detected errors could be corrected, potentially helping to alleviate the state space explosion problem, and simplify the analysis report.

5.6.2 Qualitative Results

These related to the first and second criteria for success and the second and third research questions, i.e. do Petri nets increase the quality of the design specification, and what are the shortcomings of the state-of-the-art Petri net tool and how can it be improved? The design quality response variables were expressed in terms of comprehensibility (e.g. use of hierarchy, annotation, timing), and scalability. This data was captured from the CPN Tools integrated development environment at each iteration of model design through screenshots, and note-taking. An overview of the results is shown in Table 5.8.

Criteria for Success Goal 1&2 / System-of-System Engineering Level	Comprehensibility Hierarchy, annotation, timing	Scalability
Analysis	Able to express problem (operational process, including timing specification) using CPN Tools.	Yes.
Design and Architectural	Able to express solution design (including timing specification) using CPN Tools.	Toolset scalability issues for net of 53 subnets, approximately one hundred and twenty places.

Table 5.8 'Qualitative results from close air support study'

These qualitative (and quantitative) results are now evaluated further in relation to the modelling features identified as desirable for the specification of system-of-systems.

1. Abstraction
In terms of UML 2.0's activity diagram, activity decomposition can be specified using class and activity diagrams. Class diagrams can show composition associations between activities and instance cardinality. Activity diagrams detail the refinement of an activity. They can also allocate activities (behaviour) to structure (classes) via partitions (swim lanes). These partitions can also show multiple allocations.

In terms of the Petri net formalism, activity decomposition can be specified using high-level Petri nets with hierarchy. Implementations of Petri nets with hierarchy include substitution transitions, and place and transition fusion.

Results from the first study (close air support specified using CPN Tools) indicated that hierarchy needs to be determined in advance of modelling. In the study, tactical data link (specifically Variable Message Format) messages were used in a bottom-up approach to derive hierarchy from the suggested message functions. Hierarchy was defined for models at the system-of-systems engineering level (analysis, and design and architectural) as well as within models (a hierarchy based on function was used).

CPN Tools implementation of coloured Petri nets with hierarchy uses substitution transitions and fusion places. Substitution transitions were used primarily for abstraction as the sockets and ports used by this method were viewed as a means of explicitly specifying required and provided interfaces to the decomposed transition.

Fusion places can also be used to represent abstraction but there is no explicitly associated net page at a higher-level of abstraction as there is with the substitution transition method. An advantage of fusion places is the ability to share the same information between multiple processes, e.g. if information needs to be passed from an interface to more than one component system at a point in time.

Instantiation was used in conjunction with substitution transitions and found to be essential in minimising model size. Generalisation, i.e. use of common components at the design and architectural level of abstraction was also regarded as essential for the same reason.

The study highlighted that there was no explicit composition, cardinality, or allocation concrete notation with nets. The present means of achieving these in CPN Tools would be annotation and/or extra net elements. Extra net elements would be needed to capture multiple allocations.

2. Modularisation
In terms of UML 2.0's activity diagram, modularisation can be specified using class and activity diagrams. Class diagrams can specify the activities and the data items associated with them (including cardinality).

Activity diagrams show activities and the associated control flow of input (output) data item instances (parameter types are defined by class diagrams). Class and collaboration diagrams can also be used to specify provided and required interfaces between classes.

In terms of the Petri net formalism, modularisation can be specified using high-level Petri nets with hierarchy. Implementations include substitution transitions, and place and transition fusion.

Results from the first study (close air support specified using CPN Tools) indicated modularisation was primarily achieved using substitution transitions (and port and socket places) to represent component system interfaces. As discussed for the abstraction feature, place fusion can also be used to partition a model but does not explicitly associate the lower abstraction level net with the higher abstraction level net.

Information exchange protocol was described between and within component systems using colours (types) to define the exchanged information and net elements (places, transitions, arcs, arc inscriptions, annotation) to specify the control of the exchange.

CPN Tools makes no provision for model re-use (e.g. searching for suitable existing models to use in a bottom-up approach) and their management (e.g. versioning).

Capture of existing component systems including their performance parameters was not undertaken.

Data items input and output by activities can be specified through colours (types) and variable bindings. Cardinality would need to be specified through annotation.

3. Data typing
In terms of UML 2.0's activity diagram, classes defined previously in class diagrams specify data typing.

In terms of the Petri net formalism, timed, high-level Petri nets specify data typing.

Results from the first study (close air support specified using CPN Tools) indicated that use of CPN Tools implementation of timed, coloured Petri nets enabled capture of the information needed for close air support using a combination of simple, compound, and timed colours (types). There was no ability to refer to variable values unless the same variable bindings were propagated through the entire net.

There is no equivalent one diagram, static description in nets. Operations (and the required parameters) provided by system components were made more explicit using arc inscriptions and annotation, and their associated data types were provided by local place colours (types).

4. Adequate toolset implementation
In terms of UML 2.0's activity diagram, many open source and commercial toolsets offering various levels of integrated UML development exist.

In terms of the Petri net formalism, several implementations of high-level Petri nets offering various levels of integrated net development exist.

Results from the first study (close air support specified using CPN Tools) indicated that CPN Tools provided a useful integrated net environment for the specification of systems-of-systems but shortfalls were identified in the areas of: improved analysis reports, examples and best practice, and large model support (navigability, syntax-checking, versioning, error-reporting, and animation).

5. Timing
In terms of UML 2.0's activity diagram, extension via the Profile for Schedulability, Performance, and Time (to be replaced by Profile for Modelling and Analysis of Real-time and Embedded Systems) is needed in order to specify (non-functional) timing properties. The resulting activity diagram is static so performance analysis, or investigation into analysis-of-alternatives is only achievable through model conversion.

In terms of the Petri net formalism, timed, high-level Petri nets with hierarchy specify timing information.

Results from the first study (close air support specified using CPN Tools) were based on CPN Tools implementation of timed, coloured Petri nets with hierarchy. Stochastic and deterministic functions within the toolset were used to introduce random placement of requests, delays, and timeouts. Time, cost and quality performance indicators need to be identified in advance of the analysis.

Simulation was used in conjunction with toolset monitors in the performance analysis net to calculate close air support request fulfilment duration. An analysis-of-alternatives net used physical asset known performance data in the simulation time and was used to calculate cost information per activity, per component process, and for the overall process. Analysis-of-alternative nets can be used to advise the timing duration ranges of performance analysis nets.

Timing can impose order on tokens but cannot guarantee the prevention of token overtaking (queuing places were defined for this purpose). Timing was primarily used to reflect deterministic and stochastic activity durations for the purposes of performance analysis.

6. Verification and validation

In terms of UML 2.0's activity diagram, there is no full formal syntax and semantics and diagrams cannot be executed.

Verification and validation is done using static inspection of the graphical notation which may include extensibility profiles such as Profile for Schedulability, Performance, and Time, and the Profile for Modelling QoS and Fault Tolerance Characteristics and Mechanisms.

In terms of the Petri net formalism, there is formal syntax and semantics and nets can be executed using a well-defined execution algorithm (simulation). Additionally, exhaustive verification can be achieved by calculating the reachability graph of the net and checking structural properties such as deadlock. Timed, coloured Petri nets and simulation were used to conduct performance analysis and analysis-of-alternatives.

Results from the first study (close air support specified using CPN Tools) were based on CPN Tools simulation modes for initial investigations into analysis, and design and architecture model behaviour (six errors were detected: infinite loop; token overtaking; potential deadlocks; incorrect predicates; missing initial values; and incorrect disabled transition), performance analysis (the calculation of close air support request fulfilment time was demonstrated), and analysis-of-alternatives (the calculation of the cost of the close air support process using Variable Message Format as the communications physical resource was demonstrated). For large system-of-systems models, interactive simulation was time-intensive. This was managed by building nets incrementally and simulating the new net elements.

Verification and validation of close air support using simulation is constrained by the size of the system-of-systems model. Simulation can be used to check the behaviour of an entire model and analyse performance where timing is used but it cannot exhaustively verify the correctness of the entire model.

CPN Tools was used to calculate a reachability graph for analysis, and design and architecture nets (three errors were detected: infinite loop; token overtaking; and missing logic), and temporal logic was used to confirm correct behaviour of one information exchange protocol across the generated reachability graph. Temporal logic queries are text-based and require experience to formulate correct queries.

Modeller experience helps significantly in interpretation of the analysis report produced by CPN Tools and to highlight unexpected analysis results. It is unlikely that all these errors would have been detected within the UML activity diagrams using static inspection alone.

For large system-of-systems models, the state space explosion problem was alleviated through largeness avoidance techniques. Reachability graphs were also calculated for timed nets using CPN Tools. These require careful management in terms of removing sources of non-determinism within the net.

Verification and validation of close air support using reachability graph calculation is constrained by the number of states in the system-of-systems. Model-checking means ignoring parts of the system-of-systems either through abstraction or considering a

subset of the system-of-systems. However, model-checking is automatic and exhaustively checks the model.

7. Precision in specification of requirements (scalability, concurrency, state-based specification, information-based specification, event-based specification)
In terms of UML 2.0's activity diagram, it has intuitive, graphical concrete syntax but does not have fully formal syntax and semantics. Activity diagrams can be used for concurrent, scalable, state, event, and data-based specification (in conjunction with class diagrams). They have the ability to specify continuous (streaming) and discrete, non-streaming activities.

Activity definition (Activity in UML) defines an activity independently of how it is used in a diagram, it does not specify where input (output) to it originates (goes to).

Activity usage (Action in UML) defines how an activity is used in the definitions of higher-level activities.

Activity instance is the enabling of an activity operating on input (output) item instances and associated timing.

Item type (Classifier in UML) specifies the type of input (output) item to (from) an activity independently of where it is used.

Item usage by activity definition (Parameter in UML) defines how an item is used in an activity definition independently of where the activity is used. Parameters are named indicating the kind of item (parameter type).

Item usage by activity usage (Pin in UML) defines the connection point between a flow line and a parameter at an activity.

Item instance refers to item used by an activity instance.

Parameters refer to items input to (output from) activities. Their types are defined in class diagrams and they are named. Items used by activities are specified by pins labelled with parameter name and type.

Presently, activity diagrams have no means of specifying: persistent data store across activity executions; the number of concurrent executions allowed for single usage activities; and resources generated (consumed) for activity execution (pre and post-conditions on activities are supported but do not specify effect on execution).

In terms of the Petri net formalism, timed, high-level Petri nets are used for concurrent, state, data, event-based specification. Petri nets ability to scale requires management due to their limited concrete syntax.

Results from the first study (close air support specified using CPN Tools) indicated that at the system-of-systems design level, systems-of-systems have non-streaming activities, i.e. terminating, discrete-event (rather than continuous) where items are accepted at the start of activity execution, processed, and output at the end of activity execution. Specification of continuous or streaming activities (i.e. activities dealing

with inputs and outputs continuously during their execution) would require stochastic Petri nets.

In the study, nets were developed to specify analysis, and design and architecture models of close air support. At analysis level, operational processes were described. This led into solution specification at design and architecture levels. Again, these levels described process-based information exchange with associated events, and states. The specification was unambiguous in the structural sense.

Tokens are not accepted by transitions in process of execution and can be queued. More than one token input can be specified to a transition but it is not possible to specify acceptance of one token and then a late token. Multiple outputs can have probability applied to them. There is no concept of persistent data store accessible across transition executions.

It is possible to describe a function independently of how it is to be used by other functions and how the function is to be used in the context of other functions but it means using separate nets and annotation. Groups have to be created in CPN Tools in order to clone and re-use parts of nets in multiple model locations. Nets were able to specify resources generated (consumed) for transition execution.

In CPN Tools, colour (type) definitions exist independently of the activities they are used in.

Due to their generic concrete syntax, nets rely on extra net elements and annotation (in comparison to activity diagrams) to relate domain and system specification concepts (e.g. iteration, decisions, cardinality, operations, parameters, constraints). This means resulting nets are much larger than the equivalent activity diagram and can lead to scalability issues. Scalability issues were encountered in close air support at the design and architectural level of abstraction for fifty-three subnets and approximately one hundred and twenty places. Toolset syntax and semantic-checking duration and editing response times increased significantly. Manageability and navigability of nets within the toolset were severely compromised.

The close air support study assumed no multiplicity of component systems or processes realising functions. The concern with multiplicity is the correctness of the execution path if functions are undertaken by more than one component or process. Also, the study did not investigate re-entrancy, i.e. if a recovery protocol is specified for a communications failure, when it completes, execution may return to earlier transitions leading to incorrect behaviour.

These results are summarised in Table 5.9.

System-of-Systems Modelling Need	**UML Activity Diagram**	**Petri Nets**
1. Precision in requirements specification		
Formal syntax & semantics	No.	Yes.
Process-based	Yes.	Yes.
Multiplicity	Yes.	To be determined.
Re-entry	Yes (requires management).	To be determined.

System-of-Systems Modelling Need	UML Activity Diagram	Petri Nets
Discrete	Yes.	Yes.
Data flow	Yes.	Yes.
Resource usage	No.	Yes.
Scalable	Yes.	No (requires management).
State	Yes.	Yes (to a greater extent than UML).
Control flow	Yes.	Yes.
Concurrency	Yes.	Yes.
Independent activity description	Yes.	Yes (separate net).
Independent data description	Yes.	Yes (colours).
Persistent data	No.	No.
Interfaces	Yes (plus class diagram).	Yes (annotation & hierarchy).
Information exchange protocol	Yes (plus schedulability profile).	Yes (timed nets).
Analysis size	More compact than nets.	Larger models than UML.
Design & Architecture size	More compact than nets.	Larger models than UML.
2. Verification and validation		
Formal syntax & semantics	No.	Yes.
Static inspection	Yes.	Yes.
Dynamic inspection (simulation)	No.	Yes.
Behaviour checking		Yes.
Performance analysis		Yes (timed nets).
Analysis-of-alternatives		Yes (timed nets).
Exhaustive analysis		No.
Complete specification		Yes (constrained by net size).
Reachability graph calculation	No.	Yes.
Structural properties		Yes (e.g. deadlock, boundedness).
Temporal logic queries		Yes (e.g. correct protocol).
Largeness avoidance		Yes (abstraction, net division).
Exhaustive analysis		Yes.
Complete specification		No (dependent on scope).
QoS	Yes (plus QoS profile).	Yes (annotation).
3. Abstraction		
Decomposition	Yes (plus class diagram).	Yes (substitution transitions & fusion places).
Activity composition	Yes (plus class diagram).	Yes (annotation, separate net).
Cardinality	Yes (plus class diagram).	Yes (annotation, separate net).
Allocation	Yes (swim lane).	Yes (annotation).
4. Modularisation		
Interfaces	Yes (plus class, collaboration diagrams).	Yes (substitution transitions, fusion places, & colour).
Information exchange protocol	Yes (plus class, collaboration diagrams and schedulability profile).	Yes (timed nets).
Top-down, bottom-up support	Yes (top-down), Yes (bottom-up).	Yes (hierarchy & cloning).
5. Timing		
Duration, timeout, arrivals	Yes (plus schedulability profile).	Yes (timed nets, deterministic & stochastic functions).

System-of-Systems Modelling Need	UML Activity Diagram	Petri Nets
Static	Yes.	No (simulation).
6. Data typing		
Domain concepts	Yes (plus class diagram).	Yes (timed, coloured nets).

Table 5.9 'Summary of study results from the specification of close air support using Petri nets'

5.6.3 Evaluation Conclusions

Referring to Tables 5.7-5.9, in terms of the first criteria for success, 'Precisely specify the close air support example (research questions 2 and 3)', Petri nets were used in a top-down engineering approach to unambiguously (in terms of model structure) capture the operational processes, component systems and information exchange involved in close air support at analysis, design and architecture levels of model abstraction. A bottom-up approach was used to identify the functional hierarchy used in the models from the existing close air support tactical data link message set. These abstraction levels described process-based information exchange with: associated events; component system interfaces; states; type of information exchanged; information exchange protocol; execution control flow; initial request arrival timing, event durations and timeouts; interfaces and operations used by component systems; and potential failure states (e.g. underlying communications failure). In terms of the desirable modelling needs of systems-of-systems, from the first study on close air support, Petri nets meet an additional two attributes over activity diagrams (formal syntax and semantics, and specification of resource usage). However, the study strongly suggested that Petri nets did not scale well, attributable to their generic concrete syntax. Nets rely on extra net elements and annotation (in comparison to activity diagrams) to relate domain and system specification concepts.

In contrast, activity diagrams offer multiple specification concepts in their concrete graphical syntax compared to the Petri net concrete graphical syntax. This means specification with activity diagrams is usually more concise than the equivalent Petri net specification but not necessarily more understandable. Activity diagram practitioners and non-practitioners need to have some understanding of the multiple underlying concepts and associated graphical notation. From the first study on close air support, Petri nets appear to be able to meet the precision in specification of system-of-systems requirements need and describe close air support at the system-of-systems analysis level of abstraction completely and visually (for the benefit of domain users), ready for correctness verification. Although the study was able to specify close air support at the design and architecture level of abstraction completely, navigation and management of the resulting model was severely degraded. It is not clear exactly how concise the Petri net specification is in comparison to specification with activity diagrams. The study also highlighted labelling and annotation within nets can be open to interpretation to stakeholders but this trait also affects activity diagrams.

For the second criteria for success, 'Determine the scalability of the close air support system-of-systems model implemented using Petri nets (research question 2)', the study indicated Petri nets may not scale according to the size of the system to be

modelled. Scalability issues using the toolset were encountered at the design and architectural level of abstraction for fifty-three subnets and approximately one hundred and twenty places. Due to their generic concrete graphical syntax, nets tend to use extra net elements and annotation to relate domain and system specification concepts (e.g. iteration, decisions, cardinality, operations, parameters, constraints). This means the resulting net is much larger than the equivalent activity diagram and will require careful management to achieve scalability.

Based on analysis of the first and second criteria for success, Petri nets would be recommended as a means of improving activity diagrams. In order to scale, system-of-systems specification using Petri nets needs to be suitably abstracted. Although close air support was specified at analysis, and design and architecture levels of abstraction, hierarchy based on function was determined in advance of modelling and detail minimised as far as possible. Use of toolset instantiation and common components at the design and architecture levels of abstraction were essential in achieving a complete specification that was also readable.

For the last criteria for success, 'Confirm if the same Petri net verification and validation techniques used in the telephone exercise are effective in the close air support system-of-systems specification models (research question 1)', CPN Tools was used to explore verification and validation of the close air support specifications at the analysis, design and architecture levels of abstraction. In terms of the desirable modelling needs of systems-of-systems, from the study, Petri nets meet an additional three attributes over activity diagrams (formal syntax and semantics, dynamic inspection, and reachability graph calculation). Close air support highlighted the fact that reachability graph calculation provides exhaustive verification but only across a restricted (in terms of model size and detail) specification. In comparison, simulation provides verification across the whole specification (constrained by model size, toolset, and underlying hardware) but is not an exhaustive means of checking model correctness.

Simulation detected infinite loop, token overtaking, potential deadlocks, incorrect predicates, missing initial values, and incorrect disabled transitions specification errors in the close air support system-of-systems. Simulation also enabled domain users to be involved in the model correctness-checking process. These errors would not necessarily be highlighted during static inspection of the equivalent UML activity diagrams. When timing was introduced into the close air support nets, simulation could be used to undertake performance analysis (the calculation of close air support request fulfilment time was demonstrated), and analysis-of-alternatives (the calculation of the cost of the close air support process using Variable Message Format as the communications physical resource was demonstrated). Again, this is not possible with activity diagrams.

As mentioned, simulation is not an exhaustive method of checking model behaviour is correct. Reachability graphs for the close air support analysis, and design and architecture models were calculated for this purpose. As indicated in Table 5.7, reachability graph analysis detected three errors in the close air support system-of-systems specifications (the infinite loop, token overtaking, and missing logic). If the first two errors had not been detected by simulation, reachability graph analysis would have alerted the modeller to their presence. Exhaustive verification is not possible on

native activity diagrams. Temporal logic was also used to confirm correct behaviour of one information exchange protocol across the generated reachability graph.

Based on analysis of the last criteria for success, again Petri nets would be recommended as a means of improving activity diagrams.

The close air support study assumed no multiplicity of component systems or processes realising functions. The concern with multiplicity is the correctness of the execution path if functions are undertaken by more than one component or process. Also, the study did not investigate re-entrancy, i.e. if a recovery protocol is specified for a communications failure, when it completes, execution may return to earlier transitions leading to incorrect behaviour.

From Table 5.9 and the close air support case study, the confirmed benefits of the Petri net formalism are:

1. Analysis capability regarding the modelled design specification. The Petri net formalism facilitates both simulation and reachability graph calculation based on its full formal syntax and semantics. This capability is essential in helping to preserve functional correctness of the system-of-systems specification.
2. Specification capability. The CPN Tools implementation of timed coloured Petri nets with hierarchy offers the ability to graphically represent state, event, concurrent, performance, and data-based behaviour of a system-of-systems at different levels of detail. Nets can be logically divided to represent components facilitating top-down and bottom-up engineering approaches in system-of-systems engineering.

From Table 5.9 and the close air support case study, the confirmed weaknesses of the Petri net formalism are:

1. State space explosion. Logical structure-dependent, the calculated state space graph can have infinite reachable states. Largeness avoidance techniques may be able to help alleviate the problem.
2. Scalability.

In terms of shortfalls to the specification of system-of-systems requiring further investigation in relation to Petri nets, the study highlighted:

1. Verification and validation. The close air support study indicated both simulation and static analysis can be used to detect erroneous behaviour in the model. However, further insight into how it can be used to check the completeness of a system-of-systems specification would be useful, especially when used together with specification of multiplicity.
2. Multiplicity of component systems or processes realising functions. The concern with multiplicity is the correctness of the execution path if functions are undertaken by more than one component or process.
3. Re-entrancy. If a recovery protocol is specified for a communications failure, when it completes, execution may return to earlier transitions leading to incorrect behaviour.

Chapter 6 Case Study (Exchange Network Parameters)

6.1 Introduction

This chapter implements the case study design for a second time for two reasons. The first is to demonstrate the experiment is replicable and reliable, i.e. the process outlined in chapter 3 relating to construction of the Petri net models was followed again in order to check the desired response variables (at each iteration of model design). In this way the evidence obtained from the second study could be verified against the evidence obtained from the first study to demonstrate similar results using the Petri net formalism treatment.

The second reason for executing a second study was to investigate the results that needed further clarification from the first study, in particular the specification of multiplicity and re-entrancy, and the role of verification and validation in helping to ensure completeness and correctness of a system-of-systems design.

Similar to chapter 5, and using the case study objective and research questions identified in chapter 3, the case study exercise begins by defining the criteria for success by which the Petri net approach is measured. The objective of the case study and research questions from chapter 3 stated:

Case Study Objective: evaluate the strengths and weaknesses of Petri nets in terms of the system-of-systems problems identified in chapter 1.

Research Questions:
1. Do Petri nets improve the functional correctness of the system-of-systems design specification?
2. Do Petri nets increase the quality of the design specification?
3. What are the shortcomings of the state-of-the-art Petri net tool and how can it be improved?

Criteria for Success derived from the research questions above:

Goal 1 (research questions 2 and 3): Precisely specify the exchange network parameters example. Use Petri nets to capture the operational processes, components and information exchange involved in the system-of-systems completely, concisely and correctly at analysis, design, and architecture phases. Metrics: Check if Petri net elements can describe operational processes, components, information to be exchanged, information interfaces, information exchange protocols, multiplicity, and re-entrancy for the exchange network parameters system-of-systems. Note syntactical, semantic, and feature support of selected toolset.
Goal 2 (research question 2): Determine the scalability of the exchange network parameters system-of-systems model implemented using Petri nets.

Metrics: Explore the use of hierarchy within Petri nets to check if they can be used to create a scalable specification model of exchange network parameters.
Goal 3 (research question 1): Confirm if the same Petri net verification and validation techniques used in the telephone and close air support exercises are effective in the exchange network parameters system-of-systems specification models. Use Petri nets and the selected toolset to explore verification and validation of the exchange network parameters specifications at the analysis, design and architecture phases. Metrics: Employ static (state space) analysis of nets to check for well-known and user-defined properties in models. Employ dynamic analysis (simulation) of nets to explore correctness and completeness of behaviour and efficiency of specifications. Functional correctness is expressed in terms of number of errors detected by simulation; and number of errors detected by static analysis. Investigate the application of largeness avoidance techniques.

Table 6.1 'Criteria for success for the specification of exchange network parameters using Petri nets'

This process is described in the next section.

6.2 Specification of Exchange Network Parameters using Coloured Petri Nets

6.2.1 Description of Exchange Network Parameters

From military doctrine [121, Appendix E], exchange network parameters is a Combat Net Radio network management process defined as:

'..covers the subnetwork operations for a MIL-STD-188-220 subnetwork. If a station moves into or through different subnetworks, the Data Link addresses and operating parameters may change in each subnetwork, and exchange network parameters will automate the change of parameters and address from one subnetwork to another'.

Similar to the close air support study, this system-of-systems problem required further investigation. Currently, the only source of military documentation for exchange network parameters is [121, Appendix E] and this was consulted in conjunction with subject matter experts. It became clear that the problem of exchange network parameters was relatively new in the military tactical data link domain but the concepts behind it are related to those presented by the Dynamic Host Configuration Protocol internet standard [122].

[121, Appendix E] does not describe a generic process for dynamic network configuration management in the same way [122] does. It specifies a solution targeted towards a particular communications bearer for the Variable Message Format tactical data link, combat net radio. Consequently, it was difficult to obtain a textual description of the problem at a system-of-systems level from the standard [121, Appendix E]. Instead, both the military standard and the internet dynamic host

configuration protocol [122] were used as source documents to derive an exchange network parameters system-of-systems specification. The dynamic host configuration protocol offers an extra configuration process between the client and control nodes compared to the exchange network parameters protocol. Where a process relates to configuration over and above that of exchange network parameters, it is presented in italics in sections 6.2-6.5. In addition, exchange network parameters subject matter experts believed there to be omissions and implied requirements within [121, Appendix E]. These were to be made explicit during the course of the study.

For the purposes of the study, the scope of the problem was summarised in terms of functions to be provided, assumptions, messages involved, and information exchanges as follows:

The functions associated with exchange network parameters are persistent storage of network parameters and allocation of network addresses on a static (permanent) or dynamic (leased) basis.

The problem assumes: the roles involved are 'client node' (makes the request to join the network), 'control node' (responds to requests to join network), and 'relay node' (passes requests and responses to and from clients and control nodes where direct communication is not possible); fixed or wireless communications links susceptible to problems are used in the message exchange; and the message exchange uses a combination of broadcast, multicast, and unicast messages.

The messages involved are join request (issued from the new client node); initial join response (initial offer of network parameters from control node to client node); offer echo (confirmation of initial offer from client node to selected control node); *acknowledgement (confirmation from selected control node to client node regarding offered parameters); negative acknowledgement (denial from selected control node to client node regarding offered parameters);* and decline (denial from client node following detection of its proposed address being in-use).

In terms of information exchanges, the exchange network parameters process is triggered by the presence of a new node and its broadcast of a join request (which may be relayed to remote control nodes). Control node(s) respond to the client node join request with initial accept or reject response(s) containing the proposed network address, address lease period, and network parameters. Upon receipt of the initial response(s), the client node echoes confirmation of the initial accept offer to the selected control node using broadcast and relay if necessary.

The control node checks if the proposed address is already assigned, updating the configuration database and sending the client node a positive acknowledgement if it is able to satisfy its initial offer message. If the proposed address is already assigned, the control node sends a negative acknowledgment to the client node. When it receives a positive acknowledgment message from the control node, the client node also checks to see if the network address it has been assigned is already in use. If it detects a duplicate address, the client nodes sends a decline message and begins the network join process again. Where no duplicate address exists, the client node is configured and able to participate in the network. *If it receives a negative acknowledgement from the control node, the client node also begins the network join*

process again. If no initial offers are received by the client node, or if its initial offer is rejected, it begins the network join process again.

6.2.2 Petri Net Construction Method

Similar to the approach adopted for the telephone example in Appendices A-D and the close air support study in chapter 5, the modeller was keen to specify the exchange network parameters problem space in a way that promoted the need for a flexible, adaptable solution without being prescriptive and considered its specification from analysis, design, and architecture abstraction levels. In doing so, simplification through abstraction enables analysis of certain aspects of the exchange network parameters protocol by avoiding implementation detail. In terms of problem scope, the rules of interaction for exchange network parameters are considered rather than how messages are encoded or stored. The exchange network parameters function of processing a new network client node arrival is modelled (collection and update of the configuration details for the network is also modelled but is captured in much less detail and is included to show that this is a necessary function). Multiple entities are considered and the formats of the messages listed above are abstracted. Lease renewal, parameter updates, leaving the network, collection of network configuration data, and relaying processes were not included in this study. Dynamic rather than statically configured client nodes are focused upon.

The exchange network parameters problem was functionally decomposed into a series of functions and sub-functions (activities). These functions were then used with the concept summary above to suggest operational processes. The operational processes outline a particular timed ordering of activities and information exchange to be performed by roles so that the function (or service) may be realised. Optimised where possible in terms of resources, the operational process level helps to specify the overall function of the exchange network parameters system-of-systems to domain users and developers, and drive its lower-level design and implementation. At the analysis stage, functions (together with the information and information exchange protocol used by these functions) rather than the physical components able to meet these functions are specified.

To begin with, the net at the operational process (conceptual or analysis) model level of abstraction was created. Functions (activities) from the processes were mapped to net transition elements and information exchanged was assigned to net place elements following the control sequence presented within [121, 122] and guidance from domain experts. Colours (types) were defined in the toolset according to the information exchanged at each place. Compound or structured type definitions were used to specify the information exchanged. Roles (owners of the identified functions and sub-functions) were indicated by text labels on the page allocated to the net within the toolset.

As indicated previously in this section, one of the functions of the system-of-systems (the processing of a new network client node arrival in the exchange network parameters problem) was focused on. This describes the problem of a client node making a join request to a control node. The roles, control sequences, processes and information exchanged are presented in Fig. 6.1.

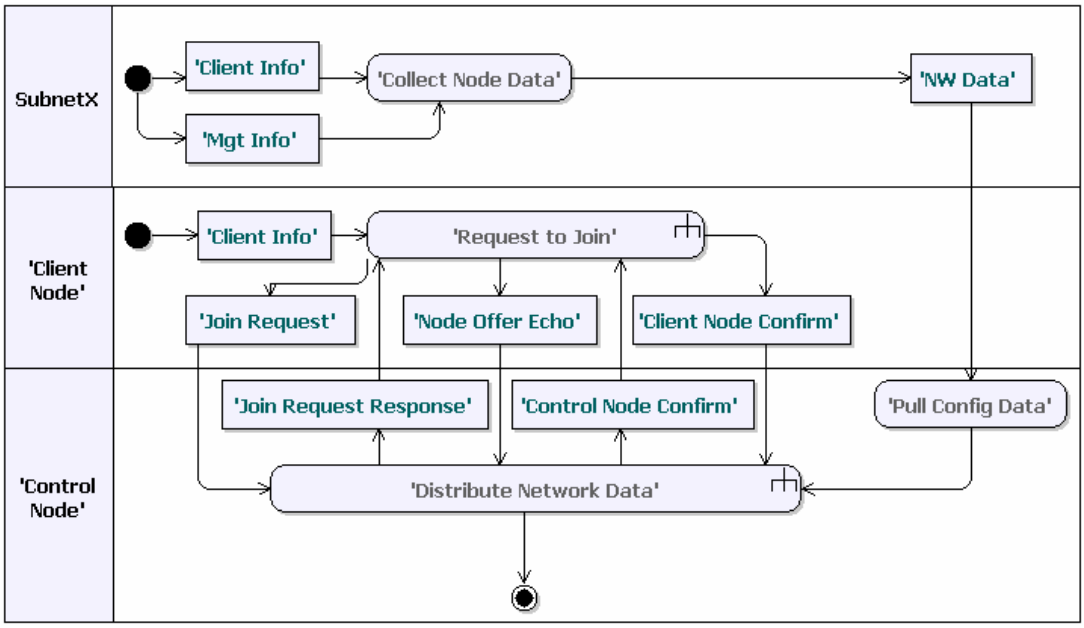


Fig. 6.1 'Exchange network parameters/dynamic host configuration protocol join request UML activity diagram'

From Fig. 6.1 it can be seen that two roles, 'client node' and 'control node', have operational processes assigned to them. These operational processes are presented as a sequential series of activities. The detail of groups of these activities can be abstracted at a higher level of abstraction under parent activities such as 'Request to Join' or 'Distribute Network Data' (shown using ⊞ notation in Fig. 6.1). Fig. 6.1 also shows that five pieces of information are exchanged between the roles, 'Join Request', 'Join Request Response', 'Node Offer Echo', *'Control Node Confirm' (Dynamic Host Configuration Protocol)*, and 'Client Node Confirm'.

A coloured Petri net model of the operational processes represented in Fig. 6.1 was attempted. Fig. 6.1's function (or service) level of granularity and naming was based on the existing standards [121, 122] and underlying communications infrastructure information sets (exchange network parameters transactions and the associated function performed by assets). This information was used to help manage the construction of the net by developing a hierarchy of functions to promote its readability and scalability. The function of 'Exchange Network Parameters (new network client node arrival)' was decomposed into two main sub-functions: 'Request to Join' and 'Distribute Network Data' (a third sub-function of 'Collect Node Data' was included to highlight persistent network configuration storage). Both of the main sub-functions are realised using the processes identified from existing standards and domain experts in Fig. 6.1. These processes were examined in turn in order to establish:

1. Executed activities together with their pre and post information states (i.e. input and output interfaces).
2. Control of activity execution.
3. Suggested roles (owners) associated with the control of activities.

4. Naming of activities, information and roles.
5. Net symbols that should be used on the net at the highest abstraction level (primary parent net) and associated sub-pages to accurately represent the hierarchy within the model.

The control flow of the two main sub-functions shown in Fig. 6.1 follows the exchange of information between Request and Distribute. The Collect sub-function is intended to occur periodically within and between subnetworks.

At the highest abstraction level in the hierarchy, the pre-condition of the request to join function executing is the arrival of a client node. Using [121, 122], suggested fields for new client node information are: unique node identifier; node type; network address; and optional node capabilities.

When a join request has been assembled, the post-condition of join request executing is outgoing information, containing the join request details. This information is sent to the role dealing with assignment of network parameters (control node). Its content can be derived from [122] and existing exchange network parameters message content [121]. Suggested fields for the join request information are: unique node identifier; network address; and optional node capabilities.

Considering the join request function's process in more detail, the client node information is used in the activity of assembling the join request. This assembly activity could be decomposed further into sub-activities relating to the nature of the assignment i.e. static or dynamic; inclusion of error recovery if an offer from a control node is not received, or the initial request was rejected. For the purposes of this case study, dynamic address allocation is considered and the result of the assembly activity is that dynamic network address assignment is requested by sending a join request.

At the highest level of abstraction in the hierarchy, according to [121, 122] and Fig. 6.1, the pre-condition of the assign function executing is incoming join request information from the client node, containing the join request information specified above.

The post-condition of assign executing is outgoing information, containing initial response details. This information is sent to the requesting role (client node). Its content can be derived from [121] and exchange network parameter message content. Suggested fields for denial response information are: unique node identifier; network address; and optional reason for denial. For initial acceptance response information, suggested fields are: unique node identifier; network address; and optional configuration parameters.

Considering the assign function's process in more detail, the join request information is used in the activity of making an initial offer (if there is a network address available). The check availability of network address activity could be decomposed further into sub-activities. For the purposes of this exchange network parameters case study, further activity decomposition is omitted. The result of the process is an initial accept or reject response.

As part of the request function at the highest level of abstraction, the pre-condition of the receive initial response function executing is incoming response information from the control node, containing the information specified above.

According to [121, 122], if an acceptance response is received there is one post-condition of receive initial response executing. This is outgoing offer echo information containing notification that the initial offer response was accepted by the client node. Suggested fields for offer echo information are: unique node identifier; network address; and optional configuration parameters.

As part of the assign function at the highest level of abstraction, the pre-condition of the receive offer echo function executing is the incoming offer echo information from the client node, containing the information specified above.

According to [122] there are two post-conditions of receive offer echo executing. One is that the control node has checked the proposed address is not in use and sends outgoing positive acknowledgement information containing confirmation of the offer. The other is negative acknowledgment information denying the offer. Suggested fields for positive acknowledgment response information are: unique node identifier; network address; and optional parameters. For negative acknowledgement response information, suggested fields are: unique node identifier; network address; and optional reason for negative acknowledgement.

As part of the request function at the highest level of abstraction, the pre-condition of the client node receive positive/negative acknowledgment function executing is the incoming positive/negative acknowledgment information from the control node, containing the information specified above.

According to [122] there are two post-conditions of receive positive/negative acknowledgment executing. One is that the client node has checked the proposed address is not in use and the process ends with the client node in a configured state. The other is the client node detects that the proposed address is in use and sends an offer decline to the control node. Suggested fields for this decline response are: unique node identifier; network address; and optional parameters.

As part of the assign function at the highest level of abstraction, the pre-condition of the control node receive decline function executing is the incoming decline information from the client node, containing the information specified above.

According to [121, 122] there is one post-condition of receive decline executing. The control node updates the network configuration database and the process ends.

All the above information (including the extra dynamic host configuration protocol configuration detail) was then manually mapped to Petri net constructs. As before, the net was constructed from a planner point-of-view. Conditions were allocated to places and colours (types) were defined to represent the fields of messages suggested above. Naming convention followed Fig. 6.1's as closely as possible. For the net relating to the 'exchange network parameters (new network client node arrival)' function, the two main sub-functions (request and distribute) were represented as abstracted transitions ('Request_to_Join' and 'Distribute_NW_Data') with more detail for each presented on

subnets ('Requester' and 'Assigner'). The collect sub-function is shown as 'Collect_Node_Data' (and detailed on subnet 'Collector'). Each abstracted transition has associated input and output sockets (interfaces). Together these form the highest level of abstraction in the hierarchy (parent net). The operational processes that realise each of these sub-functions are expressed on additional net pages (subnets) together with their associated input and output ports (interfaces). A successful or unsuccessful client node network configuration outcome from the assign function was selected at random within the model. The derived net's highest level of abstraction is shown in Fig. 6.2.

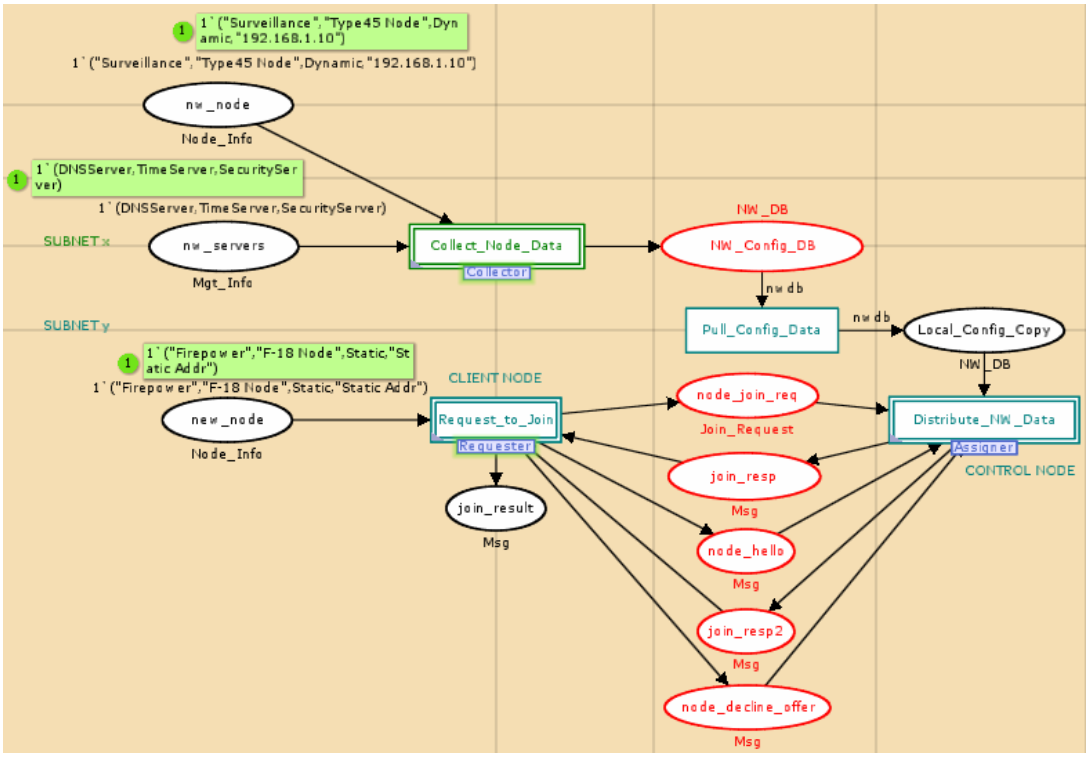


Fig. 6.2 'Exchange network parameters (new client node arrival) parent net'

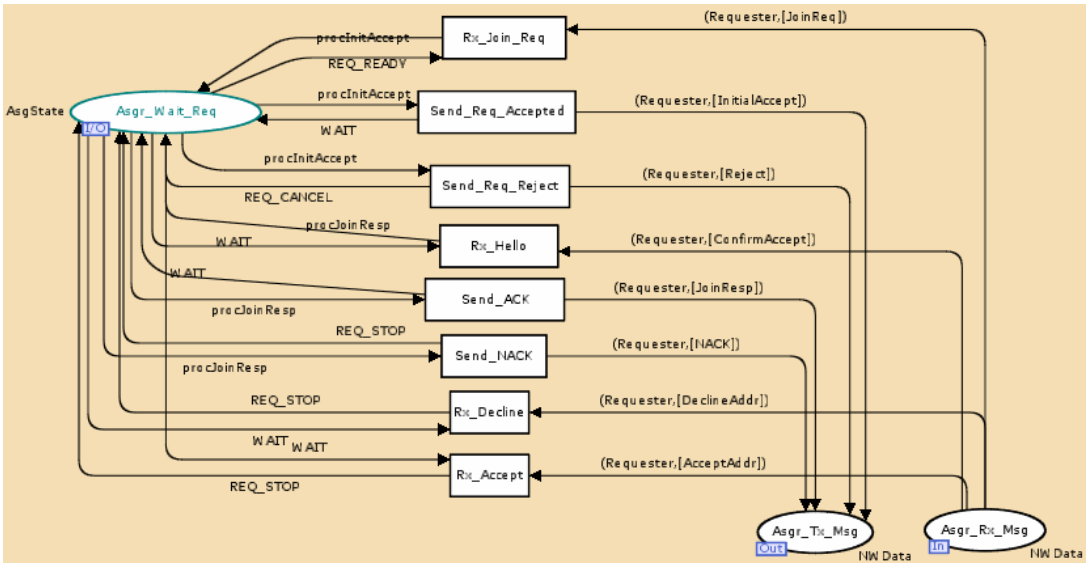


Fig. 6.3 'Net detailing state of the join request-response information exchange transaction (ASGR_Join_Req_Xchg) undertaken by the Assigner role'

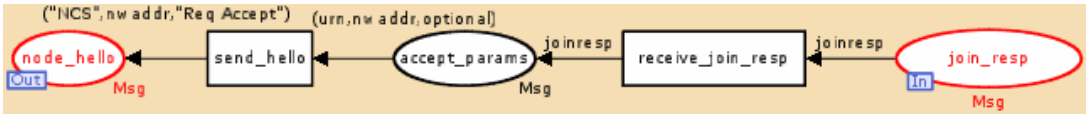
Nets were also used to specify the state of the information exchange protocol relating to the join request and join request assignment functions introduced in the operational process of Fig. 6.2. 'Requester' (Join_Request_Service) and 'REQR_Join_Req_Xchg' (Fig. 6.3) subnets taken together with corresponding 'Assigner' (Join_Assign_Service) and 'ASGR_Join_Req_Xchg' specify how each role could track its state in terms of what information is sent to and what information is expected from its partner role(s) on the network. If a communications failure occurs, suitable recovery can be designed for. It should be noted that with the addition of timing to the net, timeouts could be specified in the net of Fig. 6.2. Again, this specification information is in addition to the operational process and aims to be non-prescriptive in terms of how tracking is to be implemented.

In this section, coloured Petri nets and hierarchy have been used to specify the join request function of exchange network parameters at an analysis level of abstraction i.e. the operational process level. From Figs. 6.2-6.3, in terms of design readability, net places capture process state in terms of the information input and output to net transitions (activities); net colours (types) define the information used by activities; net arc inscriptions govern the information required and produced by activities; and net arcs dictate the control flow of execution i.e. the order of activity execution and information exchange. As long as the interpreter of the net has some experience of the syntax and semantics of Petri nets, and use is made of toolset features such as textual annotation and colouring, the specification represented by the net in terms of flow of execution is unambiguous.

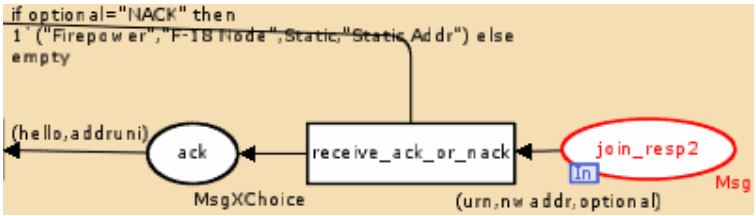
6.3 Analyses with Petri Nets and further Specification

6.3.1 Dynamic Analysis (Simulation)

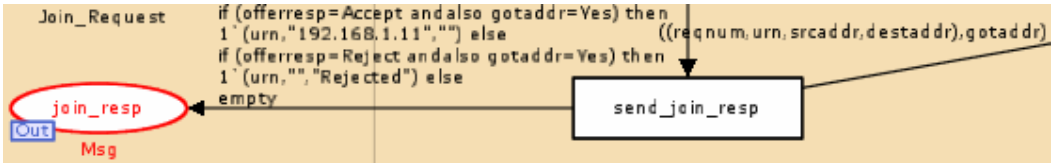
To investigate functional correctness, simulation of the derived net in Fig. 6.2 was used to provide confidence in the correctness of the behaviour and logic specified by it. Using the toolset it was possible to interactively step through enabled transitions in the net from a given initial marking until there were no more enabled transitions. In doing so, problems were revealed within the 'Assigner' and 'Requester' subnets detailing the activities undertaken by the control and client nodes. With the 'Requester' subnet, simulation revealed missing logic to deal with receipt of an initial reject response from the control node (Fig. 6.4, a), and missing logic to check for receipt of positive acknowledgment (ACK) on input arc to 'ack' place (Fig. 6.4, b). In the 'Assigner' subnet, simulation detected missing logic producing the accept or reject initial response message (Fig. 6.4, c), and incorrect logic to send positive or negative acknowledgement due to specification of incorrect parameter information in the arc inscription.



a) Requester subnet: missing logic for reject response



b) Requester subnet: missing logic to check for acknowledgment



c) Assigner subnet: missing logic for producing initial offer response

Fig. 6.4 'Sample of errors detected by interactive simulation'

Once these problems were corrected by amending the relevant arc inscriptions, basic interactive simulation indicated that the hierarchy of processes appeared to produce the desired behaviour for one join request. In addition, places were added to the 'Assigner' and 'Requester' subnets to reflect process end states of no network address availability, receipt of a join request rejection, and achievement of join request retry limit (following receipt of negative acknowledgments).

