COMPUTING SCIENCE

Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML

Jeremy Bryans, John Fitzgerald, Richard Payne, Klaus Kristensen

TECHNICAL REPORT SERIES

No. CS-TR-1406 January 2014

TECHNICAL REPORT SERIES

No. CS-TR-1406 January, 2014

Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML

J. Bryans, J. Fitzgerald, R. Payne and K. Kristensen

Abstract

This paper describes a pilot study in the use of model-based techniques in system of systems (SoS) engineering. The focus is on the derivation of specifications for new constituent systems that are to be integrated with an existing SoS. The pilot study is based on a commercial application in the home audio/video domain and illustrates the application of architectural modeling guidelines to the description of a content-streaming SoS using SysML and the formal modeling language CML. Analysis of the models leads to the derivation of a specification sufficient for constituent systems to guarantee a key leader election property of the SoS.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

BRYANS, J., FITZGERLAD, J., RICHARD, P., KRISTENSEN, K.

Maintaining Emergence in Systems of Systems Intergration: a Contractual Approach using SysML
[By] J. Bryans, J. Fitzgerald, R. Payne, and K. Kristensen
Newcastle upon Tyne: Newcastle University: Computing Science, 2014.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1406)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1406

Abstract

This paper describes a pilot study in the use of model-based techniques in system of systems (SoS) engineering. The focus is on the derivation of specifications for new constituent systems that are to be integrated with an existing SoS. The pilot study is based on a commercial application in the home audio/video domain and illustrates the application of architectural modeling guidelines to the description of a content-streaming SoS using SysML and the formal modeling language CML. Analysis of the models leads to the derivation of a specification sufficient for constituent systems to guarantee a key leader election property of the SoS.

About the authors

Jeremy is a Senior Research Associate in the School of Computing Science, and a member of Centre for Software Reliability. His research interests are in the modelling and analysis of collaborating systems, and in the development of trustworthy policies for their interaction. He is currently a Co-Investigator on the EPSRC project Trusting Dynamic Coalitions, on which he works on designing and composing provenance policies for coalition members. Part of his time is spent on the EU project COMPASS, on which is developing semantic foundations for a modelling language for Systems of systems.

John Fitzgerald is Professor of Formal Methods in Computing Science, and Director of the Centre for Software Reliability at Newcastle. He is a specialist in the engineering of resilient systems, particularly in rigorous analysis and design tools. He leads the international COMPASS project, which is developing technology for engineering complex "Systems of Systems" and heads Newcastle's research into co-modelling and co-simulation in the design of fault-tolerant cyber-physical systems in several EU and UK-funded projects. He studied formal proof (PhD, Manchester Univ.), before joining Newcastle, where he worked with British Aerospace on the design of avionic systems in the 1990s. He went on to study the industrial application of formal modelling (specifically the Vienna Development method – VDM) as a SERC Fellow and later as a Lecturer at Newcastle. He returned to the University in 2003, having established the design and validation team in a successful SME in the embedded processor market. John is Chairman of FME, the main European body bringing together researchers and practitioners in formal methods of systems development. He is a Fellow of the BCS, and a member of the EPSRC College, the ACM and IEEE.

Richard Payne obtained his PhD in 2012 at Newcastle University under the supervision of Prof. John Fitzgerald, titled Verifiable Resilience in Architectural Reconfiguration. As part of his PhD, Richard provided a basis for the formal verification of policies defined using a reconfiguration policy language (RPL) for the governance of resilient component-based systems. Richard worked as an RA on the Ministry of Defence funded SSEI project and was involved in the 'Interface Contracts for Architectural Specification and Assessment' sub task, investigating the use of contract-based interface specification in system of systems architectural models. Richard is now working on the COMPASS project, on the use of model-based techniques for developing and maintaining systems of systems, involved with work in architectural modelling, fault modelling and tool development.

Klaus Kristensen (MSc cum laude Computer Science with Mathematics, University of Aalborg, Denmark) has over 18 years' experience working as a software engineer and software architect in companies including Bang & Olufsen, Motorola and LEGO, and is an expert in Audio/Video systems engineering. In the COMPASS project, he leads work on the instantiation and evaluation of model-based SoS engineering methods in the AV domain.

Suggested keywords

SYSTEM INTEGRATION
SYSTEMS OF SYSTEMS
SYSML
EMERGENCE
MODEL-BASED SYSTEMS ENGINEERING

Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML

Jeremy Bryans, John Fitzgerald, Richard Payne
Newcastle University, United Kingdom.
firstname.lastname@ncl.ac.uk

Klaus Kristensen
Bang & Olufsen, Denmark.
KRT@Bang-Olufsen.dk

Abstract. This paper describes a pilot study in the use of model-based techniques in system of systems (SoS) engineering. The focus is on the derivation of specifications for new constituent systems that are to be integrated with an existing SoS. The pilot study is based on a commercial application in the home audio/video domain and illustrates the application of architectural modeling guidelines to the description of a content-streaming SoS using SysML and the formal modeling language CML. Analysis of the models leads to the derivation of a specification sufficient for constituent systems to guarantee a key leader election property of the SoS.

Keywords: system integration, systems of systems, SysML, emergence, model-based systems engineering

1. Introduction

Systems of Systems (SoSs) are groups of independently owned and managed systems which collectively offer emergent functionality that cannot be provided by the constituent systems (CSs) alone (Maier 1996, Jamshidi 2008). As reliance comes to be placed on the emergent behaviors that SoSs deliver, successful systems engineering demands methods and tools that allow this reliance to be justified. Consequently, although SoSs are not themselves new, there has been a marked growth of interest in the potential of SoS engineering (Kemp et al. 2013).

The systems engineer working in an SoS environment faces important challenges, not only in SoS development, but in the long term, as the CSs evolve, possibly quite independently of the SoS itself (Boardman and Sauser 2006, INCOSE 2011). In this setting, one of the key difficulties is managing the integration of new or modified CSs. As a CS is introduced or updated, there is a need to assess the risk of undesired behaviors compromising the delivery of the emergent SoS-level behavior. This is not only a technical challenge: the integration of a CS entails specification of the contractual relationship between the existing CSs and the new element, affecting procurement.

