Model-based Engineering of Emergence in a Collaborative SoS: Exploiting SysML & Formalism

Claire Ingram, Richard Payne, John Fitzgerald and Luis Diogo Couto

TECHNICAL REPORT SERIES

No. CS-TR-1449 **February 2015**

TECHNICAL REPORT SERIES

No. CS-TR-1449 February, 2015

Model-based Engineering of Emergence in a Collaborative SoS: Exploiting SysML & Formalism

C.Ingram, R. Payne, J. Fitzgerald, L. Diogo Couto

Abstract

A collaborative SoS is a system composed of constituent systems (CSs), which are independent and voluntarily cooperate without an agreed SoS director. Engineering emergent behaviour is just one of the key engineering challenges for which support is needed. In this paper we illustrate for the first time an integrated collection of model-based techniques for verifying behaviours and properties, with the aim of assisting the engineering of collaborative systems of systems (SoSs). We provide an illustration of an approach that flows from requirements and architectural modelling to the use of formal techniques, integrating methods drawn from software and systems engineering fields to tackle engineering challenges in SoS. The approach incorporates architectural modelling (implemented in SysML) before transitioning to a formal modelling notation which has been developed specifically for SoSs. This formal modelling approach supports a wide range of analysis and verification techniques, such as: design space exploration; requirements verification; and consistency checks. We also discuss how our approach can be incorporated into a standard systems engineering approach.

Bibliographical details

INGRAM, C., PAYNE, R., FITZGERALD, J., DIOGO COUTO, L.
Model based Engineering of Emergence in a Collaborative SoS: Exploiting SysML & Formalism
[By] C.Ingram, R. Payne, J. Fitzgerald, L. Diogo Couto

Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1449)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1449

Abstract

A collaborative SoS is a system composed of constituent systems (CSs), which are independent and voluntarily cooperate without an agreed SoS director. Engineering emergent behaviour is just one of the key engineering challenges for which support is needed. In this paper we illustrate for the first time an integrated collection of model-based techniques for verifying behaviours and properties, with the aim of assisting the engineering of collaborative systems of systems (SoSs). We provide an illustration of an approach that flows from requirements and architectural modelling to the use of formal techniques, integrating methods drawn from software and systems engineering fields to tackle engineering challenges in SoS. The approach incorporates architectural modelling (implemented in SysML) before transitioning to a formal modelling notation which has been developed specifically for SoSs. This formal modelling approach supports a wide range of analysis and verification techniques, such as: design space exploration; requirements verification; and consistency checks. We also discuss how our approach can be incorporated into a standard systems engineering approach.

About the authors

Claire Ingram earned her PhD at Newcastle University in 2011, with doctoral research on early stage design and requirements data. She has worked on the DESTECS project, which supported fault-tolerant embedded systems, and the COMPASS project, examining model-based techniques for SoSs. Her research interests lie in empirical software engineering and systems engineering, SoSs and architectures.

Richard Payne's research interests include architectural modelling, systems of systems, and verification tools for formal methods. Following his PhD (Newcastle University 2012) on models of dynamic reconfiguration, he worked on the UK Ministry of Defence project on Interface Contracts for Architectural Specification and Assessment, and the COMPASS project, designing advanced modelling and verification techniques. He is currently working on the INTO-CPS working on methods and modelling for a tool chain on Cyber-Physical Systems.

John Fitzgerald is a full Professor in Computing Science at Newcastle University, UK, where he leads research on model-based engineering and the Cyber-Physical Systems Lab. He recently led the international COMPASS project on model-based methods for systems of systems engineering. With a background in verification technology (PhD, Manchester Univ., 1991), he has contributed to methods and tools applied commercially in areas as diverse as chip design and options trading. He is a member of INCOSE, ACM, and IEEE, and is a Fellow of the BCS.

Luís Diogo Couto is a PhD student in Computer Engineering at Aarhus University, Denmark. His PhD work is focused primarily on extensibility of software systems. Other research interests include formal methods, tooling and modelling. He is working on the COMPASS project, developing tools and techniques for formal model analysis. He is also one of the primary developers of the Overture tool for formal modelling.

Suggested keywords

SYSTEMS OF SYSTEMS
REQUIREMENTS ENGINEERING
ARCHITECTURAL MODELLING
FORMAL MODELLING
ANALYSIS
VERIFICATION
SYSML
CML

Model-based Engineering of Emergence in a Collaborative SoS: Exploiting SysML & Formalism

Claire Ingram, Richard Payne, John Fitzgerald
Newcastle University
{firstname.lastname}@newcastle.ac.uk

Luis Diogo Couto
Aarhus University
ldc@eng.au.dk

Abstract. A collaborative SoS is a system composed of constituent systems (CSs), which are independent and voluntarily cooperate without an agreed SoS director. Engineering emergent behaviour is just one of the key engineering challenges for which support is needed. In this paper we illustrate for the first time an integrated collection of model-based techniques for verifying behaviours and properties, with the aim of assisting the engineering of collaborative systems of systems (SoSs). We provide an illustration of an approach that flows from requirements and architectural modelling to the use of formal techniques, integrating methods drawn from software and systems engineering fields to tackle engineering challenges in SoS. The approach incorporates architectural modelling (implemented in SysML) before transitioning to a formal modelling notation which has been developed specifically for SoSs. This formal modelling approach supports a wide range of analysis and verification techniques, such as: design space exploration; requirements verification; and consistency checks. We also discuss how our approach can be incorporated into a standard systems engineering approach.

1 Introduction

A system of systems (SoS) is composed from independent systems that collectively deliver an emergent capability on which reliance may be placed. The engineering of dependable SoSs presents a number of issues (Dahmann 2014). For example, the engineering of emergent end-to-end capabilities that will reliably meet some minimum requirements (e.g., in terms of dependability or availability) is a challenge (Sanders and Smith 2012). This is exacerbated when incomplete information may be available about each constituent system (CS), as is the case in many SoSs. In addition, SoSs experience continual change (e.g., Maier 1998, Abbott 2006, Boardman & Sauser 2006); this requires SoSs to identify accurately and quickly when changes deployed by one CS (perhaps without warning) may be incompatible with other aspects of SoS behaviour. SoSs usually cannot be taken offline for testing, and are so large and complex that it can be difficult to create realistic test laboratories. Furthermore, it can be difficult to engage fully with the requirements and goals of the many stakeholders in an SoS (Hallerstede et al 2012). These difficulties become particularly acute when considering *collaborative* SoSs (Maier 1998) which lack central direction or engineering management.

Several research and industrial groups are tackling SoS engineering challenges. For example, the INCOSE SoS Working Group is active in sharing best practice and identifying current issues (Dahmann 2014). Two recent EU-funded research projects have identified the engineering of emergent behaviours as a key problem area for SoS engineering, and suggested that model-based approaches should be explored for potential solutions (T-AREA-SoS 2013, Road2SoS 2013). Two further projects, COMPASS[1] and DANSE[2], have pursued this,

[1] http://www.compass-research.eu

developing methods and tools for modelling the architecture and functionality of SoSs, and assessing their applicability in several application domains. Model-based approaches address complexity by allowing developers to focus on models that abstract from details that are not relevant to the emergent properties of interest. In the area of software engineering, the use of modelling notations with formal mathematical semantics has been shown to allow consistent automation in analysis, offering opportunities to assess the quality of designs by simulation and by static techniques (Fitzgerald, Larsen and Woodcock 2013). However, there has been so far only limited work on the potential for integrating such “formal” methods with established systems engineering and SoS engineering practices to assist the design and maintenance of challenging collaborative SoSs.

In this paper, we present a proof-of-concept study of an approach which integrates formal modelling techniques with model-based engineering of dependable collaborative SoSs. This study is the first attempt at demonstrating whether it is possible and/or beneficial to obtain a flow from requirements and architectural modelling to the use of formal techniques in a single study. Specifically, we adopt requirements and architectural modelling, realised in SysML (OMG 2010), alongside formal models of data, functionality and communication, to support machine-assisted analysis of the emergent behaviour of a collaborative SoS. We employ a contractual approach to SoS modelling, which means that CSs need not be completely specified in the SoS model, as long as the assumptions and commitments of each CSs’ externally-visible behaviours are available. This is useful for collaborative SoSs in which the independent CSs’ owners may have commercial reasons not to fully disclose internal logic. Further, although we have targeted collaborative SoSs, we believe that our techniques can usefully be employed in other types of SoS which face similar challenges in coping with autonomy, emergent behaviour, and continual evolution.