Specifying the join request-response process using net elements, using simulation to explore design iterations of the net, and correcting detected errors helped highlight incompleteness in the exchange network parameters standard. Implicit or omitted requirements are listed below (including Petri net specification of the first two requirements).

1. A reject response should be issued to the client node if there are no network addresses available.

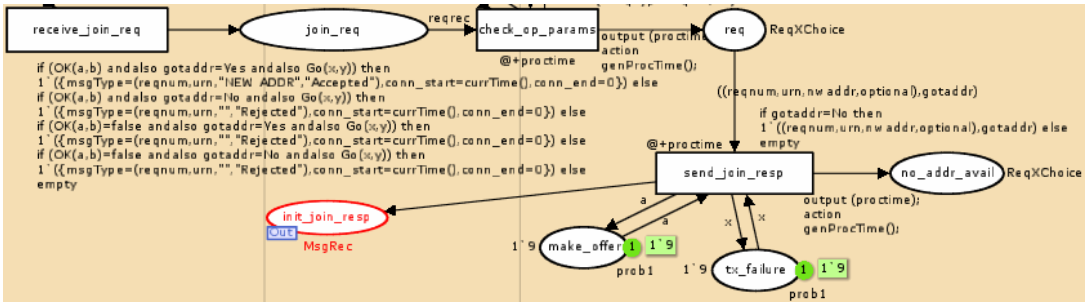


Fig. 6.5 'Specification of reject response'

2. Error recovery (timeouts and retry attempts) if no offer response is obtained from the control node or no offer confirmation (or denial) is received from the client node.

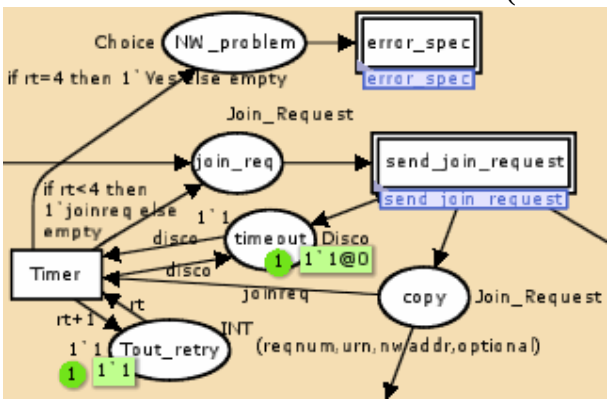


Fig. 6.6 'Specification of error recovery'

3. How the accept broadcast forces a duplicate address check and rejection by client node of accept offer response if a clash is detected.
4. Rejection of unauthorised control node/client node messages.
5. Specification of pre-requisite network configuration parameters, for example, subnetwork operational frequency.

6.3.2 Static Analysis (Reachability Graph Analysis)

The nets in Figs. 6.2-6.3 are the results of being able to execute the process-based net, identifying its shortcomings, and amending the constructs. Although simulation is very much part of an iterative net development process, interactive simulation of large nets can be extremely time-consuming and does not provide an exhaustive means of net verification. State space analysis (described in chapter 2) is used to complement simulation and provide this deeper level of verification.

CPN Tools' standard analysis of the state space (calculated for each of the nets created in Figs. 6.2-6.3 and described further for Fig. 6.2) is shown in Table 6.2.

State Space	Fairness Properties
Nodes: 480416	----------
Arcs: 542418	Assigner'addr_avail 1 Impartial
Secs: 1200	Assigner'addr_not_in_use 1
Status: Partial	Fair
	Assigner'receive_decline 1
Scc Graph	Fair
Nodes: 464919	Assigner'receive_join_req 1
Arcs: 495927	Fair
Secs: 52	Assigner'send_join_resp 1
	Impartial
	Assigner'send_join_resp2 1
	Fair
	Collector'process_data 1
	Fair
	Collector'send_node_info 1
	Fair
	Collector'send_server_info 1
	Fair
	Collector'send_subnet_data 1
	Fair
	Overview'Pull_Config_Data 1
	Fair
	RequesterTimer 1 Fair
	Requester'check_nw 1 Fair
	Requester'receive_ack_or_nack 1
	Fair
	Requester'receive_join_resp 1
	Fair
	Requester'send_decline_addr 1
	Fair
	Requester'send_hello 1 Fair
	Requester'send_join_request 1
	Fair
	Requester'static_or_dynamic 1
	Fair

Table 6.2 'Standard state space analysis report for Fig. 6.2'

The state space graph report of Table 6.2 calculated for an initial marking of one token for the net in Fig. 6.2 highlights there is a problem with the net for two reasons. First of all, in the given state space calculation limit of twelve hundred seconds, the state space explosion problem was encountered, resulting in the calculation of a partial reachability graph. Secondly, on inspection of the standard state space report produced by CPN Tools for the partial reachability graph, the 'Fairness Properties' section (providing information about how often individual transitions occur) reveals the presence of infinite occurrence sequences which had not been detected using simulation. The section shows two transitions, 'addr_avail' and 'send_join_resp' within the 'Assigner' subnet, as having the 'impartial' fairness property (these occur infinitely often in any infinite occurrence sequence) and the remaining transitions as fair (these occur infinitely often in all infinite occurrence sequences where they are infinitely often enabled). Initially, retry limits were introduced in the 'Requester' and 'Assigner'

subnets to remove the infinite occurrence sequences. State space analysis was conducted again on the revised net producing the standard report sample in Table 6.3.

State Space	
Nodes:	96835
Arcs:	120170
Secs:	600
Status:	Partial
Scc Graph	
Nodes:	96835
Arcs:	120170
Secs:	20

Best Integer Bounds	Upper	Lower
Assigner'Retry 1	1	1
Assigner'addr_in_use 1	1	0
Assigner'conf 1	1	0
Assigner'join_req 1	1	0
Assigner'req 1	99	0
Collector'nodeinfo 1	1	0
Collector'serverinfo 1	1	0
Collector'subnet_data 1	1	0
Overview'Local_Config_Copy 1	1	0
Overview'NW_Config_DB 1	1	0
Overview'join_resp 1	1	0
Overview'join_resp2 1	1	0
Overview'join_result 1	1	0
Overview'new_node 1	1	0
Overview'node_decline_offer 1	1	0
Overview'node_hello 1	1	0
Overview'node_join_req 1	1	0
Overview'nw_node 1	1	0
Overview'nw_servers 1	1	0
Requester'accept_params 1	1	0
Requester'ack 1	1	0
Requester'copy 1	1	0
Requester'dupe_addr 1	1	0
Requester'join_req 1	1	0
Requester'retry 1	1	1
Requester'timeout 1	1	0

Table 6.3 'Standard state space analysis report following addition of retry limits'

Again, in the given state space calculation limit of six hundred seconds, the state space explosion problem was encountered, resulting in the calculation of a partial reachability graph. This time the 'Fairness Properties' section confirmed infinite occurrence sequences had been removed. On inspection of the 'Boundedness Properties' section, the presence of multiple tokens on places (rather than the expected one token) indicated there was still a token generation problem leading to the accumulation of tokens on the 'req' place in the 'Assigner' subnet.

6.3.3 Specification and Verification of Re-entrant Error Recovery and Multiplicity

Use of simulation enabled the problem to be traced back to the timeout specified in the 'Requester' subnet. The specified timeout relied on a reset following successful receipt of an initial response message from the 'Assigner'. The re-entrant error-

recovery process specified in the 'Assigner' subnet did not provide a mechanism for resetting the 'Requester' timeout based on the failed request (i.e. no network addresses available). No timeout reset resulted in erroneous join requests being generated for the 'Assigner' subnet to process. The original error recovery process specified in the 'Assigner' subnet (Fig. 6.7) was designed to loop back to the 'join req' place when no network address was available and attempt network address allocation again. However, initiation of this loop back should have reset the timeout in the 'Requester' subnet associated with the first failed join request attempt.

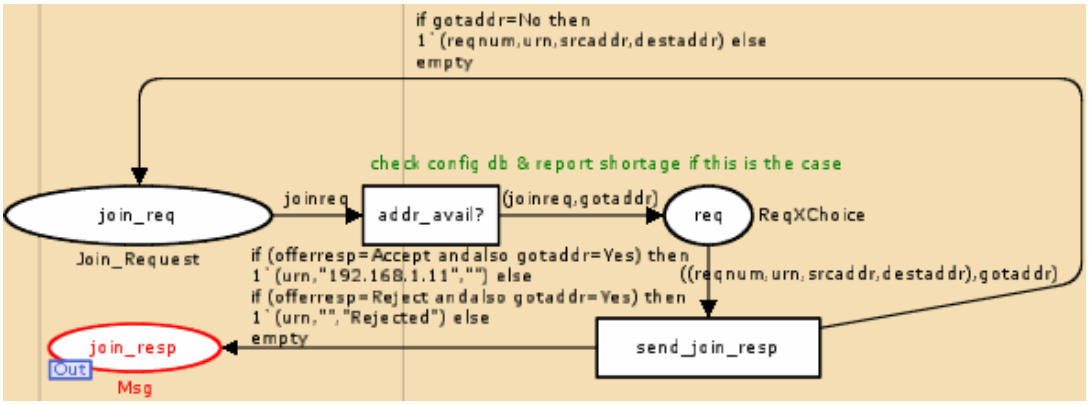


Fig. 6.7 'Error recovery specification in Assigner subnet'

Doing so was viewed as inconsistent with the modular structure of the exchange network parameters system-of-systems specification so an alternative specification method was considered. Instead, the loop was removed from the net and an arc inscription used in the 'Assigner' net to produce a reject request response from the client node (Fig. 6.8). Receipt of this reject response would reset the timeout in the 'Requester' subnet and give the option to the client node of attempting the join request again. In this way, modular self-consistency of the functions in the exchange network parameters system-of-systems was maintained.

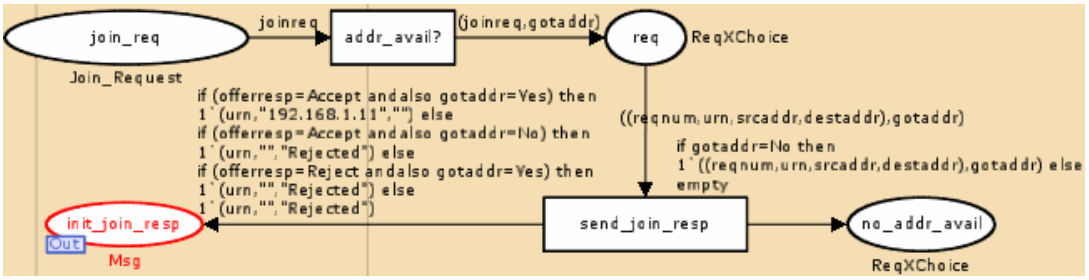


Fig. 6.8 'Amended Assigner net'

Once this timeout loop was removed from the net, its full reachability graph was re-calculated in less than one second with ninety-four nodes, one hundred and fourteen arcs and thirteen dead markings.

Multiplicity was then considered in more detail in this exchange network parameters study by specifying one requester process and one assigner process. Simulation was used to execute this model with one join request token to check its logic and

behaviour. This token represented one new request for network configuration (e.g. one client node) leading to execution of one instance of requester and one instance of assigner processes. Two tokens were then used to represent two requests for network configuration across the same one instance of requester and one instance of assigner processes. This model could represent one physical client node with two network interfaces making sequential configuration requests for each interface or it could represent one logical client node and two independent configuration requests (again sequential). This model highlighted the need for join request messages to have a unique node identifier and a transaction identifier. Token overtaking was not considered an issue in the case of one instance representing a physical host but would require management in the logical scenario when timing was implemented in the net and priority of requests to the control node was an issue.

Two instances of the client node process and one instance of the control node process were then modelled. Simulation was used to execute this model with one join request token for each client node process. These tokens represented two new requests for network configuration (e.g. two client nodes) leading to concurrent execution of both instances of requester and one instance of assigner processes. This model could represent two physical client nodes making concurrent configuration requests to one physical control node or a logical scenario as described in the first paragraph on this page. Execution of this multiplicity model highlighted the need for a unique node identifier in both the message and assigned to the process instance (so that messages could be routed back to the correct originator from the control node), and the possibility of token overtaking and need to manage this.

Two instances of the control node process and one instance of the client node process were modelled next (even though the exchange network parameters standard implies and subject matter experts confirm there should only ever be one active control node per subnetwork). Simulation was used to execute this model with one join request token for the client node process. This model could represent two physical control nodes processing concurrent configuration requests from one physical client node or a logical scenario as described in the first paragraph on this page. To capture the dynamic host configuration protocol specification of the request reaching both control node process instances (rather than being directed to a specific control node), a copy of the request is produced for the interface place of each control node (these need to be separate places rather than one shared one in order to specify that each control node should process a copy of the request rather than having a non-deterministic situation where all requests could be consumed by the same control node). Responses are passed back to the client node from both control nodes, the client node selects one and directs its response back using the control node's unique node identifier. This model highlighted the need for join request messages and control node process instances to have an assigned unique node identifier and message transaction identifier; a wait process in the client node to accumulate, select, and discard offers per request transaction; group requests by transaction identifier in order to configure the timeout mechanisms; and to keep multiple instances in models to a minimum for complexity purposes.

Finally, two instances of the control node process and two instances of the client node process were modelled next (even though the exchange network parameters standard implies and subject matter experts confirm there should only ever be one active

control node per subnetwork). Simulation was used to execute this model with one join request token for the client node process. This model could represent two physical control nodes processing concurrent configuration requests from two physical client nodes or a logical scenario as described in the first paragraph on page 114. To capture the dynamic host configuration protocol specification of the request reaching both control node process instances, a copy is produced for the interface place of each control node (these need to be separate places rather than one shared one in order to specify that each control node should process a copy of the request rather than having a non-deterministic situation where all requests could be consumed by the same control node). Responses are passed back to the relevant client node from both control nodes, the client node selects one response and directs its confirmation back using the control node's unique node identifier. This model highlighted the need for join request messages and control and client node process instances to have an assigned unique node identifier; a wait process in the client node to accumulate, select, and discard offers per request transaction; group requests by transaction identifier in order to configure the timeout mechanisms; and to keep multiple instances in models to a minimum for complexity purposes.

In all multiplicity cases, re-entrancy was specified per function rather than between (interfaced) functions. Use of multiplicity highlighted a further requirements clarification to [121] in terms of exchange network parameters information exchange (listed on page 110).

6. Transaction and unique node identifiers present in join request messages and the assignment of a unique node identifier to each process instance.

During the analysis of multiplicity specification using nets, net scalability issues were demonstrated in the second study on exchange network parameters by gradually increasing the number of instances of the 'Requester' subnet. It was found that toolset performance was severely degraded when a total of sixty-seven subnets (incorporating approximately three hundred and forty places) was reached (representing sixteen client nodes). Both net readability and navigation were extremely difficult and time-consuming and net scalability compromised.

In this section, the second study on exchange network parameters has shown that two forms of analyses for coloured Petri nets can verify functional correctness of the model they represent. These were execution of the net (simulation) and analysis of the net's reachability graph (static analysis or model-checking). Following these analyses, models can be amended and enhanced, improving design quality. Simulation and static analysis applied to the models also helped to identify incompleteness in the [121] standard (identifying six requirements clarifications) and explicitly specify the information exchange protocol within the exchange network parameters join request and response problem using multiplicity. Scalability issues with nets were highlighted during examination of multiplicity specification and re-entrancy was found to be an issue when error recovery specification involved updating state dependencies between interfaced modules.

6.4 Addition of Timing to Petri Net Model

As demonstrated by the telephone (Appendix C) and close air support (section 5.4) exercises, time-dependent actions such as timeouts, processing delays or deadlines are essential to capture the efficiency or performance of a system and facilitate validation of its design. As well as efficiency specification, time-dependent actions also enhance a system's behaviour specification in terms of correctness and completeness. Activity ordering alone is insufficient to capture overall system behaviour precisely. Tokens representing information in larger-scale systems will be processed according to the time they entered the system, time involved in their consumption and generation, and involvement in delays and transfer failures. Timing will be needed to specify the ordering multiple tokens receive over and above any activity sequence they experience.

Currently, the exchange network parameters problem has been specified at an operational process (analysis) model level of abstraction and used as the first stage in large-scale, system-of-systems development. Typically, this viewpoint is useful for gaining a shared understanding of the problem concept and the intended technical and non-technical audience would include analysts, developers and domain users. The introduction of timing information to the problem at this abstraction level would help enable domain users and developers to decide whether the modelled concept was efficient and adequate for input into the design stage. Assessing performance would involve checking if the modelled processes reached desirable behaviour states (including recovery from undesirable states) within realistic time and resource estimates. Improving the efficiency of the process means looking for new or different ways to realise desirable behaviour within defined time, cost and quality parameters.

To examine alternative options for the process, it was necessary to determine the time, cost and quality performance indicators for the exchange network parameters (new network client node arrival) process and implement these in the model. Examples of these indicators include join request fulfilment time, communications resource usage (and related costs), and successful join request fulfilment time within a certain time limit. The natural inclination would be to minimise the first two and maximise the last one but all three need to be taken in context with the strategy of the system-of-systems involved. In the case of exchange network parameters dynamic network configuration, it is essential to understand the economic and operating environment for which it is used, and which (if any), of the performance indicators carries more weight than the others.

In the exchange network parameters example, as the system-of-systems was specified originally from a planner's point-of-view, it is assumed that they are concerned with striking a trade-off between quality and cost parameters for dynamic network configuration. Depending on the criticality of the mission to be supported by dynamic network configuration, this may mean the planner would be interested in maximising successful join request fulfilment and communications resource allocation without necessarily maximising join request fulfilments on first attempt for new network participants (requesters). If these are the desired parameter outcomes, a planner may use the design models to identify platforms with existing communications equipment able to satisfy these outcomes in the supported mission's operational environment.

Dynamic analysis (simulation) is used in conjunction with timing in the net. Timing delays were introduced at various intermediate places within requester and assigner processes using both stochastic and deterministic distributions to represent random request placement and delays between each activity in the overall exchange network parameters (new network client node arrival) process. Transmission duration between role processes was captured for the send and receipt of a join request. A record declaration was used for each join request in order to store the model time at which the 'send_join_request' activity executes. This was viewed as the start of the attempt by the underlying communications infrastructure to connect the requester with the assigner. Again, a time delay was introduced here to the record token to represent the delay of the underlying communications infrastructure. The toolset data collection functionality was used to compare the model time following execution of the assigner's 'receive_join_req' transition (Fig. 6.9) with the start time of the transmission of the request.

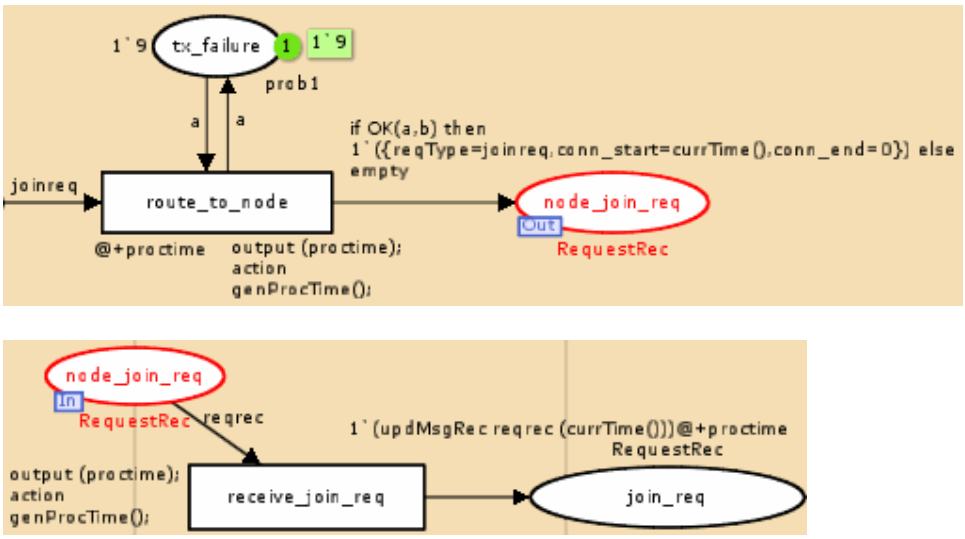


Fig. 6.9 'Requester and assigner performance analysis subnets and transmission duration'

Average transmission duration information can be used to setup a realistic timeout delay (and specify recovery, in the case of a join request resend, together with resend attempt limit) in the model to be enabled once the join request is sent. The transmission duration thresholds will vary depending on the underlying communications infrastructure used to carry the network join request. On the internet, end nodes normally use an ethernet-based backbone [123] as a fixed or wireless communication mechanism, [121] focuses on combat net radio as the communications mechanism between military platform nodes. Timing could be advised from their real-world implementations. If a resend limit is reached following successive timeouts between requester and assigner nodes, possible reasons for such a failure could be specified (as requirements for a successful implementation of this system-of-systems). For example, participant nodes' communications software implementations would have to support the protocols necessary for dynamic configuration; relay and routing mechanisms would be necessary to accommodate physical network communication barriers such as line-of-sight or subnetworks; and a degree of redundancy in control nodes distributing network configuration parameters would be expected.

In order to capture the duration of a successful join request fulfilment, toolset data collection functionality was used to observe the 'client_configured' transition in the 'Assigner' subnet (Fig. 6.10). When the transition fired with bindings of an accepted request, the current model time was captured. This model time represents the duration for successful client node configuration. Factors influencing this duration would be message transmission times, process activity selection and ordering, and process activity duration times (some of these would be influenced by the number of resources undertaking an activity, availability of network addresses, accuracy of network address allocation and supplied configuration data, and join request acceptance criteria).

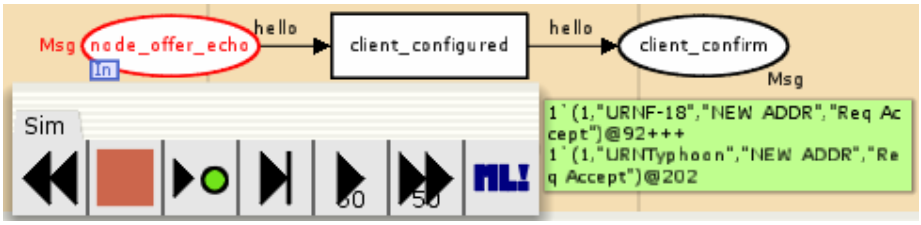


Fig. 6.10 'Capture of model time for client_configured transition'

Analysis-of-alternatives, as conducted for the first case study on close air support (section 5.4), could be used to allocate known existing physical assets to activities; obtain the associated costs per activity, per process, and overall; and apply this knowledge to the process-based model. For example, allocating combat net radio as the underlying enabling communications mechanism may be at a cost of x amount to the military. Using the process modelled with Petri nets in Fig. 6.11 (now tailored for exchange network parameters), obtaining the average message transmission time associated with line-of-sight combat net radio, the average duration for each activity, the probability of transmission failure in the join request process (currently specified as 10%), and the probability of join requests made that are accepted (currently specified as 90%), the average successful join request duration can be calculated and linked to resource cost. Fig. 6.12 illustrates the use of automatic simulation replications on the net of Fig. 6.11 to calculate an average successful join request time based on the probabilities given above and the message transmission time results using the discrete distribution. With multiplicity, simulation can be used to estimate effect on average successful join request times with multiple physical client node processes executing and interacting with one control node process.

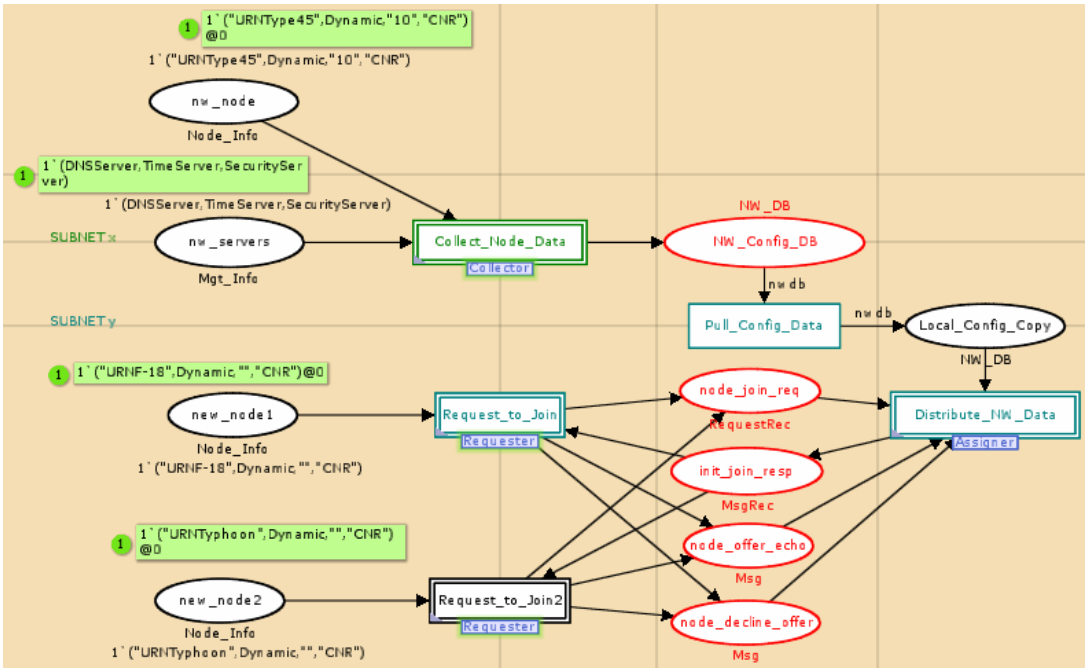


Fig. 6.11 'Exchange network parameters performance analysis parent net'

```
fun simulateConfigs(n:int) =
  let
    fun setCNR() = (avg_proc_time:=30)
    fun setEN() = (avg_proc_time:=15)
  in
    procdist := exp;
    setCNR();
    CPN'Replications.nreplications n;
    setEN();
    CPN'Replications.nreplications n;

    procdist := disc;
    setCNR();
    CPN'Replications.nreplications n;
    setEN();
    CPN'Replications.nreplications n
end
```

Discrete Distribution (Data Collection Monitor: Req_Success_Monitor)

REP 1 (30 time units)

1	229	229.000000	229	229
2	461	230.500000	206	255
1	219	219.000000	219	219

6 possible successes, 3 simulation replications:
4 successes, avg. 227.25 time units.

REP 2 (15 time units)

0	0	0.000000	0	0
2	259	129.500000	129	130
1	102	102.000000	102	102

6 possible successes, 3 simulation replications:
3 successes, avg. 120.33 time units.

Fig. 6.12 'Automatic simulation replication used to calculate average duration for a successful join request'

Considering the specification of exchange network parameters as supplied by [121], the overall process is shorter than that specified by the dynamic host configuration protocol. This is due to there being one control node providing a response to the initial join request rather than multiple offers being provided by more than one control node (and subsequent selection of a control node from which to accept an offer). Checking

for in-use addresses prior to issue and acceptance of an address offer is undertaken by control and client nodes in the dynamic host configuration protocol specification but only by the client node in exchange network parameters (via an accept response). This means the onus is with the control node to maintain as accurate a record of client nodes and their associated network addresses as possible.

In exchange network parameters, when a client node obtains an accept response and network parameters including a network address, the response is directed to all subnetwork nodes. Based on simulation of the model, it was detected that it is possible for a control node to offer an address to a client node and fail to receive confirmation of its successful configuration in time. If the address is then marked as available and offered to another incoming client node, duplicate addresses will exist on the subnet. This situation can be avoided by explicitly stating in the standard that the timeout value has to be of suitable duration to allow a client node to confirm its successful configuration, or not to re-use offered addresses, and that a client node should ensure that it ceases to use its allocated address upon receipt of a 'reject from subnetwork' message from the control node. Use of timing highlighted a further clarification to [121] in terms of exchange network parameters logic and behaviour specification (listed on pages 110 and 115).

7. Adequacy of timeout to allow client node to confirm successful configuration, and/or control node refraining from re-using offered network addresses; and client node must stop using network address upon receipt of a subnetwork reject message from control node.

It should be noted that although simulation was primarily used in this section to validate the models, it is also possible to conduct static analysis (as per recommendations from the telephone example in Appendix C) to verify their correctness.

In this section, timing in coloured Petri nets has been used to enhance correctness, specification completeness (identification of additional requirement clarification to [121]) and conduct performance analysis. Use of timed colours (types), and suitable inscriptions on output arcs (in conjunction with stochastic or deterministic functions) help to specify duration of activities (and execution control flow) such as information exchange, and timeout error recovery in the event of a communications failure.

6.5 Design and Architectural Levels of Abstraction for Exchange Network Parameters

Sections 6.2-6.4 have focused on using Petri nets to specify the exchange network parameters process at an analysis level of abstraction. Hierarchy and timing have been added to further enhance a specification in terms of scalability, understandability, readability, correctness and completeness. Both model-checking and simulation were employed iteratively in verification and validation of the constructed analysis level net. Before deciding if the criteria for success in relation to the exchange network parameters study have been met and conclude chapter 6, Petri nets are checked to see if they can address the problem of specifying exchange network parameters at design and architecture levels of abstraction. This is the objective of this section.

6.5.1 The Design Level

The purpose of the design level of abstraction is the lead into the specification of a solution to the problem described by the analysis level. Again, a functional decomposition approach was used. This time it was used in conjunction with the parent net developed for the analysis level to think about how this net's main activities (e.g. 'Request_to_Join' and 'Distribute_NW_Data' would eventually be realised by physical implementations. To keep the design flexible, two components, 'Make_Join_Request Component' and 'Assign_NW_Data Component', were used to depict the solutions that would realise each of the main activities. These are shown in Fig. 6.13.

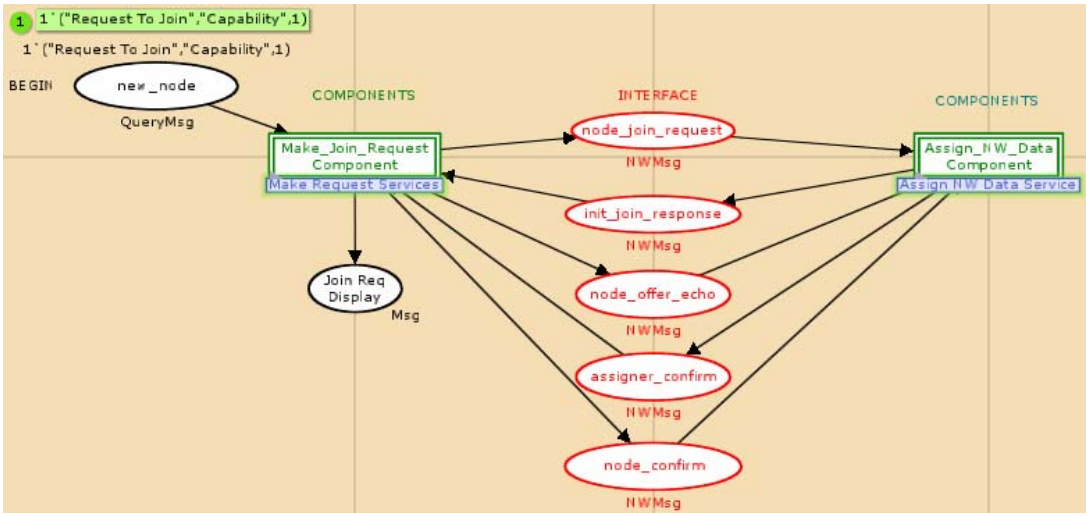


Fig. 6.13 'Design level parent net'

It can be seen that Fig. 6.13 closely resembles the parent net of the analysis level except for the new place colours (types). The next level of design decomposition for the two components aimed to capture the functional service(s) each would be expected to realise. Again, work developing the analysis level net helped suggest functional services for the design level by thinking about the purpose of the processes used to realise the main activities. 'Make_Join_Request Component' would be responsible for providing join request setup, initial join request response, client node confirm offer setup, *control node confirm response*, and client node final confirm setup services. 'Assign_NW_Data Component' would be responsible for providing join request response, initial join request setup, client node confirm offer response, *control node confirm setup*, and *client node final confirm response* services. These services are shown at the next lower abstraction level providing greater detail in Fig. 6.14.

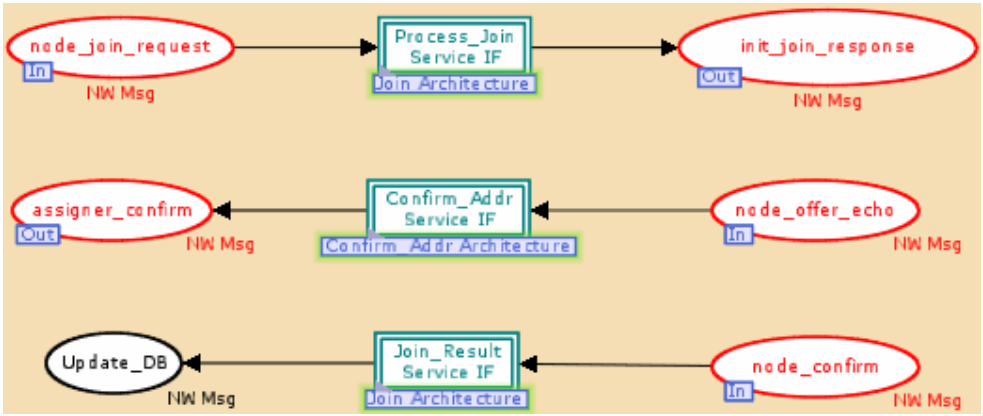


Fig. 6.14 'Services of Assign_NW_Data_Component'

Having introduced the components and functional services at the design level, the next lower abstraction level providing greater detail, i.e. detailed design or architecture was focused on. Rather than develop a separate model at this stage, as the architecture level appeared to naturally manifest the next lower abstraction level of the design level, the design level model was further decomposed to capture the architecture level.

6.5.2 The Architecture Level

The purpose of the architecture level is detailed design of the services identified at the design level and flexible capture of the components required to realise these individual services. Constituent components were considered for each functional service resulting in the identification of a common component pattern for the six services associated with client node join request-response. The common components consisted of an asset communications interface, transmit and receive (network) interfaces, and a communications controller interface to co-ordinate the sequencing of activities to and from the other two common components. The common component architecture is shown for the 'Process_Join Service' in Fig. 6.15.

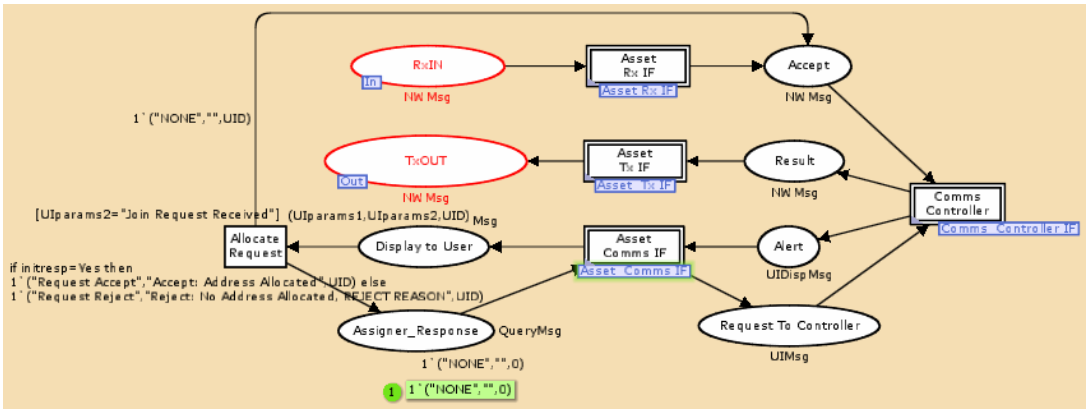


Fig. 6.15 'Join architecture'

From Fig. 6.15, it can be seen that net places are used capture the input and output information for the user interface, network and controller common components. Colour (type) definitions were lifted for re-use from the telephone (Appendix D) and close air support (section 5.5) example nets and adapted accordingly (definition labels reflect the nature of the interface, e.g. 'NWMsg' aims to reflect that places associated with this type are both input and output interfaces to asset network communications interface components). The intention with this labelling convention was improved net clarity and comprehension. Place types were based on character strings rather than enumerated types for flexibility reasons. The tuples in the type were populated with the functions implemented by each common control component and the associated parameters via logic on transition output arcs. Logic on transition output arcs within each of the common components was amended as necessary. As an example, consider the network transmit common component in Fig. 6.16.

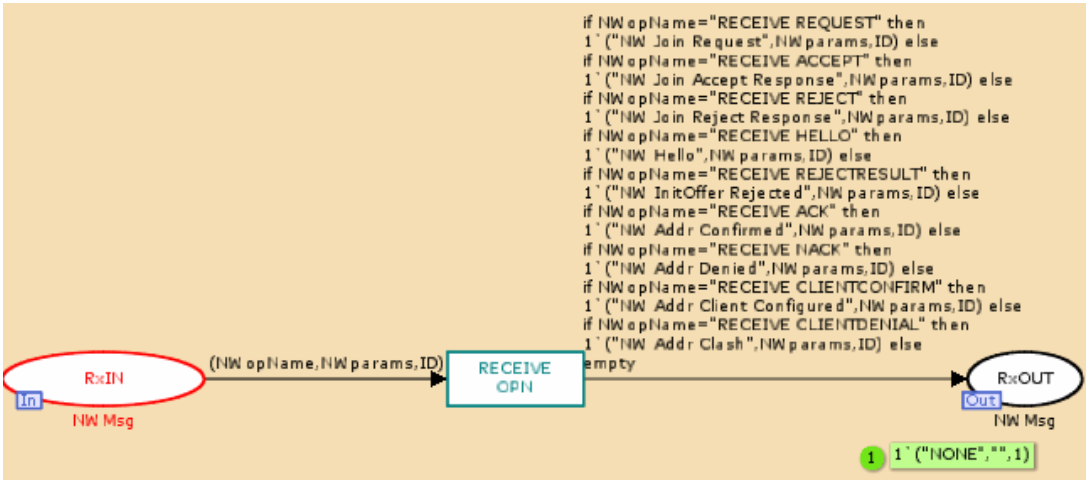


Fig. 6.16 'Receive common component subnet'

Fig. 6.16 shows the subnet of the receive common component. Its transition is labelled as 'RECEIVE OPN' to reflect the function the component provides to the controller component. On the transition output arc (within the transmit component subnet) to the output interface place ('TxOUT'), there is logic to output a token with 'NWopName' (a tuple within 'NWMsg' compound type) populated with required functions such as 'RECEIVE REQUEST' or 'RECEIVE ACCEPT'. In this way, the net specifies use of the receive component's 'RECEIVE OPN' function by the transmit component more explicitly. The 'NWparams' tuple within 'NWMsg' is populated with values relevant to the function of the message described by 'NWopName'. 'ID' is populated to differentiate between initiated join requests. The other two common components, communications interface and controller, are designed to reflect the same interface principles as those discussed above for the network components.

Considering the original parent net of the design level in Fig. 6.13, the specification of the exchange network parameters process at this level was across fewer model subnets. When the architecture level of Fig. 6.15 was reached and the next lower abstraction level providing greater detail of the common component interfaces was completed, the modeller was extremely conscious of the requirement to manage the levels of abstraction. The toolset can present each level of abstraction as a separate

page within a folder (or binder). These pages can be selected between using their tabs. By the common component interface level of abstraction for the request and assignment services, thirty-three pages and tabs were present and it was tedious work identifying and selecting relevant pages. At this stage, experience gained with the telephone (Appendices A-D) and close air support (section 5.5) examples was used to rationalise the model where possible, making use of the toolset's features and those of hierarchical coloured Petri nets. The main source of rationalisation was the common component interface nets.

6.5.3 Verification of the Design and Architecture Levels and further Specification

At this stage, simulation was employed to check the structure and logic of the model and was able to detect incorrect logic on transition output arcs. Errors included: missing or incorrect predicates (highlighted by incorrect or missing display notifications for the common communication interface component or incorrect information messages for the network component); and missing initial values on input places required by common component interfaces.

The necessary corrections were made and static analysis based on one initiated request performed. One error was highlighted through model-checking and production of a full state space calculation. The original standard report from the toolset produced four dead markings, one of which indicated that a reject result was sent from the 'Make_Join_Request Component' only when the component had received a rejected request response from the 'Assign_NW_Data Component'. Upon investigation it was discovered that logic was missing in the 'Comms Controller IF' component to produce this message randomly based on receipt of an accepted request response. This logic relates to the fourth omitted requirement (list on page 110) in [121] where the client node should be able to reject a response from a control node if there is reason to believe it is unauthorised.

In this section, coloured Petri nets and hierarchy were used to specify the new network client node arrival sub-function of exchange network parameters at design and architecture levels of abstraction, i.e. the solution specification level. From Figs. 6.13-6.16 above, net places capture state in terms of the information input and output to net transitions (activities); net colours (types) define the structure of the information used by activities (operations and parameters); net arc inscriptions govern the information required and produced by activities (including operations needed between components); net arcs dictate the control flow of execution i.e. the order of activity execution and information exchange; toolset hierarchy facilitates levels of abstraction within the model (at the design level component and services, at the architecture level common components realising design level services) and offers instantiation for re-use of existing subnets; toolset colour palette and annotation improves readability of nets; and finally, dynamic and static analyses permit verification and validation of the models.

In terms of scalability, a model of one exchange network parameters sub-function with thirty-three subnets and approximately eighty places (and being kept as generic as possible employing instantiation) permitted calculation of a full state space graph.

6.6 Evaluation of Exchange Network Parameters Study

The design objective of the exchange network parameters study was derived from chapter 1's second criteria for success. This stated that the strengths and weaknesses of Petri nets regarding the greater formalism of dynamic behaviour in systems-of-systems and the role of Petri nets as a means of engaging stakeholders were to be determined. Following the case study design again from chapter 3, nets have been used to specify, verify and validate the military exchange network parameters problem. As specified by the study plan, data (evidence) was collected at design iterations of each model using screenshots of the model and simulations, standard reports from CPN Tools, model source code from CPN Tools, and project team notes. This evidence was presented in the report of the study in this chapter and discussed further in this section.

6.6.1 Quantitative Results

These related to the third criteria for success and the first research question, i.e. do Petri nets improve the functional correctness of the system-of-systems design specification? The functional correctness response variables were expressed in terms of the number of errors detected by simulation, and number of errors detected by static analysis. In addition, for this study, functional correctness was also conveyed by the number of requirements clarifications highlighted for the standard [121]. This data was captured from the CPN Tools integrated development environment at each iteration of model design through simulation (and screenshots, note-taking) or reachability graph calculation (and CPN Tools standard analysis report, screenshots, note-taking). An overview of the results is shown in Table 6.4.

Criteria for Success Goal 3 / System-of-System Engineering Level	**Simulation Number of Errors Detected**	**Static Analysis Number of Errors Detected**	**Resulting number of clarifications to the standard**
Analysis	5	2	7
Design and Architectural	2	1	0

Table 6.4 'Quantitative results from exchange network parameters study'

From Table 6.4 it can be seen that use of simulation detected seven functional errors during the specification of exchange network parameters. With textual or UML-based specification, given the nature of the errors detected through simulation of the Petri net model, it would have been extremely difficult to detect the same errors using both these means of static specification.

Use of static analysis detected three functional errors in the specification of exchange network parameters. As well as the seven errors detected using simulation, static analysis was able to detect infinite looping and token generation within a model and localise the part of the net where the problem was. From examination of the errors detected by static analysis at the system-of-systems engineering analysis level, it can be seen that if the same errors were not detected using simulation, they would have been detected through calculation of the reachability graph for the net. In this way, static analysis offers a means of exhaustively checking a net on behalf of the modeller but the modeller needs to know how to interpret the analysis report in order to isolate

the detected errors and this is a time-intensive process. The modeller also needs to be aware of the state space explosion problem and means of largeness avoidance (previously discussed in chapter 5 and Appendices B-D).

Given CPN Tools in its standard form, simulation (or the Petri net 'token game') is a more intuitive means of stepping through the behaviour of a model (depending on its size) to check its functional correctness. It was used as an initial means of model verification prior to conducting static analysis so that detected errors could be corrected, potentially helping to alleviate the state space explosion problem, and simplify the analysis report.

Both forms of analyses also contributed to the identification of seven enhancements (listed on pages 110, 115, and 120) to the existing exchange network parameters standard [121].

6.6.2 Qualitative Results

These related to the first and second criteria for success and the second and third research questions, i.e. do Petri nets increase the quality of the design specification, and what are the shortcomings of the state-of-the-art Petri net tool and how can it be improved? The design quality response variables were expressed in terms of comprehensibility (e.g. use of hierarchy, annotation, timing), and scalability. This data was captured from the CPN Tools integrated development environment at each iteration of model design through screenshots, and note-taking. An overview of the results is shown in Table 6.5.

Criteria for Success Goal 1&2 / System-of-System Engineering Level	**Comprehensibility Hierarchy, annotation, timing**	**Scalability**
Analysis	Able to express problem (operational process, including timing specification) using CPN Tools. Visually, net difficult to read for non-practitioner (need for best practice in terms of layout).	Yes (multiplicity specification used to demonstrate toolset scalability issues for a net of sixty-seven subnets, approximately three hundred and forty places i.e. representation of sixteen client nodes).
Design and Architectural	Able to express solution design (including timing specification) using CPN Tools. Again, net difficult to read for non-practitioner.	Yes.

Table 6.5 'Qualitative results from exchange network parameters study'

These qualitative (and quantitative) results in relation to the criteria for success defined at the beginning of this section are now evaluated further. The first and third criteria with metrics relating to the gaps in Petri net specification knowledge (identified from the first study) are focused on.