Model-based methods are seen as a way of addressing significant challenges in SoS Engineering (Cantot and Luzeaux 2011). Explicit models expressed using standard frameworks and languages such as SysML can help to clarify areas of incompleteness and ambiguity in specifications, and act as a lingua franca for stakeholders to negotiate interfaces. Models that are expressed in languages with mathematical semantics (so-called *formal models*) allow deep analyses such as the discovery of defects such as feature interactions and deadlocks that could compromise the delivery of the SoS emergent behavior.

In the COMPASS project (http://www.compass-research.eu), we aim to advance support for model-based techniques to provide methods and tools that are rich enough to express the characteristics of SoSs but also have formal foundations, enabling machine-assisted analysis of emergent properties. In this paper we describe and evaluate an approach to assist in the task of SoS integration through the specification of CS interfaces, leveraging system modeling in SysML with formal methods to support the verification of emergent SoS

properties. In this paper we focus on ICT interfaces; assumptions that a CS may make at a given interface, and the commitments that it makes provided the assumptions are satisfied, can be recorded. We present a proof-of-concept study led by an industrial user of the technology that has allowed us to evaluate the potential of these techniques.

We first outline the COMPASS technology for SoS modeling and analysis (Section 2) and consider relevant work in SoS, architecture and interface specifications, and formal modeling (Section 3). Section 4 introduces our approach to defining interface and contractual specifications. Section 5 describes the proof of concept study through its SysML models, and Section 6 details the formal modeling and analysis work. Finally in Section 7 we evaluate the approach.

2. COMPASS Technology

In COMPASS, we aim to deliver guidelines for the systems engineering of SoSs covering requirements, architecture and integration, including the presentation of modeling patterns in SysML (Object Management Group OMG 2012). In order to facilitate formal verification, we have developed the COMPASS Modeling Language (CML) (Woodcock et al. 2012) which is capable of recording data, user-defined data types with invariants, functionality (both algorithmically and in terms of precondition/postcondition pairs) and behavior. In CML, an SoS is modeled as the parallel composition of concurrent processes (modeling the CSs). CML supports a rich collection of static and dynamic analysis methods, including some forms, such as consistency checking, that deliver results which can be readily reflected back to the SysML architectural level. Other forms of analysis, including model-checking and symbolic proof, can be used to verify emergent properties. CML is formal, with semantics expressed using Unifying Theories of Programming (Hoare and Jifeng 1998).

Tools to support the COMPASS technology include a SysML base built on Atego's Artisan Studio tool, and an Eclipse-based platform for CML which includes tools for model creation and analysis plugins that permit the exploration of models by simulation, the generation and management of dynamic tests, and analysis by proof and model-checking. A translator from SysML to CML allows coupling between the two, and allows extension to other modeling notations.

3. Related Work

We propose the contractual description of architectural interfaces within the widely used notation SysML (Object Management Group OMG 2012). SysML allows basic operation signatures to be defined at interfaces, and pre- and postconditions to be specified textually, though these are rarely used in practice. This has much in common with the Design by Contract (DbC) software engineering technique (Meyer 1988), used to constrain software operations. In previous work, we have considered the role of nonfunctional properties and DbC in architectural interfaces (Payne and Fitzgerald 2010) and in (Bryans, Payne et al. 2013) we show how interfaces in SysML can be translated into CML. Related work in this area is surveyed more comprehensively in (Bryans, Payne et al. 2013).

(Mordecai and Dori 2013) also aim to develop a model-based approach to the design of interfaces and interactions among disparate systems. They seek to ensure interoperability of the SoS by developing a framework for integration in which the integration domain is a treated as a system in its own right. We take a complementary approach: interfaces and contracts are developed separately and combined, then subjected to appropriate (in our case formal) analysis.

The concept of emergence is used in the literature to describe a range of phenomena and there

is still debate about its precise meaning. We are interested in nominal emergence in the sense of (Baldwin 2013) and our definition aligns with that of Maier: *An SoS performs functions not resident in any single component system* (Maier 1998)**.**

The study we use (a Bang & Olufsen Audio/Video network) has also been used in work on the verification of deadlock (Antonino et al. 2014) with CML.

4. Interface and Contract Modeling

The task of the SoS integrator is formidable, and "SoS integration can be a complex, risky, long, and frustrating effort" (Mordecai and Dori 2013). In developing a new SoS, an integrator must have the ability to determine that the combination of proposed contracts and interfaces leads to the desired emergent behavior. When maintaining an existing SoS they must know that any changes to contracts do not adversely affect emergent behavior.

The COMPASS engineering guidelines include processes for model-based requirement engineering, architectural modeling and system integration in SoS using SysML (Holt and Perry 2013). In order to support SoS integration, the authors propose the definition of CS interfaces (the publically visible behavior of the CS) using the *Interface Definition Pattern* (Perry 2013) comprising: the Interface Identification View (IIV), which identifies the interfaces of a CS and their relation to the CSs that use them; the Interface Connectivity View (ICV), which shows how interfaces and ports are connected in a SoS; the Interface Description View (IDV), which defines the interfaces in terms of the operations they provide; the Interface Behaviour View (IBV), which identifies typical scenarios at the interface; and the Protocol Definition View (PDV), which describes protocols to which the interface must conform. Not all views are required when modeling SoS interfaces; in this paper, we use only the ICV, IDV and PDV.

In previous work, we proposed interface definitions as a way of dealing with CS independence and SoS integration (Bryans et al. 2013). However, the interface definition is limited to scenarios, protocols, and operation signatures. The verification of emergent behavior requires a richer model of the CS, showing the influence of the CS's state on the functionality available at the interface. We refer to this richer definition, taken together with the interface definition, as a *contract*. We therefore propose a *contract specification*, which contains two views in addition to the Interface Definition Pattern: the *contract definition view* models the internal operations and state values of the contract, and the *contract protocol view* describes how combinations of these operations and values influence the behavior visible at the CS interface.