We introduce the model-based approach in Section 2 and discuss how it can be adapted to a general systems engineering approach. In Section 3 we describe an SoS case study, drawn from the traffic management domain. Sections 4, 5 and 6 present our requirements models, our architectural modelling and our formal modelling outputs respectively. The techniques we describe have been influenced by software, computation and communication; there is further work to be done before formal techniques can fully represent general engineering systems that combine cyber, physical and human elements. We discuss work towards achieving this in our conclusions in Section 7.

2 An Approach to Model-based SoS Engineering

We employ both semi-formal and formal modelling techniques in the proof of concept study. Specifically, the illustrated approach integrates SysML (OMG 2010) and COMPASS Modelling Language (CML) (Woodcock et al. 2012). We use SysML when defining modelling guidelines, requirements engineering and architectural modelling. CML, a formal modelling language, is used for several forms of automated analysis. Formal modelling languages are modelling techniques with a discrete mathematical sematics, which are used in software engineering to develop unambiguous models of software-intensive systems. Like mathematical models in other engineering disciplines, such a model can be used to generate predictions about the finished system and its behaviour (Rushby 1993). This allows software engineers to apply rigorous analysis and verification of requirements and design choices at any

[2] http://www.danse-ip.eu

stage in a system's lifecycle (Woodcock et al 2009). We build on these principles with an approach that consists of the following steps:

1. Requirements Engineering. We use an approach designed for SoSs, called SoS-ACRE.
2. Architectural Modelling. We use a pre-existing pattern to reduce the modelling effort.
3. Transition to a formal modelling language, and analysis of the formal model.

Having generated a formal model we employ the following analysis techniques (addressed in later sections):

- Rapid prototyping. We perform some design space exploration activities to optimize design choices, using a simulator on an executable model for immediate feedback.
- Validation of SoS requirements.
- Verification of SoS contract. This activity allows us to detect incompatibilities, mismatches and inconsistencies at an early design stage during integration of CSs.
- Additional kinds of formal analysis. We briefly introduce other possible analyses.

2.1 Adapting Our Techniques to a Systems Engineering Approach

In this section, we consider how the approach we have outlined can be incorporated into a general systems engineering process. We use the Systems Engineering Handbook (INCOSE, 2011) as a guide for a systems engineering approach, although of course this should be tailored as necessary.

Exploratory Research Stage. We use in this paper an approach (in Section 4) suitable for the capture of requirements from a variety of stakeholders' perspectives. Our suggested technique for managing SoS requirements, SoS-ACRE, also incorporates steps designed to consider the different stakeholder perspectives at the SoS level. This allows inconsistencies, conflicts or hidden dependencies between stakeholders to be identified.

Concept Stage. The Concept Stage may involve identifying candidate solutions and selecting one, based on a firm evaluation. Analyses of the types we describe in Section 6 can be conducted to identify potential inconsistencies and design problems and to support selections made between candidate solutions; system models can be archived as design rationale to support decisions.

Development Stage. This stage usually includes detailed planning and validation and verification (V&V) activities. Our approach supports varied analysis techniques for V&V of requirements, architectural choices, and detailed designs. In some SoSs, separate teams may be able to exchange contractual specifications of their CSs as they progress through the development stage, to assess for design errors.

Production Stage. This stage typically includes iterative processes for producing or implementing the new SoS (or its CSs). Models can continue to be used during the production stage; e.g., any proposed changes or evolutions can be assessed first by model-based techniques to achieve the optimal decisions or to identify potential inconsistencies which may propagate changes unexpectedly or result in degradation of the SoS emergent capabilities.

Utilisation and Support Stages. Many SoSs will spend many years in utilisation and support. Throughout its life, model-based techniques that can be used to assess proposed changes to any of the CSs.

3 A Border Traffic SoS Case Study

In this section, we introduce our proof of concept case study, a simple example of a collaborative SoS drawn from the field of traffic management. A Traffic Management System

(TMS) exists to monitor and direct current traffic loads in a specific locale. Typically a TMS has access to a number of different systems for monitoring traffic (such as induction loops, radar or infrared detectors, and video cameras) as well as a number of systems which behave as actuators that can influence traffic. Actuators could include, for example, dynamic message boards to advise motorists of faster routes or upcoming delays, dynamic speed limits, responsive traffic lights or dynamic lane closures. TMSs also include control rooms which accept data from traffic monitors, perform some analysis and issue instructions to actuators in an attempt to achieve some overall goals, such as reducing journey times, reducing air pollution or improving road safety. Road management is typically managed on a regional basis, with each region or country responsible for a geographical area.

Our case study is a hypothetical cross-border traffic scenario, which centres on a simplified view of a one-way highway connecting two countries, *A* and *B*. The international border between A and B bisects the road. In reality, TMSs implement a large number of traffic management functions, but in our model we concentrate on modelling the behaviour of just one function, which is called *queue tail detection.* Queue tail detection is designed to improve road safety on high-speed highways. From time to time incidents such as traffic accidents result in a queue (line) of slow or stationary vehicles on a high-speed route. We wish to ensure that vehicles moving at high speed do not encounter the queue unexpectedly and collide. To tackle this, a speed corridor is established, spanning several kilometres and culminating in back of the slow-moving queue. Dynamically adjustable speed limits are used to ensure that approaching vehicles reduce speed gradually over the length of the corridor, until they are travelling at a safe target speed when they encounter the slow-moving queue. In our hypothetical example, we assume that the span of highway crosses the international border, and that traffic flows from Country A to Country B. A queue may form inside B, such that traffic arriving from A encounters it suddenly. Queue tail detection in this case requires collaboration, with B requesting from A that certain speed limits are imposed on traffic approaching and crossing the border towards B. We present some examples in Figure 1. For example, the top-most incident has occurred at the location labelled -7, creating a queue spanning a distance *d* of 4km. In this case the speed corridor operates solely within B. However, the third incident occurs at location -1, with a resultant queue of 5km; therefore this case requires a cross-border speed corridor in order to ensure traffic safety. Actuators (e.g., dynamic speed limits) capable of implementing this speed reduction are distributed on each side of the border, but each country retains ownership and operation of its own actuators and does not permit its neighbour to access them directly.

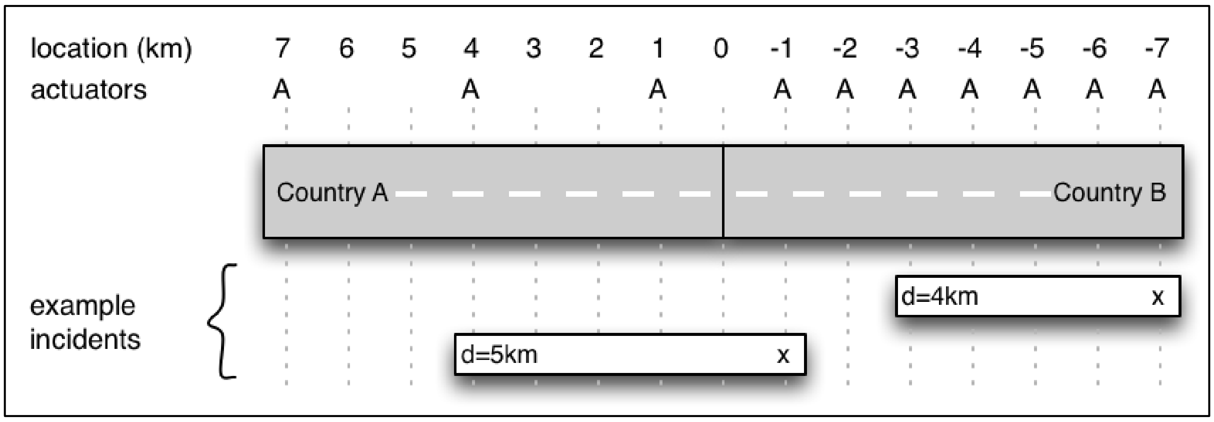


Figure 1 – Illustrative examples of incidents on a cross-border highway

This example is selected as it exhibits key characteristics associated with collaborative SoSs, including independent constituents, emergent behaviour, distribution, lack of an overall manager and so on. This makes it competent for demonstrating our selected approach.