1. Abstraction

In terms of UML 2.0's activity diagram, activity decomposition, cardinality, and allocation can be specified.

In terms of the Petri net formalism, activity decomposition can be specified using high-level Petri nets with hierarchy. Implementations of Petri nets with hierarchy include substitution transitions, and place and transition fusion.

Results from the second study (exchange network parameters specified using CPN Tools) indicate that hierarchy needs to be determined in advance of modelling. In the study, exchange network parameter and dynamic host configuration protocol messages were used in a bottom-up approach to derive hierarchy from the suggested message functions. Hierarchy was defined for models at the system-of-systems engineering level (analysis, and design and architectural) as well as within models (a hierarchy based on function was used).

CPN Tools implementation of coloured Petri nets with hierarchy uses substitution transitions and fusion places. Substitution transitions were used primarily for abstraction as the sockets and ports used by this method were viewed as a means of explicitly specifying required and provided interfaces to the decomposed transition.

The study confirmed there was no explicit composition, cardinality, or allocation concrete notation with nets. The present means of achieving these in CPN Tools would be annotation and/or extra net elements. Extra net elements are needed to capture multiple allocations.

2. Modularisation
In terms of UML 2.0's activity diagram, modularisation can be specified using class and activity diagrams. Class and collaboration diagrams can also be used to specify provided and required interfaces between classes.

In terms of the Petri net formalism, modularisation can be specified using high-level Petri nets with hierarchy. Implementations include substitution transitions, and place and transition fusion.

Results from the second study (exchange network parameters specified using CPN Tools) indicated modularisation was primarily achieved using substitution transitions (and port and socket places) to represent component system interfaces.

Information exchange protocol was described between and within component systems using colours (types) to define the exchanged information and net elements (places, transitions, arcs, arc inscriptions, annotation) to specify the control of the exchange.

CPN Tools makes no provision for model re-use (e.g. searching for suitable existing models to use in a bottom-up approach) and their management (e.g. versioning).

Data items input (output) by activities can be specified through colours (types) and variable bindings. Cardinality needs to be specified through annotation.

3. Data typing

In terms of UML 2.0's activity diagram, classes defined previously in class diagrams specify data typing.

In terms of the Petri net formalism, timed, high-level Petri nets specify data typing.

Results from the second study (exchange network parameters specified using CPN Tools) indicated that CPN Tools implementation of timed, coloured Petri nets enabled capture of the information needed for the case study using a combination of simple, compound, and timed colours (types). There was no ability to refer to variable values unless the same variable bindings were propagated through the entire net.

There is no equivalent one diagram, static description in nets. Operations (and the required parameters) provided by system components were made more explicit using arc inscriptions and annotation, and their associated data types were provided by local place colours (types).

4. Adequate toolset implementation
In terms of UML 2.0's activity diagram, many open source and commercial toolsets offering various levels of integrated UML development exist.

In terms of the Petri net formalism, several implementations of high-level Petri nets offering various levels of integrated net development exist.

Results from the second study (exchange network parameters specified using CPN Tools) indicated that CPN Tools provided a useful integrated net environment for the specification of systems-of-systems but shortfalls were identified in the areas of: improved analysis reports, examples and best practice, and large model support (navigability, syntax-checking, versioning, error-reporting, and animation).

5. Timing
In terms of UML 2.0's activity diagram, extension via the Profile for Schedulability, Performance, and Time (to be replaced by Profile for Modelling and Analysis of Real-time and Embedded Systems) is needed in order to specify (non-functional) timing properties. The resulting activity diagram is static so performance analysis, or investigation into analysis-of-alternatives is only achievable through model conversion.

In terms of the Petri net formalism, timed, high-level Petri nets with hierarchy specify timing information.

Results from the second study (exchange network parameters specified using CPN Tools) were based on CPN Tools implementation of timed, coloured Petri nets with hierarchy. Stochastic and deterministic functions within the toolset were used to introduce random placement of requests, delays, and timeouts. Time, cost and quality performance indicators need to be identified in advance of the analysis.

Simulation was used in conjunction with toolset monitors in the performance analysis net to show that it is possible to derive average successful join request fulfilment duration using automatic simulation. Timing was primarily used to reflect

deterministic and stochastic activity durations for the purposes of performance analysis.

6. Verification and validation
In terms of UML 2.0's activity diagram, there is no full formal syntax and semantics and diagrams cannot be executed (static inspection verification and validation).

In terms of the Petri net formalism, there is formal syntax and semantics and nets can be executed using a well-defined execution algorithm (simulation). Additionally, exhaustive verification can be achieved by calculating the reachability graph of the net and checking structural properties such as deadlock. Timed, coloured Petri nets and simulation were used to conduct performance analysis.

Results from the second study (exchange network parameters specified using CPN Tools) were based on CPN Tools simulation modes for initial investigations into analysis, and design and architecture model erroneous behaviour (seven errors were detected: missing reset logic; incorrect predicates; and missing initial values), performance analysis (the calculation of average successful join request fulfilment time was demonstrated using automatic simulation replications), and incomplete specification (for the examined subset of exchange network parameters, seven omitted or implied requirements from the standard were highlighted).

For large system-of-systems models, interactive simulation was time-intensive. This was managed by building nets incrementally and simulating the new net elements. Simulation can be used to check the behaviour of an entire model and analyse performance where timing is used but it cannot exhaustively verify the correctness of the entire model.

CPN Tools was used to calculate reachability graphs for analysis, and design and architecture nets (three errors were detected: infinite looping; no retry limits; and missing logic to reset timeouts). Modeller experience helps significantly in interpretation of the analysis report produced by CPN Tools and to highlight unexpected analysis results. It is unlikely that all these errors would have been detected within the UML activity diagrams using static inspection alone.

Model-checking means ignoring parts of the system-of-systems either through abstraction or considering a subset of the system-of-systems. However, model-checking is automatic and exhaustively checks the model.

7. Precision in specification of requirements (scalability, concurrency, state-based specification, information-based specification, event-based specification)
In terms of UML 2.0's activity diagram, it has intuitive, graphical concrete syntax but does not have fully formal syntax and semantics. Activity diagrams can be used for concurrent, scalable, state, event, and data-based specification (in conjunction with class diagrams).

In terms of the Petri net formalism, timed, high-level Petri nets are used for concurrent, state, data, event-based specification. Petri nets scalability requires careful management due to their limited concrete syntax.

Results from the second study (exchange network parameters specified using CPN Tools) indicate that at the system-of-systems design level, systems-of-systems have non-streaming activities, i.e. terminating, discrete-event (rather than continuous) where items are accepted at the start of activity execution, processed, and output at the end of activity execution. Specification of continuous or streaming activities (i.e. activities dealing with inputs and outputs continuously during their execution) would require stochastic Petri nets.

In the study, nets were developed to specify analysis, and design and architecture models of exchange network parameters. At analysis level, operational processes were described. This led into solution specification at design and architecture levels. Again, these levels described process-based information exchange with associated events, probabilities and states. The specification was unambiguous in the structural sense.

Tokens are not accepted by transitions in process of execution and can be queued. More than one token input can be specified to a transition but it is not possible to specify acceptance of one token and then a late token. Multiple outputs can have probability applied to them. There is no concept of persistent data store accessible across transition executions.

In CPN Tools, colour (type) definitions exist independently of the activities they are used in.

Due to their generic concrete syntax, nets rely on extra net elements and annotation (in comparison to activity diagrams) to relate domain and system specification concepts (e.g. iteration, decisions, cardinality, operations, parameters, constraints). This means resulting nets are much larger than the equivalent activity diagram and can lead to scalability issues unless carefully managed. Scalability issues were forced in exchange network parameters when multiplicity was examined at the analysis level of abstraction for sixty-seven subnets and approximately three hundred and forty places. Toolset syntax and semantic-checking duration and editing response times increased significantly. Manageability and navigability of nets within the toolset were severely compromised.

The exchange network parameters study examined multiplicity of processes realising functions. Specification of multiplicity to reflect multiple, concurrent physical entities was achieved using toolset instantiation of processes. It was necessary to ensure tokens produced for equal consumption by multiple processes were output to input places dedicated (rather than shared) to each process. Simulation of nets specifying multiplicity aided completeness specification in terms of information exchange content.

The study also investigated re-entrancy and it was found that due to the modular nature of system-of-systems specification re-entrancy was easier to implement within modules or to specify alternative courses of actions in separate models where this was not possible.

Overall case study results are presented in Table 6.6.

System-of-Systems Modelling Need	UML Activity Diagram	Petri Nets
1. Precision in requirements specification		
Formal syntax & semantics	No.	Yes.
Process-based	Yes.	Yes.
Multiplicity	Yes.	Yes (toolset instantiation feature).
Re-entry	Yes (requires management).	Yes (requires management).
Discrete	Yes.	Yes (timed coloured nets. Continuous time activity specification may be required at system level & timed stochastic nets would be needed for this).
Data flow	Yes.	Yes.
Resource usage	No.	Yes.
Scalable	Yes.	No (requires management).
State	Yes.	Yes (to a greater extent than UML).
Control flow	Yes.	Yes.
Concurrency	Yes.	Yes.
Independent activity description	Yes.	Yes (separate net).
Independent data description	Yes.	Yes (colours).
Persistent data	No.	No.
Interfaces	Yes (plus class diagram).	Yes (annotation & hierarchy).
Information exchange protocol	Yes (plus schedulability profile).	Yes (timed nets).
Analysis size	More compact than nets.	Larger models than UML.
Design & Architecture size	More compact than nets.	Larger models than UML.
2. Verification and validation		
Formal syntax & semantics	No.	Yes.
Static inspection	Yes.	Yes.
Dynamic inspection (simulation)	No.	Yes.
Behaviour checking		Yes (including detection of behavioural gaps).
Performance analysis		Yes (timed nets).
Analysis-of-alternatives		Yes (timed nets).
Exhaustive analysis		No.
Complete specification		Yes (constrained by net size).
Reachability graph calculation	No.	Yes.
Structural properties		Yes (e.g. deadlock, boundedness).
Temporal logic queries		Yes (e.g. correct protocol).
Largeness avoidance		Yes (abstraction, net division).
Exhaustive analysis		Yes.
Complete specification		No (dependent on scope).
QoS	Yes (plus QoS profile).	Yes (annotation).
3. Abstraction		
Decomposition	Yes (plus class diagram).	Yes (substitution transitions & fusion places).
Activity composition	Yes (plus class diagram).	Yes (annotation, separate net).
Cardinality	Yes (plus class diagram).	Yes (annotation, separate net).

System-of-Systems Modelling Need	UML Activity Diagram	Petri Nets
Allocation	Yes (swim lane).	Yes (annotation).
4. Modularisation		
Interfaces	Yes (plus class, collaboration diagrams).	Yes (substitution transitions, fusion places, & colour).
Information exchange protocol	Yes (plus class, collaboration diagrams and schedulability profile).	Yes (timed nets).
Top-down, bottom-up support	Yes (top-down), Yes (bottom-up).	Yes (hierarchy & cloning).
5. Timing		
Duration, timeout, arrivals	Yes (plus schedulability profile).	Yes (timed nets, deterministic & stochastic functions).
Static	Yes.	No (simulation).
6. Data typing		
Domain concepts	Yes (plus class diagram).	Yes (timed, coloured nets).

Table 6.6 'Summary of overall case study results'

6.6.3 Evaluation Conclusions

From the results in Table 6.6, in terms of the first criteria for success, 'Precisely specify the exchange network parameters example (research questions 2 and 3), Petri nets were used again in a top-down engineering approach to unambiguously (regarding model structure) capture the system-of-systems problem and solution spaces and the operational processes, component systems and information exchange involved. A bottom-up approach was used to help identify the functional hierarchy used in the models from the existing exchange network parameters and dynamic host configuration protocol message sets.

Different abstraction levels were used to describe process-based information exchange with: associated events; component system interfaces; states; type of information exchanged; information exchange protocol; execution control flow; initial request arrival timing, event durations and timeouts; interfaces and operations used by component systems; and potential failure or success states (e.g. underlying communications failure, successful join request response) together with their probability.

As well as reinforcing the results of the first study on close air support for the first criteria for success (section 5.6.3), the second study on exchange network parameters was used to explore the shortfalls the first study did not cover (section 6.3). The metrics of the first criteria refer to the ability of nets to specify multiplicity and re-entrancy.

Specification of multiplicity contributes to development of a model closer to a real-life scenario and use of simulation in conjunction with multiplicity can help the modeller detect omitted logic, information or behaviour. In the case of the second study on exchange network parameters, specification of multiplicity was achieved through the toolset instantiation feature and consideration of multiple request tokens.

Multiplicity enabled detection of omissions in the exchange network parameters standard. Use of multiplicity together with simulation for performance analysis purposes can also enhance verification and validation of models. This is discussed later in this section under the third criteria for success. It was noted that use of multiplicity can quickly lead to increased model complexity and its specification benefits from an incremental process such as the one followed for the second study (i.e. using a minimum number of instances for each process in order to check modelled behaviour).

From the examination of multiplicity, the second study also strongly suggested that the scalability of Petri nets needs careful management, attributable to their generic concrete syntax. Nets rely on extra net elements and annotation (in comparison to activity diagrams) to relate domain and system specification concepts and can quickly become complex and difficult to read in terms of the number of elements and associated annotation.

The other shortfall referred to by the metrics of the first criteria for success relates to the ability of nets to specify re-entrancy. It was noted that while it is possible to capture alternative courses of action like error recovery, if the model is partitioned into modules using hierarchy ports and sockets, it can be difficult to facilitate cross partition adjustment of net elements that are not interfaced. Even if place fusion is implemented, re-entrancy creates dependencies between modules in addition to their information exchange interfaces. Re-entrancy is easier to capture if it can be self-contained within a module. In the second study, capturing the error state within a subnet and then specifying a means of recovery in a separate net was a more satisfactory error recovery specification method.

For the second criteria for success, 'Determine the scalability of the exchange network parameters system-of-systems model implemented using Petri nets (research question 2)', the first study indicated Petri nets do not scale well according to the size of the system to be modelled. The second study also highlighted scalability needs careful management. Performance issues related to toolset navigability were encountered at the analysis level when considering multiplicity for sixty-seven subnets and approximately three hundred and forty places. Due to their generic concrete graphical syntax, nets tend to use extra net elements and annotation to relate domain and system specification concepts (e.g. iteration, decisions, cardinality, operations, parameters, constraints). This means the resulting net is much larger than the equivalent UML activity diagram (that can also lead to decreased readability) leading to these scalability issues.

For the last criteria for success, 'Confirm if the same Petri net verification and validation techniques used in the telephone and close air support exercises are effective in the exchange network parameters system-of-systems specification models (research question 1)', CPN Tools was used to explore verification and validation of the second study on exchange network parameters specifications at the analysis, design and architecture levels of abstraction. Both studies on close air support and exchange network parameters highlighted the fact that reachability graph calculation provides exhaustive verification but only across a restricted (in terms of model size and detail) specification. In comparison, simulation provides verification across the

whole specification (constrained by model size, toolset, and underlying hardware) but is not an exhaustive means of checking model correctness.

The use of simulation detected token overtaking, incorrect predicates, missing timeout resets, missing retry limits, and missing initial values specification errors in the exchange network parameters system-of-systems. In addition, use of interactive simulation (together with detecting and correcting errors) helped to engage domain users and highlight omissions and implied requirements in the referenced standard [121]. These errors and omissions would not necessarily be highlighted during static inspection of the equivalent UML activity diagrams. When timing and probability were introduced into the exchange network parameters nets, simulation could be used to undertake performance analysis (the calculation of the average successful join request fulfilment time was demonstrated based on automatic simulation replications). Multiple instances of a process can be introduced into nets to specify the interaction of multiple physical entities realising functions. When simulated, model behaviour can be checked again for omissions in referenced standards (e.g. in the second study on exchange network parameters, specification of multiplicity highlighted the need for transaction identifiers and unique node identifiers to be assigned to process instances as well as messages) and efficiency. Performance analysis with multiplicity allows the modeller to consider the effect of multiple instances on overall or individual process duration. Again, this analysis is not possible with activity diagrams.

As mentioned, simulation is not an exhaustive method of checking model behaviour is correct and complete. Reachability graphs for the exchange network parameters analysis, and design and architecture models were calculated for this purpose. As indicated in Table 6.4, reachability graph analysis detected three errors in the exchange network parameters system-of-systems specifications (infinite looping, and missing logic). If these errors had not been detected by simulation, reachability graph analysis would have alerted the modeller to their presence. Exhaustive verification is not possible on native activity diagrams.

Following execution of the second study on exchange network parameters, the benefits of the Petri net formalism remain as indicated by the first study on close air support, and are:

1. Analysis capability. Constructing models using Petri nets allows the modeller to apply simulation and reachability analysis techniques to the models and check them for correctness, and completeness.
2. System-of-systems specification capability. Petri nets enable unambiguous specification in terms of model structure and have extensions permitting capture of the attributes desirable for system-of-systems requirements specification.

The weaknesses of the Petri net formalism are:

1. State space explosion. Calculation of a Petri net model reachability graph is an exhaustive means of correctness-checking but is subject to the logical structure of the model. Successful calculation can be helped by largeness avoidance techniques discussed in Appendices B-D and the case study on close air support in section 5.5.3.
2. Scalability and readability. The generic concrete syntax of Petri nets means resulting models tend to be large and potentially more difficult to interpret. Successful

specification using nets requires management of the system-of-systems problem and solution scopes prior to model construction and implementation of a best practise strategy governing their creation and management.

For organisations using Petri nets in the specification of systems-of-systems, the choice appears to be exhaustively verifying a restricted system-of-systems specification, or part-verifying a complete system-of-systems specification, or managing and using a combination of the two forms of analyses. The important issue for systems-of-systems is to specify the requirements correctly in the first instance at a system-of-systems engineering level rather than at a system level. Petri nets offer a means of unambiguously (in the model structure sense) specifying requirements so that they can be automated to achieve consensus between technical and non-technical audiences and examine behaviour correctness and completeness. The challenge will be in managing the models and deriving best practice to maximise specification and the two forms of verification and validation possible with Petri nets.

.

Chapter 7 Conclusions

7.1 Introduction

This chapter reviews the work of this thesis, comparing it to the original criteria for success outlined in chapter 1. The industrial perspective of the research is discussed in relation to the Knowledge Transfer Partnership from which it was funded, and directions for further work are suggested.

7.2 Review of Research

7.2.1 System-of-Systems Level Design Specification and Analysis Problem

Chapter 1 discussed the concept of system-of-systems, defining the term for the purposes of the thesis, and highlighting areas of difficulty in their engineering. The overall problem to be addressed was defined as:

1. There is no complete, correct, and consistent capture of the problem and solution design specification (particularly information exchange) at the system-of-systems level.
2. There is inadequate verification and validation of the design specification providing low levels of assurance that the design will lead to desirable implemented behaviour.

Within this overall problem, two research issues were identified:

1. Where the dynamic, behavioural diagrams of UML can and cannot be used to model and analyse system-of-systems.
2. Determining how Petri nets can be used to improve the specification and analysis of the dynamic model of a system-of-systems specified using UML.

7.2.2 Potential of Petri Nets

The Petri net formalism is introduced in chapter 2 together with justification for its use as a potential solution to the system-of-systems design specification and analysis problem. Desirable system-of-systems modelling language features (described in chapter 1) such as: abstraction; modularisation; data typing; adequate toolset support; timing; verification and validation; precision in specification of requirements; and scalability, concurrency, state, information, and event-based specification were discussed in relation to Petri nets. It was indicated that hierarchical, high-level, timed nets could offer the modelling language features deemed necessary for system-of-systems specification. Chapter 2 set the context for the evaluation of these hierarchical, high-level, timed nets in relation to system-of-systems specification and analysis in the case study of chapters 5 and 6.

7.2.3 Research Approach, Weaknesses of UML, and Petri Net Formalism

Due to the unstructured nature of system-of-systems problems, an alternative research approach to a tightly controlled experimental investigation is desirable to help in their understanding. The case study research approach offers an opportunity to gain this understanding despite scepticism regarding its suitability as a research method. Chapter 3 discusses the technique and a systematic approach to the design and execution of the case study used in thesis chapters 5 and 6 is outlined.

The first study on close air support is introduced in chapter 4 following discussion of where the UML dynamic model diagrams can and cannot be used to model and analyse system-of-systems. The main weakness of all the UML behavioural diagrams is their lack of well-defined concrete and abstract syntax and static and dynamic semantics. This means the behavioural diagrams of UML cannot be executed or used to derive a reachability graph for the purposes of exhaustive correctness checking. Both functions are viewed as essential in helping produce correct specifications for systems-of-systems. Chapter 4 also highlighted the UML activity diagram as the most applicable candidate diagram for enhancement by Petri nets in system-of-systems specification. This was due to it being the only behavioural diagram able to specify overall system behaviour; hierarchy support; scalability; and support for timing via an extension profile.

By specifying a well understood problem (a telephone system, Appendices A-D) using the Petri net formalism, its potential strengths and weaknesses in relation to system-of-systems specification were listed. Using this information, chapter 4 concludes by suggesting how Petri nets can be used instead of UML activity diagrams and introduces the first study, close air support, indicating the characteristics of interest it holds for system-of-systems engineering in general.

7.2.4 Case Study

Chapters 5 and 6 execute the case study design presented in chapter 3. The design objective of the case study was derived from chapter 1's second criteria for success. This stated that the strengths and weaknesses of Petri nets were to be determined regarding the greater formalism of dynamic behaviour in systems-of-systems and their role as a means of engaging stakeholders. The case study (from the defence domain), is used to answer the research questions:

1. Do Petri nets improve the functional correctness of the system-of-systems design specification?
2. Do Petri nets increase the quality of the design specification?
3. What are the shortcomings of the state-of-the-art Petri net tool and how can it be improved?

Criteria for success were derived from these research questions in relation to the Petri net specification of the close air support and exchange network parameters studies. Specification of each study was carried out at different system-of-systems engineering design levels using Petri nets. Study response variables were captured at each iteration of model design in order to measure whether the criteria for success were met.

The first study, close air support, confirmed the following strengths of Petri nets in relation to system-of-systems specification: capability to analyse model behaviour; and capability to specify range of model behaviour (state, event, concurrency, data, timing and hierarchy). The following weaknesses were also confirmed: state space explosion; and scalability. A number of shortfalls were highlighted in terms of: multiplicity of component systems or processes realising functions; verification and validation and its use in checking specification completeness; and re-entrancy and its effect on correct behaviour.

The second study from the military domain, exchange network parameters, confirmed the strengths and weaknesses identified by the first study on close air support. In terms of the shortfalls, the second study was able to explore re-entrancy and multiplicity, concluding both could be specified using Petri nets and recommending best practice for their use in system-of-systems modelling.

7.2.5 Research Results

Based on the results of the case study of chapters 5 and 6, the strengths and weaknesses of Petri nets in relation to the formalism of dynamic behaviour specification in system-of-systems were identified. Based on these, a final enhancement to UML was proposed (Appendix E).

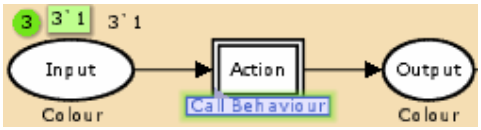


Fig. 7.1 'Petri net replacement for UML CallBehaviourAction activity diagram element'

This enhancement complements UML with timed, coloured Petri nets with hierarchy, providing a mapping between activity diagram nodes and Petri net elements (see Fig. 7.1 for an example activity diagram element replacement). The enhancement uses simulation for initial verification of specifications and performance analysis of timed specifications, and reachability graph calculation for exhaustive verification of specifications. Timed, coloured Petri nets with hierarchy address the specification part of the overall system-of-systems problem, where as simulation and reachability graph calculation address the assurance part of the overall system-of-systems problem.

7.3 Evaluation of Research

Using the research aims and criteria for success defined in chapter 1, an evaluation of the work in this thesis is discussed.

1. Indicate the strengths and weaknesses of the behavioural diagrams of UML regarding the specification, and verification and validation of systems-of-systems.

To address the first main research issue, chapter 4 discusses the UML dynamic model and the behavioural diagrams it is composed of in relation to the overall system-of-systems problem. The chapter summarises each diagram in relation to a system-of-

systems specification feature and concludes that the main weakness of the UML behavioural diagrams is their lack of formal dynamic semantics, i.e. it is not possible to execute or perform exhaustive checking of models built using the syntax and semantics of the three types of UML diagram. From the summary of the three UML behavioural diagrams against system-of-systems specification features, UML activity diagrams were deemed most applicable to the specification of systems-of-systems. Activity diagrams offered the ability to support overall behaviour specification, hierarchy, scalability, and timing via an extension profile.

2. Determine the strengths and weaknesses of Petri nets regarding the greater formalism of dynamic behaviour in systems-of-systems, i.e. their specification, and verification and validation. This should cover Petri nets' ease of use; comprehensibility; scalability; state, data, and event-based modelling capability; concurrency modelling capability; and verification and validation capability. The role of Petri nets as a means of engaging stakeholders should also be examined.

To address the second main research issue, chapter 2 examines the desirable features a modelling language should offer for the specification of systems-of-systems (abstraction, modularisation, data typing, adequate toolset support, timing, verification and validation, precision in specification of requirements, and scalability, concurrency, state, information, and event-based specification) in relation to Petri nets. The findings of the chapter indicated Petri nets can provide each of these features (with scalability to be explored further) graphically via high-level, timed nets with hierarchy.

Chapter 4 considers a well understood problem (a telephone system) in order to further explore the potential usefulness of Petri nets in creating models of systems-of-systems. This exercise confirmed the need for high-level nets as well as the definition of a suitable hierarchy and adoption of an adequate Petri net toolset. The chapter summarised perceived strengths and weaknesses of Petri nets in system-of-systems development based on the experience of modelling the telephone system.

Chapters 5 and 6 use the findings from Petri net specification of the telephone system in chapter 4 for a system-of-systems case study. Chapter 5 executes the systematic case study design described in chapter 3 for the first system-of-systems study from the military domain, close air support. Close air support was used to further demonstrate the benefits, weaknesses, and shortfalls Petri nets bring to specification and analysis of large-scale systems-of-systems. The study confirmed greater formalism of dynamic systems-of-systems behaviour via Petri nets' analysis capability (simulation and reachability graph calculation) and specification capability (state, event, concurrency, data, timing, and abstraction). Both were considered Petri net strengths. Their graphical concrete syntax and ability for execution also enabled stakeholders to be involved in correct and complete behaviour specification. Petri net weaknesses included state space explosion and scalability.

Chapter 6 considers the problem of exchange network parameters (from the military domain) for the second study, confirming the strengths and weaknesses, and exploring further the shortfalls identified by the close air support study. Both studies also highlighted the need for best practice to be established within organisations tailored to their system-of-systems specification requirements.

3. Show how Petri nets can be used to replace UML activity diagrams and address the overall problem of system-of-systems specification and analysis.

To address the third main research issue, chapter 2 sets the context for the evaluation of Petri nets in the specification of systems-of-systems by considering the notation in relation to desirable systems-of-systems specification features. Chapter 4 lists the strengths and weaknesses of UML activity diagrams in relation to systems-of-systems desirable specification features. Based on this list, activity diagrams appear to be the most applicable UML behavioural diagram for use in systems-of-systems specification. Their main weakness identified in chapter 4 is the lack of formal dynamic semantics. It is not possible to execute or perform exhaustive checking of models built using the present activity diagram syntax and semantics. Chapter 4 suggests how Petri nets could be used to enhance UML with a notation offering formal semantics, and graphical concrete syntax. A well-known problem (telephone system) is specified using Petri nets in order to identify the potential strengths and weaknesses of the notation in the modelling and analysis of systems-of-systems.

4. Demonstrate and evaluate the feasibility of the Petri net solution to the overall problem of system-of-systems specification and analysis using a case study approach.

To address the fourth main research issue, chapters 5 and 6 demonstrate the feasibility of specifying two system-of-systems studies using Petri nets, summarising the strengths, weaknesses, and shortfalls of the notation based on case study research questions and criteria for success for each study. Chapter 5 conducted a study on specification and analysis of a close air support system-of-systems, ending with an evaluation of the study results. As part of this analysis and the work of chapters 2 and 4, the modelling needs of systems-of-systems were considered in relation to activity diagram and Petri net modelling notations and the actual results from the study. Petri nets were found to offer the following attributes over activity diagrams: formal syntax and semantics; specification of resource usage; dynamic inspection; reachability graph calculation; and non-static specification using timing. Appendix E details the proposed Petri net enhancement of activity diagrams.

In terms of the overall system-of-systems specification and analysis problem, Petri nets are able to provide analysis capability through simulation and reachability graph based on their full formal syntax and semantics. Petri nets provide specification capability enabling representation of state, event, concurrent, performance, and data-based behaviour of a system-of-systems at different levels of abstraction detail.

The main weaknesses of Petri nets identified by the thesis are their ability to scale to the size of the system to be modelled and the state space explosion problem associated with model-checking. As shown in Appendices A-D and the thesis case study of chapters 5 and 6, both weaknesses can be managed to an extent using abstraction at the system-of-systems engineering level. Based on application of careful management and the results from the second study on exchange network parameters of chapter 6, Petri nets are proposed as an enhancement to the activity diagram to help address both parts of the overall system-of-systems level design specification and analysis problem.

The research aims and criteria for success defined in chapter 1 have been met based on the work presented in this thesis. The remainder of this chapter elaborates on this work, discussing its industrial context, and highlighting areas for further research.

7.4 Discussion

Overall, the proposed Petri net formalism of dynamic behaviour in the specification of system-of-systems has been successful. The proposal significantly improves upon existing specification techniques for systems-of-systems and meets the criteria for success defined in chapter 1. In particular, Petri nets can be viewed as complementary to de-facto UML activity diagrams. They can be used in conjunction with them or as a system-of-systems specification notation in their own right (Appendix E details the mapping between activity diagram nodes and Petri net elements necessary for specification of system-of-systems behaviour). Due to their underlying mathematical description, Petri nets enhance the specification and analysis capability currently possible with UML activity diagrams. Model execution and analysis of a model's reachability graph can be undertaken by the modeller in conjunction with system-of-systems' stakeholders to achieve correct and complete system-of-systems behaviour specification early in the system-of-systems' lifecycle.

As indicated, the two major advantages Petri nets have over UML are the analysis and specification capabilities available to the modeller for precisely specifying and assuring the system-of-systems design. Due to their formal syntax and semantics, Petri nets permit unambiguous (from a model structure viewpoint), visual specification of system problem and solution spaces using a small range of graphical modelling elements. Slightly disappointing was the readability of nets, time-intensive net construction process (particularly for new and non-practitioners), and toolset support for system-of-systems models likely to involve large numbers of subnets and continuous model evolution. Useful system-of-systems specification with Petri nets requires definition of best practice over many years (regarding model construction and management, annotation, and practitioner training) and selection (or development) of an adequate net modelling framework. The work of the thesis has shown the feasibility of using Petri nets to specify process-based systems-of-systems, capturing the desirable system-of-systems features of state, event (including event probability), concurrency, data, timing, and abstraction.

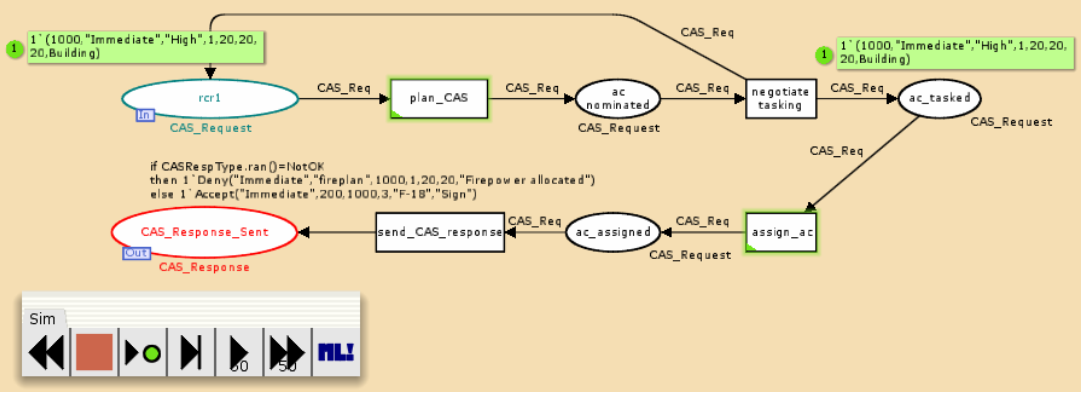


Fig. 7.2 'Infinite net problem detected by interactively executing net'

The formal syntax and semantics of Petri nets also means that constructed models can have well-defined algorithms for execution and reachability graph calculation (model-checking) applied to them. These facilitate simulation and exhaustive model-checking analyses on the models for the purposes of checking the correctness and completeness of modelled behaviour. When timing is introduced to a Petri net model and the model is simulated, performance analysis and analysis-of-alternatives can be conducted on the model. These analyses can be used to further validate the modelled behaviour with domain users prior to hand-off to subsequent system-of-systems' development stages. In the thesis, simulation was used to demonstrate this form of model validation and to initially verify model design iterations. The interactive form of simulation (Fig. 7.2) is a useful means of stepping through model execution for the purposes of sharing the specification with non-practitioners and gaining consensus in the correctness and completeness of model behaviour. More automatic forms of simulation are useful for the purposes of obtaining a range of pre-defined performance indicator values by varying different model parameters in order to assess the efficiency, correctness, and completeness of the model logic and behaviour. While simulation can be used to check the behaviour of a Petri net model, it cannot normally be used to exhaustively verify correctness of a complete system-of-systems model specification.

Reachability graph calculation on the model is an exhaustive form of correctness verification but was shown to be subject to the state space explosion problem discussed further below. Design errors not detected by static inspection or simulation of the Petri net model were shown to be highlighted through reachability graph calculation. The CPN Tools toolset produces a standard analysis report for interpretation by the modeller (again, practitioner training in the toolset selected for an organisation is recommended). In addition, the modeller can also construct non-standard temporal logic queries to be run against the calculated reachability graph for the purposes of confirming or denying certain model properties. Reachability graph calculation is dependent on the scope of the system-of-systems specification. Again, slightly disappointing for both forms of model analysis was toolset reporting support. For the thesis case study of chapters 5 and 6, both forms of analyses were viewed as extremely positive for system-of-systems design assurance. However, the positive extent of their contribution is dependent on the ability of the modeller to interpret, model, and verify the system-of-systems problem with domain experts and interpret the feedback provided by the toolset regarding the analyses.

The thesis case study, particularly the second study on exchange network parameters, highlighted the effectiveness of Petri nets in a real, industrial system-of-systems specification situation. As part of the Knowledge Transfer Partnership responsible for funding the project, the company and university partners were able to use the study to demonstrate how the graphical, concrete syntax of nets can be used to specify a process-based reference standard and then detect omissions and ambiguities within it for defence domain customers. It is also anticipated that Petri nets will be used as part of an overall system-of-systems engineering approach across multiple customer domains to improve the specification, and verification and validation of systems-of-systems. Further case studies in non-military domains would serve to further investigate and validate the results obtained by the case study presented in this thesis.

The case study approach as a research method helped the company partner assess the strengths, weaknesses, and shortfalls of a new method in a cost-effective manner

within one particular environment. The usefulness of the outcome was the result of following a systematic approach to undertaking the case study. It could be argued that without such a planned approach, a means of comparing and analysing valid results between the studies would not have been established. The functional correctness and design quality measures were selected to reflect developer and end-user views of correctness and quality. The two studies were selected to be representative of typical system-of-systems problems and deemed comparable based on objective system-of-systems characteristic measures listed in chapter 1. Both studies verified similar strengths and weaknesses of Petri nets in system-of-systems specification, and did so for two examples from the same application domain. Although this may suggest that use of Petri nets would exhibit the same strengths and weaknesses across all domains, this generalisation cannot be made for certain (at least not until further case studies conducted in non-military domains provide further validation).

The two major disadvantages of the Petri net formalism of dynamic behaviour are their ability to scale according to the size of the system to be modelled (and subsequent model readability) and the state space explosion problem affecting model-checking.

It should be noted that in terms of managing their ability to scale, the system-of-systems specification needs to be suitably abstracted, i.e. scoped appropriately. Although the case study systems-of-systems could be specified at analysis, and design and architecture levels of abstraction, it demonstrated that detail in nets addressing a wide specification scope should be generalised as far as possible with a suitable hierarchy determined in advance of modelling. Advantage should also be taken of a Petri net toolset features such as re-use of nets within a model (CPN Tools instantiation feature), colouring, and annotation in order to achieve the desired specification.

Functional decomposition was used to identify a hierarchy within models and was influenced by a combined top-down and bottom-up engineering approach. This flexible function-based approach is deemed similar to the one presented by service-oriented architecture where re-usable functions or services are identified and then composed from legacy and new component systems to build applications. The functions or services provided by existing message sets in close air support and exchange network parameters systems-of-systems were used to suggest a hierarchy within models in conjunction with standards specification documents. In this way, a service-based architecture helped to address scalability of the specification. The architecture also promoted non-prescriptive solution specification in analysis, and design and architecture level models by identifying and focusing on the services to be realised by components.

In the case of design at the system-of-systems level, operational services are identified using a combination of top-down and bottom-up engineering approaches. Eventually, existing physical systems that implement these services as closely as possible are then selected (or procured) based on the original system-of-systems design and derived component system specifications. Currently, this system-of-systems composition is a static, pre-planned, manual process. Given the highly dynamic and heterogeneous nature of the military domain (and domains such as health, traffic management, and policing), a future research challenge would be the dynamic composition of the

services offered by available physical component systems in an operational environment. Part of this research challenge would involve addressing service advertising (including quality of service and component evolution); security; and semantics of exchanged data (both in terms of service discovery and the information exchanged between component systems). Another part of the challenge would involve accurately modelling older legacy systems, particularly those where there is inadequate documentation; and assessing where to begin the process when the system-of-systems problem is of the scale highlighted in chapter 1.

In order to employ the benefits of model-checking on a system-of-systems model, the largeness avoidance abstraction techniques discussed in Appendices B-D and the study of chapter 5 on close air support should be applied. An incremental approach was taken to construct models in the close air support study to judge which sub-functions to abstract detail away from so that a reachability graph could be calculated. In addition, a compositional largeness avoidance technique can also be applied to component systems of the complete system-of-systems specification (see Appendices B-D, sections B.1.5, C.1.3, and D.1.5 for details).

For organisations using Petri nets in the specification of systems-of-systems, the choice appears to be exhaustively verifying a restricted system-of-systems specification, or part-verifying a complete system-of-systems specification, or ideally managing a combination of the two forms of analyses. The important issue for systems-of-systems is to specify the requirements correctly and completely in the first instance at a system-of-systems engineering level rather than at a system level. Petri nets offer a means of unambiguously (in the model structure sense) specifying requirements visually so that they can be automated to achieve consensus between technical and non-technical audiences and examine behaviour correctness as part of an iterative development process.

Without this greater formalism of dynamic behaviour, execution and reachability graph analysis of the modelled behaviour could not be undertaken and exploited by the modeller (this is the case with native UML activity diagrams). No interactive simulation or discussion of reachability graph analysis results could be used by the modeller to engage stakeholders in assuring specification completeness and correctness. The challenge will be managing the models and deriving best model construction practice over many years to take advantage of the two forms of verification and validation possible with Petri nets and promote scalability and readability within Petri net-based models.

In terms of lessons learned during the course of this thesis, the main one was the realisation that system specification languages accepted as an industry standard are not without flaws, and should be thoroughly reviewed prior to their implementation within an organisation according to its specification objectives. In this way, weaknesses of the language in relation to specification requirements can be assessed and alternative, complementary approaches sought where applicable. The other lesson learned related to terms used commonly within a domain and the importance of identifying a common definition and shared understanding of them amongst stakeholders of a project.

In general, the proposed Petri net formalism can be viewed as complementary to the existing UML activity diagram and is a successful solution to the system-of-systems design specification and analysis problem in its own right.

7.5 Further Work

The research undertaken by this thesis could be extended in several ways and these are highlighted in this section.

7.5.1 Evaluation for a Different Domain

The thesis military domain case study of chapters 5 and 6 demonstrated that Petri nets meet many of the desirable requirements for specification of systems-of-systems. This could be investigated further by undertaking a case study from a non-military domain and lead to further research on the Petri net formalism in addition to providing information about their effectiveness. Use of Petri nets in real specification situations offer an alternative evaluation environment.

7.5.2 Specification Evolution

Although use of Petri nets is intended as a system-of-systems specification language, it may be useful to investigate many versions of the same specification and examine changes throughout its evolution. This insight into the way specifications change over time may be able to advise best practice for evolution of models.

7.5.3 Model Transformation

Given the time and dependability demands associated with large-scale systems-of-systems, the need for efficient and systematic development methods is essential. System-of-systems requirements engineering can be viewed as a key design phase and commonly UML use case and activity diagram-driven. A systematic, automated transformation approach between use case, activity and Petri net models (considering meta-models describing the transformation from UML use cases and activity diagrams to Petri nets) would offer a formal validation framework, and ideally help the production of more precise, complete, and correct system-of-systems specifications.

7.5.4 Toolset Development

The case study of chapters 5 and 6 highlighted that the toolset selected for use in the thesis could be improved in terms of its support for system-of-systems specification. If developing a modelling framework for system-of-systems specification, key functional areas include: the navigability of the model (particularly if there are large numbers of activities specified at lower abstraction levels); versioning and re-use of existing models (including locating suitable existing models and maintenance of consistency between models); analysis report generation and format of produced output; simulation involving network communication between models developed remotely; and ability to select graphics to enhance the basic net elements. The ability to perform model transformation (section 7.5.3) could also be incorporated into such a supporting toolset for system-of-systems specification. In addition, work exploring

largeness avoidance and reduction techniques in Appendix B indicated these areas should also be considered in toolsets supporting system-of-systems design analysis.

7.5.5 Dynamic Composition of Functions (Services)

The functional decomposition approach adopted by the specification examples used in this thesis is a similar one presented by service-oriented architecture. In the thesis, operational functions (services) were identified to establish a hierarchy within the Petri net analysis, design, and architecture models. In terms of implementation, legacy and new physical assets would be selected and integrated on the basis of their perceived ability to realise one or a number of particular services. Currently, this is a pre-planned, manual task. A future research challenge is the dynamic composition of services offered by available physical component systems in an operational environment. Part of this research challenge would involve addressing service advertising (taking into account quality of service and component evolution); security; and semantics of exchanged data (both in terms of service discovery and the information to be exchanged between component systems).

7.5.6 Semantics

Highlighted during the Petri net specification examples in the thesis, a key long term systems-of-systems research problem concerns data semantics and use of ontology. Underpinned by semantic interoperability, additional long term research areas are evolution of component systems, and security. Semantic interoperability affects the data (and meta-data) to be exchanged intra- and inter-organisationally within a system-of-systems. Continuous component evolution demands management of amendments to components in terms of the services they realise, their associated data and data format but to be useful, meaning of terms must be consistent. Security in domains such as defence and health is critical and complex, involving the management of appropriate static and dynamic access to confidential services and exchanged data between autonomous components within the systems-of-systems. Extension of Petri nets to include ontology-related annotation in the specification of systems-of-systems (and consequent reduction of the level of human interpretation required to understand the information exchange protocol involved) should be further investigated.

7.6 Final Summary

The research achieved by this thesis was discussed in this chapter and its positive contribution considered in relation to the criteria for success defined in chapter 1. In addition, a number of future research areas were outlined.

In the previous six chapters (and Appendix E), the context, and motivation of the research were presented, leading to an investigation of UML behavioural diagrams in relation to system-of-systems design specification and analysis, the proposal and description of Petri nets as a formalism for dynamic behaviour capture, and execution of a case study approach to identify the strengths and weaknesses of Petri nets in addressing system-of-systems design specification and analysis. The formalism has been discussed in an industrial context, and potential for it to be used across different

industrial domains was highlighted as a future research area. Additional further research areas were also indicated.

The Petri net proposal is a novel and positive step towards a solution to the system-of-systems design level specification and verification and validation problem. By providing greater formalism of system-of-systems' dynamic behaviour, the modeller can take advantage of model execution and reachability graph analysis. Undertaken in conjunction with stakeholders, these analyses can help assure completeness and correctness of the behaviour specification early in the system-of-systems' engineering lifecycle.

Appendices

Appendix A

The telephone system is specified textually as follows:

The telephone receiver is lifted by a caller and the receiver's number dialled. The caller waits for connection and hears either a ring tone or an engaged tone. If an engaged tone is received, the caller hangs up (this also applies if more than one call has been placed to the receiver, all result in a hang up). If a ring tone is received, the call is either answered by the receiver (in which case a voice call proceeds until the caller or receiver hang up) or it is not (and the caller hangs up).

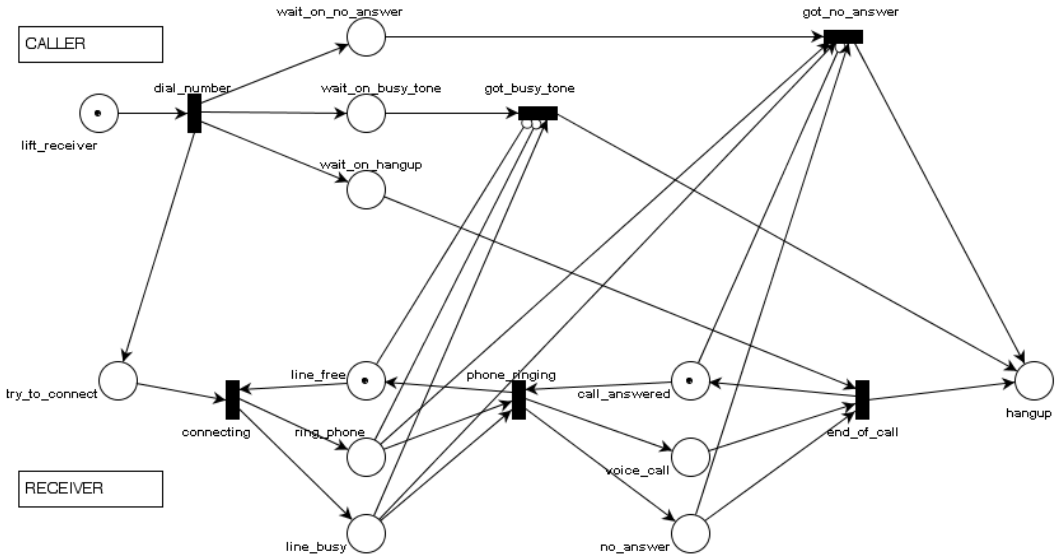


Fig. A.1 'Classic Petri net of a telephone call process'

A.1 Specification of the Telephone Process using a Classic Petri Net

Fig. A.1 was an early attempt at the telephone call model using a classic Petri net to specify the system. Initially, a classic Place/Transition net [57, 63] was used to gain practical experience of using the technique.