We propose a method that may be used alongside the activities identified in the COMPASS guidelines to ensure that the integration of a new CS respects the emergent behaviors visible at the SoS boundary. The method first uses the Interface Definition Pattern to define the SysML interfaces of the CSs in an existing SoS, and then identifies contracts that must be respected by the CSs at their interfaces. Contract specifications are defined in SysML at an appropriate level of abstraction so that, when contracts are defined, the composition is sufficiently detailed to allow analysis of emergent behavior. Using the COMPASS tools, a SysML model with a contract specification can be translated to CML which, with its formal underpinnings, makes the model amenable to formal analysis, allowing us to analyze emergent properties.

It is worth noting that a CML model may additionally record invariant restrictions on data, and may characterize operations in the form of preconditions and postconditions. Following translation of the SysML model to CML, these elements may be added by the modeler, allowing for yet more comprehensive analysis. In future work, we expect to explore the

potential for extending the contract notion to allow these features to be added at the SysML level.

In Section 5.1, we introduce an industry-led study of an Audio/Video SoS, and in Section 5.2 we show the modeling undertaken to describe the interfaces and contracts of the SoS. In Section 6 we translate the contracts we develop into a CML formal model, and describe the analysis of the SoS emergent behavior.

5. A Proof of Concept Study: Emergent Leader Election in Audio/Video Systems of Systems

5.1. Proof of Concept Study Description

A Bang & Olufsen (B&O) home Audio/Video (AV) network consists of several devices (such as audio, video, gateway and legacy audio products) which may be produced by competing manufacturers and distributed across a user's home. Such a network is an SoS: it exhibits the dimensions typical of an SoS as described in (Fitzgerald et al. 2013). The individual CSs exhibit a (potentially) wide variation in *autonomy*. They all operate at the behest of the user, but the fact that they may be legacy or non-B&O systems means that they may only offer a limited degree of controllability from the point of view of the SoS. The CSs exhibit operational *independence*; they provide stand-alone streaming or content browsing experiences, e.g. watching TV or selecting music to play. The CSs are typically *distributed* in different zones/rooms, the AV content can be local or remote, and the location of content source is often transparent to the user. Geographical distribution leads to *emergent* behaviors such as making sound follow the user around, driven by contracts between streaming and clock systems. The CSs undergo *evolutionary* development. The stakeholders will have an evolution vision that is not necessarily compliant with that of B&O. There is *dynamic reconfiguration* behavior in that products join or leave the SoS during streaming or browsing operations; products can be turned off by users or enter power-saving mode. While products have no *interdependence*, CSs rely on each other in order to deliver the emergent behavior that fulfills the SoS goal.

Constituent systems may join or leave the network at any time, but a consistent user experience (such as a playlist, current song, etc.) must be provided, and this requires availability and consistency of the system configuration data. In order to do this, a publish/subscribe architecture is employed. This in turn, requires that the underlying network is able to elect a leader from among the CSs. As there is no centralized control, the ability to elect a leader is a required emergent property of the SoS. We present a requirements definition view (RDV) in Figure 1 to specify these minimal requirements.

The RDV, constructed during the requirements engineering process (Holt and Perry, eds., 2012) provides the means to identify and define the requirements. The remainder of the process (not relevant to our argument in this paper) places the requirements in the context of the SoS stakeholders. The RDV shows that the B&O user experience requirement contains (at least) two high-level functional requirements – Audio/Visual Streaming (R2) and Remotely-Located Content-Browsing (R3). There is a constraint on the user experience, which states that the configuration must be available and consistent (R1). In this paper, we largely focus on the Identification of a Single Leader (R1.1), and briefly revisit the SoS problem of Constituent System Integration (R1.2) in Section 6.

The leadership problem is a distributed consensus problem in a network with unreliable processes and asynchronous communication. When the CSs of the network are in an election state, no publisher is present and the multi-room experience space is inconsistent and

unavailable. During the election, the devices must react to a set of local transition rules that will guarantee the desired emergent property of a leader in the network, and allow the network to enter the publisher-subscriber state.