4 Requirements Engineering

The first model-based technique we illustrate is in the definition of requirements. An SoS context can present extra challenges for the requirements engineer (Hallerstede et al, 2012). We use a requirements process, SoS-ACRE (Holt et al 2012) that is designed specifically for engineering SoS requirements. We present a subset of SoS-ACRE views here, implemented in SysML: the Requirements Description View; Contract and Constituent System Definition View; and Context Interaction View.

4.1 Requirement Definition

SoS-ACRE defines a Requirements Description View (RDV) for describing each requirement according to pre-defined attributes. An excerpt from the RDV for the cross-border traffic model is presented in Figure 2.

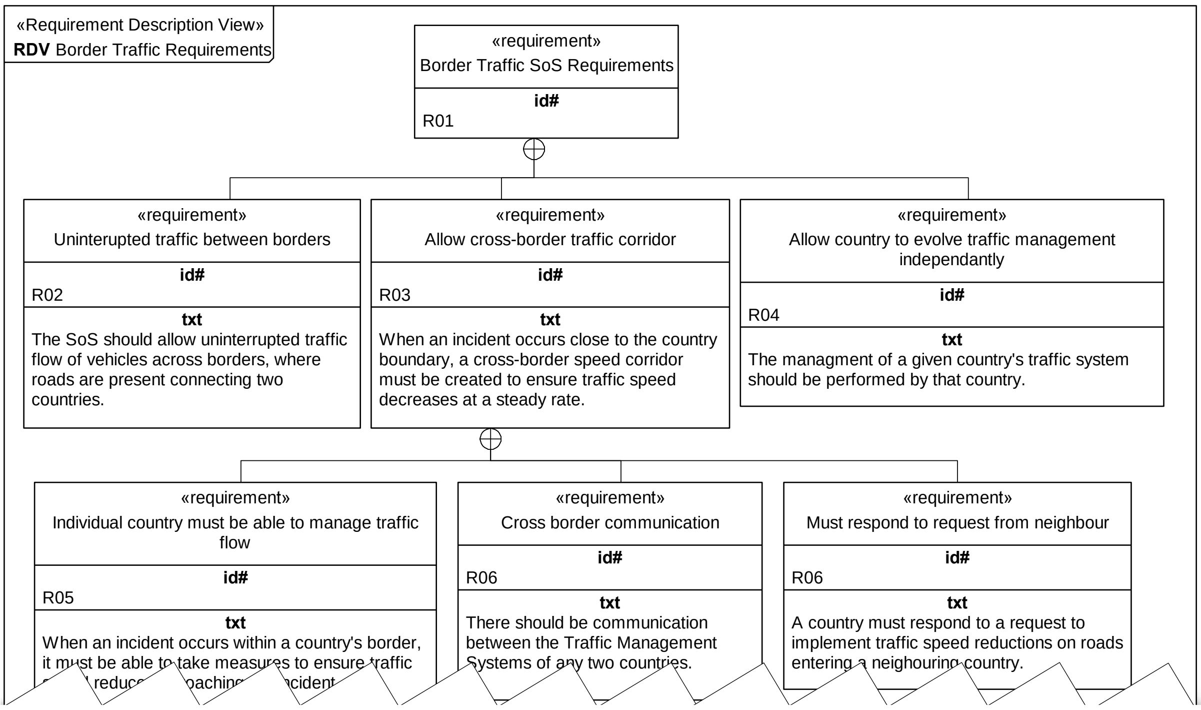

«Requirement Description View»
RDV Border Traffic Requirements
«requirement»
Border Traffic SoS Requirements
id#
R01
«requirement»
Uninterupted traffic between borders
id#
R02
txt
The SoS should allow uninterrupted traffic flow of vehicles across borders, where roads are present connecting two countries.
«requirement»
Allow cross-border traffic corridor
id#
R03
txt
When an incident occurs close to the country boundary, a cross-border speed corridor must be created to ensure traffic speed decreases at a steady rate.
«requirement»
Allow country to evolve traffic management independantly
id#
R04
txt
The managment of a given country's traffic system should be performed by that country.
«requirement»
Individual country must be able to manage traffic flow
id#
R05
txt
When an incident occurs within a country's border, it must be able to take measures to ensure traffic
«requirement»
Cross border communication
id#
R06
txt
There should be communication between the Traffic Management Systems of any two countries.
«requirement»
Must respond to request from neighbour
id#
R06
txt
A country must respond to a request to implement traffic speed reductions on roads entering a neighbouring country.

Figure 2 - Requirements Description View (RDV) for the cross-border traffic model

Three main requirements are defined in this RDV: *uninterrupted traffic between borders*; *allow cross-border traffic corridor*; and *allow country to evolve traffic management independently*. The RDV presents each requirement with a unique ID and text description. Sub-requirements are also modelled where appropriate. The unique IDs are important for achieving traceability, which is particularly important once we transition to the formal model later in our SoS design process.

4.2 Identify SoS Constituents and Stakeholders

After defining the requirements relevant to the study, we next need to place the requirements in context for the actors of the case study.

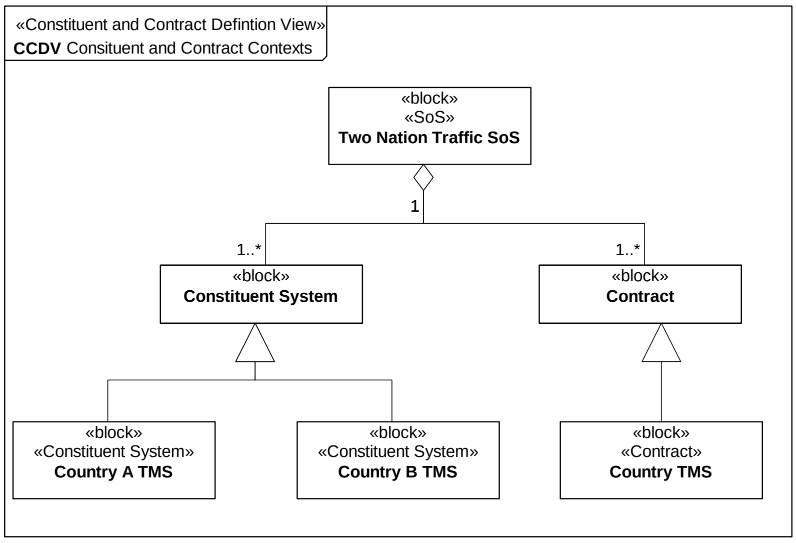


Figure 3 Constituent and Contract Definition View

In the simplified two-nation cross border example, we define a single SoS: *Two-Nation Traffic SoS.* This SoS is composed of two CSs: *Country A TMS* and *Country B TMS*. These two constituents are captured in the Constituent and Contract Definition View (CCDV) shown in Figure 3. We also identify a single contract: *Country TMS.* We will return to this concept of a contract when we begin to define the architecture for the SoS.

4.3 Requirements in Context

SoS-ACRE advocates analysing the requirements in the context of each separate CS, by generating Requirement Context Views (RCVs); in SysML, for example, RCVs could be implemented by use cases representing each CS's perspective on the SoS. Finally, these views are then combined on a separate, SoS-level view, the Context Interaction View (CIV). This is seen as a key part of SoS requirements engineering by providing an expanded view of requirements, presented in their separate contexts, for the entire SoS. This may be the first time that requirement and contexts are analysed together, and is helpful for identifying conflicts, inconsistencies and duplicated functionality, for example.

For our case study, therefore, we next analyse SoS and CS requirements in the context of the different SoS entities. Models of requirements in context for each constituent system are omitted here, but in Figure 4 (overleaf) we present a CIV that captures the requirements in relation to the *Country TMS contract,* an entity which was previously identified in Figure 3.