Due to the non-complexity and high-level of abstraction of the problem to be modelled together with familiarity with its concept and overall behaviour, the classic Petri net model's constituent elements and viewpoint were rapidly established from the textual specification. The call sequence was modelled from the viewpoint of the receiver of the call. Net places were used to store tokens relating to the state of the call progress, transitions were used to represent activities resulting in a change of state and the control sequence of activity execution was noted. Labelling of places and transitions used terminology from the textual specification. To begin with, one of the scenarios (that of an answered call) was modelled and expanded further with the two unanswered call scenarios in order to build the net shown in Fig. A.1.

The 'wait_on_no_answer', 'wait_on_busy_tone' and 'wait_on_hangup' places were originally used to represent that a call had been made and a result relating to that effort would be relayed back to the caller, i.e. the caller's attempt to place a call would either be answered successfully and end with a hang up, or it would be met with an engaged tone and end with a hang up, or it would trigger ringing but for whatever reason would not be answered and end with a hang up. Fig. A.1's initial marking consists of one token on the place 'lift_receiver' (indicating the state that one telephone call has been initiated), one token on the place 'line_free' (indicating the receiver's line is currently free), and one token on the place 'call_answered' (indicating the receiver of the placed call will respond to their telephone ringing).

Apart from providing the user with a graphical overview of the system specification, the classic Petri net was used to provide some insight into the modelled system's behaviour.

A.2 Initial Verification of the Telephone Process using a Classic Petri Net

Dynamic analysis (interactive simulation) was used to verify that the specification represented by the net in Fig. A.1 behaved as expected. For the initial marking described above and for subsequent initial markings used to reflect an engaged receiver line and an unanswered call, the functionality of the net appeared to be correct. This meant that when a call attempt was input into the system and the initial marking of the net was set up to specify one of the three scenarios, the desired output state of hang up was reached in each of the three cases. Simulation was also able to highlight the poor choice of labelling for the 'hangup' terminal state. This label insufficiently specified how the call had reached this hang up state.

When the net was amended to reflect the fact that more than one call was being placed to the receiver, i.e. more than one token on the 'lift_receiver' place, interactive simulation revealed problems with the logic modelled within the net. These included improper line reset behaviour and redundancy of the three wait places described in section A.1. The net was amended following the results from iterative simulation and is shown in Fig. A.2.

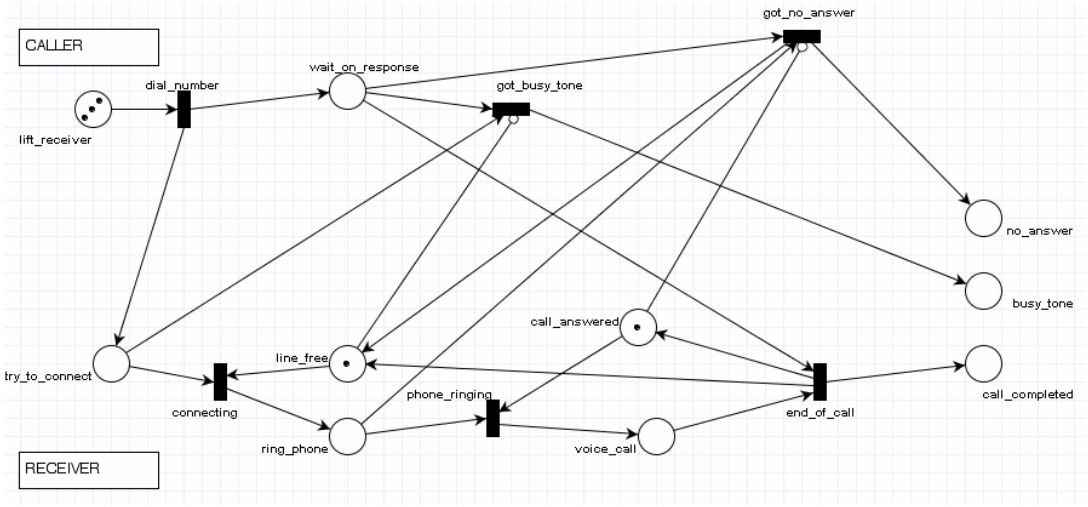


Fig. A.2 'Amended process following dynamic analysis'

Following completion of Fig. A.2's net of the simple telephone example, it was clear that classic Petri nets can quickly become large and complex (however, when finite state machines are used, the model would be even more complex). Constructed reasonably quickly due to the narrow problem scope, high level of abstraction and common concept involved, even with the few improvements made following iterative simulation analysis, the specification appeared cluttered and was difficult and time consuming to interpret. Given this information and the third criteria for success goal of determining scalability of nets (and desire to specify large-scale, complex system-of-systems), it was decided to abandon classic nets and implement a high-level Petri net of the telephone model, specifically a Coloured Petri net. In order to do so, a suitable Petri net tool was selected based on an evaluation of existing Petri net tools meeting thesis requirements (Appendix F).

Tool Survey Conclusion

From the evaluation of the four tools (CPN Tools [47], WoPeD [126], Renew [127], and PIPE [128]) carried out in Appendix F, each was highly useable in its own right. For the intended thesis case study, use of CPN Tools was recommended. CPN Tools had been briefly used earlier in the project to demonstrate Petri nets as a potential modelling tool for system-of-systems development. The toolset offers high levels of support in terms of papers, tutorials, online and offline help, and an internet mailing list forum. It is a comprehensive toolset 'out-of-the-box' allowing immediate model construction, execution and analysis. Further 'plug-in' support comes in the form of branching temporal logic implementation (ASK_CTL), graphing (Graphviz), and animation (BRITNeY) and there are plans to develop improved toolset support for state space exploration and analysis of coloured Petri net models [69].

CPN Tools can be used in a standalone environment to open multiple net models simultaneously and has 'clone' functionality to copy elements of nets between models. Related to integration of models is their export. Currently CPN Tools supports its own XML export (with published Document Type Definition, DTD). Finally, in terms of cost, CPN Tools is free to both academic and commercial organisations.

A.3 Specification of the Telephone Process using a Coloured Petri Net

To try and alleviate the weaknesses identified using classic Petri nets, a high-level Petri net was developed based on the classic net of Fig. A.2. High-level nets have the potential to introduce colour, time and hierarchy to models. Tokens have values (based on a colour or type) referring to features of the object modelled by the token. Use of colour reduces the number of places needed to reflect a state. Defined by Kurt Jensen [63], coloured Petri nets require the colour of the tokens on the input places to be considered as well as existing firing rules. This means that these high-level nets are able to facilitate more compact, natural process description.

A.3.1 The Operational Process (Conceptual) Level

A coloured Petri net representation of the example telephone operational process is shown in Fig. A.3.

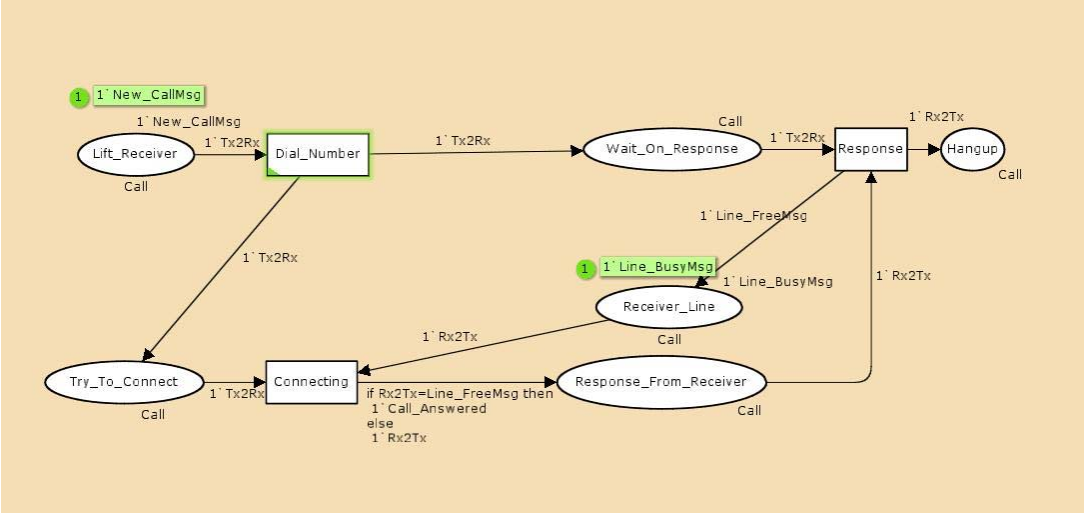


Fig. A.3 'Coloured Petri net model of telephone call process shown in Fig. A.2'

Instead of having three sink places (as in Fig. A.2) capturing the results of the initiated calls, Fig. A.3's net defines an enumerated type or colour named 'Call' with values reflecting potential results (in this case, 'Call_Answered', or 'Line_Busy', or 'Not_Answered'). All places in Fig. A.3's net have the same colour (type), and when viewed together with the labelling of each place, transition and directed arc provide seven observations:

1. A rule-based, graphical model of the call process at a high-level of abstraction.
2. Implicit cardinality of role interaction (via the initial markings of 'Lift_Receiver' and 'Receiver_Line' places) which is m:1 (many callers following the specified activity sequence can try to call one receiver).
3. Call's history (via identification of states) and 'reversibility'.
4. Implicit specification of an operational process using a sequence of activities, firing conditions and information states relating to a telephone call.
5. Implicit specification of an information exchange protocol between a caller and a receiver. The exchange between the two roles uses the caller's 'Dial_Number'

transition and the receiver's 'Connecting' transition and then the receiver's 'Connecting' transition and the caller's 'Response' transition to undertake the information transfer.
6. Implicit specification of provided and required interfaces. The 'Dial_Number', 'Connecting' and 'Response' transitions are similar to operations used in an object-oriented environment. The information produced and used within these operations is specified by the 'Call' colour (type) and can be enforced by guards and arc inscriptions.
7. Implicit viewpoint of the model (i.e. that of the receiver of the calls).

Building on these seven observations of this simple example, it can be seen that the net does not reflect individuality of callers or receivers, i.e. ability to be autonomous and behave in a different way to the sequence specified. For example, the caller could hang up before the receiver can respond to their telephone ringing or before a connection is made to receiver's line, or the receiver could facilitate a conference call between more than one caller. These activities would require tailoring of parts of the interaction for the role variations. This could be achieved using the same net (with a new partition for each role variation. The variation could be visually distinguished using the modelling tool's colour palette for this extension to the net) or a separate net (for each role variation). Unfortunately, depending on the number of role variations, this could lead to a complex net unless a suitable facilitation process is followed. Subject to a suitable facilitation process, incorporation of this variation into the net suggests nets are amenable to amendments. The ability to re-use and evolve nets will be a key requirement in the specification of large-scale system-of-systems.

Prior to any kind of net analysis, termination of the interaction described by the net in Fig. A.3 is not visually obvious and the interaction needs time and effort to follow. It can be argued that in spite of its graphical nature and use of colour (types), Fig. A.3 lacks readability in general, particularly to those unfamiliar with the Petri net modelling technique. For example, there are no graphical logical operators such as 'and', 'or' to aid readability; the viewpoint from which the call process is illustrated, cardinality and activity execution by role are implicit; information exchange protocol between the roles is implicit; provided and required interfaces are implicit; and the initial start-point of the operational process and subsequent execution path are implicit.

Again, the example highlights the potential problems of scale, complexity and readability with a flat, non-hierarchical net employing no enhancements to the net's foundation graphical nature. Larger processes involving multiple roles will quickly generate large nets. Use of hierarchy (as well as a suitable facilitation process) is essential for any chance of specification readability and scalability.

A.3.2 Use of Hierarchy

Hierarchy can structure complex Petri nets in a similar way to hierarchy within data flow diagrams. With hierarchy, a net at a certain level of abstraction (parent net) can have some or all of its transitions described in a greater level of detail by subnets (top-down decomposition). These subnets can be composed of places, transitions and other subnets. Also, hierarchy can be facilitated by linking existing lower-level subnets to transitions within parent nets (bottom-up development).

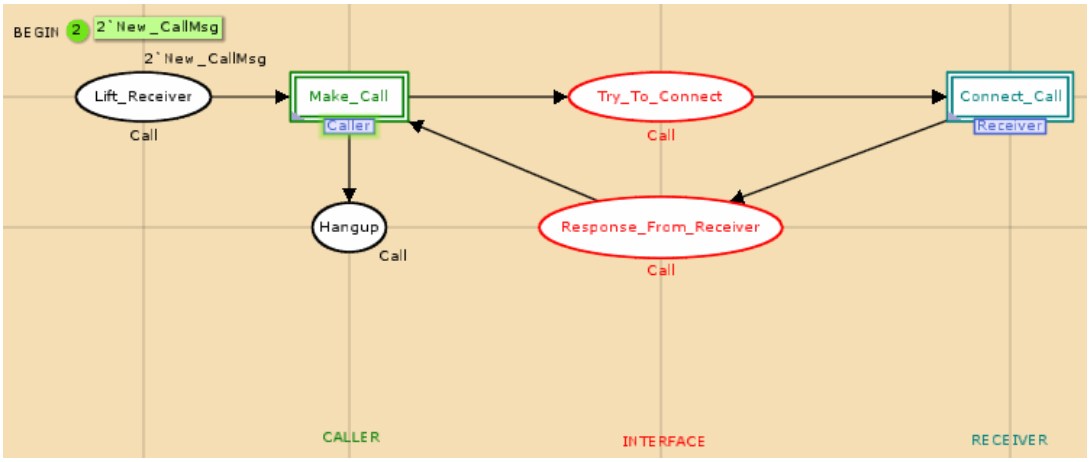


Fig. A.4 'Telephone call process parent net'

The challenge using hierarchy is deciding upon an appropriate abstraction level and viewpoint for the model. This depends on the stage of development the system model is at. So far in the telephone example, nets have been developed at a high level of abstraction, i.e. a conceptual level depicting a telephone operational process from the viewpoint of the receiver of the calls. This level is intended to describe the problem and promote understanding between non-technical and technical audience members so that a non-prescriptive solution can be designed.

This operational process specifies a sequence of activities and information exchanges performed by roles in order to achieve desirable behaviour. This initial conceptual specification then drives detailed design and implementation. Obtaining a hierarchical breakdown of the telephone operational process involved consideration of the existing flat process and the roles and activities used within it to describe an overall function.

Using the flat net, activities were then restructured using a combination of functional decomposition, function (activity)-to-role assignment, and identification of interfaces between roles. The parent net is the top level of the hierarchy, describing the operational process at its most abstract. Within it, the first main activity 'Make Call' is associated with the caller role and the second main activity 'Connect Call' is associated with the receiver role. A trigger process input, a process output and the information exchanged between the two activities (interfaces) are also included. The parent net aims to provide a coherent overview of the telephone process and indicate clearly that more detailed descriptions of the two main activities carried out by the roles are available on subnets. This parent net is show in Fig. A.4.

As mentioned, hierarchical nets can also be constructed in a bottom-up fashion. For large-scale system-of-systems, the concept of subnets as components is extremely useful both in terms of reuse of existing nets and as a means to explore variations in the design of components. Existing, amended or brand new nets relating to individual components of the system-of-systems could be swapped in and out of the composition when considering different application scenarios or designs of components. Viewing system-of-systems components as subnets may also help alleviate the state space

explosion problem in static analysis of nets. This alleviation is considered further in Appendix B, section B.1.5.

From Fig. A.4 it can be seen that hierarchy (and use of toolset colour palette and labelling) greatly improves readability of the model by allowing the modeller to employ levels of abstraction to maximise model scalability and readability. Depending on the domain being captured and a suitable net construction process for large-scale system-of-systems specification, the modeller has the option of abstracting detail when required and employing fairly compact nets. The net construction process should also recommend use of a suitable net modelling tool that provides a colour palette and ability to add annotation and graphics enhancement to nets. These will be essential features for the modelling and analysis of large-scale system-of-systems.

Hierarchy also makes use of input and output socket places to and from the decomposed transitions on the parent net. These sockets have corresponding port places on the resultant subnet describing the decomposition. The colours (types) of these socket and port places can be used to specify the types of the information used and produced by the decomposed activity. In this way, sockets and ports can be viewed as a means of explicitly specifying required and provided interfaces to the decomposed transition. This is illustrated by the 'Caller' subnet shown in Fig. A.5.

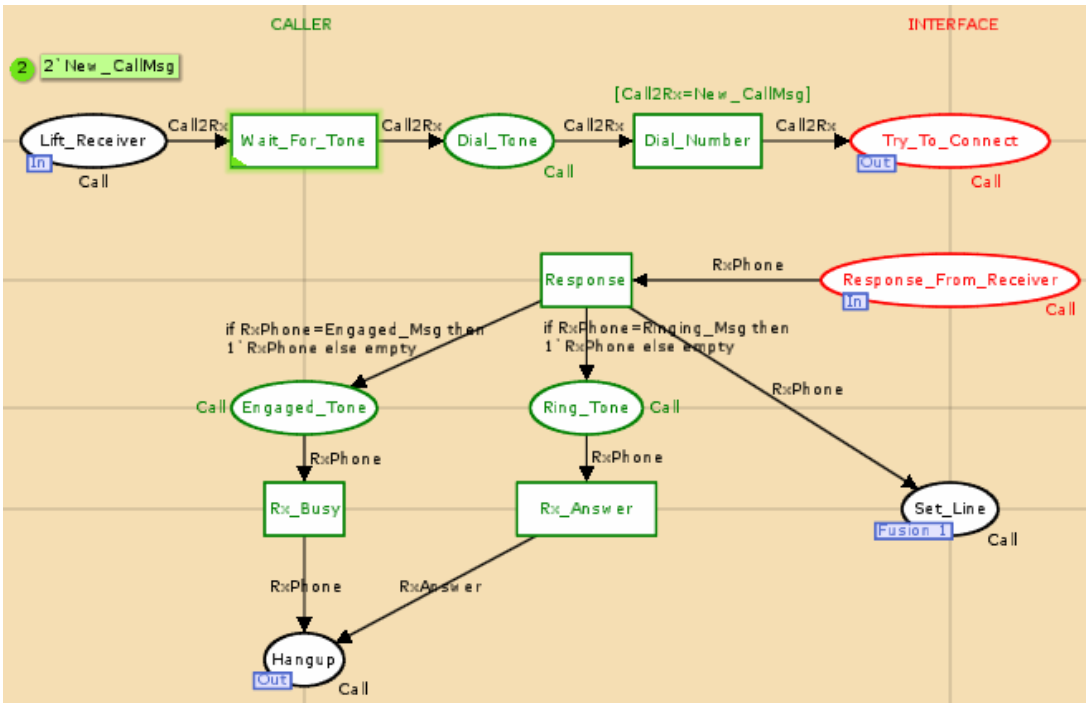


Fig. A.5 'Decomposed subnet for Caller Make_Call transition showing ports'

In Fig. A.5, there are four port places, two input ('Lift_Receiver' and 'Response_From_Receiver') and two output ('Try_To_Connect' and 'Hangup') made use of by the 'Caller' role. All port places have colour (type) 'Call'. At this high level of abstraction, i.e. the operational process level, Fig. A.5 specifies that transition or operation 'Wait_For_Tone' takes information specified by type 'Call' as input from the parent net and produces information of type 'Call' as output to the parent net.

Transition or operation 'Response' takes information of type 'Call' as input from the parent net and after further activity execution, eventually outputs information of type 'Call' back to the parent net.

Currently, the interface information at this operational process level is fairly crude. Provided and required interface information together with role function (service) and information exchange can be specified in greater detail at the design and architecture levels of abstraction. This is illustrated later in Appendix D.

A.4 Conclusions so far following Specification using Classic and Coloured Petri Nets (with Hierarchy)

Petri nets offer a mathematical, graphical modelling foundation with reasonably straightforward interpretation rules and a small range of modelling elements which can be adapted for use in different application domains.

In order to exploit this flexibility, an organisation needs to determine its objectives for using Petri nets as well as their intended audience in advance. Subsequently, best practice guidelines are essential in helping to realise these objectives. For example, an organisation may decide to use Petri nets in the analysis and design of large-scale system-of-systems. These nets would need to be understood by both skilled and unskilled Petri net practitioners. Based on the complexity issues encountered using flat classic and coloured Petri nets, at a minimum the organisation would require high-level Petri nets employing colour (type) and hierarchy.

Once these objectives had been established, systematic guidelines for net construction would need to be implemented to address: adoption of a standard net toolset; approaches to aid comprehension of nets (e.g. use of labelling convention for net elements, abstraction, toolset's colour palette, textual annotation, training in Petri nets and selected toolset); recommended approach to developing hierarchy; means of storing produced nets for future re-use and modification; maintenance of consistency with other modelling techniques used within the organisation (e.g. UML); and identification of a suitable hierarchy (again, this process would need to consider the objectives for use of Petri nets and involve domain experts in confirming the granularity of the abstraction).

So far, coloured Petri nets have been used to specify the telephone system at an analysis level of model abstraction (using an operational process to do so). A hierarchy was suggested using a functional decomposition approach for use within the model and facilitated by the toolset using port and socket places. Design and architecture levels of model abstraction are explored further in Appendix D.

Appendix B

B.1 Verification of the Telephone Process using a Coloured Petri Net

Both the non-hierarchical net of Fig. A.3 and the hierarchical net of Fig. A.4 were used in static and dynamic analyses for the purposes of comparison.

B.1.1 Dynamic Analysis (Simulation)

Again, simulation was used to verify the behaviour and logic of the specifications in Figs. A.3-A.4. For both nets it was noted that while checking multiple calls made to the receiver role, multiple tokens can accumulate on a place and are removed randomly. Prioritised ordering of their addition and removal would require definition of a queue place colour (type). A problem in the logic setting the line to the correct state, i.e. free or busy was detected in Fig. A.3. While aiming to include the correct logic for setting the line in Fig. A.4, an incorrect simulation halt was experienced due to a variable with an inappropriate colour (type) definition being used in an output arc. This meant that the variable could not be bound to a value needed to enable transition 'Amend_Line'. This error was corrected by associating the variable definition to the correct enumerated type colour (type). Iterative simulation runs helped finely tune the nets by the addition of guard conditions on transitions, e.g. 'New_CallMsg' on 'Dial_Number'. Simulation helped instill confidence that the interaction was terminating properly.

It was also noted that the parent net (Fig. A.4) provided an overview of exactly where in the model the simulation was executing. The subnet could then be selected for an overview of execution within its associated process. This was not such a significant issue in this small example but will be useful for tracking execution on nets of much larger system-of-systems. The net toolset also allows the modeller to select variable bindings during simulation or fully automate the simulation.

B.1.2 Static Analysis

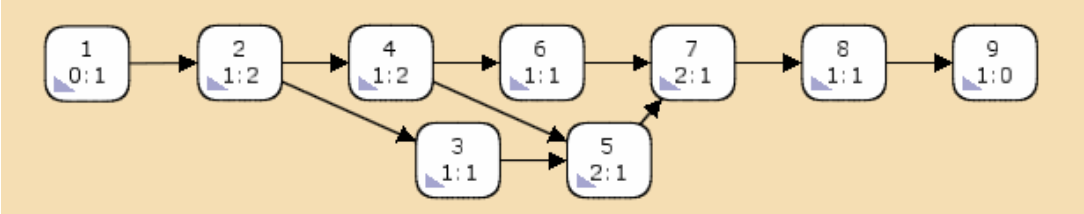
As dynamic analysis via simulation cannot guarantee that all possible execution paths of the process have been covered, static analysis of Petri nets is used to provide a more exhaustive, deeper level of verification over and above simulation alone. Static analysis using reachability tree or state space analysis was conducted to check for standard behavioural Petri net properties such as reachability, boundedness, home, liveness and fairness (depending on the Petri net tool used, it may also possible to construct non-standard property queries).

Static Analysis of non-hierarchical net

Figs. B.1-B.3 show the static analysis results for two initiated calls within the non-hierarchical net of Fig. A.3:

Reachability/State Space

Nodes: 9
Arcs: 10
Secs: 0
Status: Full



Scc Graph

Nodes: 9
Arcs: 10
Secs: 0

Fig. B.1 'Reachability (state space) and strongly connected component analyses'

Best Integer Bounds

Place	Upper	Lower
Hangup	*2*	*0*
Lift_Receiver	*2*	*0*
Receiver_Line	*1*	*0*
Response_From_Receiver	*1*	*0*
Try_To_Connect	*2*	*0*
Wait_On_Response	*2*	*0*

Best Upper Multi-set Bounds

Hangup	*1`Line_BusyMsg++1`Call_Answered*
Lift_Receiver	*2`New_CallMsg*
Receiver_Line	*1`Line_BusyMsg++1`Line_FreeMsg*
Response_From_Receiver	*1`Line_BusyMsg++1`Call_Answered*
Try_To_Connect	*2`New_CallMsg*
Wait_On_Response	*2`New_CallMsg*

Best Lower Multi-set Bounds

Hangup	*empty*
Lift_Receiver	*empty*
Receiver_Line	*empty*
Response_From_Receiver	*empty*
Try_To_Connect	*empty*
Wait_On_Response	*empty*

Fig. B.2 'Boundedness properties'

```
Home Markings                [9]

Dead Markings                [9]
Dead Transition Instances    None
Live Transition Instances    None

No infinite occurrence sequences.
```

Fig. B.3 'Home, liveness and fairness properties'

Fig. B.1 shows the results of performing reachability tree (state space) and strongly connected component analyses. The strongly connected components graph calculates the number of strongly connected components within the state space graph, i.e. components where each node has a path to any other node in the component. When there are less strongly connected components than state space nodes, infinite occurrence sequences can exist. This suggests that the net may not terminate. In this example, the full state space calculation has nine nodes and ten arcs and took less than a second. The strongly connected component graph calculated based on this state space has nine strongly connected components and ten arcs. This implies that the telephone call terminates.

Fig. B.2 shows boundedness properties for the net in Fig. A.3. The first part of Fig. B.2 shows the maximum and minimum number of tokens contained within each place. Places 'Hangup', 'Lift_Receiver', 'Try_To_Connect', and 'Wait_On_Response' always have between zero and two tokens and relate to the number of calls initiated within the process. Places 'Receiver_Line', and 'Response_From_Receiver' always have between zero and one token. This latter result indicates that incoming calls are essentially stalled on place 'Try_To_Connect' (highlighted by the maximum one token on the place, 'Response_From_Receiver') until 'Receiver_Line' obtains a 'Line_FreeMsg' token and enables transition 'Connecting'.

This was not the desired behaviour of the call process with initial marking of 'Line_BusyMsg' for the receiver. Instead, if more than one incoming call arrived for the receiver and the receiver's line was initially marked as busy, the process does not permit any of the incoming calls to connect (all terminate with 'Line_BusyMsg' on place 'Hangup'). 'Response_From_Receiver' should have been capable of hosting a maximum of two tokens. As it stands, the net in Fig. A.3 would need amendment to reflect this. In this way, integer bounds help reassure the process is working as intended.

The second part of Fig. B.2 provides details about the information held by the tokens at each place and confirmed the incorrect behaviour highlighted by the integer bounds. Considering upper multi-set bounds first of all, places 'Lift_Receiver', 'Try_To_Connect', and 'Wait_On_Response' can hold a maximum of two tokens with content always 'New_CallMsg'. Places 'Receiver_Line' and 'Response_From_Receiver' can both hold a maximum of one token. 'Receiver_Line' can only take one of the multi-set 'Line_BusyMsg' and 'Line_FreeMsg' as value. 'Response_From_Receiver' can only take one of the multi-set 'Line_BusyMsg' and 'Call_Answered' as value. Finally, place 'Hangup' can hold a maximum of two tokens with content from the multi-set 'Line_BusyMsg' and 'Call_Answered'.

From the lower multi-set bounds in Fig. B.2 it can be seen that all the places have a lower multi-set bounds of the empty multi-set. This means the markings of all the places do not remain the same. Again, given the initial marking used and the desired process behaviour, identical upper and lower multi-set and integer bounds (corresponding to a multi-set of 'Line_BusyMsg' and one respectively) would have been expected on place 'Receiver_Line'. In this way, multi-set bounds can be used to alert the user to incorrect operation of their process.

The remainder of the state space graph analysis can be seen in Fig. B.3 and relates to home, liveness and fairness properties. The process has a single home marking, a marking that can always be reached, M_9. It also has a dead marking (Fig. B.4), a marking with no enabled transitions, identical to the home marking of M_9. This marking corresponds to the state where results of the two placed calls have been received and no tokens are left at places 'Lift_Receiver', 'Wait_On_Response', 'Try_To_Connect', and 'Response_From_Receiver', i.e. the process terminates.

```
9:
Call_Business_Process'Lift_Receiver 1: empty
Call_Business_Process'Wait_On_Response 1: empty
Call_Business_Process'Try_To_Connect 1: empty
Call_Business_Process'Receiver_Line 1: 1`Line_FreeMsg
Call_Business_Process'Response_From_Receiver 1: empty
Call_Business_Process'Hangup 1: 1`Line_BusyMsg++
1`Call_Answered
```

Fig. B.4 'M_9 dead marking from state space graph'

As M_9 is a home marking it means the process can never reach a state from which it cannot terminate with the desired result.

Liveness properties also reveal that each transition is enabled by at least one reachable marking (no dead transition instances) and no transitions can become enabled again (no live transition instances).

In terms of the frequency of transition firing or fairness properties, the telephone process has no infinite occurrence sequence of transition firing. This means there is no non-determinism in the net and the dead marking is reachable.

Depending on the Petri net modelling toolset used, non-standard queries can be used to inspect the state space graph further. For example, there are several ways to find and output information associated with dead markings in the state space.

```
ListDeadMarkings();
```

Fig. B.5 'Pre-defined toolset query to output dead markings in state space graph to screen'

```
val DeadListing = SearchNodes
(
EntireGraph,
fn n => (length(OutArcs(n))=0),
NoLimit,
fn n => n,
[],
op ::
)
```

Fig. B.6 'SearchNodes query used to inspect state space graph and output list of dead markings to screen'

```
let
val fid = TextIO.openOut "SSAPhoneDeads.txt"
val _ = TextIO.output(fid, "List of dead markings: \n")
val _ = EvalNodes(ListDeadMarkings(),
fn n => INT.output(fid,n) )
val _ = TextIO.output(fid, "\nNumber of dead markings: ")
val _ = INT.output(fid,length (ListDeadMarkings()))
in
TextIO.closeOut(fid)
end
```

Fig. B.7 'Count and list of dead markings output to a file'

In Figs. B.5-B.7 above, a branching temporal logic is implemented within the toolset to permit analysis of state space graph marking or transition information. Using this mechanism, it is possible to combine pre-defined queries or construct new modeller-defined queries and undertake further checks related to model properties such as reachability and liveness. For example, the modeller may want to verify whether the designated dead markings are valid, i.e. the values of tokens on places involved in dead markings are as expected, or after reaching a certain place state another place state of interest can be reached, or that the receiver will not provide a response to the caller if a connect attempt has not been received.

Analysis of the calculated state space graph has enabled fairly exhaustive verification of the behaviour currently specified by the non-hierarchical net in Fig. A.3. The results of the static analysis shown in Figs. B.1-B.3 also highlighted that standard deadlock analysis only detects deadlock that is a result of logic and place state currently engineered into the net or when connection elements have been erroneously omitted (in this case, simulation may be able to alert a modeller to the deadlock).

By understanding the example's concept, it is known that a telephone call process could deadlock due to failure of system component(s) normally modelled at a lower abstraction level, e.g. communications hardware. This could mean that calls are not connected, disconnected, or no response reaches the caller. This potential for failure during the call process is not specified in the model at the current high-level of abstraction and would not be detected using static analysis.

Normally, process modelling focuses on the roles, resources, information exchange, control sequence and timing of a process, and does not specify the possibility of potential failure states. Instead, process modelling often makes the assumption that the underlying infrastructure is reliable. Petri net modelling of the telephone process and

the ability to conduct static analysis for deadlock detection helped to underline this fact. It also indicates that like the telephone process, processes involved in large-scale system-of-systems involve capture of an operational process as well as an information exchange protocol at different levels of abstraction.

Unless the potential for undesirable information exchange states is specified using a combination of net elements (tokens and types, firing rules, directed arcs, and activities), static analysis of Petri nets cannot identify the potential deadlock states in system-of-systems early development stages. In system-of-systems development, it is vital to specify the possibility of undesirable states as well as desirable states as early in the development cycle as possible so that they can be properly mitigated in later design stages. In this way, all associated terminal markings identified by static analysis can be checked at the operational process level of abstraction for validity using the non-standard queries discussed above.

Static analysis of the non-hierarchical net in Fig. A.3 has shown the telephone call process does terminate. The existing specification would benefit from amendment to coherently capture what the system should and should not do. Prior to incorporating any further specification amendments derived from static analysis of the flat net, static analysis of the hierarchical net was investigated.

B.1.3 Static Analysis and State Space Explosion

While a full state space graph was calculated for the basic telephone process of Fig. A.3, a known weakness of Petri nets is the complexity problem [57]. Even small sized process representations can have infinite reachable states (the state explosion problem). To alleviate this problem, methods are used to try and reduce the state space graph by focusing on its form (largeness avoidance) or a subset (largeness reduction).

Examples of avoidance methods include step-wise refinement of processes, i.e. use of abstraction and composition (calculating the state space graph based on subsets of the model), limiting the number of tokens on places (using the premise that if the process behaves as expected for a small number of tokens, it is likely to work with higher numbers), and restricting the values associated with place types. Reduction methods include condensing state space by exploiting symmetries present in processes and consideration of limited capability between nodes.

Largeness avoidance methods have been adopted as best practice and used prior to undertaking model-checking in the telephone example. The net in Fig. A.4 is used to investigate the effects of largeness avoidance techniques on the state space graph. A standard static analysis report is conducted on the hierarchical net of Fig. A.4.

```
Reachability/State Space
     Nodes:  88
     Arcs:   168
     Secs:   0
     Status: Full

Scc Graph
     Nodes:  88
     Arcs:   168
     Secs:   0

Dead Markings                  [88,87,86,85,84,78,77]
```

Fig. B.8 'Static Analysis for the hierarchical net'

Fig. B.8 shows the state space calculated for the hierarchical net of Fig. A.4. The state space graph consists of eighty-eight reachable markings and one hundred and sixty-eight transitions. As the much larger state space graph highlights, the hierarchical net of Fig. A.4 was developed to illustrate the benefit of hierarchy on net comprehensibility and scalability. Fig. A.4 also incorporates some corrections and extensions to logic identified from dynamic and static analysis of Fig. A.3 (and no inclusion of undesirable state specification as yet).

Again, a full state space graph was constructed. For a hierarchical net, the state space is normally larger than that of the equivalent non-hierarchical net. This is because the toolset 'flattens' the hierarchical net in order to calculate its state space graph. Hierarchy substitution transitions, and port and socket places contribute to the additional nodes in the state space graph. These overhead elements can be justified based on the scalability and increased readability they bring to the model.

Abstraction and compositional techniques

Jensen [90] hints that employing hierarchy and subnetting will help combat state space explosion. There is no existing work prescribing the benefits that hierarchy and subnets actually bring to largeness avoidance, scalability and comprehension in Petri net models of systems-of-systems. Where research has made use of hierarchy and subnetting, the reported benefits tend to be related to readability, scalability, re-use of coloured Petri net models, or performance analysis [77, 79, 82, 94, 95] in software systems. However, research from three sources was particularly relevant to hierarchy and subnetting and combating state space explosion.

First, in [52], Chukwuogo uses an object-oriented model transformation approach to address scalability and largeness avoidance in large-scale software applications. Initially, an application is modelled using UML diagrams and transformed into hierarchical coloured Petri nets for the purposes of model-checking. To combat the state explosion problem, Chukwuogo uses the information gained transforming the UML models to Petri net models to abstract out the detail of certain components during state space calculation.

Although Chukwuogo was able to observe reductions in duration and size of state space graph calculation using two small case studies, the method was not applied to a large-scale industrial software application and did not relate specifically to large-scale system-of-systems' applications. No mention of the concept of timing or reliance upon

network connectivity and communications was made in the work. By Chukwuogo's own admission, the method used did not follow or present a systematic approach, nor was there any inclusion of model-checking results at the individual module level, i.e. a bottom-up approach. The work appeared to be focused on addressing software systems using a top-down approach and untimed, hierarchical, coloured Petri nets.

Secondly, the results from Petrucci et al [91-93] considered the benefits modularity in nets (specifically Modular Petri nets) brings to reduction of the state space graph. Modular Petri nets are used to compose a model of a system by separating a flat net into modules. Fused transition points in the net are treated as module interaction points. Petrucci et al's work on compositional analysis was focused on exploring a module's local behaviour without considering all possible interleavings with the local behaviour of the other modules in the system model. In [91,93], local state space graphs are calculated per module in parallel with a global synchronisation graph for the system while in [92] the approach incrementally generates reachable states based on LTL\X properties.

Given that the majority of industrial Petri net applications rely on classical or coloured Petri nets and fused place points, an attempt was made by Petrucci et al in [93] to adapt the method to use fusion places. The result was to transform such a net into a fused transition net but no systematic approach for converting classic or coloured Petri nets was given. The research did indicate that best results were obtained for modules exhibiting strong cohesion and loose coupling, a main characteristic of constituent systems within systems-of-systems.

In the CPN Tools toolset, modularity (or subnetting) of system models is facilitated either through hierarchy and associated port and socket places or fused places (fused transitions are not supported although [97] did propose such an extension). Hierarchy permits an overview of a model's functional decomposition through a parent net and port and socket places. The transitions present in the parent net are substituted by more detailed nets shown on separate pages. In simulation, a modeller can track the location of the next enabled transition using the parent net. Fusion places can also be used to capture modularity but there is no explicitly associated parent net page. Each subnet on a page is independent and passes information to another subnet page using a set of places (fusion places). To mimic the abstraction depicted by the parent net of a hierarchical approach, a net can use the fusion places to pass (receive) information to (from) a subnet but the only way of associating the net at the higher abstraction level with the subnet at the lower abstraction level is via the labelling of the fusion places. Port and socket hierarchy makes the association with the lower abstraction level in a more explicit way.

An advantage of fusion places in modularity is the ability to share the same information between multiple processes. With hierarchy and port and socket places, if information needs to be passed from an interface to more than one component at a point in time, sufficient copies of the token need be deposited on the interface place for consumers to remove. Hierarchy's main advantage is enabling scalability via explicit abstraction. Detailed subnets can be copied and re-used (similar to a programming procedure) with each copy able to receive separate inputs and return separate outputs (compare with parameterised procedure calls). Use of either abstraction technique depends on the modelling context.

Petrucci et al's largeness reduction method was noted for future research in conjunction with the largeness avoidance techniques employed within CPN Tools. In Petrucci et al's work, there was no explicit treatment of modularity as a means of largeness avoidance or aid to comprehension. This is explored later in section B.1.5.

The third work of interest comes from Bonnefoi et al [98, 99] and deals with design and analysis of intelligent transport systems using UML and Petri nets. Approaches [98] consider integration of continuous and discrete characteristics of an intelligent transport system into a discretised Petri net model (coloured Petri nets are transformed to Symmetric nets in order to take advantage of the symmetries in the system and reduce the state space) and possibility of transforming UML models to symmetric nets [99]. [99] discusses UML component diagrams as a means of modularising the system by describing components and their interfaces to one another. Both the discretisation and use of symmetric nets as a means of largeness avoidance and reduction are worth bearing in mind for future system-of-systems research.

B.1.4 Largeness Avoidance by Abstraction

Based on the work of Chukwuogo [52], the effect of re-calculating the state space graph based on abstracting out details of modules within a net designed to capture the specification of a system at a certain level of abstraction (or viewpoint) is investigated. It is anticipated that a module (or component) in this case can refer to both an activity in a net employed at the conceptual (operational process) level or a design component in a net employed at the design and architecture levels of abstraction. In this section, detail is abstracted out of the main activities and their associated processes in the telephone net currently described at the conceptual level of abstraction.

Initially, detail associated with the caller role's 'Make_Call' transition is abstracted out. No decomposition of the 'Make_Call' transition from the parent net to a separate subnet was included. Instead, a minimal set of net elements were used to ensure that the provided and required interfaces of the 'Make_Call' transition consumed and produced the same information as before for the parent net. The process detail associated with the Receiver role's 'Connect_Call' transition remained the same. The revised parent net showing abstraction of the caller role is shown in Fig. B.9.

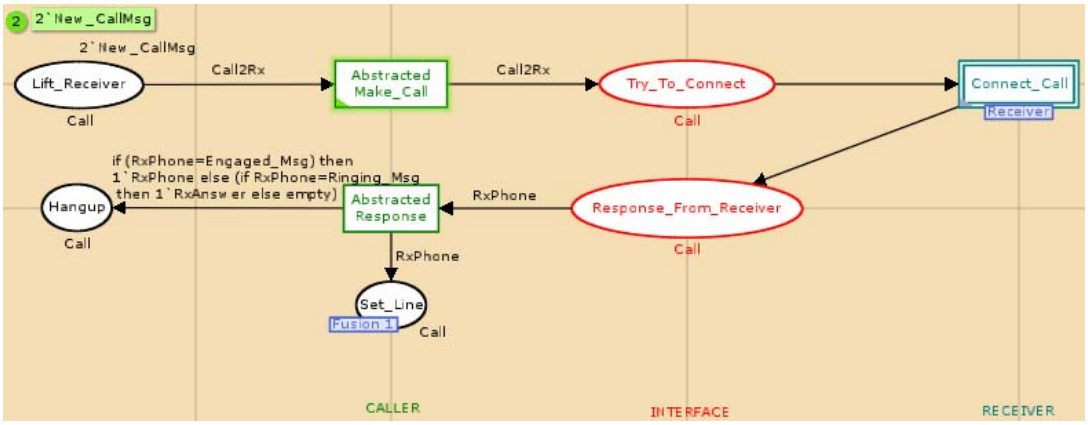


Fig. B.9 'Abstracted caller role used for re-calculation of state space graph'

From Fig. B.9 it can be seen two transitions, 'Abstracted Make_Call' and 'Abstracted Response' are needed to facilitate the provided and required interfaces for the 'Make_Call' transition decomposed in the original hierarchical net of Fig. A.4. The re-calculated state space graph results for Fig. B.9 are shown in Fig. B.10.

```
Reachability/State Space
     Nodes:  44
     Arcs:   68
     Secs:   0
     Status: Full

Scc Graph
     Nodes:  44
     Arcs:   68
     Secs:   0

Dead Markings                    [44,43,42,41,40,38,35]
```

Fig. B.10 'Static analysis for the abstracted caller role within the hierarchical net'

Fig. B.10 shows that for the original hierarchical net, by considering one of its two components at an abstracted level and one at a detailed level, the state space graph has halved in size (in terms of time, the original hierarchical net and the abstracted net calculations took less than one second as reported by the toolset). The technique was carried out again. This time the detail was abstracted out of the process associated with the receiver role's 'Connect_Call' transition and the caller's 'Make_Call' transition reverted back to the decomposition presented in the original hierarchical net. The revised parent net showing abstraction of the receiver role is shown in Fig. B.11.

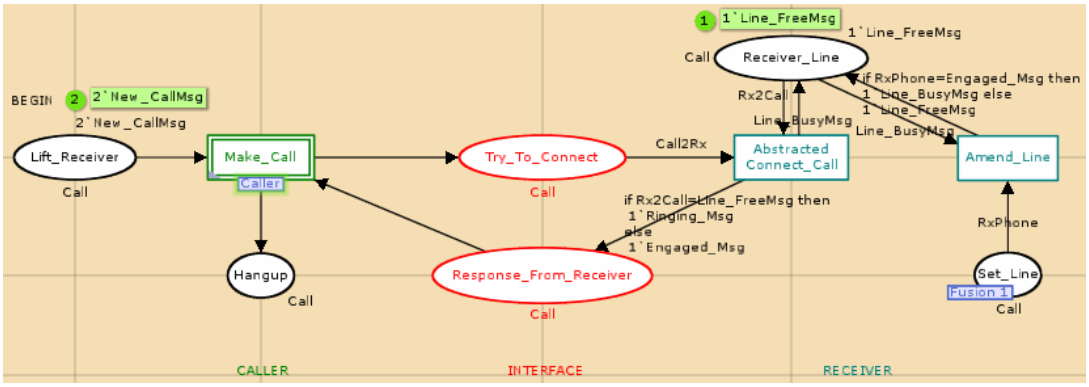


Fig. B.11 'Abstracted receiver role used for re-calculation of state space graph'

From Fig. B.11 it can be seen two transitions, 'Abstracted Connect_Call' and 'Amend_Line' are needed to facilitate the provided and required interfaces and telephone line reset for the 'Connect_Call' transition decomposed in the original hierarchical net of Fig. A.4. The re-calculated state space graph results for Fig. B.11 are shown in Fig. B.12.

```
Reachability/State Space
     Nodes:  77
     Arcs:   146
     Secs:   0
     Status: Full

Scc Graph
     Nodes:  77
     Arcs:   146
     Secs:   0

Dead Markings                    [77,76,75,74,73,68,62]
```

Fig. B.12 'Static analysis for the abstracted receiver role within the hierarchical net'

Again, Fig. B.12 shows that for the original hierarchical net, by considering one of its two components at an abstracted level and one at a detailed level, the state space graph has reduced in size (in terms of time, the original hierarchical net and the abstracted net calculations took less than one second as reported by the toolset). This time the abstraction benefit is not as great due to there being a greater number of core net elements required in the abstracted net from the decomposed 'Connect_Call' subnet (mainly from the telephone line reset logic). Before drawing any conclusions the effect of a compositional approach to state space calculation is now considered.

B.1.5 Largeness Avoidance by Composition

The abstraction technique used above takes advantage of a top-down, functional decomposition approach and aims to remove one subnet's elements from the state space while maintaining the overall structure or composition of the net. Another technique investigated also takes function into account but does so in a compositional or bottom-up way. Unlike Petrucci et al's largeness reduction approach to the state space problem using modularity, the aim of the modular approach is largeness avoidance. Where Chukwuogo [52] used hierarchy within nets to identify components to abstract the detail from, the work did not report on model-checking individual components. The interface information within the telephone system is used to derive its constituent modules (in this case, processes). These are model-checked individually.