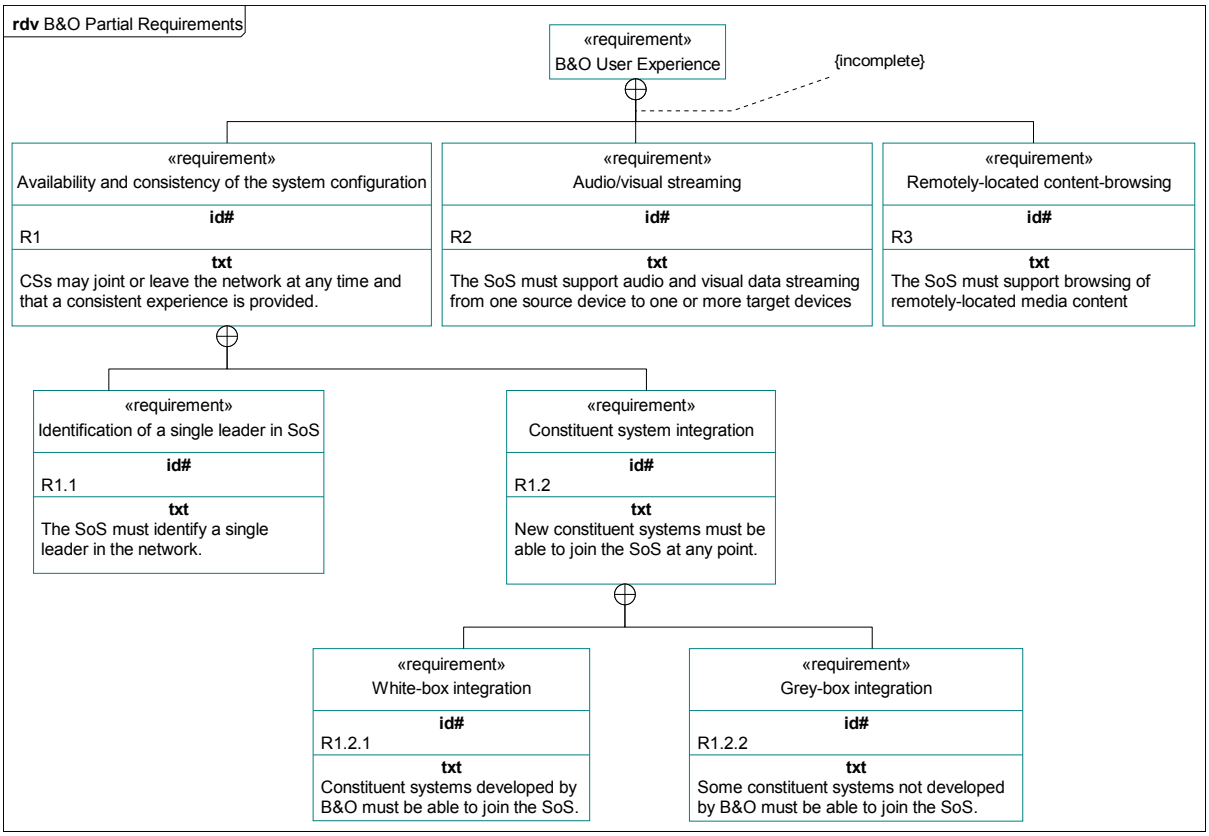

Figure 1. Requirements definition view for B&O AV SoS

5.2. Audio/Video Systems of Systems Architectural Modeling

At a high-level structural view, as depicted in the SysML block definition diagram (BDD) in Figure 2, the B&O AV SoS is composed of devices which are capable of participating in a leader election (which we therefore refer to as LE Devices) and a Transport Layer. Figure 2 states that the LE Device contract may be 'implemented' by several AV Devices of varying degree of openness towards the integration into a given SoS. The categorization consists of *blackbox*, *greybox* and *whitebox* CS. Whilst any device may be integrated that satisfies the contract, the degree to which they may be modified by the SoS integrator varies. A *blackbox* CS must be integrated without any changes; the integrator has full control over the architecture of a *whitebox* CS and may therefore make arbitrary changes; and a *greybox* CS allows limited internal changes to be made for integration purposes. The Transport Layer contract is specialized by a Network.

Having defined this SoS composition, we can begin to consider: 1) the connections within the B&O AV SoS and the interfaces between the contractual specifications, and 2) the behavior of the LE Device and Transport Layer contractual specifications.

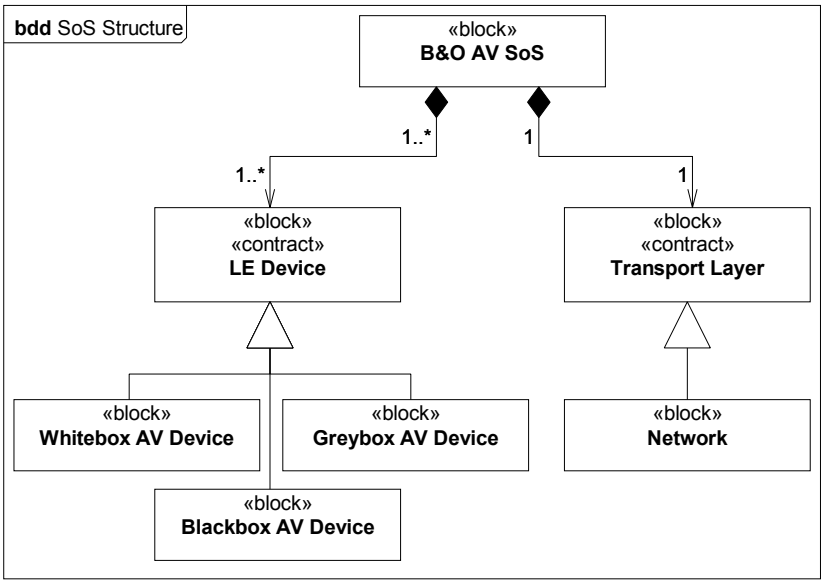

Figure 2. B&O AV SoS Composition Structure

Interface Definition

We first consider the SoS connections and interfaces, and use the COMPASS interface definition pattern to populate these views. In Figure 3, we present the interface connectivity view (ICV) for a simple SoS consisting of only two LE Devices and a single Transport Layer; a larger SoS would have multiple LE Devices connected to the single Transport Layer. The view shows that there are only two interfaces between these contractual specifications: *rec* and *send*. The *rec* interface is provided by the LE Device and required by the Transport Layer, whilst the *send* interface is provided by the Transport Layer and required by the LE Device. The interface naming has been given from the perspective of the LE Device.

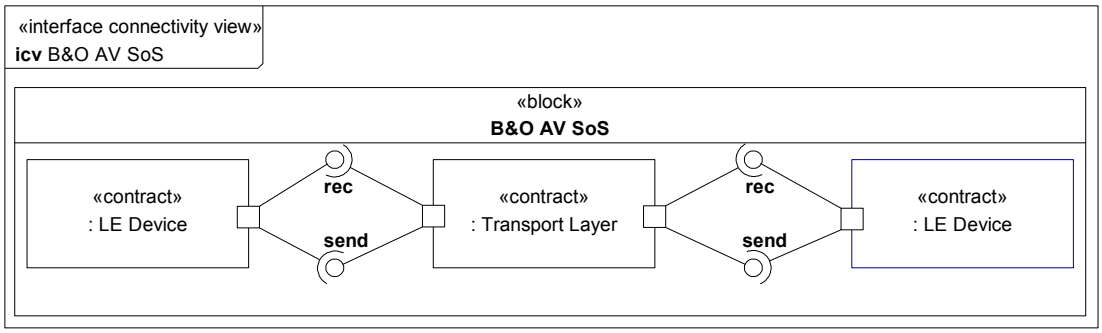

Figure 3. Interface connectivity view for B&O AV SoS

In Figure 4, we present the interface definition view (IDV) for the SoS, which shows the operations available at the interface. This shows that these interfaces are very simple; each contains a single operation. The operations each have the same parameters; the sender (*sendId*) and receiver (*recId*) LE Device identifiers and the data (*data*) being communicated.