Although they conform to same contract, we expect that the two TMSs will not be exactly the same, and that each TMS may interpret some of the requirements slightly differently. For example, due to various internal and external pressures, two CSs offering superficially similar functionality may develop different business drivers or goals and therefore different behaviour. This type of inconsistency could be identified from examining the contracts for each TMS.

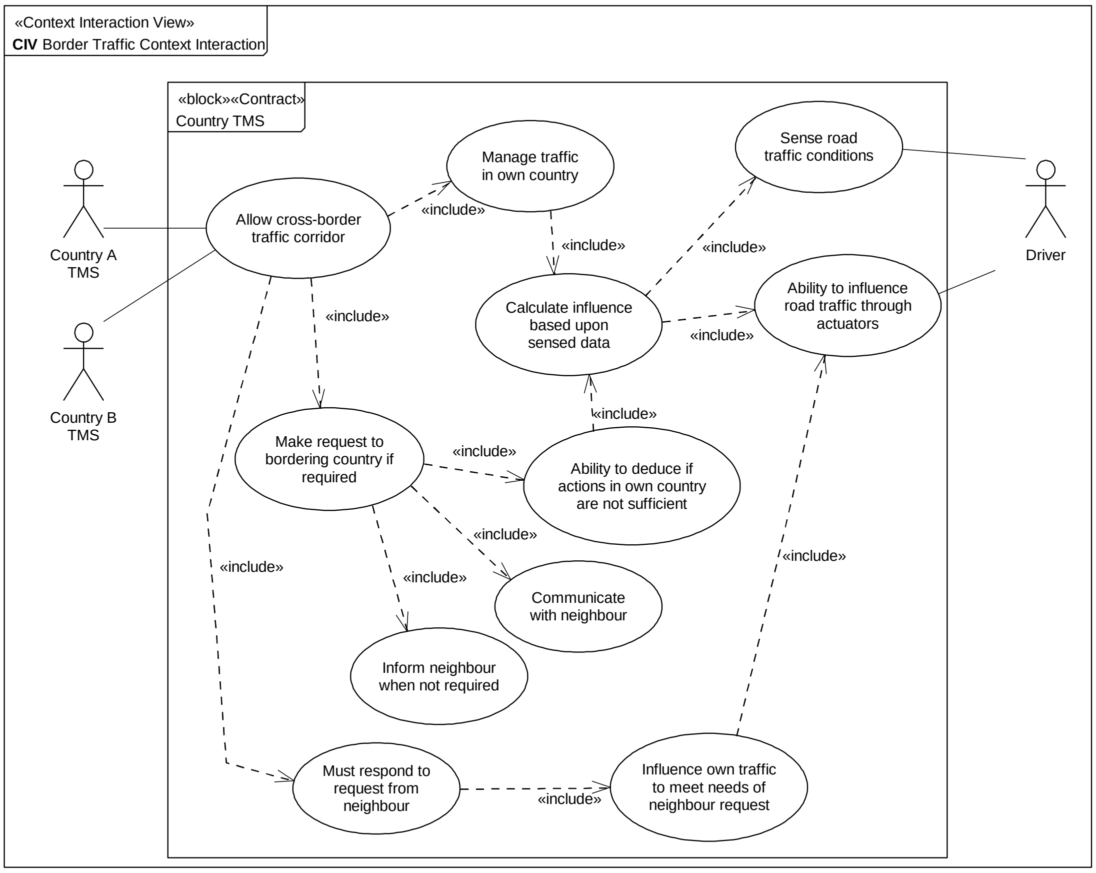

«Context Interaction View»
CIV Border Traffic Context Interaction
«block»«Contract»
Country TMS
Country A TMS
Country B TMS
Driver
Allow cross-border traffic corridor
Manage traffic in own country
Sense road traffic conditions
Calculate influence based upon sensed data
Ability to influence road traffic through actuators
Make request to bordering country if required
Ability to deduce if actions in own country are not sufficient
Communicate with neighbour
Inform neighbour when not required
Must respond to request from neighbour
Influence own traffic to meet needs of neighbour request
«include»

Figure 4 - Context Interaction View

5 SoS Architectural Modelling

We take a contractual approach to modelling the architecture of the Border Traffic SoS. We use the "Contracts Pattern", a low-level modelling pattern suitable for contractual specification of SoSs which was proposed by Bryans et al. (2014). We use this to define the Country TMS contract identified in the requirements engineering phase. The Contract Pattern identifies several viewpoints on the SoS model – defining the contract conformance relationships, connections between contracts, functionality offered, and contractual behaviour. In this section, we present a selection of these viewpoints to describe the architecture of our case study. The Contract Conformance View in Figure 5 (overleaf) shows that the SoS in this example comprises two TMS constituent systems (*Country A TMS* and *Country B TMS*), both of which conform to the *Country TM*S contract. The SoS itself conforms to the *BorderTrafficSoS* Contractual SoS. At present this element is considered simply to contain two contracts, but in the future we may extend the Contracts Pattern to include SoS-level properties which must be adhered to.

Having defined this relationship, we consider the connections present in the contractual model. In Figure 6 (overleaf), we identify how CSs conforming to the *Country TMS* contract are connected. In this case, the connection is implemented via a single interface, *tmsIF*, which is both provided and required by each contract. This interface may be further defined using the "Interface Pattern" (Perry et al 2014). The Interface Pattern refers only to the operations and state variables which are accessible at the interface of the CS, and makes no reference to details of internal implementation of functionality. Due to space requirements, we omit this detail in this paper.

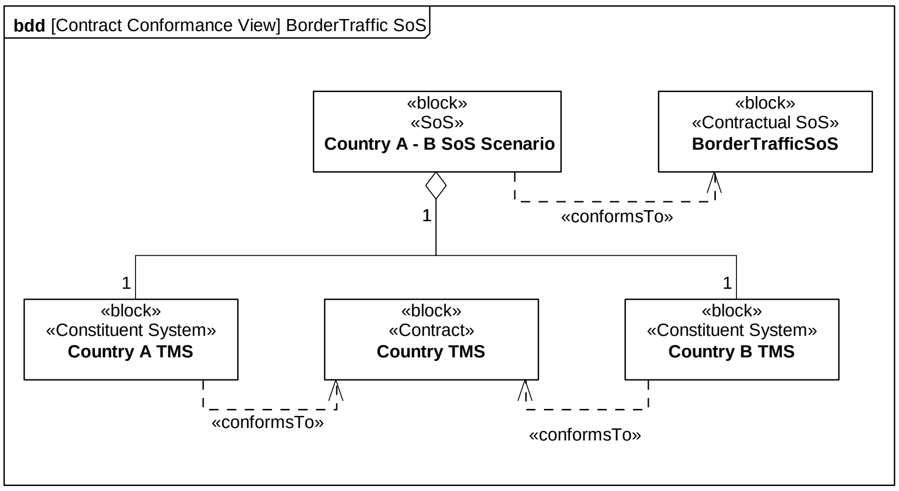

bdd [Contract Conformance View] BorderTraffic SoS
«block» «SoS» Country A - B SoS Scenario
«block» «Contractual SoS» BorderTrafficSoS
«conformsTo»
1
1
1
«block» «Constituent System» Country A TMS
«block» «Contract» Country TMS
«block» «Constituent System» Country B TMS
«conformsTo»
«conformsTo»

Figure 5 - Contract Conformance View for BorderTraffic SoS

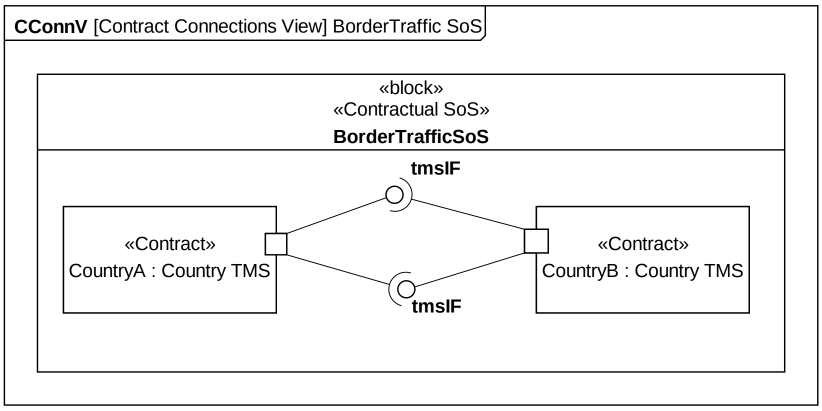

CConnV [Contract Connections View] BorderTraffic SoS
«block» «Contractual SoS» BorderTrafficSoS
tmsIF
«Contract» CountryA : Country TMS
«Contract» CountryB : Country TMS
tmsIF

Figure 6 - Contract Connections View for BorderTraffic SoS

Having identified the *CountryTMS* contract and its connections, we may begin to describe the functionality provided by the contract, through a collection of Contract Definition Views (CDVs). Aspects of functionality we can describe include: state variables; operations; and invariants. We begin by identifying the state variables (including their data types) and operations on the CDV in Figure 7.