The parent or 'overall structure' net specifies the order of interface usage as well as the activity execution sequence within the processes. In the telephone example, there are two distinct interfaces specified ('Try_To_Connect' and 'Response_From_Receiver'). One is provided by the Receiver component, and one by the Caller component. Three processes and their constituent activities use the interfaces. Two processes are associated with the Caller component and one process with the Receiver. The parent net specifies that one of the Caller's processes uses the Receiver's connecting operation first. The Receiver's process accepts the incoming call and uses the Caller's response operation. Finally, the second process associated with the Caller deals with the Receiver's response. Each of these three processes uses the interfaces to send and receive information. As long as these interfaces are adhered to, the component processes can be changed or substituted without requiring changes to the interfaces or other components.

In the compositional approach to static analysis, the three processes are considered in the order specified by the parent net. The Caller call initiation process is first in sequence and the parent net of Fig. B.13 highlights the sequence.

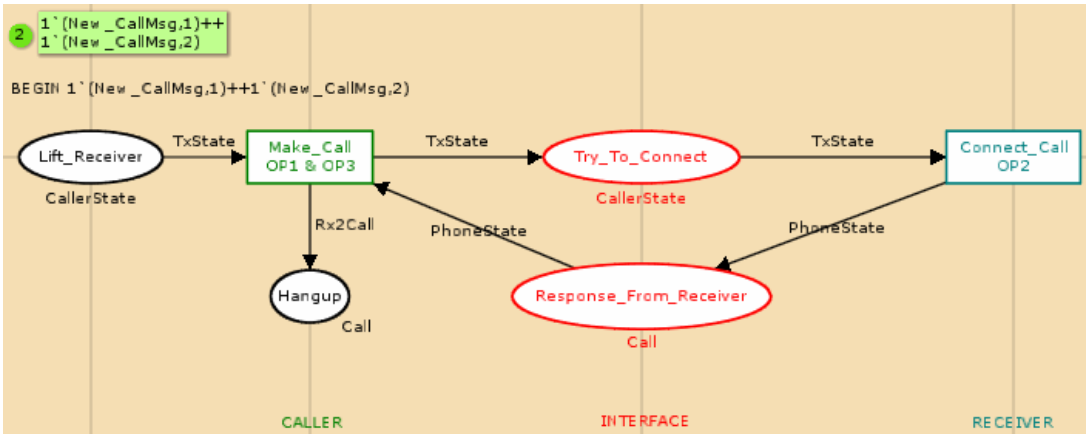
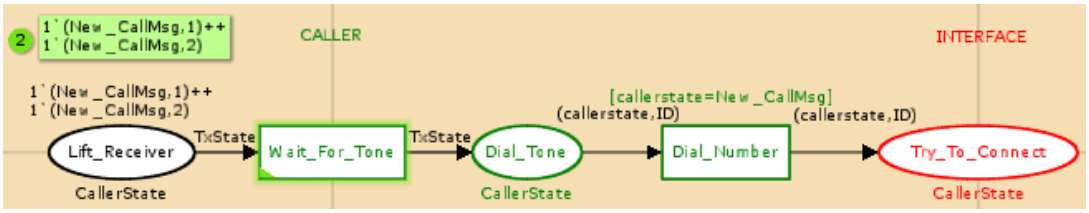


Fig. B.13 'Parent net showing control sequence and operation ownership'

The trigger input of this process is lifting the telephone receiver. The output is information required by the 'Try_To_Connect' interface place. The aim of the approach is to maintain the required inputs and outputs of these three processes as if they were still part of one whole net so that static analysis can be informed and conducted on each. For the first process, the initial marking was an attempt to model two calls so two tokens were used in the initial marking. Fig. B.14 shows the first process subnet and state space graph results calculated for this subnet.



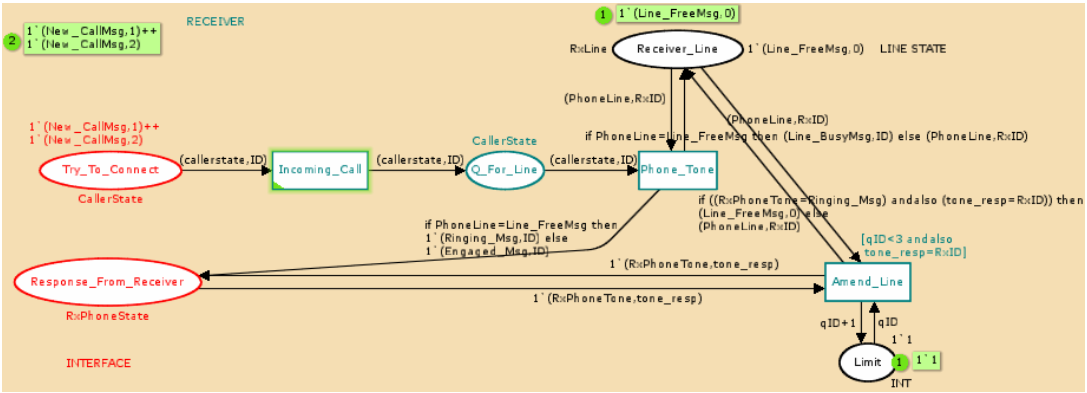

```
Reachability/State Space
     Nodes:  9
     Arcs:   12
     Secs:   0
     Status: Full

Scc Graph
     Nodes:  9
     Arcs:   12
     Secs:   0

Dead Markings                [9]
```

Fig. B.14 'Static analysis for the Caller component initiate call process in the compositional approach'

The output from the first process in the untimed net is simply the same tokens on output interface place 'Try_To_Connect' as those used in the initial marking. These were manually entered as the initial marking of the next process in the sequence, the Receiver component's respond to call process (Fig. B.15).



Reachability/State Space

Nodes: 19
Arcs: 24
Secs: 0
Status: Full

Scc Graph

Nodes: 19
Arcs: 24
Secs: 0

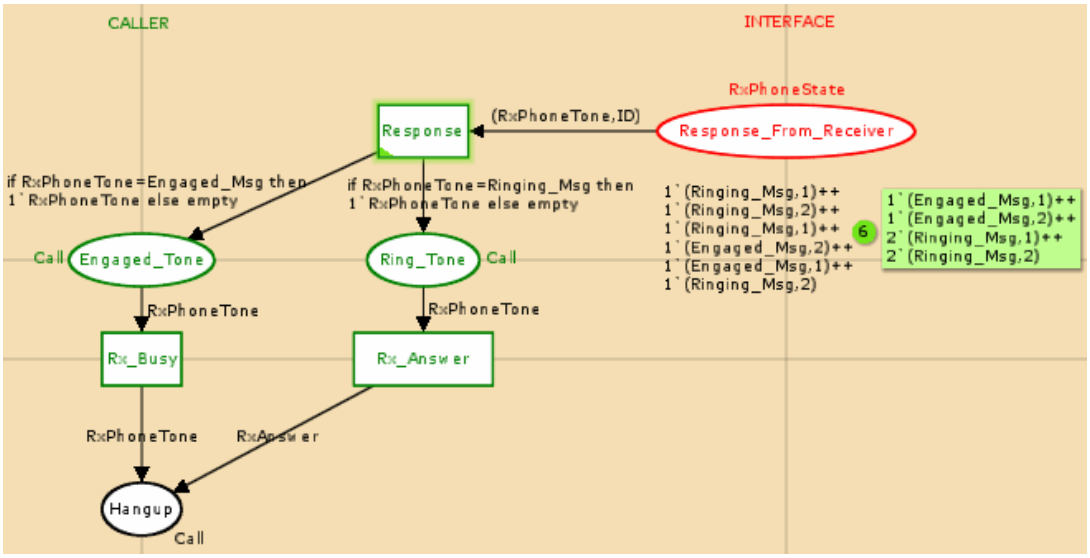
Dead Markings [16,18,19]

Fig. B.15 'Static analysis for the Receiver component respond to call process in the compositional approach'

The static analysis results for this subnet are shown at the bottom of Fig. B.15. Three dead markings were obtained. For each of these dead markings, the tokens on output interface place 'Response_From_Receiver' could have the following values:

1. 1`(Ringing_Msg,1)++1`(Ringing_Msg,2)
2. 1`(Ringing_Msg,1)++1`(Engaged_Msg,2)
3. 1`(Engaged_Msg,1)++1`(Ringing_Msg,2)

All three were manually input into the third and final process, the Caller component's process that handles responses, as shown in Fig. B.16.



Reachability/State Space

Nodes: 480
Arcs: 1592
Secs: 0
Status: Full

Scc Graph

Nodes: 480
Arcs: 1592
Secs: 0

Dead Markings *[476,477,478,479,480]*

Fig. B.16 'Static analysis for the Caller component call response process in the compositional approach'

Following inspection of the five dead markings of Fig. B.16, using the three pairs of tokens suggested by process two as the initial marking of process three in model-checking is not the correct way to obtain the true range of possible behaviour from two initiated calls in the telephone system. As it stands, the initial marking in Fig. B.16 represents six initiated calls. Instead, each of the possible pairs output from process two should be input into process three separately as an initial marking and model-checked. Composition of the dead markings obtained from the three separate model-checks will provide the complete range of behaviour for two initiated calls in the telephone system. This is shown below:

1. *1`(Ringing_Msg,1)++1`(Ringing_Msg,2)* used as first initial marking of process three.

Model-checking found three dead markings. Each dead marking's 'Hangup' place had the following markings:

2`Call_AnsweredMsg
1`Not_AnsweredMsg++1`Call_AnsweredMsg
2`Not_AnsweredMsg

2. *1`(Ringing_Msg,1)++1`(Engaged_Msg,2)* used as second initial marking of process three.

Model-checking found two dead markings. Each dead marking's 'Hangup' place had the following markings:

1`Engaged_Msg++1`Call_AnsweredMsg
1`Not_AnsweredMsg++1`Engaged_Msg

3. *1`(Engaged_Msg,1)++1`(Ringing_Msg,2)* used as third initial marking of process three.

Model-checking found two dead markings. Each dead marking's 'Hangup' place had the following markings:

1`Engaged_Msg++1`Call_AnsweredMsg
1`Not_AnsweredMsg++1`Engaged_Msg

Combining the markings obtained from the three separate pairs of initial markings to process three gives:

1. 2`Call_AnsweredMsg
2. 1`Not_AnsweredMsg++1`Call_AnsweredMsg
3. 2`Not_AnsweredMsg
4. 1`Engaged_Msg++1`Call_AnsweredMsg
5. 1`Not_AnsweredMsg++1`Engaged_Msg

These markings are the five possible outcomes from the telephone process when two calls are initiated.

It should be noted that the same compositional approach was attempted for larger numbers of initiated calls in the first process. Due to the very manual nature of the compositional approach so far and high numbers of potential inputs obtained for subsequent stages, a current weakness of this approach is the volume of values the modeller has to process.

B.1.6 Summary of Largeness Avoidance Techniques

Using a top-down approach the main functions and their underlying processes were identified. Static analysis was performed on the developed model and it was noted that state space explosion tended to be alleviated when small numbers of initial tokens were used. From this analysis insight was gained into the correct behaviour of the model. Detail was then abstracted out of each main function in turn (the abstraction approach) and static analysis performed. This lowered the state space graph size substantially (the time for two initiated calls took less than one second).

The top-down approach and information gained abstracting out the detail of each main function helped to identify the individual components at the lowest abstraction level to analyse in a compositional approach. Use of existing nets have not been considered in this thesis or where the compositional approach could help a modeller compare modules identified using the top-down method with those derived from existing systems or previous modelling attempts. Substantial state space graph calculation size benefits were noted using the compositional approach (as for the abstraction approach, reported calculation time was less than one second).

With the compositional approach, the modeller needs to consider each process and its inputs and outputs carefully. This is beneficial as it promotes further verification within the modules of the overall model. For example, following static analysis of the second process in the telephone system, the third possible marking, 1`(Engaged_Msg,1)++1`(Ringing_Msg,2), on place 'Response_From_Receiver' could reflect undesirable behaviour in the model. As this is produced as a potential outcome from process two, the second call has overtaken the first at a point in process two resulting in the first call receiving an engaged tone. The modeller may then choose to amend the model accordingly to eradicate this outcome.

In this way, the modeller can gain understanding of the behaviour of subnets and the overall net in stages, with the ability to use subsets of outputs of each process as inputs into the next, examining, understanding, correcting and amending the behaviour of the model as necessary. From the telephone example, the hierarchy used within the model was based on functional decomposition with functions implemented by components. As with systems-of-systems, the telephone system components are integrated with one another via communication at well-defined network interfaces. The components or subnets at the lowest level of abstraction are likely to contain the logic controlling the behaviour of the system. Ideally, these modules will form the basis of an organisational repository for future re-use and evolution for other systems. These modules could then be used in a bottom-up approach to development by specifying a control sequence for their integration. A parent net can be developed with abstract activities based on the function of the more detailed existing nets using the subnet's interface places as the means to perform the linking between the different abstraction levels.

Unlike static analysis of a full hierarchical model (or even a hierarchical model using the abstraction largeness avoidance technique), a modeller has the ability to narrow down the scope of analysis by examining a module in isolation. The modeller does not have to inspect a state space graph of the entire model to trace how particular dead markings that may or may not relate to the behaviour of the module have been reached. Where the entire model is non-trivial, the state space graph is likely to consist of a high number of nodes (even using the abstraction largeness avoidance technique) and manual inspection will not be a straightforward task. A compositional approach to analysis also allows a modeller unfamiliar with a model at the lowest abstraction level to investigate and verify its behaviour at its input and output interfaces prior to its integration with other modules. It may also be of benefit in promoting understanding of models to unskilled practitioners.

A current weakness of the compositional largeness avoidance technique occurs when greater numbers of initial markings are used or there are large numbers of modules to

integrate (as would be likely in large-scale systems-of-systems). Currently the approach involves much manual effort. Both automation and a model management approach would be highly recommended. It may be possible to use functional decomposition to analyse modules as part of functional groupings (and divide up the groupings between modellers) prior to their integration into the overall system-of-systems model.

Based on the investigation, use of the two approaches in conjunction with one another and with full hierarchical analysis (where this is possible) is recommended. Both largeness avoidance approaches (and use of nets to model large-scale systems) rely on a suitable hierarchy being in place. With the telephone system, top-down functional decomposition is used at conceptual, and design and architecture levels of model abstraction to modularise and provide hierarchical structure within a model. Given system-of-systems' reliance on communication between components at defined network interfaces, these communication interfaces form the boundaries between the hierarchies of components in the system-of-systems. In the model and the real-life system-of-systems, hierarchy and interfaces are key enablers of scalability and verifiability. For a net to capture a system-of-systems, the concepts of hierarchy and interfaces must be realisable within the net. Equally, for a net to be verifiable, there needs to be a way to analyse its hierarchy of modules and minimise the risk of state space explosion. The two approaches discussed above may be able to contribute in this area.

The abstraction approach can be used to remove the underlying detail from one or more components of the parent net and analyse the overall control structure and behaviour of the model. The compositional approach can be used to verify individual components at the lowest abstraction level identified by the top-down functional decomposition on an individual basis. The compositional approach can also be used with existing models, i.e. a bottom-up approach. A potential future area of research would be comparison of existing component nets and those identified using a top-down approach. Best practice approaches related to management of verification of large-scale systems models will be essential for dividing up the modelling workload.

Together with the abstraction and compositional techniques described above, other largeness avoidance techniques that can be readily employed with the toolset involve restricting potential values of types defined for places and token volumes. In the analyses, colours (types) were defined with finite values in the telephone example. For example, colour 'Call' was declared as an enumerated type with seven possible values. Variables defined as this enumerated type were bound during dynamic and static analyses to a value from a small finite set. The state space analysis was calculated using an initial marking of two tokens of type 'Call'. Given there are no loops in the nets, this essentially bounded the net to a maximum of two tokens per place.

These largeness avoidance techniques will be applied in the larger-scale case studies. They are also explored further in Appendix D during the analysis of nets at design and architecture abstraction levels.

B.2 Conclusions so far following Verification using Coloured Petri Nets

Coloured Petri nets with hierarchy provide a modeller with a specification language consisting of places, place types, directed arcs, transitions, abstraction and rules of interpretation for the net elements. These rules and elements can be described mathematically. Coloured Petri net models combine graphical notation and textual annotation (the degree of textual annotation support is toolset dependent) to capture what a system should do. It is anticipated that a modeller would then use this formal language to output a model regarded as a primary point-of-reference by other people involved with the system lifecycle. As the people involved will include skilled and unskilled Petri net practitioners alike, it is vital that this model be precise, correct, readable and understandable.

Unlike UML, system specification is not separated across different types of diagrams. Both syntax and semantics of Petri nets are described mathematically. To a skilled Petri net practitioner, the single net model in Fig. A.3 specifies information state, activity, and control sequence related to a telephone call's operational process and information exchange protocol. To an unskilled practitioner, the net in Fig. A.3 may well be very difficult to read and comprehend due to unfamiliarity with the underlying rules of interpretation, minimal textual annotation, default use of toolset colour palette, failure to identify input parameters, failure to identify relations between inputs and outputs, failure to map between net and domain, failure to see how new features could be added to the net, and the low variety of symbol elements.

The fact that a telephone system concept is being specified by the net can only be inferred through adequate use of labelling of places and transitions (some toolset's permit enhancement to a net's graphical notation via pictorial inclusion and animation). There may still be ambiguity regarding the meaning of the net as a result of the labelling convention used unless an organisation-wide convention is adopted.

The modeller using nets needs to understand the problem domain being specified so that there is accurate capture of operational process and information exchange protocol. This could be facilitated through use of domain experts, access to tried and tested models at higher and lower abstraction levels (e.g. Petri net or UML), and operational guidelines or documentation. It was useful to begin by constructing a very basic net in terms of functionality, and use dynamic and static analyses to increase confidence in its correctness before adding further information.

Simulation was used as an initial means to check the logic of the net. Petri nets were found to be useful knowledge elicitation tools in their own right for capturing the information required. Not only can nets be interactively stepped through, unskilled practitioners (e.g. domain experts), can participate with the modeller in their verification. Simulation helped highlight incorrect logic on transition output arcs, unsuitable variable bindings where the destination place type declaration did not include a required value (and simulation execution effectively halted), and the need for the modeller to exercise care in their use of unique variables in logic used on transition output arcs.

Once errors or amendments detected using simulation have been corrected, static analysis can be performed on the net to perform more exhaustive verification on the

net and its behaviour. Correct and incorrect deadlock states, and correct process logic can be examined. Desirable reachability states can be confirmed using branching logic queries (a toolset-dependent feature). The process of formulating queries and examining results from static analysis is a highly beneficial activity. In order to verify the results from the model-checker, the modeller has to understand the concepts behind the analysis so they can trace execution paths to terminal markings and reason why the places in terminal markings have certain markings. This leads the modeller to have increased insight into the logic used within their net, insight that is not necessarily gained using UML models (static or executable). Once static verification is undertaken, there can be greater confidence that the resultant net accurately and correctly describes what the system should do, ready for evolution in subsequent system lifecycle stages.

As indicated in sections A.3.2, B.1.4 and B.1.5, informed use of hierarchy may be able to help alleviate the state space explosion problem in model-checking (specifically, use of model viewpoints or abstraction levels, hierarchy within models, and component abstraction and composition). This will be essential for the deeper verification of large-scale system-of-systems. Hierarchy employed in this way can also facilitate scalability and specification of a system-of-system's operational processes and design and architecture components. In addition, careful employment of graphical enhancement, further textual annotation and a toolset's colour palette should mean nets can be used as primary points-of-reference in the system-of-systems lifecycle by both skilled and unskilled net practitioners.

As demonstrated by the telephone example, static and dynamic analyses of a net would be very much an iterative part of the system development process.

Appendix C

C.1 Validation of the Telephone Process using a Timed Coloured Petri net

Using the simple telephone example in Appendices A-B, it has been shown that classic and high-level Petri nets can be used to specify the behaviour of a system (albeit with varying degrees of success). To capture the efficiency or performance of a system and facilitate validation of its design, time-dependent actions such as timeouts, processing delays or deadlines are essential. As well as efficiency specification, time-dependent actions also enhance a system's behaviour specification in terms of correctness. Activity ordering alone is insufficient to capture overall system behaviour precisely. Tokens representing information in larger-scale systems will be processed according to the time they entered the system, time involved in their consumption and generation, and involvement in delays and transfer failures. Timing will be needed to specify the ordering multiple tokens receive over and above any activity sequence they experience. Timing information may need to be approximate, exact or both depending on the stage of development of the system. Classic Petri nets only include a basic concept of time in that actions (transitions) follow a particular execution order from an initial marking. Petri nets have been extended to incorporate the concept of time via their places, transitions, tokens, arcs or a combination of these, for example [80, 83].

Petri nets are deterministic timed nets if the delay is known, or stochastic timed nets if the delays are random, or deterministic and stochastic timed nets if a combination of fixed and random delays are present. In stochastic nets, firing time is associated with each transition indicating the delay from when the transition is enabled until it fires. Usually, the transition with the minimum remaining firing time affects the next marking of the net. Following this marking update, each newly enabled timed transition obtains a delay from the delay distribution and each timed transition enabled in the previous marking (and still enabled in the current marking), keeps its remaining delay. Transitions disabled in the current marking lose their remaining delay. Common stochastic Petri net models are by [51] and [49].

Deterministic and stochastic nets contain immediate transitions (when enabled, fire without delay), stochastic transitions (when enabled, fire after some delay sampled from a distribution), and deterministic transitions (when enabled, fire after a constant delay). Enabled immediate transitions have firing priority over enabled timed transitions. Multiple enabled immediate transitions should be specified with firing probabilities to resolve firing conflict.

This thesis uses high-level, timed coloured Petri nets. Jensen [80] extended coloured Petri nets with timed coloured Petri nets. With these nets a global clock is introduced for the net model. The state of a timed coloured Petri net consists of a marking and the global clock time. Timed coloured Petri nets can contain both timed and untimed coloured tokens. Timestamps are controlled by initial marking, transition or output arc expressions where discrete and probability distributions can be used to define the time taken for a transition to fire. Timestamps allocated to the tokens must be less than or

equal to the current model time in order to be removed. In this way, timed transitions represent the time taken by the system to perform a given task. Transitions known as immediate transitions can also fire in zero time.

CPN Tools implements timed coloured Petri nets but supports deterministic and stochastic model behaviour via discrete and continuous functions. However, in order to calculate a state space graph, models have to be discretised, i.e. evaluated to see whether they can be made deterministic and finite. Based on this, CPN Tools has a bias for stochastic behavioural support via simulation rather than static analysis. Unless infinite models are made finite, static analysis of them is intractable. For support of deterministic and stochastic Petri nets, an alternative toolset would need to be adopted. As this thesis is concerned with large-scale, discrete event system-of-systems where their behaviour is (ideally) deterministic and terminating, use of CPN Tools is maintained. Even at the system-of-systems architecture level of design, events specified will be of a discrete rather than continuous nature. Continuous specification will be required in physical monitoring at a lower level of (component system) detail. However, prioritisation between events should be enabled at a system-of-systems level.

Several research initiatives have been undertaken using timed Petri nets. These include: Christensen et al in [82] make use of timed coloured Petri nets to optimise the performance and capacity of a web server; Van der Aalst et al in [83] use interval timed coloured Petri nets to study rail time-tabling; Bulitko et al in [84] use time interval Petri nets to analyse real-time damage limitation on ships; Van der Vorst et al and Makajic-Nikolic et al use timed coloured Petri nets to examine supply chains [86, 87]; Dahl et al consider interval timed coloured Petri nets in penetration testing [85]; Kwantes uses timed coloured Petri nets to analyse a banking clearing process [88]; and Schomig et al use stochastic Petri nets to model business processes in [89].

All the approaches [82-89] are useful in providing guidance on development of performance models and contributing to parts of the validation of system-of-systems. Their approaches deal with continuous management, proactive and retrospective analysis of physical products but do not take into account the unique characteristics of system-of-systems. For development of system-of-system performance models, the budget-holder should be considered in the process, as well as the concurrent and peer-to-peer (equal) nature of the provided and consumed functions (services) realised by the processes and their components. In system-of-systems, intangible services are realised using tangible resources in different environmental locations. The perceived quality of service of these intangible functions arises from the efficiency of the processes. Insight should be offered into assessment of different ways of realising these intangible services at an early development stage and maintained throughout the system-of-systems development stages. For example, an assessment model should help to answer whether an optimal combination of activities that leads to a reduced service response time exists.

Currently, the example telephone system has been specified at an operational process (conceptual) model level of abstraction which is the first stage in large-scale, system-of-systems development. Typically, this viewpoint is useful for gaining a shared understanding of the problem concept and the intended technical and non-technical audience would include analysts, developers and domain users. The introduction of

timing information to the telephone example at this abstraction level would help enable domain users and developers to decide whether the modelled concept was efficient and adequate for input into the design stage. Assessing performance would involve checking if the modelled processes reached desirable behaviour states (including recovery from undesirable states) within realistic time and resource estimates. Improving the efficiency of the process means looking for new or different ways to realise desirable behaviour within defined time, cost and quality parameters.

To examine alternative options for the process, it was necessary to determine the time, cost and quality performance indicators for the telephone service and implement these in the model. Examples of these indicators include call fulfilment time, communications resource usage (and related costs), and call fulfilment time within a certain time limit. The natural inclination would be to minimise the first two and maximise the last one but all three need to be taken in context with the strategy of the organisation(s) involved. In the case of system-of-systems, it is essential to understand the economic and operating environment for system-of-systems services, and which (if any), of the performance indicators carries more weight than the others.

As highlighted in the telephone example, performance indicators could include average call fulfilment time, reduction in fulfilment time for priority calls, and average call duration. From the caller and receiver, i.e. as customers of a telephone service provider point-of-view, reduced cost and higher quality may be the most important performance parameters in relation to their service provider. From a receiver's point-of-view, if they are a business service-provider themselves, they are likely to be most concerned with time and quality parameters relating to customers trying to connect. From a telephone service provider's point-of-view, they need to appreciate who they are providing the service for so they can tailor their service provision strategy accordingly and use it to influence the lifecycle of their call systems.

In the example, as the system was specified originally from the receiver's point-of-view, it is assumed that they are a business service provider concerned with higher quality and lower time parameters for customers. This means the receiver would be interested in minimising call fulfilment time and maximising connection on first attempt, i.e. answered calls for its customers. As it stands, the operational process net from Fig. A.4 is modelled from the viewpoint of the receiver. Although operational process nets can be used in performance analysis, further adaptation is usually required to support the viewpoint, performance indicators and strategy of the viewpoint concerned.

Dynamic analysis (simulation) is used in conjunction with timing in the net. Timing delays were introduced on the source and various intermediate places within caller and receiver processes using an exponential distribution to represent random call placement and delays between each activity in the call process. From initial simulation runs, a record declaration was needed for each call in order to store the model time at which the 'Dial_Number' activity executes. This was viewed as the start of the attempt by the underlying communications infrastructure to connect the caller with the receiver. Again, a time delay was introduced here to the record token to represent the delay of the underlying communications infrastructure.

Once the receiver's line is checked for readiness to receive the incoming call, another delay is triggered to represent receipt of the relevant dial-tone from the underlying communications infrastructure back to the caller. The time at which the 'Response' activity executes is viewed as the end of the attempt by the underlying communications infrastructure to connect the caller's call. This model time is also stored in the call record (shown in Fig. C.1).

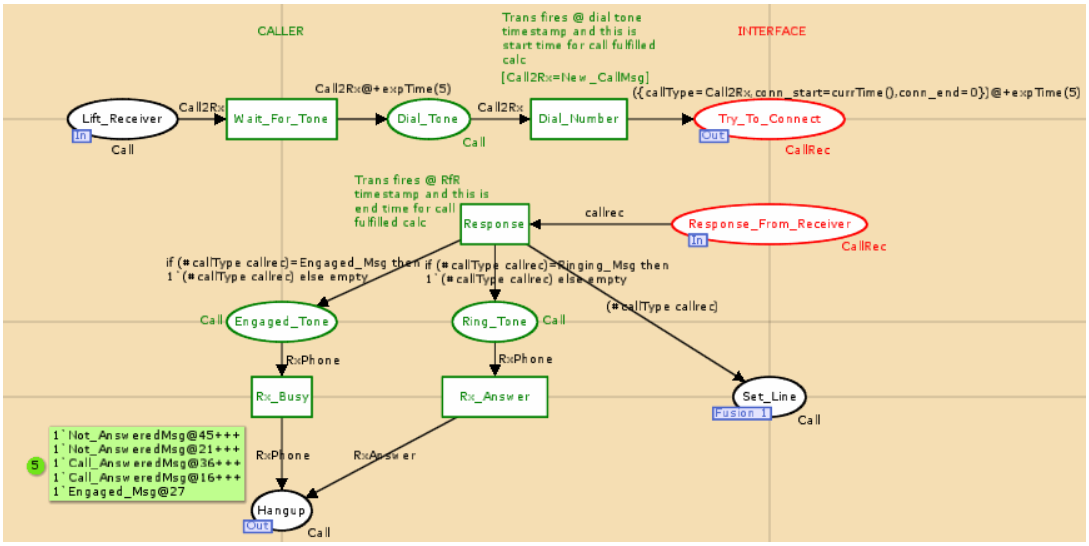


Fig. C.1 'Caller subnet within performance analysis net of telephone process'

The net toolset permits extraction of data from certain places or transitions during simulation of the operational process. This information could be used to calculate timing delays for processes associated with particular components, individual or groups of activities, and the process as a whole. In the telephone example, data collection was carried out on the 'Response' transition. Each time the transition fired, the call record was examined and the recorded model start time for the call's connection attempt was subtracted from the model time at which 'Response' fired. The result was deemed to be the connection fulfilment time for a particular call, i.e. the time between the caller completing dialling of the receiver's number and hearing an engaged or ringing tone.

In this way, the net toolset could be used to simulate a number of calls being placed from callers to the same receiver (in this case five call tokens were set up as the initial marking of the 'Lift_Receiver' place) and to obtain statistics based on the timings obtained for each call made.

Untimed statistics					
Name	Count	Sum	Avrg	Min	Max
Call_Fulfillment_Monitor	5	50	10.000000	0	30

Simulation steps executed: 35
Model time: 45

Fig. C.2 'Telephone net standard performance report for configured data collection'

Call fulfillment i.e. engaged tone or ringing tone heard
==========
{callType=Ringing_Msg,conn_start=16,conn_end=16}
{callType=Ringing_Msg,conn_start=14,conn_end=21}
{callType=Engaged_Msg,conn_start=19,conn_end=27}
{callType=Ringing_Msg,conn_start=31,conn_end=36}
{callType=Ringing_Msg,conn_start=15,conn_end=45}

Fig. C.3 'Telephone net text file report for Response transition data collection'

#data counter step time
0 1 11 16
7 2 19 21
8 3 22 27
5 4 28 36
30 5 33 45

Fig. C.4 'Telephone net log file report for Response transition data collection'

Figs. C.2-C.4 show a sample of the net toolset's output in relation to performance analysis of a net. Fig. C.2 is a standard performance report produced for any data collection configured within a net. It can be seen that there are five types of statistics shown and these statistics can be configured by the modeller. Figs. C.3-C.4 detail the content of the call record following execution of the 'Response' activity. Fig. C.4 details the execution steps of the 'Response' transition during the simulation, the corresponding model time and call fulfilment duration results for each 'Response' execution (under the data column). Fig. C.2's model steps value is two higher than Fig. C.4's last recorded step value of thirty-three. This is due to there being two final transition executions following the final execution of transition 'Response'.

Using data collection, for this particular simulation it can be seen that the five calls took a total of fifty time units for their call connections to be fulfilled at an average of ten time units per call. From the receiver's efficiency concerns, they can then decide

how to improve on this average. For example, they may choose to look at the effect of additional resources on fulfilment times, e.g. a second physical line. This can be implemented in the model by setting a new initial marking for 'Receiver_Line' of two and re-running simulation. A priority queuing mechanism could also be implemented in the model to reduce the number of connection attempts resulting in receipt of an engaged tone or unanswered ring tone.

In addition, the net toolset can be used to set up automatic simulation runs with a different selection of parameters per run. For example, simulation runs could be set up for the telephone process where calls are initiated in the system until a certain number of calls is reached, different random distributions can be used per run, the number of available lines available to the receiver can be amended, further data collection points can be implemented to examine duration of calls answered by receiver or the number of unanswered or engaged results. The net can be further amended to reflect probability of underlying network failure and capture statistics based on the number of call attempts fulfilled at the first attempt. Statistics generated can be presented textually or graphically.

It should be noted that all timings used in the telephone process are estimates. For systems-of-systems, it would be anticipated that timings would be informed by subject matter experts, archived, to-be architecture models describing the physical component(s) realising the process or actual implemented components. Based on the results from performance analysis, process logic may benefit from amendment. This could involve consolidation of activities under a different role, activity removal, additional resources, or activity re-ordering.

At this stage in development, no physical components have been decided upon to realise the activities and processes specified. Of interest is using knowledge of (legacy or planned) physical assets to help optimise engineering of the operational process level via analysis-of-alternative scenarios. As well as the timing statistics incorporated above, Salimifard et al [94] report on using nets to allocate physical resources and costs to activity execution. This work is highly relevant to the development of large-scale system-of-systems and can be adapted to it, facilitating analysis-of-alternatives in operational process engineering. The adaptation for the telephone system is shown in Figs. C.5-C.6.

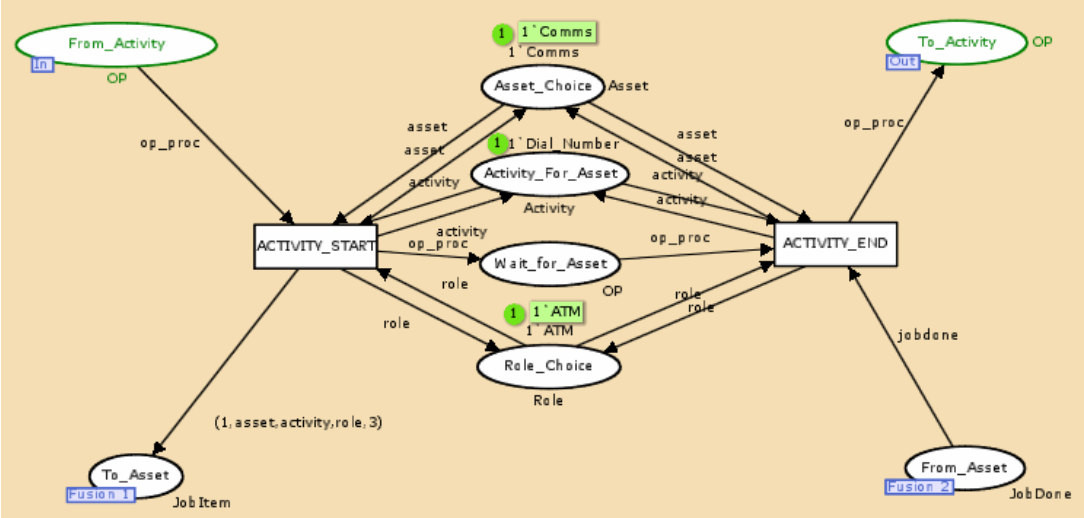


Fig. C.5 'Activity subnet detailing physical asset and role to perform Dial_Number activity'

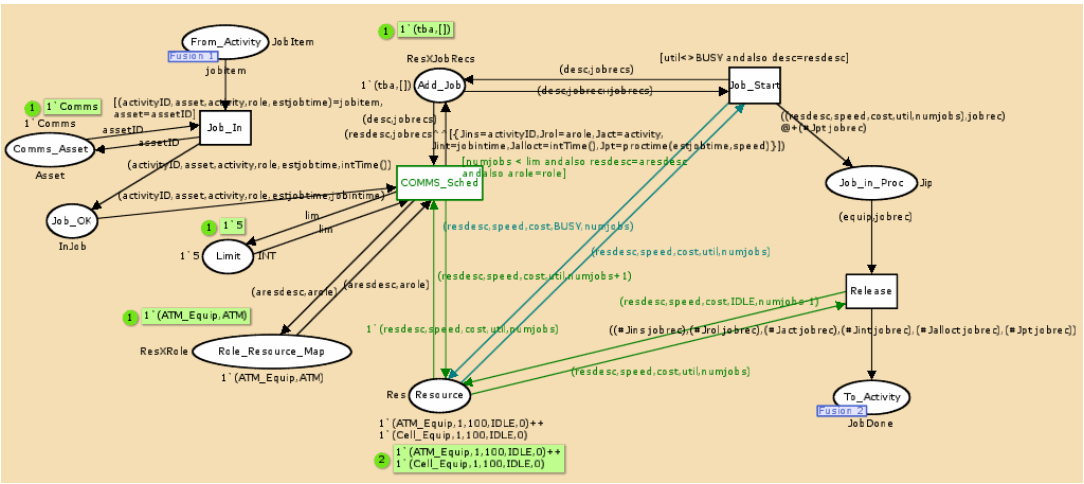


Fig. C.6 'Physical asset net scheduling and processing jobs based on required role'

Fig. C.5 is an instance of a subnet constructed to specify how its parent activity (in this case 'Dial_Number') should be processed in terms of physical asset and role within the asset. 'Dial_Number' is to be processed by the 'Comms' physical asset, specifically the 'ATM' role. Fig. C.6 is a net specifying how the 'Comms' physical asset takes in activities or jobs for processing. In the net, 'Dial_Number' activity arrives and checks are made to see that it is meant to be processed by this asset and the role identified. The job is then added to the asset's overall schedule (associating it with the 'ATM' role) by the 'COMMS_Sched' transition. Jobs are then processed according to their estimated processing time, rate of the role, and cost of the role. At the end of the processing, the role is released and the processing information (time and cost involved) are returned to the subnet of Fig. C.5. The subnet then returns control back to its parent net and the next activity's processing information can be obtained in a similar way. This information can be extracted from the overall net using the toolset in order to gain timing and cost information per activity, per

component process, and for the process overall based on the physical assets used to realise the component activities and processes.

C.1.1 Static Analysis of Timed Coloured Petri Nets

Of further note is static analysis of an untimed net amended for performance analysis through the addition of timing. Based on the thesis work with timed coloured Petri nets using the net toolset, CPN Tools, and recommendations from Jensen [90], static analysis of timed nets requires careful management. According to Jensen [90,100], non-determinism in nets means that a marking cannot be uniquely determined following an enabled transition's execution. For non-deterministic nets, the same state space cannot be generated twice due to this unpredictable behaviour. [90] suggests evaluation of a non-deterministic net to check whether it can be made into a deterministic one. Measures include: restricting the numbers of tokens in the net through initial markings or places that keep track of and enforce a limit the number of tokens; analysing one case of probability at a time; and removing the use of functions from the net (including random distribution functions) and substituting variables that can be bound from a small range of values defined by their associated type. Although these measures can apply to untimed as well as timed nets, timed nets are at much greater risk of experiencing the state space explosion problem than their equivalent untimed net during static analysis.

In static analysis of timed coloured Petri nets, state space graph calculation considers all potential times that can be associated with tokens as well as their colour (types). Sole description of activity ordering in a model can lead to an ambiguous or erroneous behaviour specification when multiple tokens are involved in nets (due to lack of priority and potential for overtaking). Timing can help reduce this ambiguity by associating a timestamp with a token. It may still be possible for tokens to overtake one another depending on introduced delays, timeouts and failures but static and dynamic analyses should help the modeller ascertain why certain output has been achieved. Timed nets will be essential in system-of-systems behaviour specification. Management, including best practice approaches towards their analysis is also essential. The telephone process is considered with this in mind.

For the telephone process example, an untimed operational process net was used in static analysis until a stable behavioural model was achieved. This model was then enhanced with timing information with a view to improving behaviour specification and undertaking validation. From associated simulation of the timed net, further logic amendments were made to the net. Before detailing some of these, static analysis was carried out on the net in order to verify the effects of these enhancements on its behaviour.

Even after following the recommended measures to ensure non-determinism in the analysed net, static analysis using the timed net resulted in a dramatic increase in the size of the state space graph and its terminal markings in comparison to the untimed net used as the basis for implementing the timing information. Inconsistent dead marking results were obtained across analysis runs prior to exercising the non-deterministic measures. These measures included minimising the initial marking to represent two initiated calls and using a small colour (type) to capture delay for three transitions (activities). Recording of call fulfilment time was also removed from the

net and the arrival times of the two calls were separated by one unit using hard coding in the initial marking (Fig. C.7).

When static analysis of the timed net was run, a full state space was calculated in three seconds resulting in four thousand two hundred and twenty-five nodes and nine thousand one hundred and one arcs. Fifty terminal markings were obtained. For the equivalent untimed net, i.e. essentially the net elements and inscriptions of the timed net with timing in colours (types) definitions removed, a full state space was calculated in one second with one hundred and sixty nodes and three hundred and thirty-five arcs. Five terminal markings were obtained. On further inspection of the state space graph and standard report obtained from the timed net, the difference in size and number of terminal markings was attributed to the way timestamp information is used in the state space graph calculation.

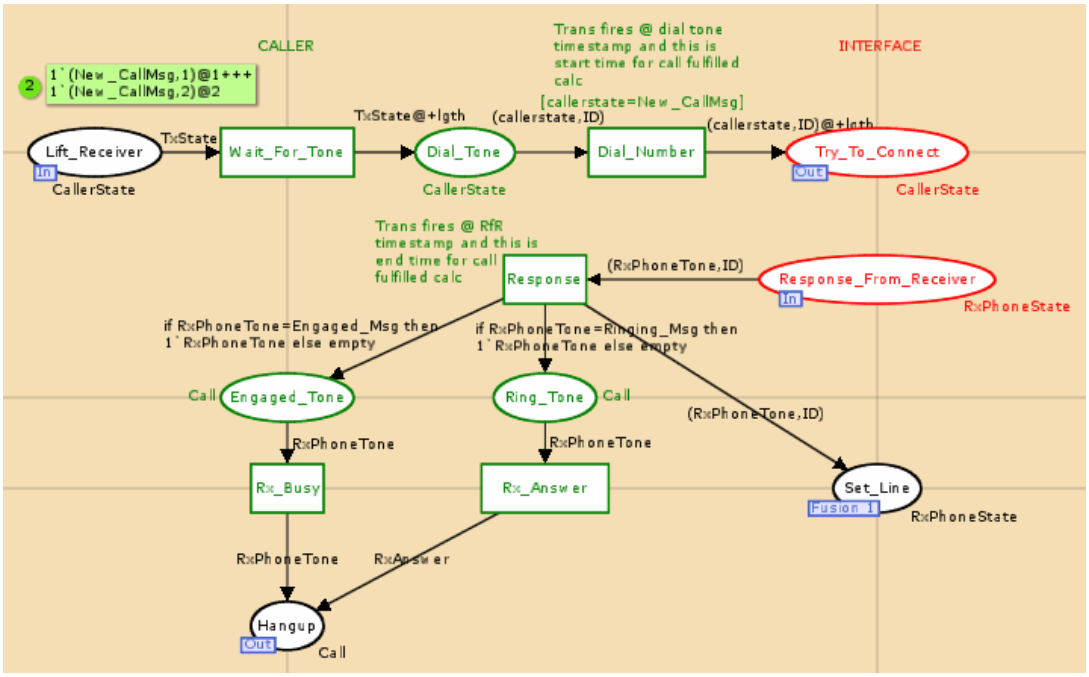


Fig. C.7 'Timed net of telephone process showing initial marking and delays'

From Fig. C.7, the initial marking and time delay introduced (given by the 'lgth' variable binding) following execution of transitions 'Wait_For_Tone' and 'Dial_Number' can be seen. The binding of variable 'lgth' can be selected from a small integer colour ranging between one and four. The two initial marking tokens have timestamps of one and two units respectively and there are three output arc inscriptions which are used to add three further time delays of between one and four units. In static analysis, the net toolset evaluates the state space graph for every possible binding of 'lgth'. In the timed telephone example, after execution of 'Phone_Tone', possible timestamps at place 'Response_From_Receiver' for the token that started with timestamp one (following addition of the three time delays) range from values four to thirteen. For the token timestamped with two, possible values at 'Response_From_Receiver' range from five to fourteen. Table C.1 shows how these ranges were reached.

		Transition	Wait_For_Tone	Dial_Number	Phone_Tone
Initial Marking	**Model Time**	Delay	@+lgth	@+lgth	@+lgth
1`(New_CallMsg,1)@1	1		2,3,4,5	3,4,5,6,7,8,9	4,5,6,7,8,9,10,11,12,13
1`(New_CallMsg,2)@2	2		3,4,5,6	4,5,6,7,8,9,10	5,6,7,8,9,10,11,12,13,14

N.B. lgth=1,2,3 or 4

Table C.1 'Possible timestamp ranges following execution of the three transitions'

The earliest model time a call result on place 'Hangup' can be obtained is four units. At four units, 'Receiver_Line' place would have a timestamp of four and the first call token (initially timestamped one) would have produced a 'Call_AnsweredMsg' or 'Not_AnsweredMsg' (due to the initial marking of place 'Receiver_Line' of 'Line_FreeMsg' it is not possible for the result of the first call to be 'Engaged_Msg'). The lowest timestamp the second call (initially timestamped two) would have on place 'Response_From_Receiver' following this result would be five. The second call would also produce a call result on place 'Hangup' of either 'Engaged_Msg', 'Call_AnsweredMsg' or 'Not_AnsweredMsg'. As part of the second call process, it will reset the receiver's line to free if it successfully connected through to the receiver. The last transition in the net, 'Amend_Line' resets the line as required, updating the 'Receiver_Line' place's timestamp in the process. Following completion of both calls, in this case where the lowest possible timestamp values have been used, the terminal marking will show one of five possible results in place 'Hangup' and the second call having a final timestamp of five (it is known that the first call has a minimum timestamp of four).

Using the above analysis, Table C.1 and inspection of the terminal markings, the fifty terminal markings in the timed standard analysis report can be accounted for. The state space graph terminal markings centre around ten pairs of timestamps (i.e. four and five, five and six, up to thirteen and fourteen) with five possible values in place 'Hangup'. The five markings from analysis of the untimed net have essentially been repeated for each pair of timestamps. It can be seen that this timed analysis only used two tokens representing calls in the initial marking and short delays of one to four units. Subsequent analysis with three calls in the initial marking resulted in the state space explosion problem (within the set processing limit of twenty minutes). Further work was undertaken using static analysis of untimed nets, timed nets and abstraction of process detail within hierarchical nets to see if there were ways to alleviate the state space explosion problem in timed nets.