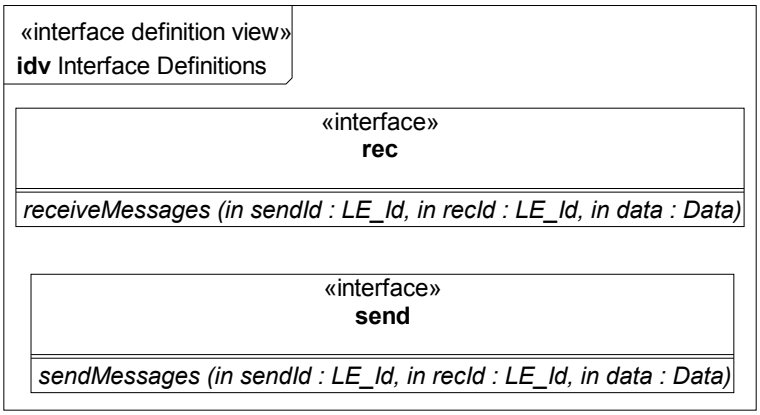

Figure 4. Interface definition view for B&O AV SoS

The final part of the interface definition is concerned with the ordering of messages. The interfaces are simple and, once initialized, there is no required ordering when sending and receiving messages at either the LE Device or the Transport Layer ports, as shown in Figure 5.

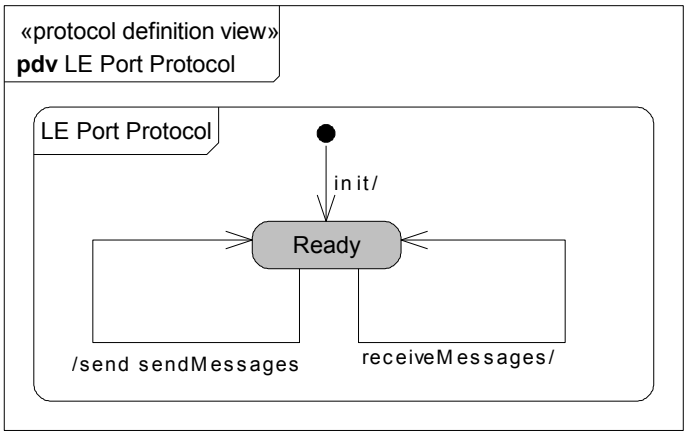

Figure 5. Protocol definition view for LE Port in B&O AV SoS

As mentioned in Section 4, this interface specification is not sufficient to use as a specification to guarantee the requirement for the Leadership Election emergent behavior. As such, we further model the LE Device and Transport Layer contractual specifications.

LE Device Contract

Figure 6 presents a contract definition view, which provides more details of the LE Device contract. The diagram includes the (private) operations and values of the contractual specification, along with the public operation provided by the *rec* interface. These operations are used to perform two main functions: to receive and process information from other LE Devices, and to use this processed information to make claims about the device's leadership status. The device transmits information about its claim and the strength of that claim to the Transport Layer through the Transport Layer's send operation.

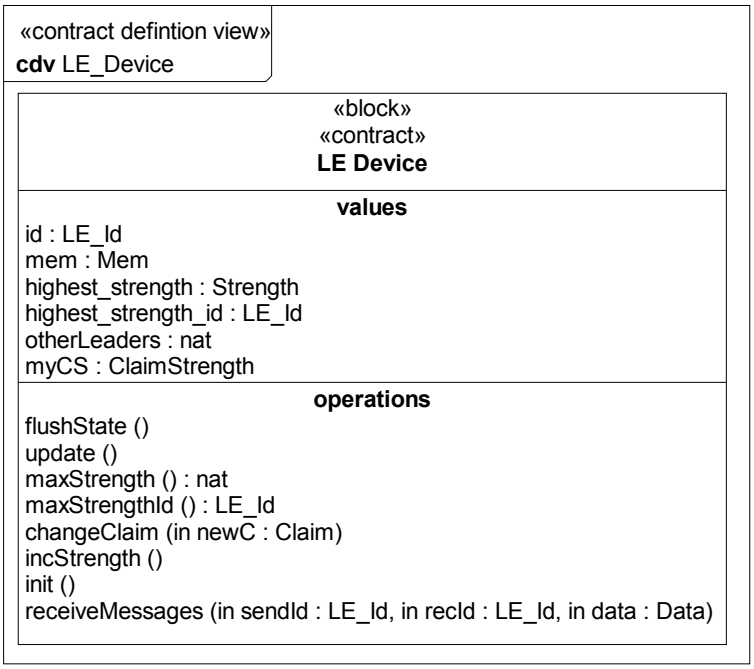

Figure 6. Contract definition view depicting LE Device contract specification

In Figure 7, we present the contract protocol view for the LE Device contract. This diagram allows us to describe the ordering of operation calls performed by an LE Device. Using this diagram, we describe the states the device may enter (*Leader*, *Follower* or *Undecided*).