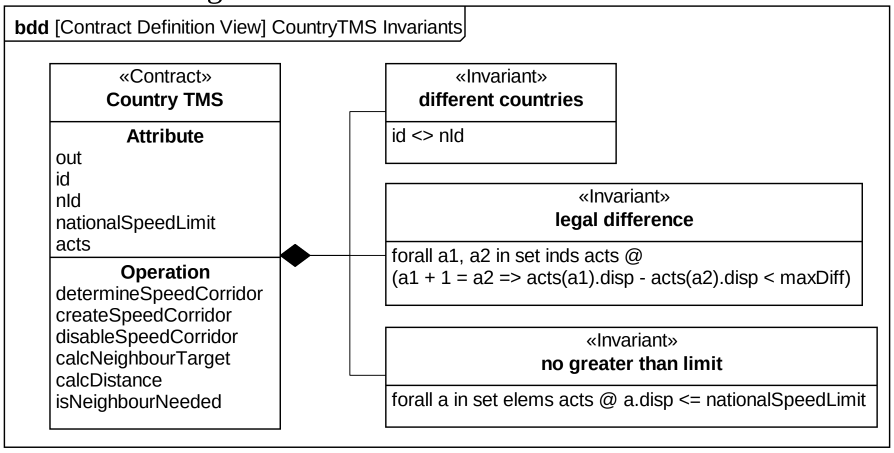

bdd [Contract Definition View] CountryTMS Invariants
«Contract» Country TMS
Attribute
out
id
nId
nationalSpeedLimit
acts
Operation
determineSpeedCorridor
createSpeedCorridor
disableSpeedCorridor
calcNeighbourTarget
calcDistance
isNeighbourNeeded
«Invariant» different countries
id <> nId
«Invariant» legal difference
forall a1, a2 in set inds acts @
(a1 + 1 = a2 => acts(a1).disp - acts(a2).disp < maxDiff)
«Invariant» no greater than limit
forall a in set elems acts @ a.disp <= nationalSpeedLimit

Figure 7 - Contract Definition View for CountryTMS Contract - Detailing Invariants

In Figure 7 we identify four variables: *id*,which is the unique identifier for the TMS; *nId*, which is the neighbouring country's identifier; *nationalSpeedLimit*, which is the speed limit for the given country; and *acts*, which is the collection of actuators in the country. Several operations are also identified; we return to these later.

Next, conditions are defined that must hold over the life of the contracts. The conditions are referred to as *invariants*. Invariants are predicates which dictate properties over the internal state of the contract, which must hold true given the changes to state performed by the contract operations. For example, Figure 7 identifies three invariants that describe: restrictions on the identifiers of the contracts; legal difference between speeds displayed on actuators; and an assurance that speed limits greater than host country's legally enforceable limit are not displayed. The operations of the CountryTMS are defined in the extract of the CDV in Figure 8.

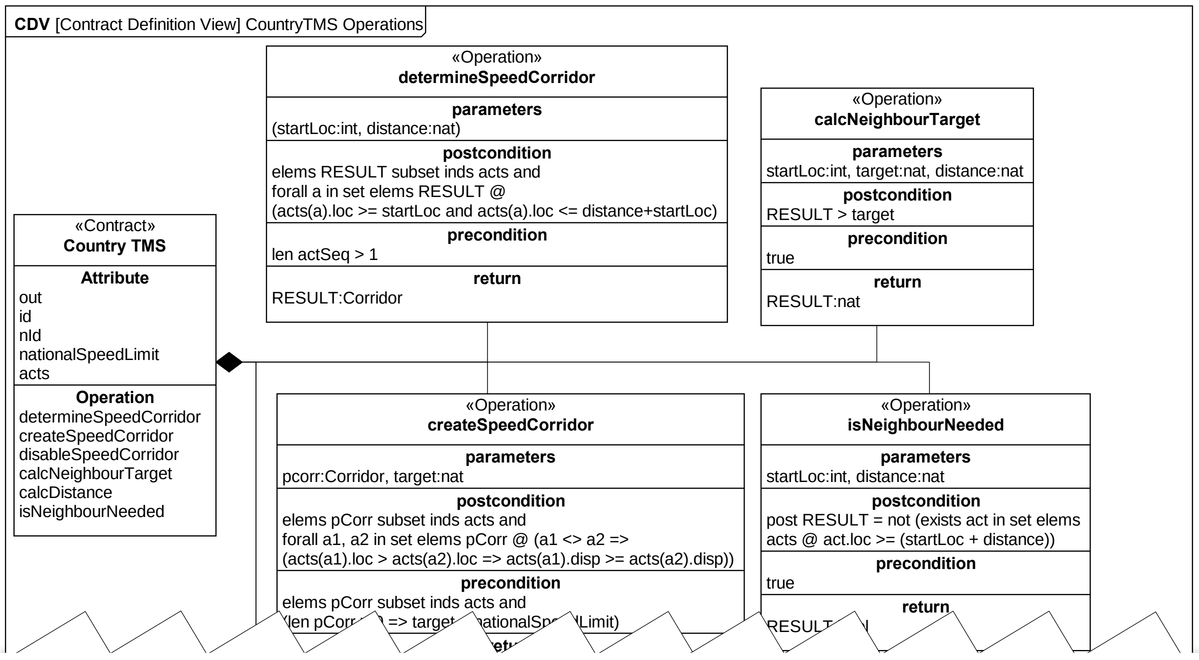

CDV [Contract Definition View] CountryTMS Operations
«Contract»
Country TMS
Attribute
out
id
nId
nationalSpeedLimit
acts
Operation
determineSpeedCorridor
createSpeedCorridor
disableSpeedCorridor
calcNeighbourTarget
calcDistance
isNeighbourNeeded
«Operation»
determineSpeedCorridor
parameters
(startLoc:int, distance:nat)
postcondition
elems RESULT subset inds acts and
forall a in set elems RESULT @
(acts(a).loc >= startLoc and acts(a).loc <= distance+startLoc)
precondition
len actSeq > 1
return
RESULT:Corridor
«Operation»
calcNeighbourTarget
parameters
startLoc:int, target:nat, distance:nat
postcondition
RESULT > target
precondition
true
return
RESULT:nat
«Operation»
createSpeedCorridor
parameters
pcorr:Corridor, target:nat
postcondition
elems pCorr subset inds acts and
forall a1, a2 in set elems pCorr @ (a1 <> a2 =>
(acts(a1).loc > acts(a2).loc => acts(a1).disp >= acts(a2).disp))
precondition
elems pCorr subset inds acts and
(len pCorr => target nationalSpeedLimit)
«Operation»
isNeighbourNeeded
parameters
startLoc:int, distance:nat
postcondition
post RESULT = not (exists act in set elems
acts @ act.loc >= (startLoc + distance))
precondition
true
return
RESULT

Figure 8 - Contract Definition View for CountryTMS Contract - Detailing Operations

The stereotyped *Contract* block includes eight operations – two of which are interface operations not defined further in the CDV. The remaining six are defined in accordance with requirements of the Contract Pattern: *determineSpeedCorridor*, *disableSpeedCorridor*, *createSpeedCorridor*, *isNeighbourNeeded*, *calcNeighbourTarget* and *calcDistance*. The Contract Pattern requires that, we must define parameter and return types and must provide preconditions and postconditions for operations. These additional conditions continue the contractual approach of the Contracts Pattern, by stating formally the reliances (preconditions) and guarantees (postconditions) placed upon the inputs and outputs of the operation. For example, *determineSpeedCorridor* is an operation defined in Figure 8; it is supplied with a precondition, which requires that for a speed corridor to be identified, the country must have more than one actuator. If this condition is satisfied, then the operation guarantees to return a speed corridor that adheres to the postcondition. The postcondition for this operation requires that the location of all actuators in the given corridor must fall between the identified target location, and a location which is *d* distance from that point (where *d* is the desired distance of the corridor). Preconditions and postconditions are written in CML, a notation described in more detail in Section 6.