Calculation of a state space graph was attempted for a timed net and its untimed structural equivalent, i.e. identical net elements and no timed colour (type) definitions or delays. Both types of nets employed deterministic measures. Variations in parameters and results are shown in Tables C.2-C.5:

STATE SPACE GRAPH	Untimed 1	Timed A	Timed B
Initial marking	1`(New _CallMsg,1)++ 1`(New _CallMsg,2)	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2
Nodes and arcs	160 nodes, 335 arcs.	4225 nodes, 9101 arcs.	655 nodes, 1289 arcs.
Generation time	1 sec.	4 secs.	1 sec.
Terminal markings	5	50	20
Delay range	N/A	1..4	1..2

Table C.2 'Comparison between untimed and timed state space graph calculation'

STATE SPACE GRAPH	Untimed 2	Timed C	Timed D
Initial marking	1`(New _CallMsg,1)++ 1`(New _CallMsg,2)++ 1`(New _CallMsg,3)	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2+++ 1`(New _CallMsg,3)@3	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2+++ 1`(New _CallMsg,3)@3
Nodes and arcs	2174 nodes, 6585 arcs.	Explosion problem.	6390 nodes, 16453 arcs.
Generation time	2 sec.	1200 secs (limit set).	12 secs.
Terminal markings	9	N/A	36
Delay range	N/A	1..4	1..2

Table C.3 'Comparison between untimed and timed state space graph calculation'

STATE SPACE GRAPH	Untimed 1	Timed E	Timed F
Initial marking	1`(New _CallMsg,1)++ 1`(New _CallMsg,2)	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2	1`(New _CallMsg,1)@0+++ 1`(New _CallMsg,2)@0
Nodes and arcs	160 nodes, 335 arcs.	51 nodes, 77 arcs.	78 nodes, 163 arcs.
Generation time	1 sec.	Less than 1 sec.	Less than 1 sec.
Terminal markings	5	5	2
Delay range	N/A	1..1	1..1

Table C.4 'Comparison between untimed and timed state space graph calculation'

STATE SPACE GRAPH	Untimed 3	Timed G
Initial marking	1`(New _CallMsg,1)++ 1`(New _CallMsg,2)++ 1`(New _CallMsg,3)++ 1`(New _CallMsg,4)	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2+++ 1`(New _CallMsg,3)@3
Nodes and arcs	26729 nodes, 105193 arcs.	163 nodes, 304 arcs.
Generation time	606 secs.	Less than 1 sec.
Terminal markings	14	7
Delay range	N/A	1..1

Table C.5 'Comparison between untimed and timed state space graph calculation'

Unless time delays and token numbers are kept to a minimum (careful judgement of what constitutes this minimum is needed to avoid adversely affecting system behaviour), timed nets show a marked difference in their calculated state space graphs. This applies even after deterministic measures are applied to the timed net. Table C.2 Untimed 1 and Timed A and Table C.3 Untimed 2 and Timed C results illustrate this considerable size difference where the timed net uses three delays of

between one and four time units. Timed B and Timed D (Tables C.2-C.3) show the effect of reducing the range of the three timed delays to between one and two time units. The associated state space graphs are significantly smaller. In the case of Timed D, restricting the delay range further has made calculation of the state space graph tractable within the given processing limit of twenty minutes. In cases Timed A, Timed B and Timed D all final markings are correct within the given initial marking and delay range parameters.

Table C.4's Timed E shows the effect of reducing the delay to one unit with two tokens (also separated by one unit) in the initial marking. Comparing it directly with its equivalent untimed net in Table C.4 (Untimed 1), it can be seen that five terminal markings (all desirable) are obtained from a smaller state space graph. From this result it can be concluded that the initial marking and delay range parameters of Timed E produce a specification of the telephone process where not only is the activity execution sequence detailed, the tokens are also prioritised within the net. Due to this token prioritisation through timing, there is no need to calculate reachable markings arising from alternative delay value choices (there is only one possible value, that of one unit). This contributes to the decreased size of the state space graph. The one unit delay does not constrict the desirable final markings of the telephone process: all five desirable dead markings achieved from the untimed net were obtained for this timed net.

This is not the case in Table C.5 Timed G. A delay of one unit with three tokens (also separated by one unit) in the initial marking interferes with the desirable behaviour of the process. It restricts the behaviour of the model as the timing introduced dictates that the final, third call token will always encounter a free receiver line. The terminal markings of one call answered, two calls engaged and one call unanswered, two calls engaged will not be obtained in this model with these parameters. Timing dictates that the call token responsible for the busy line will always reset its status to free for the arrival of the third call token. Table C.3 Timed D specification for three tokens incorporates sufficient timing flexibility to allow for the outcome of two engaged call results following the first call.

C.1.2 Largeness Avoidance by Abstraction and Timed Coloured Petri Nets

The perceived benefits regarding state space graph calculation using process abstraction within hierarchical nets were highlighted in Appendix B, section B.1.4. This top-down process abstraction technique was repeated for the timed parent net. It was noted that in Chukwuogo's work [52], no analysis was undertaken on timed coloured Petri nets using the abstraction approach.

The detail of the process associated with the caller role's 'Make_Call' transition was abstracted out of the timed net, i.e. the subnet of Fig. C.7 was removed. No decomposition of the 'Make_Call' transition from the parent net to a separate subnet was included. Instead, a minimal set of net elements were used to ensure that the provided and required interfaces of the 'Make_Call' transition consumed and produced the same information as before for the parent net. The process detail associated with the Receiver role's 'Connect_Call' transition remained the same. For the two delays abstracted out of the 'Make_Call' process detail, a new equivalent delay variable with

range between two and eight was substituted at the parent net level. The third delay (part of the 'Connect_Call' process detail) remained the same, i.e. between one and four. Following abstraction there were now two delay variables representing the original range of values rather than one. As Table C.6 shows, these two variable ranges were reduced in line with Tables C.2-C.5 to examine the effect on state space calculation:

STATE SPACE GRAPH	Timed H (activity 'Make_Call' abstracted)	Timed I (activity 'Make_Call' abstracted)	Timed J (activity 'Make_Call' abstracted)
Initial marking	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2+++ 1`(New _CallMsg,3)@3	1`(New _CallMsg,1)@1+++ 1`(New _CallMsg,2)@2+++ 1`(New _CallMsg,3)@3
Nodes and arcs	2562 nodes, 5137 arcs.	42148 nodes, 107304 arcs.	3147 nodes, 7241 arcs.
Generation time	2 secs.	481 secs.	4 secs.
Terminal markings	50	90	36
Delay range	2..8 and 1..4	2..8 and 1..4	2..4 and 1..2

Table C.6 'Abstracted Make_Call process in timed state space graph calculation'

Abstraction of just one of the two parent net activities appears to significantly reduce the state space graph size and calculation time. For example, Table C.6 Timed H is behaviourally equivalent to Table C.2 Timed A but state space size and timing of Timed H are approximately halved using abstraction. Timed I is behaviourally equivalent to Table C.3 Timed C which suffered from the state space explosion problem. This time a full state space graph could be calculated for Timed I. Finally, Table C.6 Timed J is behaviourally equivalent to Table C.3 Timed D. Again, state space size has been approximately halved and calculation time reduced by two-thirds using abstraction.

C.1.3 Largeness Avoidance by Composition and Timed Coloured Petri Nets

The perceived benefits regarding comprehension of nets and state space graph calculation using process composition within hierarchical nets were highlighted in Appendix B, section B.1.5. This bottom-up compositional technique was repeated for the timed net using the interface information of the telephone process. As before, the telephone process was separated into three processes using knowledge from conducting the abstraction approach on a timed net and the earlier compositional approach on an untimed net. Based on this knowledge, the analysis began with two initiated calls and a delay value of one time unit in order to keep output manageable.

The first process owned by Caller was isolated by its input and output interface places and marked with an initial marking of two tokens separated by one time unit. The delay range within this process was also limited to one time unit. Simulation was used to check that the changes had not negatively impacted the behaviour of the net and then static analysis was conducted. From the one terminal marking information needed for the interface of the next process in sequence could be obtained.

1`(New_CallMsg,1)@+3++1`(New_CallMsg,2)@+4 was input into process two (owned by Receiver). Again, the delay within the second process was set to one time unit and the configuration was checked using simulation. Static analysis was then conducted. Two terminal markings were obtained. The output from each terminal marking place 'Response_From_Receiver' was extracted:

1. *1`(Ringing_Msg,1)@+4++1`(Ringing_Msg,2)@+5*
2. *1`(Engaged_Msg,2)@+5++1`(Ringing_Msg,1)@+4*

Both were input separately into the last process, owned by Caller.

The results on place 'Hangup' from three and two terminal markings respectively are listed below:

2`Call_AnsweredMsg
2`Not_AnsweredMsg
1`Not_AnsweredMsg++1`Call_AnsweredMsg

1`Not_AnsweredMsg++1`Engaged_Msg
1`Call_AnsweredMsg++1`Engaged_Msg

Collating both gives a possible five results from two initiated calls separated by one time unit and processed within a model with delays set to one time unit and receiver status of line free. This matches the abstracted result for two initiated calls in Timed E above.

The compositional approach outlined above was repeated to examine the effects of increasing numbers of tokens with fixed and variable delay range (of one to two time units).

STATE SPACE GRAPH	**Timed Process 1**	**Timed Process 2**	**Timed Process 3**
Initial marking	1`(New_CallMsg,1)@1+++ 1`(New_CallMsg,2)@2+++ 1`(New_CallMsg,3)@3	1`(New_CallMsg,1)@3+++ 1`(New_CallMsg,2)@4+++ 1`(New_CallMsg,3)@5	1. 1`(Engaged_Msg,3)@6+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,2)@5 2. 1`(Engaged_Msg,2)@5+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,3)@6 3. 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,2)@5+++ 1`(Ringing_Msg,3)@6
Nodes and arcs	9 nodes, 10 arcs.	19 nodes, 20 arcs.	1. 15 nodes, 15 arcs. 2. 13 nodes, 13 arcs. 3. 16 nodes, 18 arcs.
Generation time	0 secs.	0 secs.	1. 0 secs. 2. 0 secs. 3. 0 secs.
Terminal markings	1	3	1. 3 2. 3 3. 4
Delay range	1..1	1..1	N/A

Table C.7 'Compositional approach using fixed delay range of one time unit'

STATE SPACE GRAPH	Timed Process 1	Timed Process 2	Timed Process 3
Initial marking	1`(New_CallMsg,1)@1+++ 1`(New_CallMsg,2)@2+++ 1`(New_CallMsg,3)@3+++ 1`(New_CallMsg,4)@4	1`(New_CallMsg,1)@3+++ 1`(New_CallMsg,2)@4+++ 1`(New_CallMsg,3)@5+++ 1`(New_CallMsg,4)@6	1. 1`(Engaged_Msg,2)@5+++ 1`(Engaged_Msg,4)@7+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,3)@6 2. 1`(Engaged_Msg,4)@7+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,2)@5+++ 1`(Ringing_Msg,3)@6 3. 1`(Engaged_Msg,3)@6+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,2)@5+++ 1`(Ringing_Msg,4)@7 4. 1`(Engaged_Msg,2)@5+++ 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,3)@6+++ 1`(Ringing_Msg,4)@7 5. 1`(Ringing_Msg,1)@4+++ 1`(Ringing_Msg,2)@5+++ 1`(Ringing_Msg,3)@6+++ 1`(Ringing_Msg,4)@7
Nodes and arcs	12 nodes, 14 arcs.	34 nodes, 37 arcs.	1. 19 nodes, 19 arcs. 2. 24 nodes, 26 arcs. 3. 22 nodes, 24 arcs. 4. 20 nodes, 22 arcs. 5. 25 nodes, 30 arcs.
Generation time	0 secs.	0 secs.	1. 0 secs. 2. 0 secs. 3. 0 secs. 4. 0 secs. 5. 0 secs.
Terminal markings	1	5	1. 3 2. 4 3. 4 4. 4 5. 5
Delay range	1..1	1..1	N/A

Table C.8 'Compositional approach using fixed delay range of one time unit'

Based on Tables C.7-C.8, there is a pattern of output from process one. Given a certain number of tokens as input, the same number of tokens is output with their timestamps incremented by two. It can also be seen that process two dictates the volume of collation involved for process three. For example, in Table C.8, process two outputs five terminal markings. Each of these markings is considered in turn and each marking on place 'Response_From_Receiver', used as a separate input into process three. Process two is now considered for five and ten initial markings in Table C.9.

STATE SPACE GRAPH	Timed Process 2	Timed Process 2
Initial marking	1`(New_CallMsg,1)@3+++ 1`(New_CallMsg,2)@4+++ 1`(New_CallMsg,3)@5+++ 1`(New_CallMsg,4)@6+++ 1`(New_CallMsg,5)@7	1`(New_CallMsg,1)@3+++ 1`(New_CallMsg,2)@4+++ 1`(New_CallMsg,3)@5+++ 1`(New_CallMsg,4)@6+++ 1`(New_CallMsg,5)@7+++ 1`(New_CallMsg,6)@8+++ 1`(New_CallMsg,7)@9+++ 1`(New_CallMsg,8)@10+++ 1`(New_CallMsg,9)@11+++ 1`(New_CallMsg,10)@12
Nodes and arcs	58 nodes, 64 arcs.	694 nodes, 781 arcs.
Generation time	0 secs.	0 secs.
Terminal markings	8	89
Delay range	1..1	1..1

Table C.9 'Compositional approach using fixed delay range of one time unit in process two'

Before drawing conclusions, the exercise is repeated using a variable delay range of one to two time units.

STATE SPACE GRAPH	Timed Process 1	Timed Process 1
Initial marking	1`(New_CallMsg,1)@1+++ 1`(New_CallMsg,2)@2	1`(New_CallMsg,1)@1+++ 1`(New_CallMsg,2)@2+++ 1`(New_CallMsg,3)@3
Nodes and arcs	31 nodes, 42 arcs.	111 nodes, 180 arcs.
Generation time	0 secs.	0 secs.
Terminal markings	12	36
Delay range	1..2	1..2

Table C.10 'Compositional approach using variable delay range of one to two time units'

Table C.10 shows that introduction of a small delay range significantly increases the number of nodes and terminal markings for two and three initiated calls. Considering process one from Table C.10, for two initiated calls, there are now twelve markings to examine in order to collate the input for process two. Investigating these manually will be time-consuming and based on Table C.9, introducing the same small delay range in process two will produce the same significant rise in terminal markings and subsequent collation effort.

Compositional analysis of timed nets can be used to improve readability and comprehension of the net as well as alleviate state space explosion. When starting the analysis, it is advisable to keep initial markings and introduced delays to low numbers and ranges without adversely affecting the behaviour of the net. Advice on achieving this configuration information is expected to come from experience constructing the untimed equivalent of the timed nets. It is anticipated that attempts at full and largeness avoidance analyses would have been made on these previously. Seeding of initial markings can then be gradually increased if appropriate to do so.

Summary of largeness avoidance techniques in timed nets

In Appendix B and sections C.1.2-C.1.3, abstraction and compositional largeness avoidance techniques were applied to untimed and timed nets. Analysis of timed nets (full, abstraction, or composition approaches) requires more careful planning than analysis of their untimed equivalents and benefits immensely from the experience gained working with untimed nets. General best practice when configuring analysis using full, abstraction, and compositional approaches is to initially restrict numbers of tokens in the net and the range of values that can be assigned to time delays. Simulation can then be used to check the net behaves as expected before conducting static analysis. Results from static analysis in each approach (where it has been possible to generate a full state space graph) can then be used to verify the results obtained using the other approaches.

Using a top-down, functional decomposition approach to identify a suitable hierarchy within the model of the telephone example meant that after attempting full model analysis, the abstraction approach was the next natural technique to apply. The hierarchy provided an overview of the abstraction levels within the model and the interface boundaries of modules. A common level of detail could then be identified upon which to select a module, remove its underlying detail and still maintain the integration structure of the model based on module interface boundaries. As well as combating the state space explosion problem, the abstraction approach also helps the modeller to become familiar with the different levels of detail used in the model, dependencies between modules, required interface information, and order of execution.

After conducting the abstraction approach to largeness avoidance, the compositional approach was employed. This is a bottom-up approach to largeness avoidance. At this stage, the bottom-up approach considers modules at a low level of abstraction identified by top-down, functional decomposition. Existing nets of modules available from a repository were not considered in this thesis. This is a potential future area of research. Identification of the nets capturing detailed descriptions of modules is aided by the hierarchy within the model. Order of execution between modules is obtained from nets at a higher abstraction level within the model. The detailed nets are then isolated using their interfaces and analysed individually using outputs from one subnet to seed the next in execution order. Due to this low level analysis, the modeller can gain in-depth knowledge of the behaviour of parts of the whole. The compositional approach may also highlight a greater number of issues for improvement or correction than would otherwise be possible from analysis of the whole net.

Both approaches are complimentary in the sense that results from one can help verify results from the other. It is also anticipated that results from the abstraction approach may guide the direction of compositional analysis so that behaviour of a particular module or grouping of modules is examined.

These two approaches to largeness avoidance are currently highly manual, relying completely on a suitable hierarchy both at the model level and within each model. Unless reasonably low numbers of terminal markings are obtained from either approach, examination of results can be extremely time intensive and tedious. Hierarchy is key to alleviating this issue and may be able to suggest suitable division of the model for workload purposes. Automation may also be feasible (and is

considered in Appendix D). Again, process and best practice regarding model construction and management will be essential to the successful design and specification of large-scale systems.

Amendments made to the model during validation

In terms of amendments made to the telephone process, experimentation using timed nets and static and dynamic analyses revealed that the original untimed net specification required improvement. The 'Set_Line' place was not always empty upon simulation termination. While trying to understand the standard report generated from static analysis on the timed net, it was noted that a non-empty 'Set_Line' place contributed to an erroneous (redundant) terminal marking.

The error was due to one of the firing conditions for transition 'Amend_Line' being dependent on a value of 'Line_BusyMsg' from place 'Receiver_Line'. 'Amend_Line' should have been able to fire, remove the token from 'Set_Line' and update 'Receiver_Line' accordingly. Instead it was stalled unless the current value in 'Receiver_Line' was 'Line_BusyMsg'. The inscription on output arc 'Receiver_Line' was changed to a variable. Depending on this variable's binding, the 'Amend_Line' transition would either reset the receiver's line to free or maintain its status as engaged. This meant the transition always fired and correctly removed the token on 'Set_Line'.

Further simulation analysis of line reset behaviour highlighted the need to differentiate between the calls so that the line was reset correctly by the call that had successfully connected. The new inscription logic and place colours (types) incorporating message and caller identification are shown in Fig. C.8.

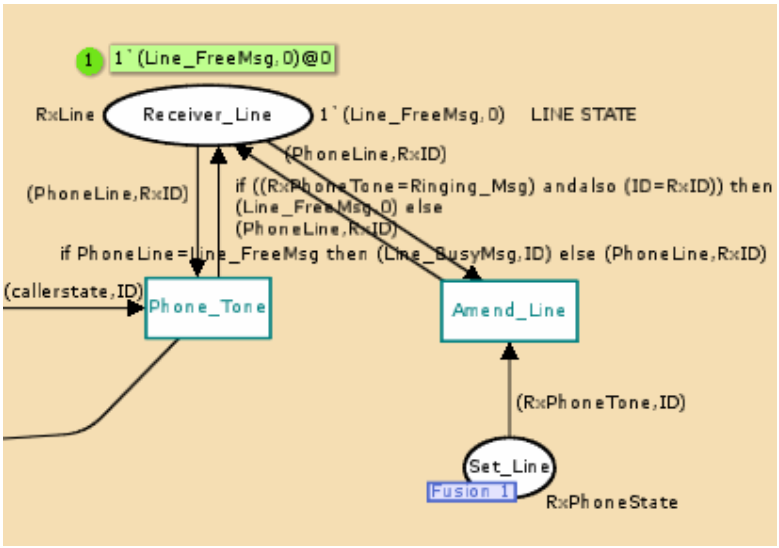


Fig. C.8 'Amended net logic to reset the receiver line'

It was also noted that within the model, the same timestamp can end up being allocated to both calls resulting in transitions involving each to become enabled simultaneously or random removal of the call tokens when the transition (enabled by their presence on the input place and current model time matching their timestamp) fires. To control this behaviour, a call queue may be introduced to prioritise calls

based on their arrival time in the system (and perhaps contribute to a defined performance indicator by increasing call connection rates where a call reaches the receiver and is placed in a queue) or a function can be written to ensure different timestamps are provided to the tokens. Use of the function solution may not be amenable to static analysis.

C.2 Conclusions so far following Validation using Timed Coloured Petri Nets

In order to explore validation of the Petri net specifications, Appendix C added timing information into the telephone process net. Using the net toolset, untimed nets were enhanced in this way for the purposes of achieving deeper correctness, performance and efficiency (Table C.11). Performance analysis and analysis-of-alternatives were achieved using a combination of timed places, delays, data collection points throughout the net and simulation.

VALIDATION PURPOSE	**Timed Net Simulation**	**Timed Net Static Analysis**
Enhanced Correctness	X	X
Performance Analysis	X	
Analysis-of-Alternatives (Resource-based Costing)	X	

Table C.11 'Validation purposes in relation to dynamic and static analyses'

Initially, performance indicators for the system based on time, cost and quality parameters were identified in context with the viewpoint from which the system was being developed. In this way, enhancements related directly to performance indicator optimisation were made to the system model. During the course of introducing time to the model, simulation and static analysis also highlighted further improvements to logic and colours (types) for the telephone example.

With the toolset, automatic simulation executions via replication and parameterisation enabled comparison to be made of results using different random distributions, numbers of resources and delays. Textual reports can have their statistical output customised by the modeller and graphical output can be plotted from standard reports.

Based on the work of Salimifard [94], section C.1 also made use of timed nets to conduct more detailed analysis-of-alternative scenarios. Although the conceptual level of abstraction featured in the telephone example describes the problem rather than a solution, knowledge of legacy or planned physical assets which can realise activities and processes modelled by the abstract net can be used to conduct an analysis-of-alternatives. This differs from simply seeding a simulation execution with timing delays because it allows the modeller to explicitly allocate a resource (with associated timing and cost information) to an activity (or process) and extract timing and cost information for the overall model based on different resource allocation. It may also be able to facilitate business process re-engineering using the knowledge of existing or planned resources to influence ordering of activities based on their known timing and cost attributes.

Finally, timed nets have a lower threshold for infinite state space graphs. Even with the simple telephone example, the state space explosion problem was encountered with as few as three initiated calls and three delays of between one and four units (and a configured twenty minute maximum calculation time). It was recommended that equivalent untimed nets (i.e. nets consisting of the same elements, colours and inscriptions but no timed places) be used to produce a stable behavioural net for the purposes of developing a net for use in validation. While dynamic analysis with timed nets is recommended for the purposes listed in Table C.11, static analysis of timed nets is essential for confirming behavioural properties of large-scale, system-of-systems. Static analysis of timed nets needs to be carefully managed.

The use of hierarchy and the abstraction and composition of net components is vital in increasing a timed net's largeness avoidance threshold. In addition, timed nets need to be evaluated prior to static analysis to see whether they can be made into deterministic nets. Experimentation regarding the number of tokens to use in initial markings and range of values in colour (type) definitions for delays is recommended when static analysis of timed nets needs to be conducted. Analysis results obtained using structurally equivalent untimed nets will be useful in helping to inform the results obtained from timed nets.

Based on the conclusions above using hierarchy, timing, and static and dynamic analyses, coloured hierarchical nets were then used in the specification of the telephone system at design and architecture levels of abstraction.

Appendix D

D.1 Specification, Verification and Validation of the Telephone Process at Design and Architecture Levels using Coloured Petri Nets

Appendices A-C have focused on constructing a Petri net to specify a telephone process at a conceptual level of abstraction. Hierarchy and timing were added to further enhance a specification in terms of scalability, understandability, readability, and correctness. Hierarchy also enabled investigation into whether the abstraction design used in the net could be used to help alleviate the state space explosion problem during model-checking. Both model-checking and simulation were employed iteratively in verification and validation of the constructed conceptual level net. In this Appendix, Petri nets are checked to see whether they can be used to specify a telephone system at design and architecture levels of abstraction.

D.1.1 The Design Level

The purpose of the design level of abstraction is the lead into the specification of a solution to the problem described by the conceptual level. Again, a functional decomposition approach was used. This time it was used in conjunction with the parent net developed for the conceptual level to think about how this net's main activities of 'Make Call' and 'Connect Call' would eventually be realised by physical implementations. To keep the design flexible, two components, 'Make_Call Component' and 'Connect_Call Component', were used to depict the solutions that would realise each of the main activities. These are shown in Fig. D.1:

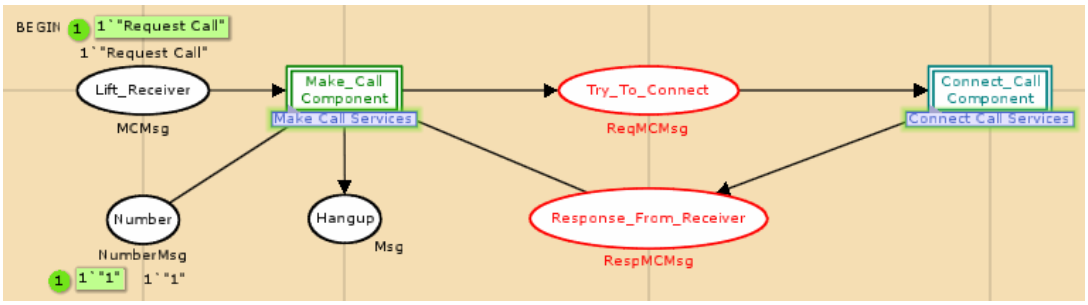


Fig. D.1 'Parent net of design level'

It can be seen that Fig. D.1 closely resembles the parent net of the conceptual level except for the additional place, 'Number' (used to represent the capture of an entered telephone number) and new place colours or types. The next level of design decomposition for the two components aimed to capture the functional service(s) each would be expected to realise. Again, the work developing the conceptual level net helped suggest functional services for the design level by thinking about the purpose of the processes used to realise the main activities. 'Make_Call Component' would be responsible for providing call setup and call response services. 'Connect_Call Component' would be responsible for providing an incoming call processing service.

These services are shown at the next lower abstraction level providing greater detail in Figs. D.2-D.3:

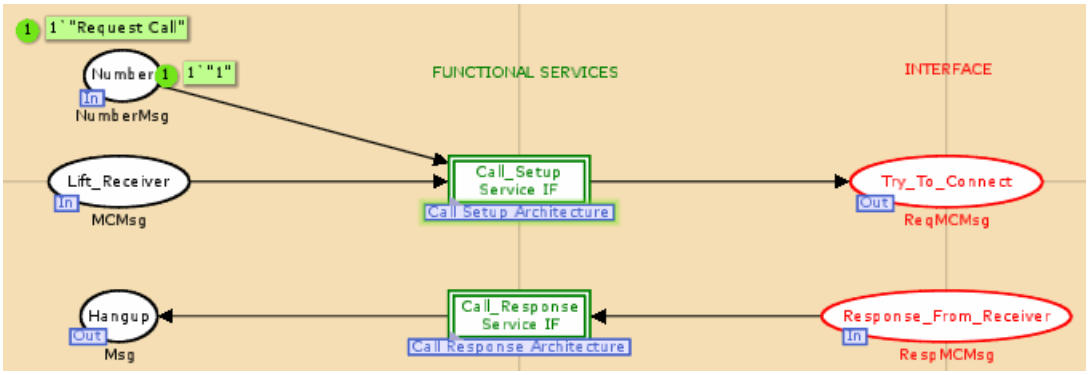


Fig. D.2 'Services of Make_Call Component'

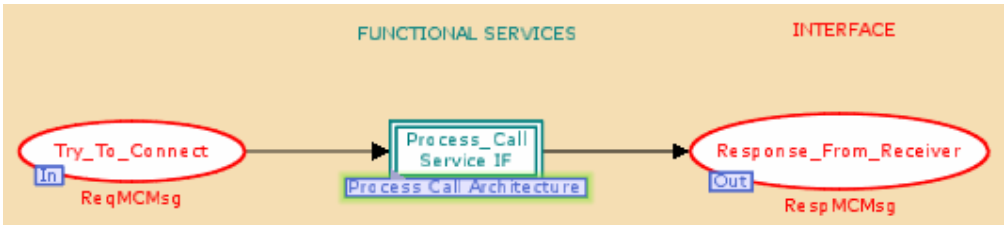


Fig. D.3 'Services of Connect_Call Component'

Having introduced the components and functional services at the design level, the next lower abstraction level providing greater detail, i.e. detailed design or architecture was focused on. Rather than develop a separate model at this stage, as the architecture level appeared to naturally manifest the next lower abstraction level of the design level, the design level model was further decomposed to capture the architecture level.

D.1.2 The Architecture Level

The purpose of the architecture level is detailed design of the services identified at the design level and flexible capture of the components required to realise these individual services. Considering the functional services, constituent components were considered for each service resulting in the identification of a common component pattern for the three services. The common components consisted of a user interface, transmit and receive (network) interfaces, and a controller interface to co-ordinate the sequencing of activities to and from the other two common components. The common component architecture is shown for the 'Call_Setup Service' and 'Process_Call Service' in Figs. D.4-D.5.

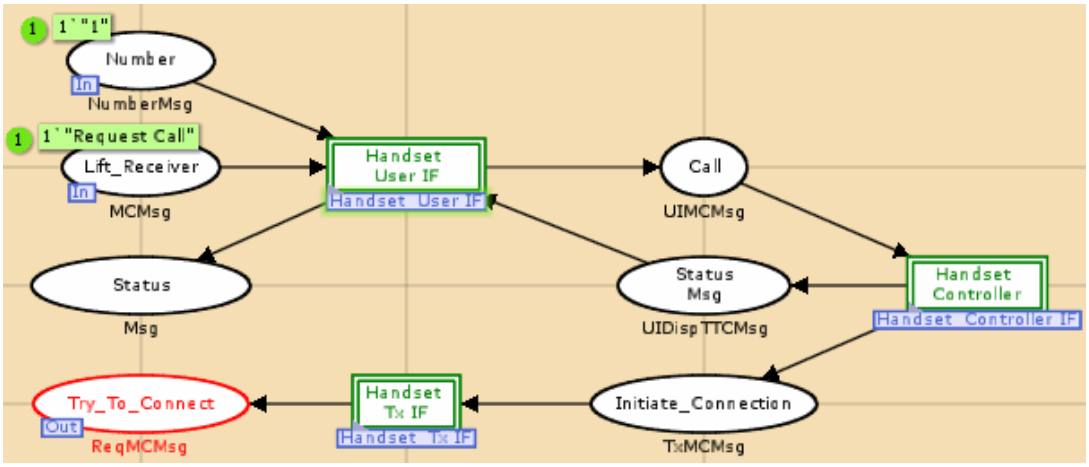


Fig. D.4 'Call_Setup service architecture'

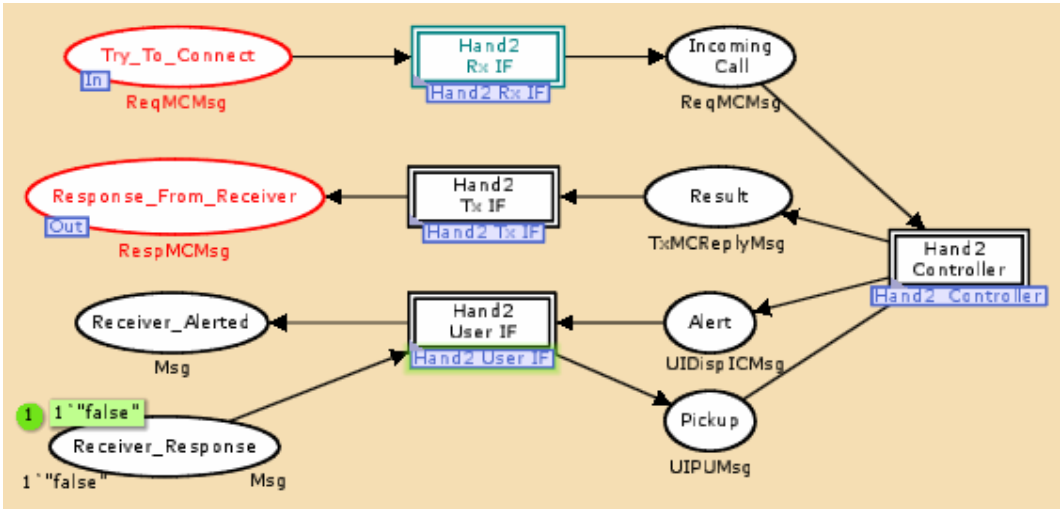


Fig. D.5 'Process_Call service architecture'

From Figs. D.4-D.5, it can be seen that net places are used capture the input and output information for the user interface, network and controller common components. In this early attempt at design and architecture levels, enumerated type definitions were lifted for re-use from the conceptual level net and labelling reflected the same terminology where possible. Figs. D.4-D.5 show that colour (type) definition labels reflect the nature of the interface. For example, 'UIDispICMsg' aims to reflect that the place is an input interface to the user interface component and is intended to be processed by the display function within this component. The intention with this labelling convention was improved net clarity and comprehension.

Considering the original parent net of the design level in Fig. D.1, the specification of the telephone system at this level was extremely concise. When the architecture level of Figs. D.4-D.5 was reached and the next lower abstraction level providing greater detail of the common component interfaces was completed, the levels of abstraction were very difficult to manage. The toolset presents each level of abstraction as a

separate page within a folder (or binder). These pages can be selected between using their tabs. By the common component interface level of abstraction, sixteen pages and tabs were present and it was tedious work identifying and selecting relevant pages using barely legible tabs (Fig. D.6). At this stage, the model was rationalised where possible, making use of the toolset's features and those of hierarchical coloured Petri nets.

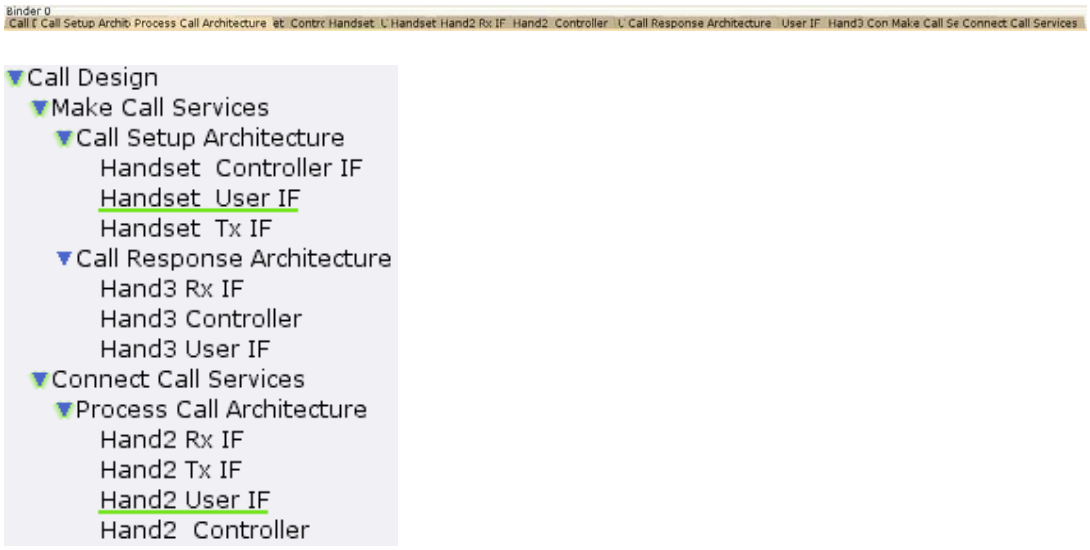


Fig. D.6 'Early model abstraction levels presented within one binder'

The next iteration of model development involved rationalisation of colours (types), subnets describing common component interfaces, removal of redundant places, and toolset presentation of pages. The main source of rationalisation was the common component interface nets.

The toolset enables a net to be used within another net (normally to reflect decomposition from an activity captured at a higher level of abstraction), similar to the concept of a procedure call in a high-level programming language. A net at a lower abstraction level providing greater detail can be used by one or more activities at a higher abstraction level. In this way, the net at the higher-level of abstraction is effectively re-using the subnet's structure, inputting and receiving information according to the defined colours (types) of the input and output places (comparable to a parameterised procedure call). The input and output information supplied to the subnet relates to the parent net transition linked with the subnet. If two separate parent net transitions are decomposed to the same subnet, instances of the subnet are effectively created by the toolset. These instances then consume (provide) input (output) values of the same colour (type) unique to their parent net transitions.

Each of the common component interfaces was examined in turn to check the internal structure and logic of their nets and decide if it would be feasible to use one net (instance) for each common component. Currently, the design and architecture model had ten separate nets representing the common components realising the three services. The user interface component's main functions are provision of notification to the user and capture of a request from the user. The request to provide notification

to the user comes from the controller component. The user inputs a request using the keypad and this is captured and passed to the controller. The transmit network interface essentially provides a send function when requested to do so by the controller and its receipt counterpart provides a receive function, accepting information from the network and passing it to the controller. Finally, the controller component processes information from the user interface and network components, deciding what information should be sent and notified. Unsurprisingly, the controller component is the most complex in terms of logic and structure.

Thinking about the architecture level in terms of the three services identified at the design level, the common components were assessed for each service. To use instances of one subnet, the interface place colours (types) of the subnet need to be common to each transition intending to be the abstraction of the subnet. This was straightforward for the user interface and network components. Place colours (types) were rationalised during this exercise. Information external to the telephone system, i.e. provided or passed to the caller was typed by 'CallerState' or 'Msg'. Colour (type) 'NumberMsg' used in the initial model attempt and the input place it defined ('Number') were removed in favour of simplifying the number of external request interfaces to the user interface component to one for the moment ('CallerState'). Interfaces between the common components were typed as 'UIMsg' (request from user interface to controller), 'UIDispMsg' (display request from controller), and 'NWMsg' (transmit request from controller or receive information request from network).

For the common controller component, inputs and outputs from and to the other common components were used to rationalise the net's structure and logic from the original three subnets developed to represent the controller component. Once this had been undertaken, the parent nets specifying the architecture of each service, i.e. 'Call Setup Architecture', 'Process Call Architecture' and 'Call Response Architecture' (decomposed from their respective design level services 'Call_Setup Service', 'Process_Call Service' and 'Call_Response Service') needed to be amended to associate each common component with its one re-usable decomposed subnet and ensure that the subnet interfaces matched those of the linked transitions in the parent net (even if no information was to be input or output by the interface places). The re-usable common controller component is shown in Fig. D.7.

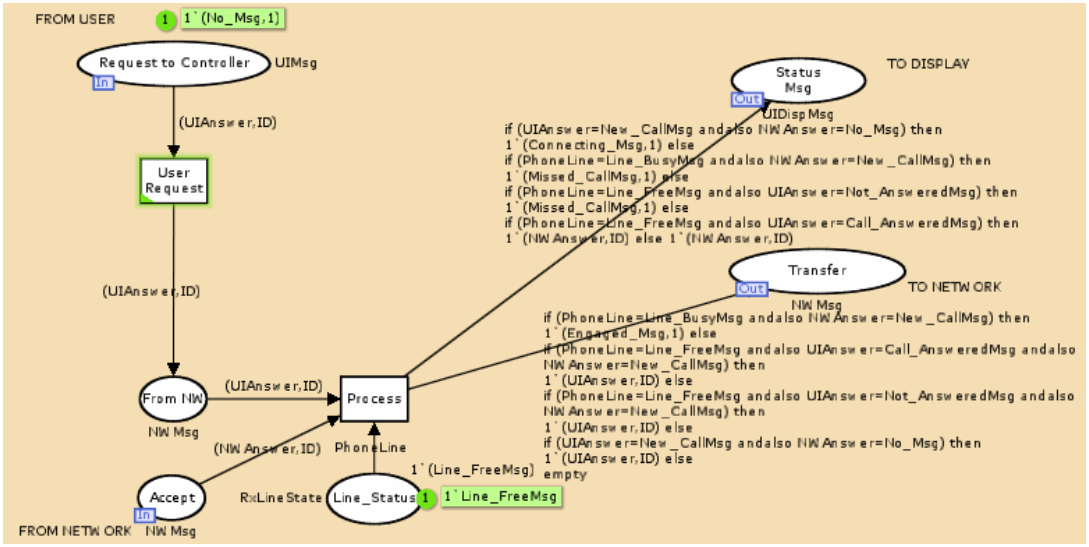


Fig. D.7 'Handset Controller common component subnet'

The resulting model now used four subnets instead of ten subnets to capture the detail of the four common components. Although the toolset still presents these subnets as ten separate pages in the model, editing of the subnets is greatly simplified since a change made to an instance of a re-usable subnet updates all its instances. Using the toolset, for each service architecture parent net, a re-usable subnet was linked to its abstract common component. In addition, two further folders (binders) were created to partition the model logically into the three service architectures making navigation, readability, comprehension and simulation much easier to achieve. This layout is shown in Fig.D.8:

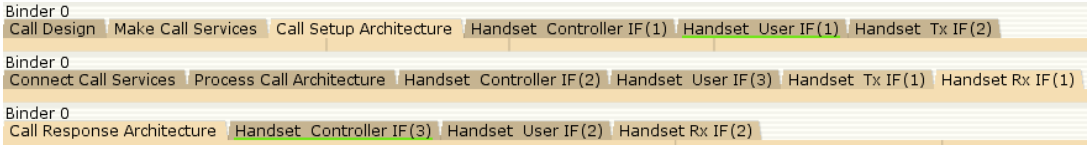


Fig. D.8 'Model abstraction levels presented within three binders'

D.1.3 Verification of the Design and Architecture Levels

At this stage, simulation was employed to check the structure and logic of the model and was able to detect incorrect logic on transition output arcs. Errors included: missing or incorrect predicates (highlighted by incorrect or missing display notifications for the common user interface component or incorrect information messages for the network component); missing initial values on input places required by common component interfaces; an unexpected disabled transition due to the same variable being used to bind values on more than one of its input arcs; and unexpected simulation halts due to missing values in enumerated type definitions associated with place types.

The necessary corrections were made and static analysis was performed based on one initiated call and the receiver of the call not responding to the call. No further errors were picked up by model-checking so the model was adapted to deal with multiple initiated calls to the receiver. Enhancements included addition of logic and net structure to set the receiver's line status to free or busy during call request processing by the common controller component, random initialisation of the response to the connection request and a change of place type (based on character strings rather than enumerated types). The latter introduction of place types composed of at least one string type was made for flexibility reasons.

Currently, the specification of the common component interfaces were informative regarding the information expected at their input and output places but lacked detail regarding the functions realised internally that are made use of by the other common components. To make this information more explicit in the model, colour (type) definitions based initially on the toolset's 'product' type definition (Fig. D.9) were implemented. Again, the main reason for doing so was flexibility. At this stage in the modelling where the model is undergoing frequent amendments, this particular compound definition was found to be quicker to adapt than using an equivalent fixed record definition and its associated syntax. Changes could be made quickly to values as necessary during iterative model updates and amendments rather than maintain enumerated or record type definitions.

```
colset Params = STRING;
colset ReqID = INT;
colset Msg = STRING;
colset QueryMsg = product OpName*Params*ReqID;
colset UIMsg = QueryMsg;
colset UIDispMsg = product OpName*Params;
colset NWMsg = UIMsg;
```

Fig. D.9 'Product compound type definitions to indicate usage of functions'

Once the place types were defined, the tuples in the type were populated with the functions implemented by each common control component and the associated parameters via logic on transition output arcs. Logic on transition output arcs within each of the common components was amended as necessary. As an example, consider the network common components in Figs. D.10-D.11.

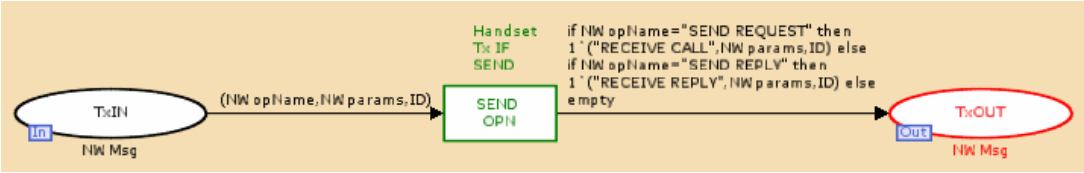


Fig. D.10 'Transmit common component subnet'

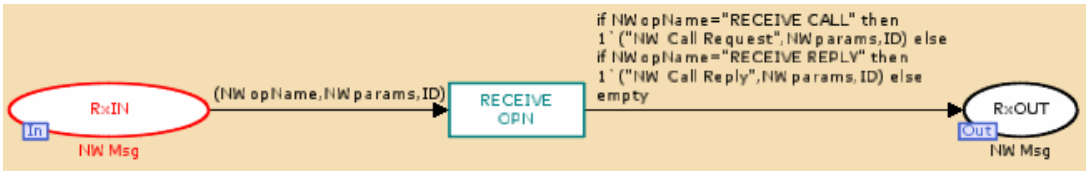


Fig. D.11 'Receive common component subnet'

Fig. D.10 shows the subnet of the updated transmit common component. Its transition is labelled as 'SEND OPN' to reflect the function the component provides to the controller component. On the transition output arc (within the controller component subnet) to the input interface place ('TxIN') of the transmit component, there is logic to output a token with 'OpName' (a tuple within 'NWMsg' compound type) populated with 'SEND REQUEST' or 'SEND REPLY'. In this way, the net specifies use of the transmit component's 'SEND OPN' function by the controller component more explicitly. The 'Params' tuple within 'NWMsg' is populated with either the number to call (in the case where a connection request is made) or the result of the connection request (in the case where a reply is made back to the caller). 'ID' is populated to differentiate between initiated calls.

Fig. D.11 shows the network counterpart to the transmit component, receive. This component's function is used by the underlying communications network to hand-off a message destined for this network node. As before with the transmit component, the receive component's transition is labelled to reflect the function the component implements, in this case 'RECEIVE OPN'.

The other two common components, user interface and controller, are designed to reflect the same interface principles as those discussed above for the network components.