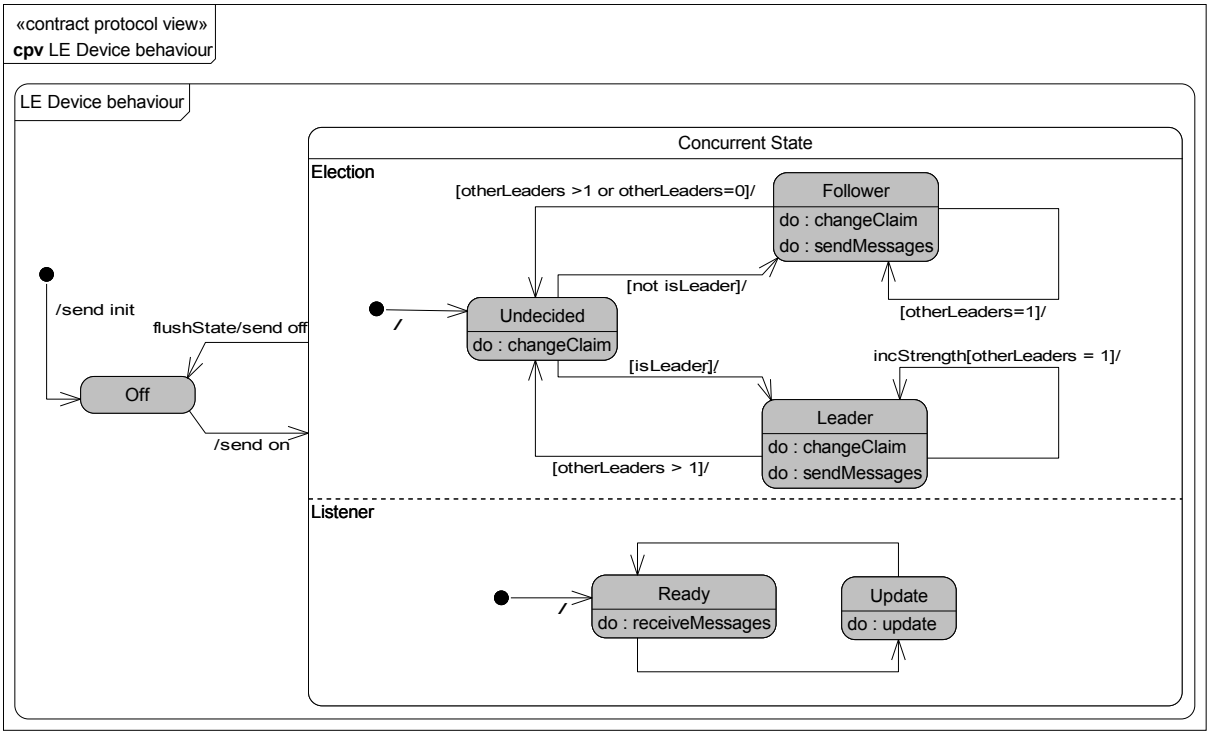

Figure 7. Contract protocol view for LE Device contract specification

From the diagram, we see that after initialization, a LE Device enters the *Off* state. From here, it may be turned on and it enters a parallel state with two sub-states: the *Listener* and *Election* states. In the *Listener* state, the LE Device repeatedly receives messages from its environment and performs the update operation. In the *Election* state, the LE Device initially enters the *Undecided* state, where it updates its claim to be undecided. After some time, the LE Device

queries its updated attributes (these are updated in the *Listener* state), and decides if it can be a leader on the network.

If the LE Device can be a leader, it enters the *Leader* state, updates its claim and sends a message to all other devices it knows about. From this state, the LE Device continuously checks if it may be a leader. If, after becoming a leader, it receives a message from an existing leader, the LE Device enters the *Undecided* state. If the LE Device may remain a leader, then it increases its strength of the claim to be a leader and again sends a message to all other devices. Allowing successful leaders to increase the strength of their claim is a heuristic to increase the likelihood of successful leaders remaining as leaders, and thereby reduce the number of new elections.

If the LE Device is not a leader, it enters the *Follower* state. Once in this state, the LE Device updates its claim to be a follower sends a message to all other devices. Once a LE Device is a follower, it remains in the *Follower* state unless either there are no leaders, or there is another LE Device claiming to be leader.

Transport Layer Contract

Figure 8 shows the CDV of the Transport Layer contractual specification. The figure presents the values and operations private to the system, along with the public operations provided by the *send* interface. This underlines the fact that the only externally visible operation is in the provided interface. The transport layer receives data from LE Devices through the provided interface, wraps it into a message containing that data and sender/receiver information, adds the message to a queue, and sends messages from the queue on to their destinations. The Transport Layer maintains a record of whether each LE Device is turned on or off in the *devOn* value.

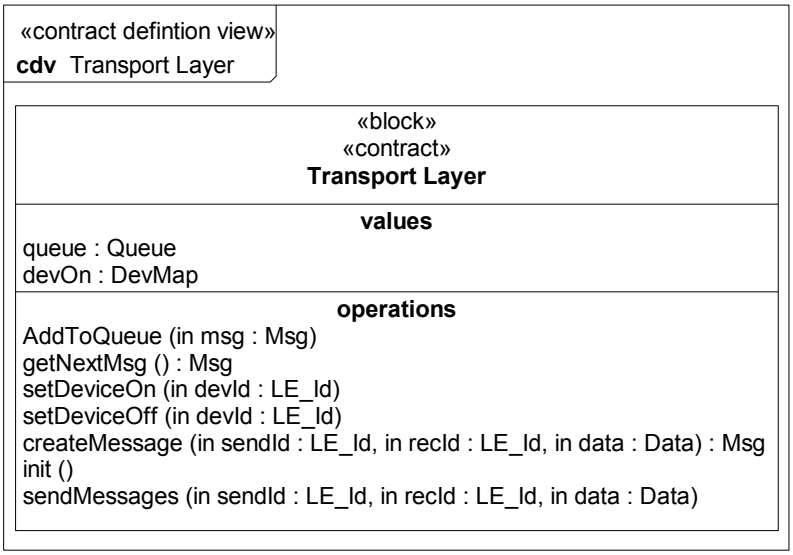

Figure 8. Contract definition view for Transport Layer contract specification

The operations relate to message and queue management and recording device status. The Transport Layer will use the *receiveMessages* operation provided by the LE Devices to send data.

In Figure 9, we present the CPV for the Transport Layer contract. Once initialized, the Transport Layer enters the *Ready* state. From this state, there are four options; either the Transport Layer receives a message from an LE Device and enters the *Reader* state, adding new messages to the queue; or the Transport Layer has messages on its queue and enters the *Writer* state where the Transport Layer attempted to be send a message to the recipient LE

Device; or it receives an status from a LE Device and enters one of the *DevMgnt* states, simply invoking the respective device management operations. Upon the completion of each of these composite states, the Transport Layer returns to the *Ready* state.