The final Contract Pattern view defined for the example is the Contract Protocol Definition View (CPDV) in Figure 9.

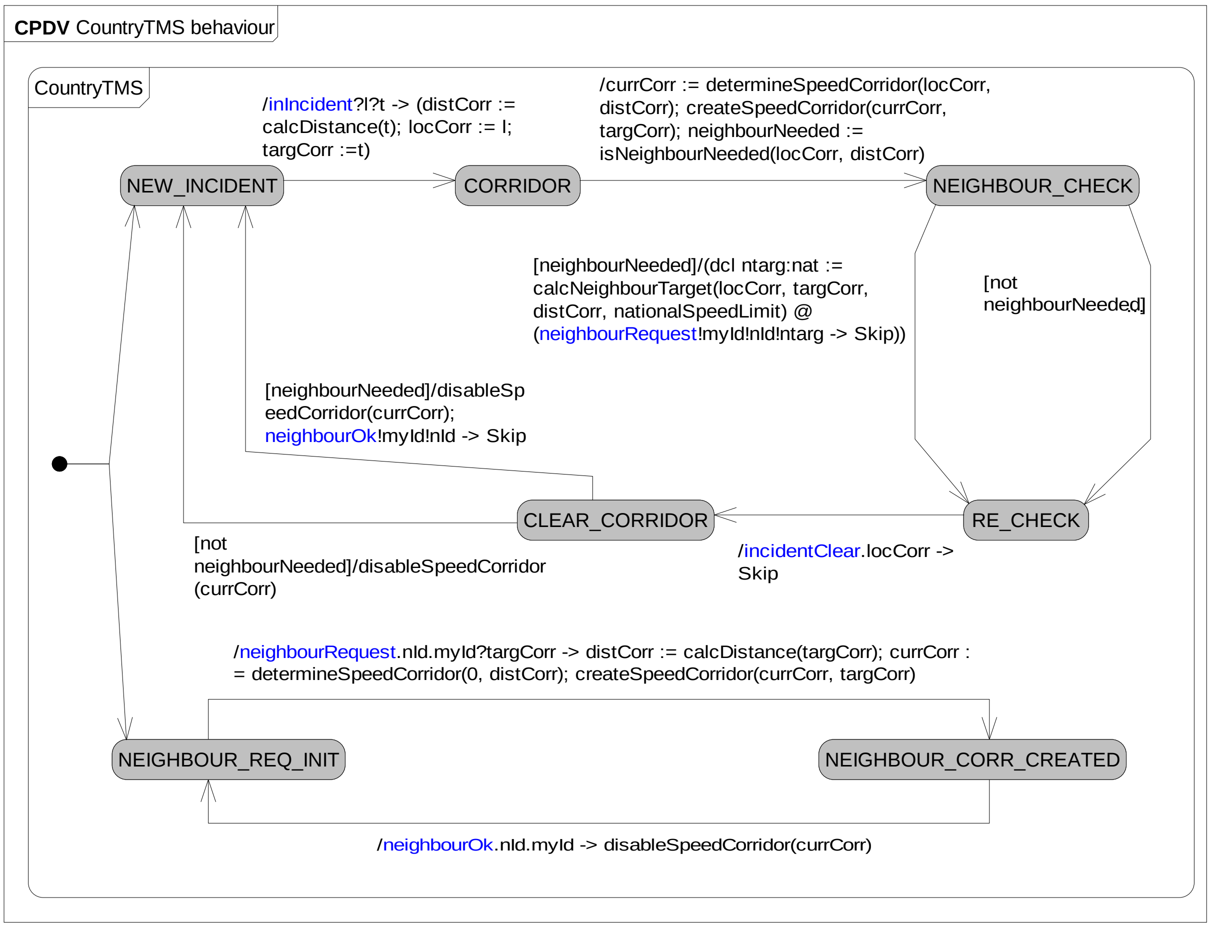

CPDV CountryTMS behaviour
CountryTMS
/inIncident?l?t -> (distCorr := calcDistance(t); locCorr := l; targCorr :=t)
/currCorr := determineSpeedCorridor(locCorr, distCorr); createSpeedCorridor(currCorr, targCorr); neighbourNeeded := isNeighbourNeeded(locCorr, distCorr)
NEW_INCIDENT
CORRIDOR
NEIGHBOUR_CHECK
[neighbourNeeded]/(dcl ntarg:nat := calcNeighbourTarget(locCorr, targCorr, distCorr, nationalSpeedLimit) @ (neighbourRequest!myId!nId!ntarg -> Skip))
[not neighbourNeeded]
[neighbourNeeded]/disableSpeedCorridor(currCorr); neighbourOk!myId!nId -> Skip
CLEAR_CORRIDOR
RE_CHECK
[not neighbourNeeded]/disableSpeedCorridor (currCorr)
/incidentClear.locCorr -> Skip
/neighbourRequest.nId.myId?targCorr -> distCorr := calcDistance(targCorr); currCorr := determineSpeedCorridor(0, distCorr); createSpeedCorridor(currCorr, targCorr)
NEIGHBOUR_REQ_INIT
NEIGHBOUR_CORR_CREATED
/neighbourOk.nId.myId -> disableSpeedCorridor(currCorr)

Figure 9 - Contract Protocol Definition View for Border Traffic Contract

This view dictates the permitted ordering of communications between contracts, events and operation calls. We identify two main behaviours: responding to an incident; and reacting to a neighbour's request. In the first behaviour, the contract describes receiving input from the environment, setting up a corridor and determining whether a neighbour must also set up a corridor. Subsequently we must wait for the environment to provide a 'clear' message before the corridor can be cleared. For the second behaviour (reacting to a request from a neighbour), the contract states that a new speed corridor should be set up given the requested target speed, which is then disabled given confirmation from the neighbouring country.

6 Formal Modelling and Analysis in SoS Engineering

Techniques employing semi-formal notations (in this case, SysML) supported by appropriate ontologies, frameworks and guidelines can support many rigorous analysis techniques. In this section, we illustrate the additional added value that may be gained through the use of a formal modelling notation alongside this, to increase the variety of analysis and validation techniques available to us. These additional techniques must be applied to a formal model of the SoS, and so we must first transition the SysML model to a model expressed in a formal modelling notation. The notation we use is CML, supported by the Symphony tool suite, both of which have been developed by the COMPASS project.

6.1 Transition to Formal Modelling Language

We have selected CML (Woodcock et al 2012) as a suitable formal notation for modelling of our SoS. CML is the first language specifically created for formal modelling of SoS, and it is flexible enough to support modelling of state as well as reasoning about the synchronisation of independent processes on events. CML models a system as a *process*; CML processes are stateful and reactive, and exchange messages through *channels*. The Symphony tool platform supports CML modelling and analysis. SysML models may be translated to CML manually by following translation guidelines (Miyazawa et al, 2013). Tool support is also available for performing automatic translation. Not all SysML constructs are currently supported, and so it may be necessary to conform to some minimum modelling idioms in order to ensure that models can be processed by automatic translators, and to complement the automatic translation with some manual translation. The automated translation tool support concentrates on single SysML blocks and state machines and the *Country TMS* CDVs in Figure 7 and Figure 8 and the CPDV in Figure 9 are the primary views processed by the translator. SysML data types and signals, not presented here, are used to generate CML data types and communication channels.

Having defined this model, we export the model into an XMI file that can then be used to generate a CML model using the translator plugin of the Symphony tool platform.