Following these changes, simulation was used first of all to verify the model. Similar issues were encountered to the early model attempt in terms of incorrect logic on transition output arcs, an unexpected disabled transition due to the same variable being used to bind values on more than one of its input arcs, and omission of initial markings on new structures added to the model (line reset function and random initialisation of receiver response to connection request). Once amendments were made and iterative simulation increased confidence that the model's behaviour was correct, static analysis based on two initiated calls was conducted (Fig. D.12).

```
Reachability/State Space
     Nodes:  33322
     Arcs:   136719
     Secs:   665
     Status: Full

Scc Graph
     Nodes:  33322
     Arcs:   136719
     Secs:   8

10 Dead Markings
```

Fig. D.12 'Static analysis for the design and architecture level model including random initial marking generation'

Upon inspection of the ten dead markings, it was noted that an expected result was missing (1`"Not Answered"++1`"Engaged"). After investigation, it was discovered that this was due to an omission in the transition output arc logic. Static analysis was repeated following this logic update and the state space explosion problem was encountered based on a toolset calculation limit of twenty minutes. The model was discretised by removing the random initial marking generator for connection response and instead set two manual initial markings of 'Ignore' and 'Pickup'. The results of model-checking are shown in Fig. D.13.

```
Reachability/State Space
     Nodes:  19656
     Arcs:   79164
     Secs:   240
     Status: Full

Scc Graph
     Nodes:  19656
     Arcs:   79164
     Secs:   4

9 Dead Markings
```

Fig. D.13 'Static analysis for the discretised design and architecture level model'

The nine dead markings contained the expected results for the given initial markings.

D.1.4 Largeness Avoidance by Abstraction

Based on the benefits reported using abstraction in largeness avoidance in Appendices B-C, the technique was investigated with the model developed for the design and architecture level of abstraction. At this level, the aim was to abstract out the detail of each of the main components and their associated services to check the effects on the duration and size of the state space graph.

The intention was to remove the detail of 'Make_Call Component' from the parent net of the design level (Fig. D.1). As before, a minimal set of net elements were used to

capture the pared down function of the 'Make_Call Component', i.e. the expected information reached its input and output interface places. The 'Make_Call Component' function was realised by two underlying services and these transitions ('Call Setup Service' and 'Call Response Service') were used to abstract out the detail of the component. The logical folder grouping related to the 'Make_Call Component' was analysed to decide upon the logic necessary to replicate the functions of the underlying subnets and produce (consume) the correct information at the interfaces with the 'Connect_Call Component'. The result is shown in Fig. D.14.

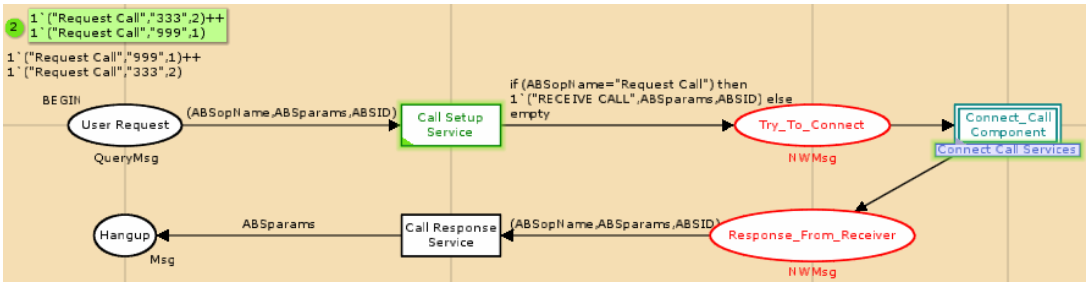


Fig. D.14 ' Abstracted Make_Call Component used for re-calculation of state space graph'

As Fig. D.14 shows, the 'Connect_Call Component' remains the same, i.e. no removal of its underlying decomposition has taken place. Model-checking was performed on the net of Fig. D.14 and then the same process was followed for the 'Connect_Call Component'. Its abstraction was slightly more complex than that for the 'Make_Call Component', largely due to the underlying function implemented by its controller component. The results of its abstraction are shown in Fig. D.15.

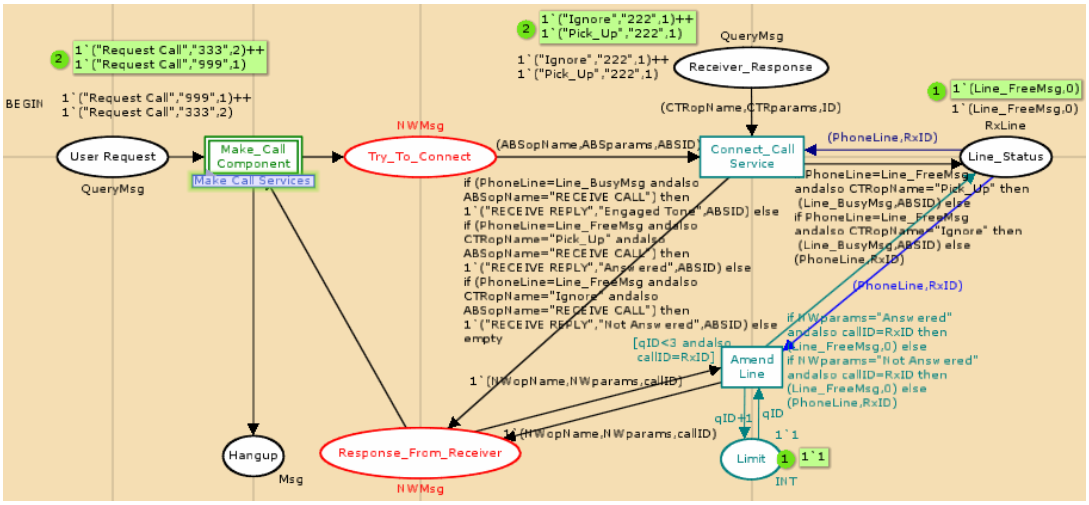


Fig. D.15 ' Abstracted Connect_Call Component used for re-calculation of state space graph'

Again, model-checking was performed on the net of Fig. D.15. The results of the state space calculations with the abstracted nets (along with the full hierarchical net based on Fig. D.1 for comparison purposes) are presented in Table D.1:

STATE SPACE GRAPH	Full Hierarchy (based on Fig. D.1)	Abstracted Make_Call Component (Fig. D.14)	Abstracted Connect_Call Component (Fig. D.15)
Initial marking	1`("Request Call","333",2)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","999",1)
Nodes and arcs	19656 nodes, 79164 arcs.	962 nodes, 2580 arcs.	2379 nodes, 6960 arcs.
Generation time	240 secs.	1 sec.	4 secs.
Terminal markings	9	9	9

Table D.1'Abstraction used in state space graph calculation at design and architecture level'

The exercise was repeated increasing the number of initiated calls to three:

STATE SPACE GRAPH	Full Hierarchy (based on Fig. D.1)	Abstracted Make_Call Component (Fig. D.14)	Abstracted Connect_Call Component (Fig. D.15)
Initial marking	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)
Nodes and arcs	Explosion problem.	30664 nodes, 119161 arcs.	Explosion problem.
Generation time	1200 secs (limit set).	938 sec.	2400 secs (limit set).
Terminal markings	N/A	20	N/A

Table D.2 'Abstraction used in state space graph calculation at design and architecture level'

From Table D.1 it can be seen that the effect of abstracting away the detail from one component and then the other is significant on duration and size of the state space graph calculation. The terminal markings were inspected from each of the abstracted component state space graph calculations in Table D.1 and found to match those determined by the full hierarchy state space graph calculation. From Table D.2, only abstraction of the 'Make_Call Component' has been successful in alleviating the state space explosion problem.

Similar to Appendices B-C, largeness avoidance using the component abstraction technique has helped alleviate the duration and size of the state space graph but it is important to note that when using the technique at the design and architecture level of abstraction, it was only successful for abstraction of each component in turn when two initiated calls were placed.

D.1.5 Largeness Avoidance by Composition

Based on the benefits reported using composition in largeness avoidance in Appendices B-C, the technique was investigated with the model developed for the design and architecture level of abstraction.

The hierarchy design for the design and architecture level of abstraction was reviewed to identify the components to isolate and the information exchange control sequence. The components at the lowest level of abstraction, i.e. those containing the logic detail are the user interface, controller, and transmit and receive (network) common components. Analysis of these would involve investigating them according to the service they are realising ('Call_Setup Service', 'Connect_Call Service', and 'Call_Response Service' respectively). Currently the four common components are described by four unique subnets which are re-used in ten instances across the three services.

The architecture level of abstraction is modularised, i.e. the architecture of the three services to investigate behaviour based on their use of the common components. In control sequence order they are: 'Call Setup Architecture', 'Process Call Architecture', and 'Call Response Architecture'.

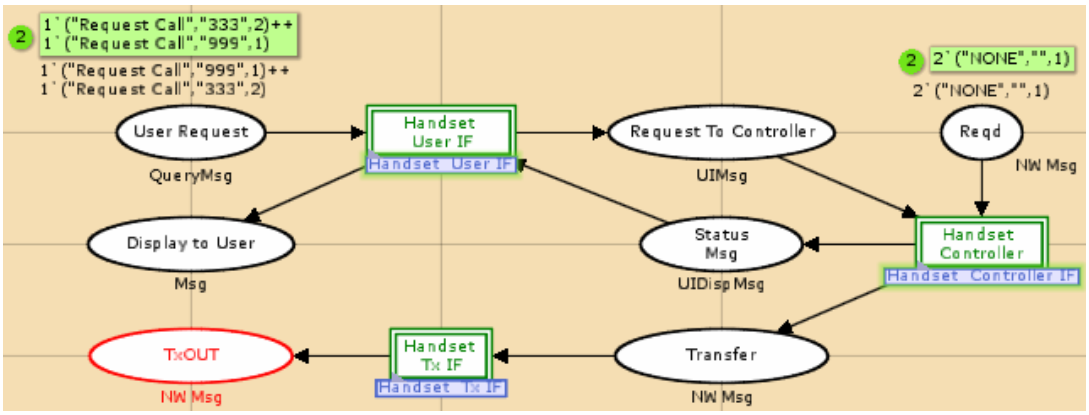


Fig. D.16 'Call Setup Architecture component'

'Call Setup Architecture' component (Fig. D.16) was isolated and examined first of all using two initiated calls. Subsequent analysis increasing the number of initiated calls indicated the presence of one terminal marking in each case until an initial marking of six calls was used. In the time limit set for state space graph calculation (twenty minutes), a full state space graph could not be calculated for six initiated calls. Where a full state space graph could be calculated, the output from this component was:

Initial marking *1`("Request Call","333",2)++ 1`("Request Call","999",1)*.

1. *1`("RECEIVE CALL","999",1)++1`("RECEIVE CALL","333",2)* at interface place 'TxOUT'.
2. *2`"Connecting"* at place 'Display_to_User.

The results from 'Call Setup Architecture' component's output interface place 'TxOUT' (from 1. above) were then used as input into the interface provided by the 'Process Call Architecture' component (Fig. D.17).

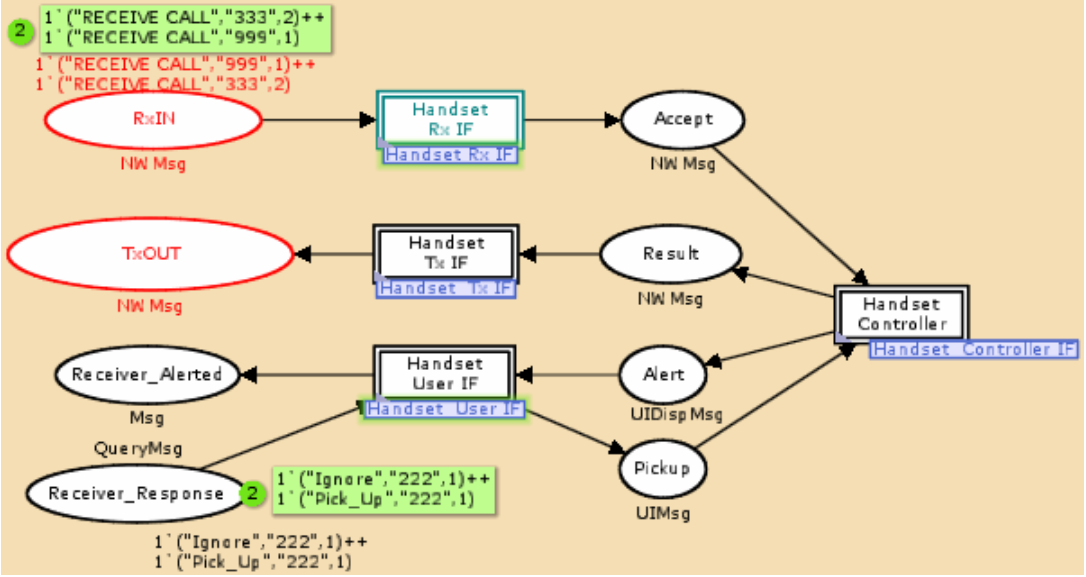


Fig. D.17 'Process Call Architecture component'

Model-checking was then performed on the 'Process Call Architecture' component and fourteen dead markings were reported. When these were examined, duplication of markings were identified on output interface place 'TxOUT'. This was due to the line reset logic in the controller common component. The transition to reset the line becomes enabled but its firing is not mandatory. This means that once the tokens that enable it are removed, it can no longer fire and the line can remain set in a busy state. With two input calls, three states of the line are possible and reported following static analysis. Even though the final marking on place 'TxOUT' is unchanged, the marking of the line reset place ('Line_Status') can have up to three potential markings, hence the duplication. The fourteen terminal markings were rationalised to a range of six possible markings on output interface place 'TxOUT':

1`("Answered",1)++ 1`("Not Answered",2)
1`("Not Answered",1)++ 1`("Engaged Tone",2)
1`("Answered",1)++ 1`("Engaged Tone",2)
1`("Answered",2)++ 1`("Not Answered",1)
1`("Not Answered",2)++ 1`("Engaged Tone",1)
1`("Answered",2)++ 1`("Engaged Tone",1)

These were then input into the last component in the control order sequence, 'Call Response Architecture' (Fig. D.18) on a pair by pair basis and static analysis performed.

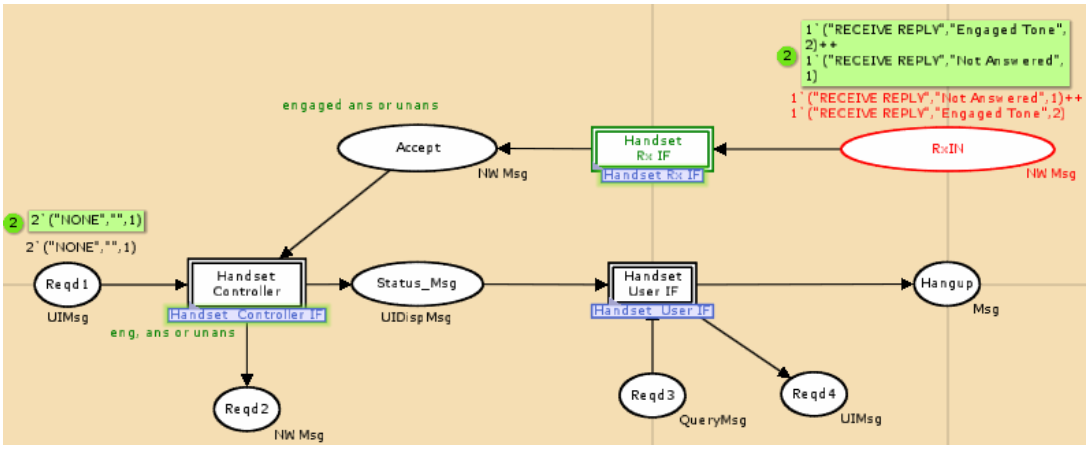


Fig. D.18 'Call Response Architecture component'

In the six cases, one terminal marking was obtained. The markings of output interface place 'Hangup' could be rationalised to three values:

1`"Answered"++1`"Not Answered"
1`"Not Answered"++1`"Engaged Tone"
1`"Answered"++1`"Engaged Tone"

These are the possible results of two calls placed to the same receiver with its line initially ready to accept calls and its user prepared to answer one call and ignore the other. Revisiting the results obtained using abstraction in largeness avoidance (Table D.1) and considering the nine dead markings obtained for two initiated calls, these can also be rationalised to the same three markings on place 'Hangup' due to the duplication caused by the line reset logic in the controller component.

The same process was repeated for three initiated calls. As for two initiated calls, the process of model-checking the first component and using its output as input into the 'Process Call Architecture' component's interface was straightforward. The result of model-checking 'Process Call Architecture' with its three input tokens was fifty-one dead markings. Rather than process these manually as done for two initiated calls, the markings for place 'TxOUT' were extracted using a non-standard branching temporal logic query into a file (Fig. D.19). Based on previous use of the compositional process, it was incorrect to simply input these markings into the final component's interface place as one multiset. Each marking on place 'TxOUT' for the fifty-one dead markings has to be processed as an independent initial marking to the component's input interface place. In Appendix B, each marking was input separately.

```
fun cond n = (Mark.Process_Call_Architecture'TxOUT 1 n <> empty);

fun ListTerminal()=PredNodes(ListDeadMarkings(),
 fn n => (cond n),
 NoLimit);
 let
  val fid = TextIO.openOut "DeadMarkingsfromResp.txt"
  val _ = TextIO.output(fid, "Markings on TxOUT: \n")
  val _ = EvalNodes(ListTerminal(),
    fn n => STRING.output(fid,st_Mark.Process_Call_Architecture'TxOUT 1 n) )
 in
   TextIO.closeOut(fid)
 end
```

Fig. D.19 'Customised query to export TxOUT markings to file'

Considering the net of the 'Call Response Architecture', two additional transitions and two places were added to the beginning and end places of the net. Execution of the first transition essentially provides input interface place 'RxIN' with a marking from a place 'Init Marking List', typed as a list of lists. This place interprets the markings exported to file from the 'Process Call Architecture' place, 'TxOUT' in batches. The second transition recognises that the results output to the 'Call Response Architecture' output interface place ('Hangup') relate to each batch of markings and captures them as such on place 'Results' typed as a list of compound elements. Finally, once the results are transferred to the list, the next batch of markings is requested via place 'Next Batch'.

The new places and transitions added to the net were verified using simulation. By using list functions it was possible to remove duplicate entries from the results list on place 'Results'. Based on a list of fifty-one batches consisting of three markings per batch, a rationalised output list indicated that three initiated calls have a possible five results:

"Answered"++"Not Answered"++"Answered"
"Answered"++"Engaged"++"Answered"
"Not Answered"++"Engaged"++"Answered"
"Engaged"++"Not Answered"++"Engaged"
"Answered"++"Engaged"++"Engaged"

Revisiting the results obtained using abstraction in largeness avoidance (Table D.2) and considering the twenty dead markings obtained for three initiated calls with 'Make_Call Component' abstracted, these can also be rationalised to the same five markings on place 'Hangup' due to the duplication caused by the line reset logic in the controller component.

Model-checking was then attempted for more thorough verification using one, two, and then five batches of three markings from the original list of fifty-one batches. The state space graph calculation reported three, nine, and two hundred and forty-three dead markings respectively (twenty-nine thousand nine hundred and seventy-five nodes, eighty-five thousand seven hundred and fifty-six arcs in two hundred and eighty-six seconds for the latter result). For each batch of three, the reason for the exponentially increasing terminal markings was that every possible ordering of the tokens present in the net and the corresponding output result were being recorded. So, for the batches of three tokens used above, there are 3^1, 3^2, and 3^5 possible

orderings of the output tokens on the results list. In terms of the node and arc numbers, the 'Reqd1' input interface place was initially marked to correspond to the number of markings on place 'Init Marking List'. The net was set up to take a maximum of one token from this place to enable the request operation within the controller component. This meant there could be a range of marking possibilities on two places using these tokens, driving up the number of nodes in the state space graph. The design of the net was revisited to see if there was a way to prevent these occurrences.

Interactive simulation was used to trace execution within the 'Call Response Architecture' net. In doing so, variable bindings of the execution could be viewed and selected. It was noted that ambiguity existed on several enabled transitions as to the value mapping a variable could take. For the 'Call Response Architecture', common components guard statements were added to the affected transitions in order to specify the bindings of variables. In the case of the 'Reqd1' input interface place, a list of lists type (similar to that used in 'Init Marking List' place) was defined so that the required information could be passed in batches of three to the controller component, matching the batches of markings. Where feasible to do so, the multiplicity of token removal (addition) from (to) places was changed so that instead of one token at a time being removed upon transition firing, a multiple (multiset) relating to the number of initiated calls was removed atomically.

The attempt to automate the interpretation of markings generated by one component for another appeared to behave as expected in simulation. When faced with exponential rises in terminal markings using only a small subset of the fifty-one batches of markings in static analysis, the design of the component's net was re-considered. Following the above amendments to make the specification within the net more precise (Fig. D.20), one terminal marking for all fifty-one batches of three markings was obtained. Through the compositional approach and use of dynamic and static analyses on component nets of the overall system, a net design that facilitated analysis of the system for an original initial marking of three calls was reached.

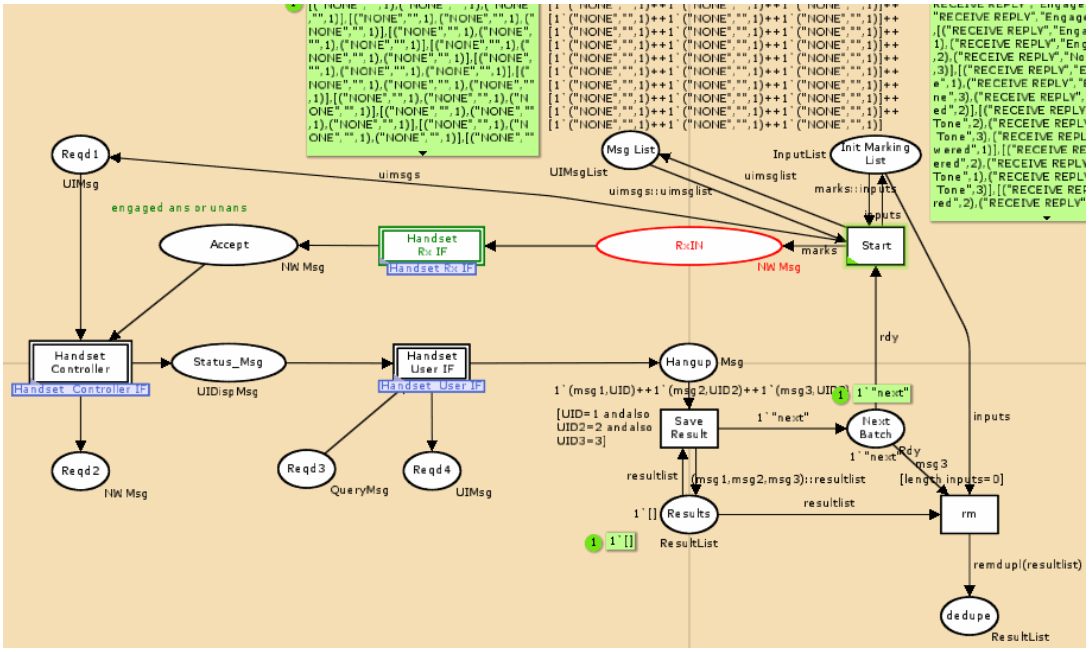


Fig. D.20 'Final version of Call Response Architecture net employing marking automation'

It should also be noted that the component nets of the design and architecture level consist of more net elements and logic than the component nets at the conceptual level of abstraction. As well as alleviating largeness avoidance in static analysis, the compositional approach can also contribute to comprehension and verification of the behaviour of component parts of the whole system using both dynamic and static analyses. Undertaking the process helped to improve the specification of one component within the model. From the component's static analysis results, the net had to be checked to see why it was producing exponential increases in terminal markings. The subsequent changes made contributed to a significant improvement in duration and size of the state space graph calculation as well as provide experience in best practice for the other nets. In addition, simulation of system component nets (rather than the net of the system as a whole) may make it more feasible for the modeller to detect, understand, improve and correct a greater proportion of behaviour than would be possible within a net of the whole system.

D.1.6 Integration of the Composition and Abstraction Approaches

Achievement of automation in the compositional approach enabled successful static and dynamic analyses of the final component net when three initiated calls were used as the initial marking. Going through the process of the compositional approach in section D.1.5 indicated that amendments to the precision of the specification of the final component would be relevant to the other two components (and in net construction in general).

More importantly, the results based on automation suggested a potentially beneficial integration of the compositional and abstraction approaches. If the results from the

compositional approach could be used to populate the interfaces associated with the abstracted component in the abstraction approach, the logic currently required by the abstracted component could be removed. The abstraction approach example where component 'Connect_Call Component' was abstracted (Fig. D.15) and the explosion problem was encountered for three initiated calls (Table D.2) was revisited.

The 'Call Response Architecture' component net was investigated to see how it could be integrated into the abstracted 'Connect_Call Component' net of Fig. D.15. 'Call Response Architecture' is the final component in control order sequence. It uses the information produced by the 'Process Call Achitecture' on its network interface place to help realise the 'Call_Response Service'. The 'Connect_Call Component' provides the 'Process_Call Service' realised by the 'Process Call Architecture'. The detail of this component can be abstracted out as long as it provides the 'Call_Response Service' and its underlying 'Call_Response Architecture' with the information identified above at the interfaces of the 'Call Response Architecture' component. This information needs to be supplied at the design level of abstraction by the 'Connect_Call Component'.

Before adding the markings, required interface information, and results lists (defined for the 'Call Response Architecture' net) to the design level of abstraction, some precision specification changes were made. The 'Call Response Architecture' common components were altered to match those used in the compositional approach. This involved replacing the instances of the common components with the revised common component nets of the 'Call Response Architecture'. No further changes were made to enhance the specification precision of the 'Call Setup Component' at this stage. It would be envisaged that the specification precision enhancements made to the 'Call Response Architecture' could be applied across the nets and use made again of four common component nets and their instances. Model-checking was performed again on the abstracted 'Connect_Call Component' based on its updated specification. The results are shown in Table D.3.

Input interface place 'Reqd1' of the 'Call Response Architecture' revised controller component was added and used at the design level of abstraction in order to pass it input interface information in line with release of batches of markings. Once these additions were made to the net, simulation was used to verify its behaviour was as expected before conducting static analysis. The model-checking results are shown in Table D.3.

STATE SPACE GRAPH	**Original Abstracted Connect_Call Component (Fig. D.15)**	**Abstracted Connect_Call Component with Specification Update**	**Fully Abstracted Connect_Call Component**
Initial marking	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)	1`("Request Call","333",2)++ 1`("Request Call","444",3)++ 1`("Request Call","999",1)
Nodes and arcs	Explosion problem.	5596 nodes, 19568 arcs.	1711 nodes, 5522 arcs.
Generation time	2400 secs (limit set).	19 secs.	2 secs.
Terminal markings	N/A	51	1

Table D.3 'Abstraction and Composition approaches used in state space graph calculation'

The results show that not only has a state space graph with fifty-one dead markings been calculated in nineteen seconds for an initial marking of three calls using only the specification updates, a fully abstracted 'Connect_Call Component' improves upon this. With all logic removed from the abstracted 'Make_Call Component' and the addition of the automation used in the compositional approach, a state space graph with one dead marking was completed in two seconds. This graph contained approximately one quarter of the nodes and arcs of the updated specification graph.

At this stage the compositional approach has provided three major benefits:

1. Identification of improvements to the specification. When applied to the original 'Connect_Call Component' abstraction net, the improvements enabled the state space graph calculation to terminate within the toolset limit and investigate the dead markings based on three initiated calls.
2. Integration with the abstraction approach to further reduce duration and size of the state space graph calculation. Input and output interface results identified by the compositional approach can be used in the abstraction approach to keep the component as abstract as possible.
3. A means of further verifying the behaviour of the model where either a full state space graph cannot be calculated for the overall system net or further assurance is sought as to correctness of the modelled behaviour. The abstraction approach could be applied and model-checked as demonstrated in Appendices B-C. The compositional approach could then be used to indicate input and output interface results, increase comprehensibility of component nets, and advise on improvements. These improvements could then be applied to the abstracted nets along with the automated results suggested by composition and model-checked.

D.1.7 Validation of the Design and Architecture Levels

In Appendix C, the concept of time was introduced into the untimed model at the conceptual level of abstraction. Timing information could further enhance the specification of the process (for example, controlling the ordering of calls, and implementing communication timeouts) and be used to check if the design of the process was efficient in terms of time and cost from particular viewpoints. The same justifications for use of timing apply at the design and architecture levels of abstraction. Timing can be used again to detail ordering of calls, communication timeouts, component processing duration times, and analysis-of-alternatives.

D.2 Conclusions from Design and Architecture Levels

Using the compositional approach at the design and architecture levels of abstraction has provided further insight into its potential usefulness in the development of Petri net models of large-scale systems. By decomposing the system model into subnets, not only is the approach trying to combat state space explosion, it promotes increased comprehension of parts of the overall system and consideration of their integration at well-defined interfaces. Used in conjunction with the abstraction approach, it could help reduce state space graph duration and size further. To be successful, a suitable hierarchy of models and a suitable hierarchy within models need to be adopted. In Appendices A-D, a functional decomposition approach was used to determine

abstraction levels within models. In addition, levels of abstraction were identified for models themselves, i.e. conceptual level and design and architecture levels.

Modelling at the conceptual level helped to improve understanding of the domain and the Petri net technique. Initially, regardless of model level of abstraction, it was beneficial to aim for simple functionality within the nets, ensure these nets were correct syntactically (using toolset's syntax checking), and then use simulation to detect structure and logic errors. Based on experience so far, it is vital to keep nets as compact as possible, rationalising and partitioning their elements properly. Static analysis is used at this point to highlight issues that are not made explicit by simulation. Once this level of net maturity is reached, a net was then evolved further, for example adding line reset logic, or timing.

In large-scale system-of-systems, their design and architecture requires specification of a combination of independent component systems. Components used at the design level in the telephone system example could be considered to be component systems in a system-of-systems. The system-of-systems architecture level details how the component systems and their realising common components communicate information across interfaces to realise services of the component system.

Model informational content at the design and architecture levels is different to that at the conceptual level but is still represented using the same set of Petri net elements. Function and the processes and sequence of activities that need to be followed to realise the function are not the focus of the design and architecture levels. Although there is still a control order sequence, this level details the physical component (not necessarily the real-life asset) and how each component combines to realise the input and output behaviour of the conceptual level. At the design and architecture level, there appears to be additional scope for: identification of common components and instantiation (although a process could be re-used by functions at the conceptual level); the ability to describe the functions made use of by each component more explicitly; and the ability to make the use of a communications network (and gateways) more explicit.

Abstraction levels (hierarchy) and their facilitation using the toolset socket and port places are the key to system-of-systems' specification, verification and validation using nets. The port and socket places modularise the model and describe operations or physical components in varying degrees of detail (with the greatest detail being at the lowest abstraction level). Like ports and sockets provide the boundaries of modules in nets, interfaces are the boundaries of component systems in a system-of-systems. These interfaces can be used at and between different abstraction levels to communicate information. The hierarchy can be used to divide a system-of-systems model into more manageable models so that abstraction and composition approached can be used to understand and verify them.

Appendix E

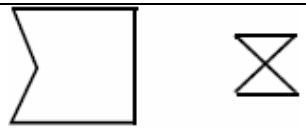
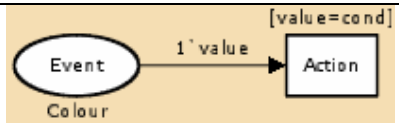
Enhancement to UML based on Petri Nets

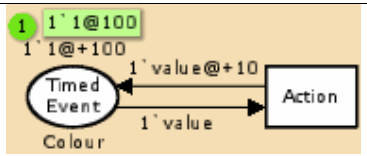
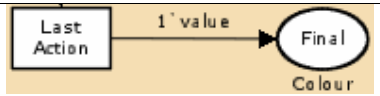
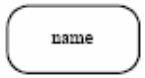
E.1 Introduction

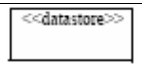
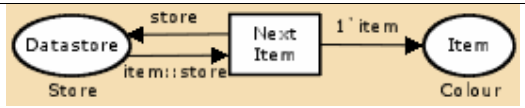
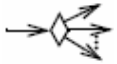
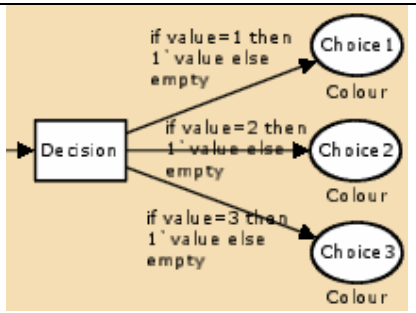
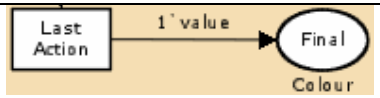
This appendix describes the mapping made from UML to Petri nets for the thesis system-of-systems specification and analysis problem. It uses a combination of natural language and syntax diagrams to describe a Petri net systems-of-systems specification language enhancement to UML.

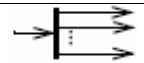
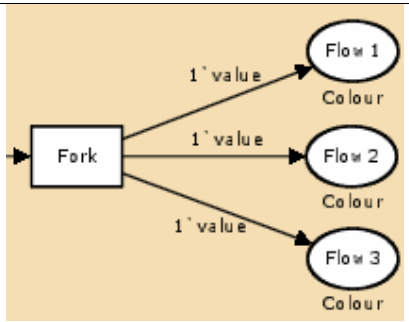
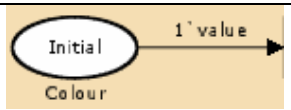
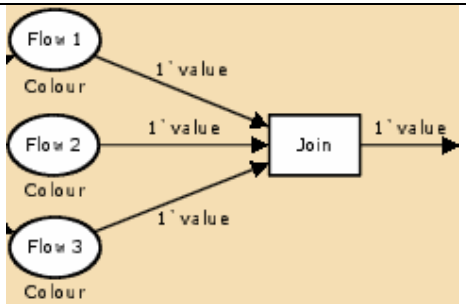
E.2 Definition of the System-of-Systems Specification End Language

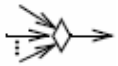
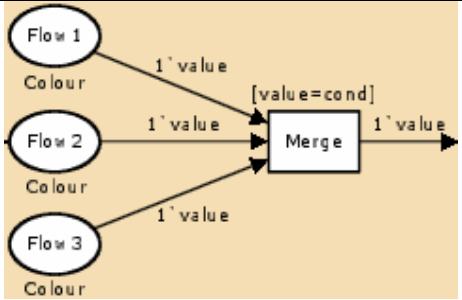
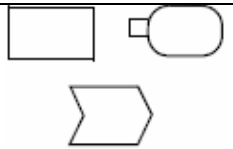
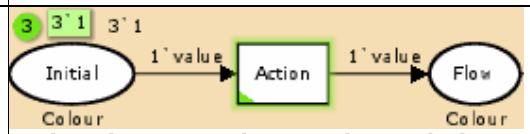
The following table presents the replacements and additions made within UML activity diagrams using Petri net constructs for the specification of systems-of-systems. The concrete syntax and description of the semantics (in natural language) are given for the replaced UML activity diagram element and the new Petri net enhancement element. For the UML activity diagram, concrete syntax and description are taken from [125].

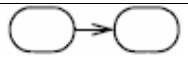
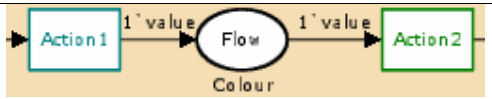
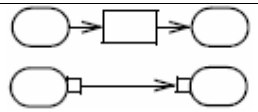
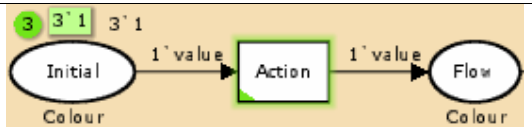
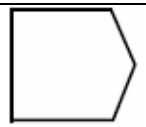
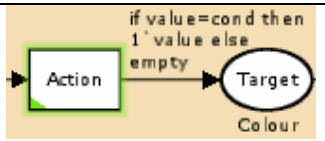
UML Activity Diagram Nodes	**Concrete Syntax & Description [125]**	**Petri Net Element(s)**
AcceptEventAction	 An AcceptEventAction is an action that waits for the occurrence of an event meeting specified conditions.	 Coloured Petri net elements place, input arc, transition, transition guard, and arc inscriptions are used to specify AcceptEventAction. Transition 'Action' is enabled when one token with a certain value ('cond') is available to be removed from place 'Event' (of type 'Colour') and bound to variable 'value'.

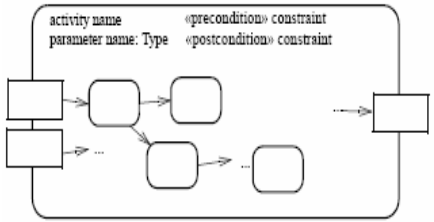
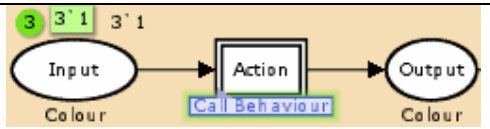
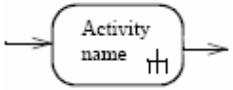
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
		 Timed Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions are used to specify a repetitive time event AcceptEventAction. Transition 'Action' is enabled when model time reaches '100' and one token is available to be removed from place 'Timed Event' (of type 'Colour') and bound to variable 'value'. The transition fires, producing a new timed token for place 'Timed Event' equal to model time plus '10'. This token timestamp determines the re-enabling of transition 'Action'.
ActivityFinalNode	 An activity final node is a final node that stops all flows in an activity.	 Coloured Petri net elements place, output arc, transition, and arc inscription are used to specify ActivityFinalNode. Place 'Final' (of type 'Colour') receives the one token output by transition 'Last Action' on its output arc. There are no additional net elements connected from place 'Final'.
Action	 An action represents a single step within an activity, that is, one that is not further decomposed within the activity. An action may have sets of incoming and outgoing activity edges that specify control flow and data flow from and to other nodes. An action will not begin execution until all of its input conditions are satisfied.	 Coloured Petri net element, transition, is used to specify Action. A transition will not execute unless all of its input conditions are satisfied i.e. the number of tokens (and potentially their value) specified by the inscriptions on its input arcs are present on the associated input places.

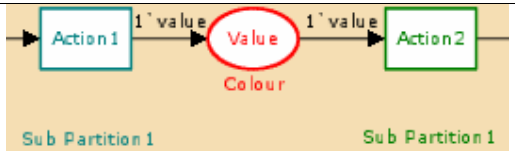
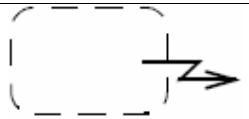
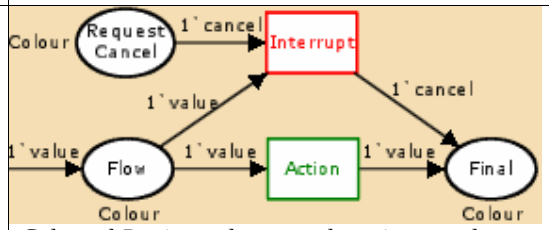
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
DataStore	 A data store keeps all tokens that enter it, copying them when they are chosen to move downstream.	 Coloured Petri net elements place and place type (list) are used to specify DataStore. Additional elements input and output arcs, transition, and arc inscriptions are used to specify an example of retrieval and update of DataStore items.
DecisionNode	 A decision node accepts tokens on an incoming edge and presents them to multiple outgoing edges. Which of the edges is actually traversed depends on the evaluation of the guards on the outgoing edges.	 Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions are used to specify DecisionNode. Execution of transition 'Decision' uses arc inscriptions to check the consumed token's content before copying one new token to one of the three output places.
FlowFinal	 A flow final destroys all tokens that arrive at it.	 Coloured Petri net elements place, output arc, transition, and arc inscription are used to specify FlowFinal. Place 'Final' (of type 'Colour') receives the one token output by transition 'Last Action' on its output arc. There are no additional net elements connected from place 'Final'.

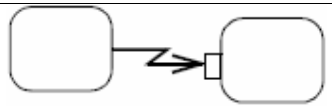
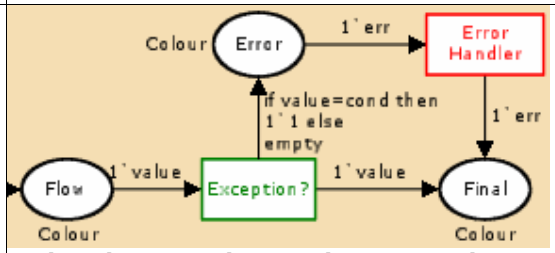
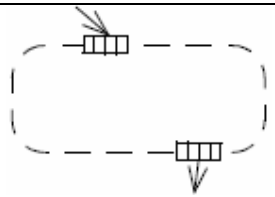
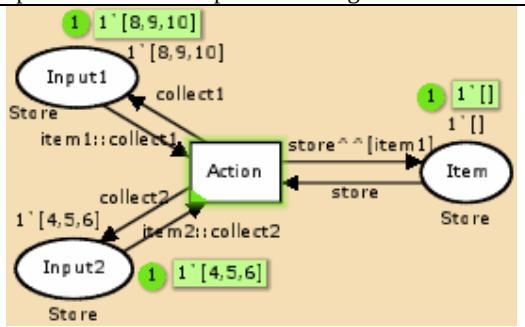
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
ForkNode	 A fork node has one incoming edge and multiple outgoing edges.	 Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions are used to specify ForkNode. Execution of transition 'Fork' uses arc inscriptions to copy one new token to all three output places.
InitialNode	 An activity may have more than one initial node.	 Coloured Petri net elements place, output arc, and arc inscription are used to specify InitialNode. There are no additional net elements preceding place 'Initial'.
JoinNode	 A join node has multiple incoming edges and one outgoing edge.	 Coloured Petri net elements place, input and output arcs, transition, and arc

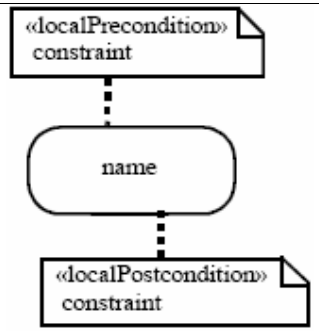
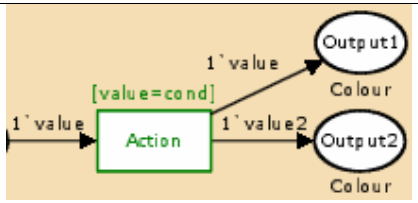
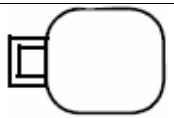
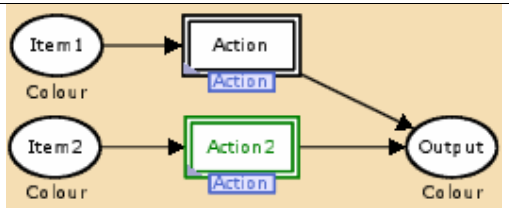
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
		inscriptions are used to specify JoinNode. Execution of transition 'Join' uses an arc inscription to consume three input tokens and copy one new token along its output arc.
MergeNode	 A merge node has multiple incoming edges and a single outgoing edge. It is not used to synchronize concurrent flows but to accept one among several alternate flows.	 Coloured Petri net elements place, input and output arcs, transition guard, transition, and arc inscriptions are used to specify MergeNode. Execution of transition 'Merge' uses arc inscriptions to consume three input tokens when its transition guard condition is met and copy one new token along its output arc.
ObjectNode	 An object node is an activity node that indicates an instance of a particular classifier, possibly in a particular state, may be available at a particular point in the activity.	 Coloured Petri net elements place and place type ('Colour') are used to specify ObjectNode. Additional elements input and output arcs, transition, and arc inscriptions are used to specify an example of an ObjectNode. Transition 'Action' is enabled when one token is available to be removed from place 'Initial' (of type 'Colour') and bound to variable 'value'. One new token is then copied to place 'Flow' (of type 'Colour').
UML Activity Diagram Paths		

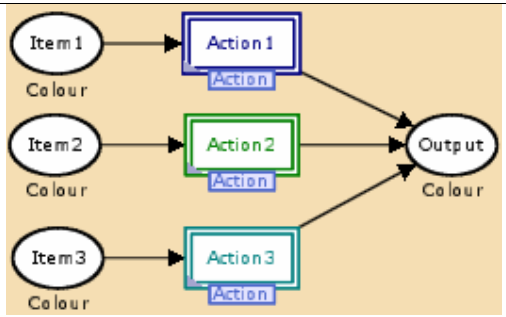
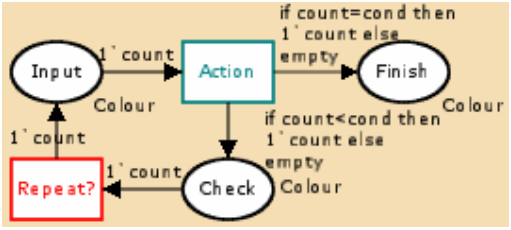
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
ControlFlow	 A control flow is an edge that starts an activity node after the previous one is finished.	 Coloured Petri net elements input and output arcs, and arc inscriptions are used to specify ControlFlow. Additional elements place and transition are used to specify an example of ControlFlow. Transition 'Action1' executes to produce one token for place 'Flow' (of type 'Colour') bound to variable 'value'. Control then passes to transition 'Action2' which becomes enabled when one token is available on place 'Flow' (of type 'Colour').
ObjectFlow	 An object flow models the flow of values to or from object nodes.	 Coloured Petri net elements input and output arcs, and arc inscriptions are used to specify ObjectFlow. Additional elements place and transition are used to specify an example of ObjectFlow. Transition 'Action' executes to consume one token from place 'Initial' (of type 'Colour') bound to variable 'value'. Flow then passes to output place 'Flow' (of type 'Colour') when one token is copied to it following execution of transition 'Action'.
SendSignalAction	 SendSignalAction is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity.	 Coloured Petri net elements place, input and output arcs, transition, and arc inscription are used to specify SendSignalAction. Transition 'Action' executes and an arc inscription compares the consumed token's value ('cond') to determine the new token to copy to place 'Target' (of type 'Colour'). Additional net elements can be used to specify the execution of an activity dependent on the token copied to place 'Target'.

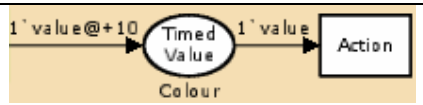
UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
UML Activity Diagram Elements		
Activity & ActivityParameterNode & CallBehaviourAction		 Hierarchical Coloured Petri net elements transitions and substitution transitions are used to specify Activity, ActivityParameterNode, and CallBehaviourAction. Additional elements place, and input and output arcs are used to specify an example of Activity, ActivityParameterNode, and CallBehaviourAction. Transition 'Action' is further decomposed by subnet 'Call Behaviour'. 'Action' has two socket interface places, 'Input' and 'Output' used by subnet 'Call Behaviour' to consume and produce tokens on its associated port places (of type 'Colour'). Subnet 'Call Behaviour' uses net elements to further specify the 'Action' activity.
		