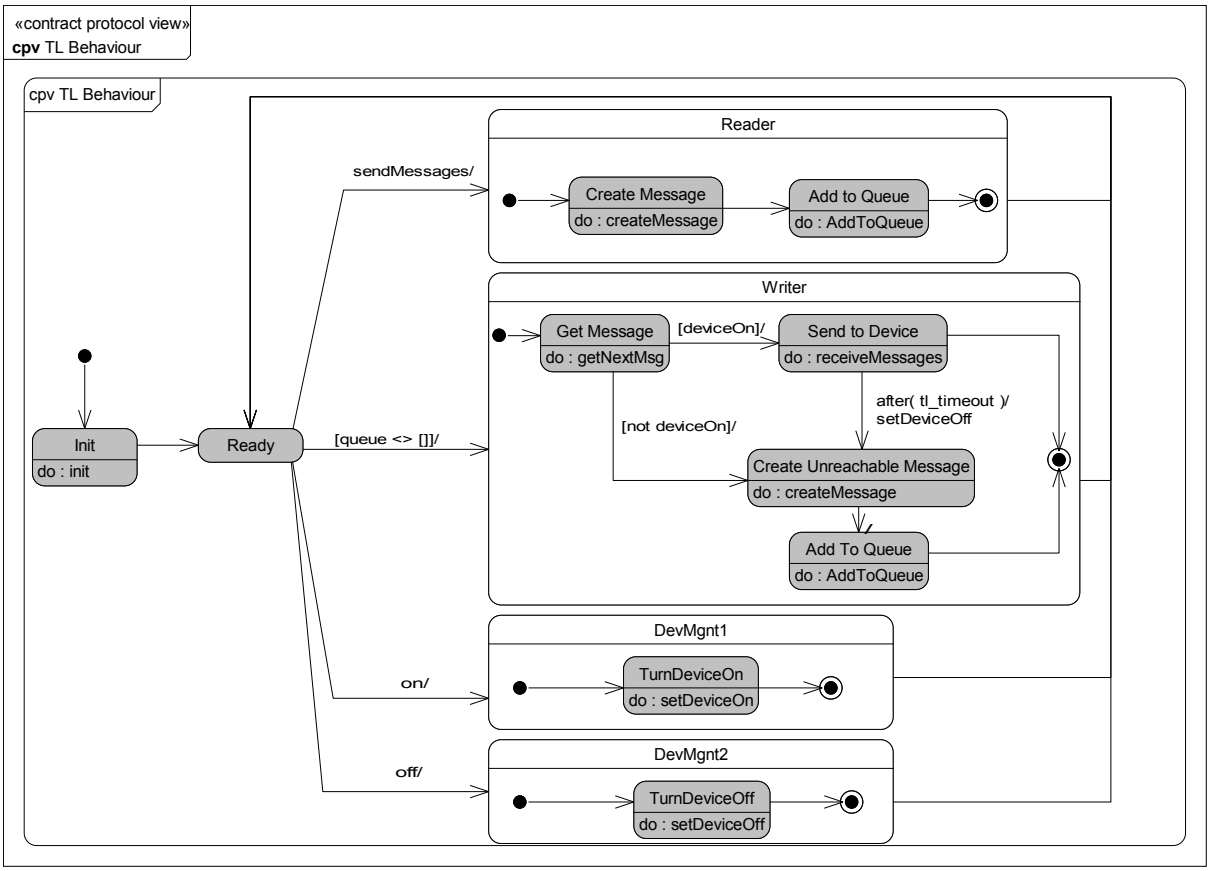

Figure 9. Contract protocol view for Transport Layer contract specification

6. Model-based Analysis: Simulation and Verification

In this section we show how the SysML model may be described in the formal modeling notation CML, following the approach to translation taken in (Bryans, Payne, et al. 2013). We go on to show how the resultant model may be used to aid analysis.

A CML model consists of a number of processes, each encapsulating scoped state variables and a description of process behavior. The process definition comprises: a set of typed values (its state variables), operations which may access and modify state, and a set of interconnected actions describing the possible events which describe the ordering of operation calls and data communications. CML provides combinators to describe a wide variety of activity, including choice, sequencing, parallel activity, and timeouts and interrupts[1].

The CML model of the contractual specifications of the LE Device and Transport Layer are derived from the SysML model outlined above. To give a flavor of the model, in this section we concentrate on the actions of the LE Device. We use the state machine diagram in Figure 7, and map SysML states to CML actions with the same name.

The LE Device begins by initializing and then behaves as the Off action, given below, in which device `id` can initially receive a message to turn on (`on!id`) and then behaves as the

[1] A complete definition of the CML language may be found in (Bryans, Canham &

Undecided action. At any point, it may be interrupted by an off event, (the operator /_\) in which case it will flush the state and turn back off.

```
Off = on!id -> (Undecided /_\ off!id -> flushState();Off)
```

In the Undecided action the LE Device changes its claim (to <undecided>), then behaves as the Listener action, in which it listens to the Transport Layer to learn about the state of the network. It then transitions into either a Leader or a Follower action on the basis of the isleader state variable. This value, set in the Listener action, determines whether the LE Device may have a valid claim to be a leader based upon its knowledge of the other LE Devices in the SoS. We have chosen to transition to the Listener action explicitly when we want the LE Device to update its view of the network, rather than consider the Listener as a separate process running in parallel with the rest of the LE Device, as in the SysML model (Figure 7). We chose this approach for the clarity of the resulting CML, as well as being a more accurate model of the behavior of a (single processor) LE Device.

```
Undecided = changeClaim(<undecided>);Listener;
            ([isleader] & Leader
             []
             [not isleader] & Follower)
```

The Listener action first behaves as ReceiveData, updates the LE Device state variables with the update() operation and then finishes (the Skip action does nothing except terminate successfully). The ReceiveData action first listens to the network for an allotted period of time. Listening is represented by n_rec!id?s?dat, where n_rec is the communication channel with three pieces of data; id is the LE Device's own identifier, s is the sending LE Device's identifier, and dat is the data being received. The use of ! and ? broadly correspond to input and output, however in this context n_rec!id?s?dat states that the device synchronizes *only* on it's own identifier, and any value of s and dat. The s and dat values are then used as input parameters to the write operation.

```
Listener = ReceiveData; update(); Skip
ReceiveData = (n_rec!id?s?dat -> write(s,dat); ReceiveData)
                    [_ n_timeout _> Skip
```

This action will time out after a period of time elapses, corresponding to the n_timeout value (this is globally defined in the CML model). After this time, the LE Device stops listening on the network.

Leader and Follower modes have a similar structure to Undecided, and we do not present the corresponding CML processes here. A Transport Layer process can be derived in a similar way from Figure 9.

We are now in a position to create a contractual view of the SoS by combining the contracts for all the CSs in the SoS. This is given as the process below:

```
process Election =
      AllLEDevices[|chans|]TransportLayer \\ {|rec,send|}
```

where AllLEDevices is the collection of Device contracts, and chans is the set of interface communications. The communications between Devices and the Transport Layer (rec and send) are hidden from view.