6.2 Formal Model Analysis: Overview

The resultant CML model can be used in various kinds of advanced analysis and property checking, which allow us to explore the quality of the initial system designs and identify at an early stage any design-related problems and conflicts. Furthermore, the Symphony tool suite can execute simulations of the CML model, allowing us to check that the global emergent behaviour of the SoS occurs as desired. We can group analysis techniques into two broad categories: validation (ensuring the SoS exhibits the desired behaviour) and verification (ensuring internal consistency and other properties). These two types of analysis can be performed independently, but naturally complement each other. We discuss some examples in more detail in this section.

6.3 Rapid Prototyping

We can leverage the simulation capabilities of the Symphony tool to execute simulations of the CML model of the SoS. This allows us to perform some “rapid prototyping” or design space exploration activities, to study the effects of various design changes applied to the model.

Typically, after translation, we have an initial, “skeleton” CML model, consisting of a reactive behaviour specification, data types and operation signatures only. In order to render the model executable, we must populate the model with more details about the algorithms which will be used to execute the operations. This involves adding operation bodies by adding statements expressed in CML. CML provides a set of familiar imperative language constructs for this purpose, such as looping structures, operation calls and conditional statements. Once the operation bodies have been supplied, the CML model can be executed in the Symphony simulator. The model can be modified and re-executed in order to experiment with different designs, checking for each modification that the required SoS behaviour is obtained. The simulator provides immediate feedback on any changes made, meaning that any problems in the design of the system can be identified here (i.e., at design time). The simulator can also be used to perform early, lightweight validation of behavioural requirements.

In the model we have created for our case study, two TMS processes are defined, one each for Country A and Country B. These two processes are then combined to form the global SoS process. Processes representing A and B operate independently, but communicate over a collection of channels. For the purposes of testing the model, we created a simple test process, which simulates the creation of two speed corridors: one corridor restricted to a single country; and another spanning both countries of the SoS. Additional test scenarios related to specific requirements can be defined as CML processes in the same way.

6.4 Validation of SoS Requirements

While rapid prototyping can help validate requirements, it is a somewhat ad-hoc technique that struggles to comprehensively cover the requirements. A more sophisticated approach is to use model-based testing (MBT) of requirements, which is available in the Symphony tool via the RT-Tester plug-in (Huang et al 2013). MBT is a very powerful technique that allows for the automatic generation of a great number of tests from a specification. MBT can be employed for many possible uses, but in this paper we couple MBT with test generation and execution, using RT-Tester and the Symphony simulator. The goal is to derive tests from the original SysML model (specifically, the CPDV in Figure 9) and then execute those tests against the simulator executions of the CML model. Instead of creating the test scenarios manually, as in rapid prototyping, test scenarios are then automatically generated from the SysML model. This ensures much more exhaustive coverage. Furthermore, since these test cases are derived from entities in the SysML model, traceability links can be established and tracked automatically by the SysML tool.

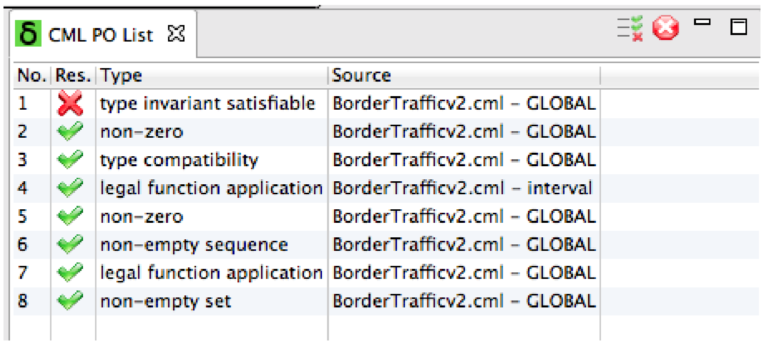
6.5 Verification of SoS Contracts

Verification of the SoS contracts consists of formally checking various kinds of properties. This allows us to detect various types of interface incompatibilities. While it is possible to formulate custom properties that are specific to a model this is a highly specialized task, and here instead we focus on presenting a class of properties that can be automatically derived from the model.

In general, these properties examine consistency of the SoS contract. Using the proof support of the Symphony tool (Couto et al 2014), we can check the satisfiability of the preconditions and postconditions of the contract operations – i.e., we check whether the SoS contracts can be fulfilled. If these properties cannot be established then we can conclude that the contract is unfulfilable, and some of its constraints must be relaxed. It is also possible to check that an executable model (e.g., as was generated after prototyping, described in Section 6.3) also respects the preconditions and postconditions of each operation – i.e., that it fulfils the contract. In addition, various kinds of general internal consistency POs may be generated, for example, that all indexed lookups in a sequence are valid.

All of these checks are carried out using a *generate and discharge* technique. The Symphony tool automatically generates a collection of properties, which must be true at a particular point in the model. The properties are then submitted to the Symphony theorem prover for *discharging*. Discharging is the activity of proving the property is true using the mathematical semantic rules of the CML language. The Symphony tool provides some automated discharge capabilities but typically not all properties can be discharged in full automatically. For this reason some properties must be proved manually, through user interaction with the tool (it's worth noting that currently not all properties are supported by the Symphony theorem prover).

A screenshot of the supported properties for our cross-border case study and the discharge results is given in Figure 10. We can see that most POs are discharged successfully by the tool.


CML PO List

No.	Res.	Type	Source
1	✗	type invariant satisfiable	BorderTrafficv2.cml - GLOBAL
2	✓	non-zero	BorderTrafficv2.cml - GLOBAL
3	✓	type compatibility	BorderTrafficv2.cml - GLOBAL
4	✓	legal function application	BorderTrafficv2.cml - interval
5	✓	non-zero	BorderTrafficv2.cml - GLOBAL
6	✓	non-empty sequence	BorderTrafficv2.cml - GLOBAL
7	✓	legal function application	BorderTrafficv2.cml - GLOBAL
8	✓	non-empty set	BorderTrafficv2.cml - GLOBAL

Figure 10 Results of Proof Obligation for CML model

PO failures should always be examined. Sometimes they indicate an issue with the model; the respective property does not hold and the model or requirements should be re-examined. Sometimes the failure is simply a case of the automatic discharge mechanism not being up to the task, and an interactive proof is needed.

6.6 Additional Formal Analysis

CML and the Symphony tool allow for other kinds of formal analysis. For example, property checking reactive parts of the model can be conducted via a model checker tool (Farias et al 2013). The model checker, like the theorem prover, allows the user to establish the truth of certain properties automatically, although this is done in a different way, and the types of properties the two tools establish are different. The model checker checks properties related to the specified reactive behaviour of the SoS, ensuring that certain kinds of reactive behaviour do not occur.

7 Conclusions and Future Work

In this paper we have provided a proof-of-concept example to demonstrate the use of both semi-formal and formal analysis techniques for the purposes of verifying aspects of an SoS, including the engineering of emergent behaviour. This type of analysis has been used by software engineers for engineering software systems with high reliability for some decades (Woodcock et al 2009). While formal modelling can be challenging to learn and integrate into an existing process, it does open up a wide range of possibilities in terms of analysis techniques and validity of analysis results, due to the highly rigorous nature of the modelling notation, and the fact that many analyses can be conducted automatically, or part-automatically. We suggest that recent innovations in the field of formal methods may be of interest to SoS engineers. For example, recent projects such as COMPASS have focussed on closer integration between SysML and formal notations (such as CML). This stepped approach makes formal techniques a more attractive prospect for modelling SoSs, particularly because formal techniques can make it possible to reason in a robust way about challenging emergent behaviour which is composed by a number of distributed and autonomous entities (for example).

We have shown that it is possible to obtain a flow from requirements and architectural modelling to the use of formal techniques. Through the use of a systematic and rigorous modelling approach, we have shown that SysML integrated with CML is a valid approach for a range of engineering activities. We recommend that an integrated approach is supported by the use of a consistent ontology, an architectural framework for SoS engineering and explicit use of traceability links. By exploiting the automated translation, we have shown the benefits of embedding systems thinking, facilitated by SysML, into the CML formalism which makes available verification and validation analyses. However, whilst we see clear benefits in enabling formal analyses on CML models derived from SysML, we require input from the SoS engineering profession in order to get more accurate assessments of cost-effectiveness in practice. In addition, several of the analysis tools (particularly the CML theorem prover and model checker) are still in development and achieving some advanced verification requires expert knowledge.