	An activity specifies the coordination of executions of subordinate behaviours, using a control and data flow model. Activity parameter nodes are object nodes at the beginning and end of flows that provide a means to accept inputs to an activity and provide outputs from the activity, through the activity parameters. CallBehaviorAction is a call action that invokes a behaviour directly rather than invoking a behavioural feature that, in turn, results in the invocation of that behaviour.	

UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
ActivityPartition	 Partitions divide the nodes and edges to constrain and show a view of the contained nodes.	 Toolset features of annotation and colouring are used to specify ActivityPartition. Additional Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions are used to specify an example of ActivityPartition. Transition 'Action1' is associated with 'Sub Partition 1' and transition 'Action2' is associated with 'Sub Partition 2'.
InterruptibleActivityRegion	 An interruptible region contains activity nodes. When a token leaves an interruptible region via edges designated by the region as interrupting edges, all tokens and behaviours in the region are terminated.	 Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions as well as toolset colouring are used to specify InterruptibleActivityRegion. Presence of tokens on places 'Request Cancel' and 'Flow' (of type 'Colour') enable execution of transition 'Interrupt' and prevent execution of transition 'Action'.

UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
ExceptionHandler	 An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node.	 Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions as well as toolset colouring are used to specify ExceptionHandler. Execution of transition 'Exception?' checks for an exception condition and copies one token to place 'Error'. Presence of one token on place 'Error' enables transition 'Error Handler' and further specification of exception handling if desired.
ExpansionRegion	 An expansion region is a strictly nested region of an activity with explicit input and outputs (modelled as ExpansionNodes).	 Coloured Petri net elements place and place type (list) are used to specify ExpansionRegion. Additional elements input and output arcs, transition, and arc inscriptions are used to specify an example of retrieval, update, and processing of two collections (type list). The transition 'Action' is enabled when one token of type 'Store' is present on places 'Input1' and 'Input2'. Transition 'Action' executes, consuming one token containing the first item in the list (type 'Store') from places 'Input1' and 'Input2'. One token is then copied to place 'Item' (of type 'Store').

UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
Local pre- and postconditions	 Local pre- and post-conditions are constraints that should hold when the execution starts and completes, respectively.	 Coloured Petri net elements place, input arc, transition, transition guard, and arc inscriptions as well as toolset colouring are used to specify Local pre- and post-conditions. Transition 'Action' is enabled when one token with a certain value ('cond') is available (the pre-condition). One token is copied to places 'Output1' and 'Output2' (the post-condition).
ParameterSet	 A parameter set acts as a complete set of inputs and outputs to a behaviour, exclusive of other parameter sets on the behaviour (express 'or' invocation).	 Hierarchical Coloured Petri net element substitution transitions as well as toolset colouring and instantiation features are used to specify ParameterSet. Additional elements place, and input and output arcs are used to specify an example of ParameterSet. Transitions 'Action' and 'Action2' are further decomposed by the same subnet 'Action'. 'Action' has two socket interface places, 'Item' and 'Output' used by subnet 'Action' to consume and produce tokens on its associated port places (of type 'Colour'). Subnet 'Action' uses net elements to further specify the 'Action' activity. Either or both of transitions 'Action' and 'Action2' can become enabled by the presence of a token on places 'Item' and 'Item2'.
CONCEPTS		

UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
Multiplicity	Expansion nodes & decision nodes.	 Hierarchical Coloured Petri net element substitution transitions as well as toolset colouring and instantiation features are used to specify multiplicity. Additional elements place, and input and output arcs are used to specify an example of multiplicity. Transitions 'Action1', 'Action2', and 'Action3' are further decomposed by the same subnet 'Action'. 'Action' has two socket interface places, 'Item' and 'Output' used by subnet 'Action' to consume and produce tokens on its associated port places (of type 'Colour'). Subnet 'Action' uses net elements to further specify the 'Action' activity. Transitions 'Action1', 'Action2', and 'Action3' can become enabled by the presence of a token on places 'Item1', 'Item2', and 'Item3' and then follow the execution sequence specified by the same subnet, 'Action'.  Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions as well as toolset colouring can also be used to specify multiplicity. Following transition 'Action' execution, output arc inscriptions

UML Activity Diagram Nodes	Concrete Syntax & Description [125]	Petri Net Element(s)
		check the content of consumed token 'count' and depending on its value one token is copied to place 'Finish' or 'Check' (the start of another iteration of the execution sequence).
Timing	SimpleTime subpackage of CommonBehaviors package [125] and Modelling and Analysis of Real-time and Embedded systems (MARTE) profile [113].	 Timed Coloured Petri net elements place, input and output arcs, transition, and arc inscriptions are used to specify an example of timing in models. Transition 'Action' is enabled when model time reaches current model time plus '10' and one token is available to be removed from place 'Timed Value' (of type 'Colour') and bound to variable 'value'.
Model execution	None.	Toolset well-defined algorithm.
Model reachability graph	None.	Toolset well-defined algorithm.

E.3 Summary

This appendix has provided a definition of the mappings made to UML activity diagrams based on Petri nets for the specification of systems-of-systems.

Appendix F

Petri Net Toolset Selection Exercise

F.1 Introduction

This appendix describes the process of selecting a Petri net development framework for the purposes of system-of-systems specification and analysis in this thesis.

F.2 Selection Process

Obtaining a comprehensive Petri net toolset can be achieved in three ways: developing the Petri net toolset in-house (this ensures all personal requirements are met but a disadvantage includes the time and effort involved. This effort can be short-circuited if there are suitable extensible frameworks available on which to build); compiling a toolset from existing Petri net analysis and graphical editing tools (again, a disadvantage is the time and effort involved in integrating the tools); or identifying a suitable existing integrated Petri net toolset and adapting it accordingly (this relies on the toolset being open and well supported in terms of documentation).

Given the time and resources available to the thesis, the latter option was chosen as the way forward. A list of criteria was identified as the basis for selecting potential Petri net toolsets for further evaluation. The list included: ability to execute Petri nets; close integration with UML; rapid, lightweight; extensible; free or low cost; intellectual accessibility; representation of service-based architecture; robustness; and networkability.

F.2.1 Selection of Toolsets for Further Evaluation

An internet Petri net toolset survey was conducted and toolsets were selected according to their latest version/maintenance release. The features listed for these toolsets did not indicate whether the toolset met all the requirements of the thesis. Reflecting on the criteria list, four toolsets were selected for further evaluation using the following features as a minimum of functionality: a graphical editor; an interactive simulator (with performance analysis capability); currency in terms of maintenance and support (including ease of installation and product stability); and a free or low cost license. Four Petri net toolsets were identified [47, 126, 127, 128] and evaluated during July/August 2008. The functionality specific to each is summarised in Table F.1.

TOOL	Features	Comments
CPN Tools **Research group (free to universities)**	High-level Petri nets support. Tried & tested?	Timed Petri nets, Coloured Petri nets, Hierarchical Petri nets. Many published papers/test cases (at least 100 papers, over 5000 licences).

TOOL	Features	Comments
	Abstraction support.	Via sub-pages, supports both top-down and bottom-up development.
	Logic support.	Via functional programming language (CPN ML based on Standard MetaLanguage, SML).
	On-the-fly-execution.	Supports manual triggering of transitions, semi-automatic triggering of a number of transitions, and automatic replication runs.
	Intuitive graphical interface.	Workspace personalisation, context menus.
	Model editing.	Several time-saving mechanisms.
	Syntax-checking of model.	Automatic correctness verification.
	State-space analysis.	Highlights dead transitions (operations not used) and dead markings (potential design error), use of built-in or custom queries to investigate state-space using CPN ML code, partial state-space verification in large models.
	Performance analysis.	Multiple simulation runs and statistical data extraction via monitors/text-based log files.
	CTL model checker extension.	Via ASK_CTL [130] state space analysis.
	Animation framework extension.	Via BRITNeY [129], supporting 2D/3D graphical representation, Message Sequence Charts, High Level Architecture (with BRITNeY).
	Graphing support extension.	Via Graphviz [131].
	Interchange file format.	XML (own Document Type Definition, DTD).
WoPeD **Research group (free to universities)** **Open source**	High-level Petri nets support.	Timed Petri nets (notation and ability to insert time for subprocesses), Hierarchical Petri nets, Predicate/Transition Petri nets, Workflow nets.
	Tried & tested?	Less than 10 papers.

TOOL	Features	Comments
	Abstraction support.	Via subprocess transition pages, supports top-down and bottom-up development.
	Logic support.	Percentage probability (notation present but do not believe it is fully implemented) and XOR/AND joins/splits.
	On-the-fly-execution.	Supports manual triggering of transitions, and automatic replication runs.
	Intuitive graphical interface.	Context menus.
	Model editing.	Suggests place/transition.
	State-space analysis.	Uses Woflan application (Workflow Analysis tool that checks if Petri Net conforms to Workflow definition) to determine whether process definition is a workflow, highlights whether all conditions in the process are proper, highlights whether all tasks in the process are not dead (operations not used), and highlights whether all tasks in the process are live, no use of built-in or custom queries to investigate state-space.
	Performance analysis.	Multiple simulation runs and statistical data extraction via pre-formatted logfile (CSV export possible).
	Graph support extension.	Ability to use diagrams option within simulation or via JGraph enhancement.
	Interchange file format.	XML, PNML.
Renew **Research group (free to universities)** **Open source**	High-level Petri nets support.	Object-oriented Petri nets, Coloured Petri nets, Timed Petri nets, Reference nets.
	Tried & tested?	At least 30 papers.
	Abstraction support.	Reference nets (synchronous channels).
	Logic support.	Java inscriptions.
	On-the-fly-execution.	Supports manual triggering of transitions, and automatic runs.
	Intuitive graphical interface.	Uses one central command

TOOL	Features	Comments
		window which works with one active editing window.
	Model editing.	Uses JHotDraw library. Suggestion of places/transitions.
	Syntax-checking of model.	Automatic correctness verification.
	State-space analysis.	Requires third party tool to perform analysis.
	Performance analysis.	Interactive and dynamic simulation only (would require enhancement).
	Animation framework extension.	Basic animation support via icons.
	Interchange file format.	XML, PNML.
Platform Independent Petri Net Editor 2 **Research group (free to universities)** **Open source**	High-level Petri nets support.	Petri nets extended with time (stochastic) and Predicate/Transition Petri nets.
	Tried and tested?	Less than 10 papers.
	Logic support.	Via weightings only.
	On-the-fly-execution.	Supports manual triggering of transitions, semi-automatic triggering of a number of transitions, and automatic replication runs (via module).
	Intuitive graphical interface.	Context menus.
	Model editing.	Basic functionality.
	State-space analysis.	Via provided module.
	Performance analysis.	Basic as it stands (would require enhancement via a module).
	Interchange file format.	PNML.

Table F.1 'The four selected toolsets and their features'

F.2.2 Comparison of Toolsets

Following usage of these toolsets, more detailed requirements were identified and translated into a list of criteria to aid further comparison of the four toolsets based on a similar process to the one presented in [132]. Ratings of aspects of desirable features for each toolset were given as 'excellent', 'good', 'fair', 'poor', and 'unsupported'. Each of these ratings has an associated point score ranging from four for 'excellent' down to

zero for 'unsupported'. In addition, the aspects were given an importance weighting of one to four (ranging from 'useful' to 'mandatory' respectively).

In terms of toolset ease-of-use, all tools had an intuitive installation procedure and were straightforward to install. All four toolsets offered a typical graphical user interface and a stable product integrating editing, simulation and analysis functions. As such, no further evaluation of these aspects was undertaken.

Regarded as one of the most critical parts of an integrated Petri net toolset, the graphical editor produces the model used in verification and validation. There are several key aspects of editors to evaluate: support for the documentation of models (e.g. export formats, automatic report generation from models); support for model layout (e.g. automatic layout via a drawing grid, alignment of model elements, ability to add text, colour, style, sizing and charts to models); support for syntax construction and checking during model editing (e.g. provision of context sensitive values and menus, provision of explicit model checking command, provision of default modelling values, not permitting place-to-place or transition-to-transition connection using arcs, warning users when undefined types are associated to places); ability to customise the graphical appearance of the tool and models and their navigation (e.g. hiding of inscriptions); ability to print (e.g. formats, all or part of a model); helpfulness of error notification; and the management of model versions. Results of the evaluation are shown in Table F.2.

Toolset	**Support for**			**Customisation of appearance**	**Printing**	**Syntax Checking**	**Error Notification**	**Version Control**
	Documentation	Layout	Syntax Building					
CPN Tools	Good.	Good.	Good.	Good.	Fair.	Good.	Fair.	Un-Supported.
WoPeD	Fair.	Fair.	Fair.	Poor.	Fair.	Poor.	Poor.	Un-supported.
Renew	Fair.	Fair.	Good.	Fair.	Good.	Good.	Fair.	Un-supported.
PIPE	Fair.	Fair.	Fair.	Fair.	Fair.	Poor.	Poor.	Un-supported.
Importance Weighting	2	1	2	2	3	4	4	2

Table F.2 'Evaluation of graphical editor key aspects for each toolset'

Critical to the thesis was the ability of the toolset to execute a model. Features essential for useful simulation are evaluated in Table F.3. These include: inclusion of break and watch points (e.g. ability to stop simulation dependent on certain conditions); support of various simulation modes (e.g. interactive ability to step through simulation, batch runs); ability to generate stand-alone simulation code from model; ability of toolset to generate animations based on model execution (e.g. simple, 'token game' or more advanced animation via GUIs). The results are shown in Table F.3.

Toolset	Simulation		Modes	Watch Points	Break Points	Code Generation	Animation	
	Interaction	Rating	Batch				Simple	Advanced
CPN Tools	Single step. Continuous.	Excellent.	Excellent.	Good.	Good.	Un-supported (Beta programming Language).	Good.	Good (BRITNeY).
WoPeD	Single step.	Fair.	Un-Supported.	Un-Supported.	Un-Supported.	Un-supported.	Good.	Un-supported.
Renew	Single step. Continuous.	Fair.	Un-Supported.	Un-Supported.	Un-Supported.	Un-supported.	Good.	Poor.
PIPE	Single step. Continuous.	Good.	Fair.	Un-Supported.	Un-Supported.	Un-supported.	Good.	Un-supported.
Importance Weighting		3	4	2	2	2	3	1

Table F.3 'Evaluation of simulation key aspects for each toolset'

Model analysis is another important tool in an integrated Petri net toolset. Aspects considered in the evaluation are tool support for reachability, liveness, fairness, temporal logic, support for results presentation (e.g. graphing, printing, export for further analysis), and support for other forms of analysis. These are shown in Table F.4.

Toolset	Reachability	Liveness	Fairness	Temporal Logic	Invariants	Statistical Analysis	Results Present-ation	Others	
								Type	Rating
CPN Tools	Excellent.	Excellent.	Excellent.	Excellent.	Un-supported.	Good.	Good.	Home state. Boundedness.	Excellent
WoPeD	Excellent.	Excellent.	Excellent.	Un-supported.	Un-supported.	Un-supported.	Fair.		
Renew	Un-supported.	Un-supported.	Un-supported.	Un-supported.	Un-supported.	Un-supported.	Un-supported		
PIPE	Excellent.	Excellent.	Excellent.	Un-supported.	Excellent.	Fair (Dnamaca).	Fair.	Home state. Boundedness. Comparison. DNAmaca.	Good.
Importance Weighting	4	4	3	3	3	4	3		2

Table F.4 'Evaluation of analysis key aspects for each toolset'

The level of support and potential future development of the toolsets are additional important aspects to evaluate. Here, toolset support covers provision of documentation (quality and quantity of user-oriented and technically-oriented documentation) and technical support (via internet, training, email and telephone). Assessment of future development is indicated along with a rating for the likelihood of these enhancements to functionality/usability taking place. The likelihood rating can be inferred from toolset usage (e.g. number of licenses, number of research groups involved, existing quality of toolset). Again, evaluation results are represented in Table F.5.

Toolset	Support	Future Development	Likelihood Rating	Tutorial/ Examples	Help	Documentation
CPN Tools	Excellent.	Next generation tool support for State Space Analysis	Good.	Good.	Good.	Good.

		[69].				
WoPeD	Fair.	No public details.	Fair (last release April 2008).	Poor.	Fair.	Fair.
Renew	Fair.	No public details.	Fair (last release July 2008).	Fair.	Fair.	Fair.
PIPE	Good.	Hierarchical Nets.	Fair (last release December 2007).	Fair.	Fair.	Fair.
Importance Weighting	3		2	3	2	3

Table F.5 'Evaluation of support/future development key aspects for each toolset'

One of the requirements outlined at the start of this document was for toolsets to support extensibility. Important properties of extensibility include the ability to import/export models and analysis results and the openness of a toolset, i.e. the ease of making modifications to the toolset to meet requirements. Openness relates to knowing the interface specifications at the architectural module level (through technical documentation). Table F.6 summarises the openness of the four toolsets.

Toolset	**Formats**		Rating	**Technical Documentation**	**Programming interface**	
	Import	Export			Type	Rating
CPN Tools	XML.	EPS, XML.	Good.	Good.	CPN ML based on SML.	Good.
WoPeD	XML, PNML.	PNG, JPG, BMP, TPN (Woflan), PNML.	Good.	Poor.	Java.	Good.
Renew	XML, PNML.	PS, EPS, XML, PNML.	Good.	Poor.	Java.	Good.
PIPE	XML.	PS, PNG, XML.	Good.	Poor.	Java.	Good.
Importance Weighting	3			3	3	

Table F.6 'Evaluation of openness aspects for each toolset'

F.2.3 Final Ranking of Toolsets

Similar to the approach used in [132], ranking of the most important features of the toolsets was made by applying a priority (importance weighting) to each aspect of the feature and a point system to the given evaluation rating of the aspect (for example, as indicated earlier, a 'good' rating would be worth three points). For each feature, a maximum scoring can be calculated based on the priorities given to its aspects in Tables F.2-F.6. This maximum scoring is shown in Table F.7 and can be viewed as a toolset profile for application in system-of-systems development. A score is then calculated for each feature by toolset. For example, for its 'Graphical Editor' feature,

CPN Tools received five 'good' (with corresponding priorities of 2, 1, 2, 2, and 4) and two 'fair' ratings (with corresponding priorities of 3 and 4) for the editor aspects, equating to 47 points (2*3 + 1*3 + 2*3 + 2*3 + 4*3 + 3*2 + 4*2) out of a possible 80 maximum (2 + 1 + 2 + 2 + 3 + 4 + 4 + 2 = 20*4) for the Graphical Editor. In this way, emphasis can be placed on aspects particularly desirable in system-of-systems development and the toolsets can be ranked by highest points accumulation (Table F.8).

Feature	Aspect	Priority	Maximum
Graphical Editor	Documentation	2	
	Layout	1	
	Syntax Building	2	
	Appearance	2	
	Customisation	3	
	Printing	4	
	Syntax Checking	4	
	Error Notification	2	80
	Version Control		
Simulation	Interactive	3	
	Batch	4	
	Watchpoints	2	
	Breakpoints	2	
	Code Generation	2	
	Simple Animation	3	
	Advanced Animation	1	68
Analysis	Reachability	4	
	Liveness	4	
	Fairness	3	
	Temporal Logic	3	
	Invariants	3	
	Statistical Analysis	4	
	Results Presentation	3	
	Other Analysis	2	104
Support/Future Development	Support	3	
	Future Development	2	
	Examples	3	
	Help	2	
	Documentation	3	52
Openness	Import/Export	3	
	Technical Documentation	3	
	Programming Interface	3	36
		Maximum Points	340

Table F.7 'Profile for system-of-systems development'

Toolset	Graphical Editor	Simulation	Analysis	Support/Future Development	Openness	Total	Rank
CPN Tools	47	52	85	42	27	253	**1**
WoPeD	26	15	50	23	21	135	**3**
Renew	45	16	0	26	21	108	**4**
PIPE	28	26	80	29	21	184	**2**

Table F.8 'Toolset ranking according to aspect scoring'

Using this ranking, knowledge of each toolset and the intended usage domain, a recommendation can be made regarding use of a particular toolset in case study exercises. Given that the toolset is likely to be used in both military and commercial sectors, emphasis must be placed on its stability and accessibility (ease-of-use and support is important to both sectors, license cost will be more of an issue in the military sector). The four toolsets were classified according to the tasks involved in engineering systems-of-systems, i.e. modelling, verification/quantitative analysis and validation/qualitative analysis and features relating to these tasks. All tasks involved in engineering a system-of-systems application are perceived to have equally high priority. Given this fact and the timescales for the thesis, it was important to select a suitably rounded toolset scoring reasonably well for all these features. As well as the collated scores, a toolset's history, number of users, and size and structure of supported state space were also considered.

F.2.4 Conclusions

From this evaluation of the four selected toolsets, given that the ratings awarded to each depended on the corresponding aspect in another toolset, each toolset was highly useable in its own right. For the intended case study exercises and thesis time available, the use of CPN Tools is recommended (the other toolsets are worthy of further investigation when there is less restriction on time). CPN Tools offers high levels of support in terms of papers, tutorials, online/offline help, and internet forums and is a comprehensive toolset 'out-of-the-box' allowing fast model construction, execution and analysis. Further 'plug-in' support comes in the form of ASK_CTL, Graphviz and BRITNeY and there are plans to develop further toolset support for state space exploration and analysis of CPN models. Although not fully investigated as yet, running multiple CPN Tools simulators and controlling communication between them appears to be possible using a combination of the CPN Tools simulator and BRITNeY. In terms of system-of-systems design, this would enable verification and validation of low fidelity models built by different agencies (perhaps retrieved from a repository of such models) across a networked environment. Regardless of this, CPN Tools can be used in a standalone environment to open multiple net models simultaneously and has 'clone' functionality to copy elements of nets between models. Related to integration of models is their export. Currently CPN Tools supports its own XML export (with published Document Type Definition). Finally, in terms of cost, CPN Tools is free to both academic and commercial organisations.

F.3 Summary

This appendix has described the process of selecting the CPN Tools Petri net toolset for the purposes of system-of-systems specification and analysis in this thesis.

References

1. ‘Theater Battle Management Core System Systems Engineering Case Study’, J.R. Collens, B. Krause, The MITRE Corporation and Lockheed Martin Integrated Systems and Solutions, 17 February 2005
2. ‘A System-of-Systems Perspective for Public Policy Decisions’, D. DeLaurentis, R.K. Callaway, Review of Policy Research, Volume 21, Number 6, November 2004, pp.829-837
3. ‘British Airways reveals what went wrong with Terminal 5’, taken from http://www.computerweekly.com/Articles/2008/05/14/230680/british-airways-reveals-what-went-wrong-with-terminal.htm on July 4th 2009
4. ‘Architecting Principles for Systems-of-Systems’, M.W. Maier, Systems Engineering, Volume 1, Number 4, 1998, pp.267-284
5. ‘System Life Cycle Processes’, ISO/IEC Standard 15288:2002
6. ‘A Consensus of the INCOSE Engineering Fellows’ (Rechtin, 2000) taken from http://www.incose.org/practice/fellowsconsensus.aspx on July 23rd 2007
7. ‘Application and Management of the Systems Engineering Process’, IEEE Standard 1220-1998, Definition 3.1.37 (2005 version of this standard)
8. ‘Architectural Framework for a System-of-Systems’, D.S. Caffal, J.B. Michael, IEEE International Conference on Systems, Man and Cybernetics, 2005, pp.1876-1881
9. ‘System of Systems Engineering’, C. Keating, R. Rogers, R. Unal, D. Dryer, A. Sousa-Poza, R. Safford, W. Peterson and G. Rabadi, Engineering Management Journal, Volume 15, Number 3, September 2003, pp.36-45
10. ‘Final Version of DSoS Conceptual Model (CSDA1)’, M. Gaudel, V. Issarny, C. Jones, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, R. Stroud, F. Taiani, Technical Report CS-TR-782, University of Newcastle upon Tyne, 2003
11. ‘Unified Modeling Language (UML)’, Object Management Group, Version 2.2 (2009 version of this standard), http://www.omg.org/spec/UML/2.2
12. ‘Darwin Among the Machines: The Evolution of Global Intelligence’, G.B. Dyson, Perseus Books Group, 1998
13. ‘An Emergent Perspective on Interoperation in Systems of Systems’, D.A. Fisher, CMU/SEI-2006-TR-003, Software Engineering Institute, Carnegie Mellon University, 2006
14. ‘Acquisition Operating Framework’, UK Ministry of Defence, taken from http://www.aof.mod.uk/index.htm on July 4th 2009
15. ‘Transforming the U.K.'s Defence Procurement System’, UK Ministry of Defence, Final Report, McKinsey & Co, 20 February 1998
16. ‘Ministry of Defence: Major Projects Report 2007’, House of Commons Committee of Public Accounts, Thirty-third Report of Session, HM Stationary Office, HC 433, 2007-2008
17. ‘Department of Defence Dictionary of Military and Associated Terms’, Joint Publication 1-02, 2001 (amended through 17 October 2007)
18. ‘System Engineering for Faster, Cheaper, Better’, K. Forsberg, H. Mooz, Centre for Systems Management, 1998
19. ‘Research to Address the Challenges of Synthetic Environment Based Acquisition’, S.H. Marshall and T. Anderson, UK Defence Evaluation and Research Agency, HM Stationary Office, 2000

20. ‘Future Combat Systems’, taken from http://www.boeing.com/defense-space/ic/fcs/bia/moreinfo.html on July 4th 2009
21. ‘Defence White Paper’, Defence Industrial Strategy, CM6697
22. ‘System of Systems – the meaning of *of*’, J. Boardman, B. Sauser, IEEE/SMC International Conference on System of Systems Engineering, US, 2006
23. ‘Programmatics of System of Systems Engineering & Integration’, J. Smith, Presentation to the National Oceanic and Atmospheric Administration (NOAA) Netcentric Seminar, May 2007
24. ‘Dynamics of Complex Systems’, Y. Bar-Yam, http://necsi.org/guide
25. ‘Glossary of Communications-electronics Terms’, Allied Communications Publication, ACP-167(H), April 1998
26. ‘The Dynamics of Complex Systems’, Y. Bar-Yam, http://necsi.org/guide/DCSchapter0.pdf, Overview, p.12
27. ‘Principles of Complex Systems for Systems Engineering’, S.A. Sheard, Third Millenium Systems LLC, http://cs.calstatela.edu/wiki/images/8/84/Sheard.doc, 2006
28. ‘A framework for information systems architecture’, J.A. Zachman, IBM Systems Journal, Volume 38, Issue 2-3, 1999, pp.454-470
29. ‘Department of Defence Architecture Framework (DoDAF) v2.0’, taken from http://www.defenselink.mil/cio-nii/policy/eas.shtml on 4th July 2009
30. ‘Ministry of Defence Architecture Framework (MoDAF) v1.2’, taken from http://www.modaf.org.uk on 4th July 2009
31. ‘Requirements Management in a System-of-Systems Context: A Workshop’, B. Craig Meyers, J.D. Smith, P. Capell, P.R.H. Place, CMU/SEI-2006-TN-015, Software Engineering Institute, Carnegie Mellon University, 2006
32. ‘Lessons from Bosnia: The IFOR Experience’, L. Wentz, XI C4ISR Systems and Services, http://www.fas.org/irp/ops/smo/docs/ifor/index.html
33. ‘Systems Architecting: Creating and Building Complex Systems’, E. Rechtin, Prentice Hall, 1991
34. ‘Department of Defence Chairman of the Joint Chiefs of Staff Instruction’, CJCSI 3170.01B, Part II – Definitions, 15 April 2001
35. ‘Global Information Grid Capstone Requirements Document’, Flag Level Review Draft, Appendix A – Part II Definitions, 28 March 2001
36. ‘IEEE 100 The Authoritative Dictionary of IEEE Standards Terms’, Seventh Edition, 2000
37. ‘Systems and software engineering – Recommended practice for architectural description of software-intensive systems’, ISO/IEC Standard 42010:2007
38. ‘The CIS Interoperability web’, 30 March 2007, http://www.ams.mod.uk/content/docs/cisintop/
39. ‘Engineering Complex Systems with Models and Objects’, D.W. Oliver, T.P. Kelliher, J.G. Keegan, McGraw-Hill, 1997
40. ‘Modeling Time in DoDAF Compliant Executable Architectures’, A. AbuSharekh, S. Kansal, A.K. Zaidi, A.H. Levis, Proceedings 2007 Conference on Systems Engineering Research, Hoboken, NJ, March 2007
41. ‘Mission Centric Reliability Analysis of C4ISR Architectures using Petri Nets’, A.E. Olmez, Proceedings IEEE SMC Conference, Washington, D.C., October 2003
42. ‘The Use of Integrated Architectures to Support Agent Based Simulation: An Initial Investigation’, A.W. Zinn, Thesis, Air Force Institute of Technology, March 2004
43. ‘Lectures on Petri Nets I: Basic Models’, W. Reisig and G. Rosenberg (Eds.), Advances in Petri Nets, Springer, 1998

44. ‘System Architecture and Operational Concept Validation through Modeling and Simulation’, R.K. Butler, L.C. Creech, A.J. Anderson, Military Communications Conference, Washington, D.C., 23-25 October 2006, pp.1-6
45. ‘Three Good Reasons for Using a Petri-net-based Workflow Management System’, W.M.P. van der Aalst, In T. Wakayama, S. Kannapan, C.M. Khoong, S.B. Navathe, J. Yates (Eds.), The Kluwer International Series in Engineering and Computer Science, Information and Process Integration in Enterprises: Rethinking Documents, Volume 428, Chapter 10, 1998, pp.161-182
46. ‘System Modelling with High-Level Petri Nets’, H.J. Genrich, K. Lautenbach, Theoretical Computer Science 13, 1981, pp.109-136
47. ‘Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems’, K. Jensen, L.M. Kristensen, L. Wells, International Journal on Software Tools for Technology Transfer, Springer Verlag, 2007, pp.213-254
48. ‘YAWL: yet another workflow language’, W.M.P. van der Aalst and A.H.M. ter Hofstede, Information Systems, Volume 30, Issue 4, June 2005, pp.245-275
49. ‘On Petri Nets with Stochastic Timing’, M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, A. Cumani, On Petri Nets with Stochastic Timing, Proceedings of the International Workshop on Timed Petri Nets, Torino, IEEE Computer Society Press, 1985, pp.80-87
50. ‘Timed Places Petri Nets with Stochastic Representation of Place Time’, C.Y. Wong, T.S. Dillon, K.E. Forward, Proceedings of the International Workshop on Timed Petri Nets, Torino, IEEE Computer Society Press, 1985, pp.96-103
51. ‘Evaluation Based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol’, G. Florin, S. Natkin, Selected Papers from the First and Second European Workshop on Application and Theory of Petri Nets, 1980, pp.280-288
52. ‘Scalability in Analysis of Software Architecture’, B.I. Chukwuogo, Thesis, Texas Tech University, August 2007
53. ‘Case Studies of Systems Engineering and Management in Systems Acquisition’, G. Friedman, A.P. Sage, Systems Engineering, Volume 1, 2004
54. ‘Five Misunderstandings About Case-Study Research’, B. Flyvbjerg, Qualitative Inquiry, Volume 12, Number 2, April 2006, pp.219-245
55. ‘A survey of controlled experiments in software engineering’, D.I.K. Sjoberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.K. Liborg, A.C. Rekdal, IEEE Transactions on Software Engineering, Volume 31, Issue 9, September 2005, pp.733-753
56. ‘Diplans: A New Language for the Study and Implementation of Coordination’, A. W. Holt, ACM Transactions on Office Information Systems, Volume 6, Number 2, April 1988, pp.109-125
57. ‘Petri Nets: Properties, Analysis and Applications’, T. Murata, Proceedings of the IEEE, Volume 77, Number 4, April 1989, pp.541-580
58. ‘Fundamentals of a theory of asynchronous information flow’, C.A. Petri, Proceedings of the 1962 IFIP Congress, 1962, pp.386-390
59. ‘Workflow Patterns’, W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Distributed and Parallel Databases, July 2003, pp.5-51
60. ‘Migration of Control in Decision Making Organisations’, A.H. Levis, S.L. Skulsky, Laboratory for Information and Decision Systems, LIDS-P 1881, 1989
61. ‘A Coloured Petri Net Model of Distributed Tactical Decision Making’, Z. Lu, A.H. Levis, Proceedings 1991 Symposium on Command and Control Research, 1991

62. ‘Intelligent Task Planning Using Fuzzy Petri Nets’, T. Cao, A.C. Sanderson, World Scientific, 1996
63. ‘Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use’, K. Jensen, Volume 1, Second Edition, Springer-Verlag, 1997
64. ‘Case studies for method and tool evaluation’, B. Kitchenham, L. Pickard, S.L. Pfleeger, IEEE Software, Volume 12, Issue 4, July 1995, pp.52-62
65. ‘A Survey of Some Theoretical Aspects of Multiprocessing’, J.L. Baer, ACM Computing Surveys, Volume 5, March 1973
66. ‘Improving UML with Petri nets’, L. Baresi, M. Pezze, Electronic Notes in Theoretical Computer Science, Volume 44, Number 4, July 2001
67. ‘Execution Strategies for Petri Net Simulations’, J.M. Grevet, L. Jandura, J. Brode, A.H. Levis, Laboratory for Information and Decision Systems, LIDS-P 1739, 1988
68. ‘Conquering Complexity: Lessons for Defence Systems Acquisition’, A. Weiss, K. Hambleton, The Stationery Office, 2005
69. ‘The ASCoVeCo State Space Analysis Platform: Next Generation Tool Support for State Space Analysis’, L.M. Kristensen, M. Westergaard, Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2007 (October 22-24)
70. ‘Littoral undersea warfare: a case study in process modelling for functionality and interoperability of complex systems’, V.D. Bindi, J. Strunk, J. Baker, R. Bacon, M.G. Boensel, F.E. Shoup, R. Vaidyanathan, International Journal System of Systems Engineering, Volume 1, Numbers 1-2, 2008, pp.18-58
71. ‘Mission Control by Coordinating Shared Resources’, W. Meyer, C. Fiedler, IEEE/SMC International Conference on System of Systems Engineering, 2006 (April 24-26)
72. ‘Modeling Interactive Systems with Hierarchical Colored Petri Nets’, M. Elkoutbi, R.K. Keller, Advanced Simulation Technologies Conference, Boston, MA, 1998 (April 5-9)
73. ‘Using Interface Prototyping Based on UML Scenarios and High-Level Petri Nets’, M. Elkoutbi, R.K. Keller, Proceedings of the 21st International Conference on Application and Theory of Petri Nets, Aarhus, Denmark, Springer-Verlag, 2000 (June 26-30), pp.166-186
74. ‘Formalization of Object Behavior and Interactions from UML Models’, J.A. Saldhana, S.M. Shatz, Z. Hu, International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Volume 11, Number 6, December 2001, pp.643-673
75. ‘UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis’, J.A. Saldhana, S.M. Shatz, Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, IL, 2000 (July 6-8), pp.103-110
76. ‘Transformation of UML-based System Model to Design/CPN Model for Validating System Behavior’, M.E. Shin, A.H. Levis, L.W. Wagenhals, Proceedings of the Compositional Verification of UML Models Workshop, Sixth International Conference on the UML, San Francisco, CA, October 2003
77. ‘An Improved Architectural Specification of the Internet Open Trading Protocol’, C. Ouyang, L.M. Kristensen, J. Billington, Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2001 (August 29-31)

78. ‘A Coloured Petri Net Approach to Formalising and Analysing the Resource Reservation Protocol’, M.E. Villapol, J. Billington, CLEI Electronic Journal, Volume 6, Number 1, December 2003
79. ‘Colored Petri Net based model checking and failure analysis for E-commerce protocols’, P. Katsaros, V. Odontidis, M. Gousidou-Koutita, Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2005 (October 24-26)
80. ‘Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use’, K. Jensen, Volume 2, Second Edition, Springer-Verlag, 1997
81. ‘Petri Nets and Industrial Applications: A Tutorial’, R. Zurawski, M. Zhou, IEEE Transactions on Industrial Electronics, Volume 41, Number 6, December 1994
82. ‘Capacity Planning of Web Servers using Timed Hierarchical Coloured Petri Nets’, S. Christensen, L.M. Kristensen, K.H. Mortensen, J.S. Thomasen, Proceedings of Hewlett- Packard OpenView University Association (HP-OVUA'99), 6th Plenary Workshop, 1999
83. ‘Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets’, W.M.P. van der Aalst, M.A. Odijk, Real-Time Systems, Volume 9, Issue 3, November 1995, pp.241-263
84. ‘Real-time Decision Making For Shipboard Damage Control’, V. Bulitko, D. Wilkins, Proceedings of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, AAAI Press, 2002, pp.37-46
85. ‘Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets’, O.M. Dahl, S.D. Wolthusen, Proceedings of the Fourth IEEE International Workshop on Information Assurance, 2006 (April 13-14), pp.157-168
86. ‘Modelling and simulating multi-echelon food systems’, J.G.A.J. van der Vorst, A.J.M. Beulens, P. van Beek, European Journal of Operational Research, Volume 122, Issue 2, Elsevier, 2000 (April 16), pp.354-366
87. ‘Bullwhip Effect and Supply Chain Modelling and Analysis Using CPN Tools’, D. Makajic-Nikolic, B. Panic, M. Vujosevic, Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2004 (October 8-11)
88. ‘Design of Clearing and settlement operations: A case study in business process modelling and evaluation with Petri nets’, P.M. Kwantes, Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2006 (October 24-26)
89. ‘A Petri Net Approach for the Performance Analysis of Business Processes’, A.K. Schomig, H. Rau, Report No.116, University of Wurzburg, Institute of Computer Science, Research Report Series, May 1995
90. ‘Application of Coloured Petri Nets in System Development’, L.M. Kristensen, J.B. Jorgensen, K. Jensen, Lecture Notes in Computer Science, Volume 3098, Springer-Verlag, 2004, pp.626-685
91. ‘Modular Analysis of Systems Composed of Semiautonomous Subsystems’, C. Lakos, L. Petrucci, Proceedings of the 4th International Conference on Application of Concurrency to System Design (ACSD 2004), Hamilton, Canada, IEEE Computer Society Press, June 2004, pp.185-194
92. ‘An Incremental and Modular Technique for Checking LTL\X Properties of Petri nets’, K. Klai, L. Petrucci, M. Reniers, Proceedings of the 27th International Conference on Formal Methods for Networked and Distributed Systems (FORTE 2007), Tallinn, Estonia, Lecture Notes in Computer Science, Volume 4574, Springer, June 2004

93. ‘Modular State Spaces and Place Fusion’, C. Lakos, L. Petrucci, Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2007), Sieldlce, Poland, University of Podlasie, June 2007, pp.175-190
94. ‘Modelling and Performance Analysis of Workflow Management Systems Using Timed Hierarchical Coloured Petri Nets’, K. Salimifard, M. Wright, Proceedings of the ICEIS 2002, Ciudad Real, Spain, 2002 (3-6 April), pp.843-846
95. ‘Petri Nets for Component-Based Software Systems Development’, L.D. da Silva, K. Gorgonio, A. Perkusich, Petri Nets, Theory and Applications, I-Tech Education and Publishing, February 2008, p.534
96. ‘Evaluating software engineering methods and tools part 9: quantitative case study methodology’, B. Kitchenham, L.M. Pickard, ACM SIGSOFT Software Engineering Notes, Volume 23, Issue 1, January 1998, pp.24-26
97. ‘A Unidirectional Transition Fusion for Coloured Petri Nets and its Implementation for the CPNTools’, J.P. Barros, L. Gomes, Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2004 (October 8-11), pp.199-218
98. ‘A discretization method from coloured to symmetric nets: application to an industrial example’, F. Bonnefoi, C. Choppy, F. Kordon, Ninth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2008 (October 20-22)
99. ‘Design, modeling and analysis of ITS using UML and Petri Nets’, F. Bonnefoi, L.M. Hillah, F. Kordon, X. Renault, Intelligent Transportation Systems Conference, ITSC 2007, 2007 (September 30-October 3), pp.314-319
100. ‘An Introduction to the Practical Use of Coloured Petri Nets’, K. Jensen, Lecture Notes in Computer Science, Volume 1492, Springer-Verlag, 1996, pp.237-292
101. ‘Evaluating software engineering methods and tools part 10: designing and running a quantitative case study’, B. Kitchenham, L.M. Pickard, ACM SIGSOFT Software Engineering Notes, Volume 23, Issue 3, May 1998, pp.20-22
102. ‘Hubble Space Telescope Systems Engineering Case Study’, J.J. Mattice, Center for Systems Engineering at the Air Force Institute of Technology, 10 March 2005
103. ‘Dictionary of Sociology’, N. Abercrombie, B.S. Turner, Penguin, 1984
104. ‘Case Study Research: Design and Methods’, R.K. Yin, 3rd edition, Sage, 2003
105. ‘Joint Tactics, Techniques, and Procedures for Close Air Support (CAS)’, Joint Publication 3-09.3, 2 September 2005
106. ‘Variable Message Format (VMF) System Management and Operating Procedures (SMOPS)’, Version 2.1, prepared by SyntheSys Systems Engineers Ltd on behalf of the Ministry of Defence Integration Authority, March 2007
107. ‘Studying Software Engineers: Data Collection Techniques for Software Field Studies’, T.C. Lethbridge, S.E. Sim, J. Singer, Empirical Software Engineering, Volume 10, Issue 3, July 2005, pp.311-341
108. ‘Guidelines for conducting and reporting case study research in software engineering’, P. Runeson, M. Host, Empirical Software Engineering, Volume 14, Issue 2, April 2009, pp.131-164
109. ‘IBM Rational Rhapsody’, taken from http://www-01.ibm.com/software/awdtools/rhapsody on 6th July 2009
110. ‘IBM Rational Tau’, taken from http://www-01.ibm.com/software/awdtools/tau on 6th July 2009
111. ‘Model Checking’, E.M. Clarke, O. Grumberg, D. Peled, MIT Press, 1999

112. ‘UML Profile For Schedulability, Performance, And Time’, Object Management Group, Version 1.1 (2005 version of this standard), http://www.omg.org/technology/documents/formal/schedulability.htm
113. ‘Modeling and Analysis of Real-time and Embedded systems’, Object Management Group, Version Beta1 (2007 version of this specification), http://www.omgmarte.org/Specification.htm
114. ‘UML For Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms’, Object Management Group, Version 1.1 (2008 version of this standard), http://www.omg.org/spec/QFTP/1.1/
115. ‘Software Performance Modeling using UML and Petri nets’, J. Merseguer, J. Campos, Lecture Notes in Computer Science Performance tools and applications to networked systems, Volume 2965, 2004, pp.265-289
116. ‘Validation of Dynamic Behaviour in UML Using Coloured Petri Nets’, R.G. Pettit, H. Gomaa, Lecture Notes in Computer Science Proceedings of UML 2000 Workshop, Volume 1939, 2000
117. ‘On the integration of UML and Petri nets in software development’, J. Campos, J. Merseguer, In 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, 2006
118. ‘Semantics and Verification of Data Flow in UML 2.0 Activities’, H. Storrle, Visual Languages and Formal Methods, 2004
119. ‘Semantics of UML 2.0 Activities’, H. Storrle, International Symposium/Human Computer Centred Systems, Rome, 2004
120. ‘UK Defence Tactical Data Link Interface Requirement Specification (Part II) Single Link - Variable Message Format’, Draft Version 2.1, prepared by SyntheSys Systems Engineers Ltd on behalf of the Ministry of Defence Integration Authority, April 2007
121. ‘MIL-STD-188-220D’, Department of Defence Interface Standard, Digital Message Transfer Device Subsystems, 29 September 2005
122. ‘Dynamic Host Configuration Protocol’, RFC 2131, Internet Engineering Task Force, March 1997, http://www.ietf.org/rfc/rfc2131.txt?number=2131
123. ‘LAN/MAN CSMA/CD Access Method’, IEEE 802.3-2008 Standard for Information technology – Part 3: Carrier Sense Multiple Access with Collision Detection (CMSA/CD) Access Method and Physical Layer Specifications
124. ‘Systems Integration: Lessons from Two Practical Experiences’, P. Layzell, IEEE Conference on Software Maintenance and Re-engineering 2005, Keynote address, March 2005
125. ‘OMG Unified Modeling Language (OMG UML), Superstructure’, Object Management Group, Version 2.2, February 2009, http://www.omg.org/docs/formal/09-02-02.pdf
126. ‘Workflow Petri Net Designer (WoPeD)’, Cooperative State University Karlsruhe, Version 2.2.0, March 2009, http://www.woped.org
127. ‘The Reference Net Workshop (Renew)’, University of Hamburg, Version 2.1.1, July 2008, http://www.renew.de
128. ‘Platform Independent Petri net Editor 2 (PIPE2)’, Imperial College, London, Version 2.5, 2007, http://pipe2.sourceforge.net/
129. ‘Basic Real-time Interactive Tool for Net-based animation (BRITNeY)’, University of Aarhus, http://wiki.daimi.au.dk/britney/britney.wiki
130. ‘Model Checking Coloured Petri Nets Exploiting Strongly Connected Components’, A. Cheng, S. Christensen, K.H. Mortensen, Proceedings of the

International Workshop on Discrete Event Systems, Edinburgh, UK, August 1996, pp.169-177
131. ‘Graph Visualisation Software (Graphviz)’, AT&T Research Labs, http://www.graphviz.org
132. ‘An Evaluation of High-End Tools for Petri-Nets’, H. Storrle, Technical Report, University of Munich, June 1998
133. ‘Dynamic Data Integration: A Service-Based Broker Approach’, F. Zhu, M. Turner, I. Kotsiopoulos, K. Bennett, M. Russell, D. Budgen, P. Brereton, J. Keane, P. Layzell, M. Rigby, J. Xu, International Journal of business Process Integration and Management, Volume 1, Issue 3, January 2006, pp.175-191
134. ‘A pattern language for designing e-business architecture’, L. Zhao, L. Macaulay, J. Adams, P. Verschueren, Journal of Systems and Software, Volume 81, Issue 8, August 2008, pp.1272-1287
135. ‘A Model Driven Architecture for Enterprise Application Integration’, A. Mosawi, L. Zhao, L. Macaulay, Proceedings IEEE Hawaii International Conference on System Sciences (HICSS-39), Volume 8, 2006 (January 4-7)