Initial results from the COMPASS project have demonstrated the benefit of integrating SysML with formal discrete-event models suitable for modelling software behaviour; our case study in this paper is intended to provide a proof of concept illustration. This approach allows SoS engineers to leverage formal analysis techniques, and in turn introduces systems thinking into CML modelling activities. A key challenge in SoS engineering is the heterogeneity of the CSs which may be drawn from different domains and include a mixture of hardware and software elements. As a result different modelling paradigms are required to represent the diverse domains typical in SoS engineering. For example, modelling notations employed in many engineering disciplines are underpinned by concepts of continuous time, which are not easily mapped to a discrete-event model. Closing this gap is an area of active research; for example recent work in "co-modelling" centres on tools which allow continuous time and discrete-event models to be designed collaboratively (Fitzgerald et al 2013). Drawing on results from the COMPASS project, new work in this area will include the EU-funded INTO-CPS project, which will develop an integrated tool chain for model-based design supporting multidisciplinary, teams co-developing requirements, designs and realisation in hardware and software.

Acknowledgements

The work presented here was supported by the EU Framework 7 Integrated Project “Comprehensive Modelling for Advanced Systems of Systems" (COMPASS, Grant Agreement 287829, http://www.compass-research.eu). Ingram and Payne also received support from the UK EPSRC platform grant on Trustworthy Ambient Systems (TrAmS-2) and the Newcastle University Science Central initiative.

Bibliography

Abbott, R., 2006. “Open at the top; open at the bottom; and continually (but slowly) evolving”. In *IEEE/SMC International Conference on System of Systems Engineering*, 2006.

Boardman, J. and Sauser, B, 2006. “System of Systems – the meaning of ‘of’”. In *Proceedings of the International Conference on System of Systems Engineering*, Los Angeles, CA.

Bryans J, Fitzgerald J, Payne R, Kristensen K., 2014. Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML. In: INCOSE International Symposium (IS2014). 2014, Las Vegas, Nevada.

Couto, L., Foster, S., Payne, R., 2014. “Towards Certification of Constituent Systems through Automated Proof”, In *Proceedings, Engineering Dependable Systems of Systems Workshop (EDSoS)*, 2014.

Dahmann, J.. “Systems of Systems Pain Points”. In Proceedings of the 24th INCOSE International Symposium. Henderson, NV, USA. INCOSE, June 2014.

Dahmann, J., Rebovich, G. and Lane, J.A, 2008. “Systems Engineering for Capabilities”, *Cross Talk: The Journal of Defense Software Engineering*, November 2008.

Farias, A., Mota, A., Didier, A., Woodcock, J., 2013. “Model Checking Support”, Technical Report, COMPASS Deliverable D33.1. [Online] http://www.compass-research.eu/Project/Deliverables/D331.pdf [Visited Nov 2014]

Fitzgerald, J.S., Larsen, P. G., Pierce, K. G. and Verhoef, M.H.G., 2013. “A Formal Approach to Collaborative Modelling and Co-Simulation for Embedded Systems”, *Mathematical Structures in Computer Science*, 23:04.

Fitzgerald J, Larsen PG, Woodcock J. 2013. "Foundations for Model-Based Engineering of Systems of Systems". In: Fourth International Conference on Complex Systems Design & Management (CSD&M) Paris, France. Springer.

Hallerstede, S., Hansen, F.O., Holt, J., Lauritsen, R., Lorenzen, L. and Peleska, J, 2012. "Technical Challenges of SoS Requirements Engineering". *Proceedings of the 7th International Conference on System of System Engineering*, IEEE SoSE 2012.

Holt, J., Perry, S., Hansen, F.O. and Hallerstede, S., 2012. "Report on Guidelines for SoS Requirements", Technical Report, COMPASS Deliverable D21.1. [Online] http://www.compass-research.eu/Project/Deliverables/D211.pdf [Visited Oct 2014]

Huang, W., Peleska, J., Schulze, U., 2013. "Test Automation Support", Technical Report, COMPASS Deliverable D34.1. [Online] http://www.compass-research.eu/Project/Deliverables/D341.pdf [Visited Nov 2014]

INCOSE 2011. "Systems Engineering Handbook: A Guide for Systems Life Cycle Processes and Activities". Technical Report INCOSE-TP-2003-002-03.2.2, International Council on Systems Engineering (INCOSE), San Diego, CA, USA. October 2011.

Maier, M. W. 1998. "Architecting Principles for Systems-of-Systems", *Systems Engineering* 1(4), 267-284.

Miyazawa, A., Cavalcanti, A., Iyoda, J., Cornelio, M., Albertins, L. and Payne, R., 2014. "Final Report on Combining SysML and CML", Technical Report, COMPASS Deliverable D22.4. [Online] http://www.compass-research.eu/Project/Deliverables/D224.pdf [Visited Oct 2014]

OMG 2010. "OMG Systems Modeling Language Version 1.2", June 2010, http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

Road2SoS, 2013. "Report on Commonalities in the Four Domains and Recommendations for Strategic Action", TR Road2SoS D5.1 and D5.2. [Online] http://road2sos-project.eu/cms/front_content.php?idcat=72 [Last visited October 2014]

Perry, S., Holt, J., Payne, R., Bryans, J., Ingram, C., Miyazawa, A., Hallerstede, S., Couto, L.D., Malmos, A.K., Iyoda, J., Cornelio, M. Peleska, J., 2014. "Final Report on SoS Architectural Models", Technical Report, COMPASS Deliverable D22.6. [Online] http://www.compass-research.eu/Project/Deliverables/D22.6.pdf [Visited Oct 2014]

T-AREA-SoS, 2013. "Strategic Research Agenda on Systems of Systems Engineering", TR, TAREA-PU-WP5-R-LU-26, Issue 2, August 2013. [Online] https://www.tareasos.eu/results.php?user=anon [Last visited October 2014]

Sanders, J. W. and Smith, G. "Emergence and Refinement". *Formal Aspects of Computing*, 24(1):45–65, 2012.

Woodcock, J., Larsen, P.G., Bicarregui, J. and Fitzgerald, J., 2009. "Formal Methods: Practice and Experience". *ACM Computing Surveys*, 41(4):1-40, 2009.

Woodcock, J, Cavalcanti, A., Fitzgerald, J., Larsen, P.G., Miyazawa, A. and Perry, S., 2012. "Features of CML: a Formal Modelling Language for Systems of Systems". In *Proceedings of the 7th International Conference on System of System Engineering.*

Biography

Claire Ingram earned her PhD at Newcastle University in 2011, with doctoral research on early stage design and requirements data. She has worked on the DESTECS project, which supported fault-tolerant embedded systems, and the COMPASS project, examining model- based techniques for SoSs. Her research interests lie in empirical software engineering and systems engineering, SoSs and architectures. She is an INCOSE member and has been an active collaborator with the SoS working group at INCOSE since 2013.

Richard Payne's research interests include architectural modelling, systems of systems, and verification tools for formal methods. Following his PhD (Newcastle University 2012) on models of dynamic reconfiguration, he worked on the UK Ministry of Defence project on Interface Contracts for Architectural Specification and Assessment, and the COMPASS project, designing advanced modelling and verification techniques. He is a member of INCOSE.

John Fitzgerald is a full Professor in Computing Science at Newcastle University, UK, where he leads research on model-based engineering and the Cyber-Physical Systems Lab. He recently led the international COMPASS project on model-based methods for systems of systems engineering. With a background in verification technology (PhD, Manchester Univ., 1991), he has contributed to methods and tools applied commercially in areas as diverse as chip design and options trading. He is a member of INCOSE, ACM, and IEEE, and is a Fellow of the BCS.

Luís Diogo Couto is a PhD student in Computer Engineering at Aarhus University, Denmark. His PhD work is focused primarily on extensibility of software systems. Other research interests include formal methods, tooling and modelling. He is working on the COMPASS project, developing tools and techniques for formal model analysis. He is also one of the primary developers of the Overture tool for formal modelling.