We are now in a position to use this model to answer the question: does the model meet requirement R1.1? One of the most intuitive ways that we can improve our confidence in the accuracy of a model is to simulate it by running a series of tests. Testing CML models is

facilitated by the Symphony tool platform[2] and we use that to investigate the behavior of the model. Figure 10 shows the Symphony tool platform running a simulation of the Leader Election model. The model is visible in the top left pane, and the events that have already been performed are visible in the top right. The user selects the next event from the pane below that, and the current status of the analysis run is visible at the bottom.

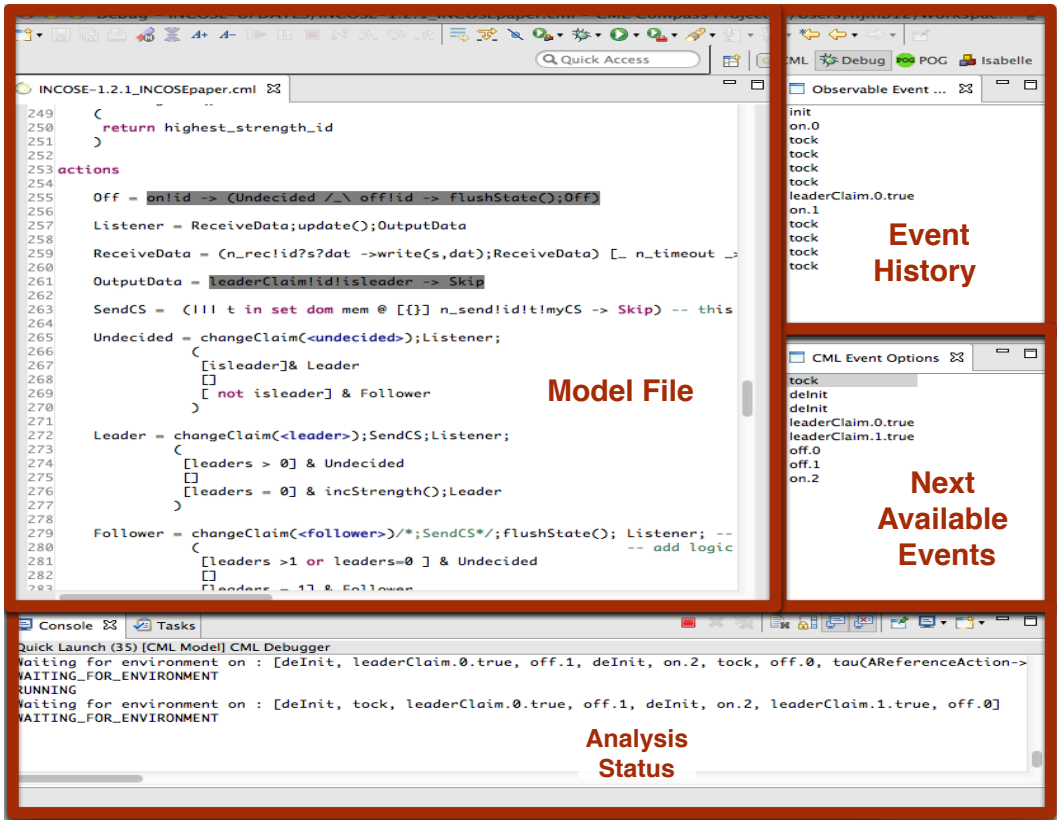

Figure 10 The Symphony tool platform

In the model under analysis we have included the channel `leaderClaim`, through which LE Devices signal a claim to be leader. Exploration of this extended model reveals that it is possible for two devices to claim to be leader simultaneously: the trace below shows both LE Device 0 and LE Device 1 claiming to be leader.

```
[init, on.0, tock, tock, tock, tock, leaderClaim.0.true, on.1,
tock, tock, tock, tock, leaderClaim.0.true, leaderClaim.1.true,
tock, tock, tock, tock, leaderClaim.0.true, leaderClaim.1.false]
```

After initialization LE Device 0 turns on, and waits four time units (the value we have chosen for `n_timeout`) to receive any network traffic that might inform it of an existing leader. None is received, as no other devices are on, and so LE Device 0 begins to act as the leader. LE Device 1 is then turned on, and four further time units pass. At this point, both LE Device 0 and LE Device 1 claim to be leader, which is contradiction to our requirement R1.1.

Following a further interval of four time units we reach a point where LE Device 0 has retained leadership, and LE Device 1 has capitulated.

The reason for this double claim is that LE Device 1 does not initially hear anything from LE Device 0. The messages sent by LE Device 0 are received by the Transport Layer before the

[2] Available at http://symphonytool.org/

Transport Layer receives the signal that LE Device 1 is on. The Transport Layer therefore creates the unreachable message straightaway, in accordance with Figure 9, and no LE Device 1 receives no messages.

The initial segment of the trace highlights a problem: the contractual specification we have given does not meet requirement R1.1, since under some circumstances two devices may both claim leadership. These conflicting leadership claims are possible in our contractual model because there is a period of instability following the switching on a device and before the SoS reaches a consistent state.

In fact, what we have illustrated here is not a problem specific to SoSs. It is a known phenomenon of distributed systems attempting to achieve a consensus. The SoS is an example of a partially synchronous system with a Global Synchronization Time (GST) (Dwork, Lynch and Stockmeyer 1998). This is the period of time after a destabilizing event (such as a device being turned on or off) that is required to pass in order to guarantee that a global consensus has been reached.

A possible solution to the problem that requirement R1.1 is not met is therefore that we make the requirement more precise. We could rephrase it as: If no LE Devices have been turned on or off for a sufficiently long time (the GST) then there will be a unique leader.

The duration of the GST could be found by simulation. We note, however, that the formal approach that underpins CML (Hoare and Jifeng, 1998) allows for other, more rigorous, forms of model analysis. One of these is refinement in the style of CSP (Hoare, 1985). Here, a simple behavioral specification - possibly a single process - describes the full range of permitted external behaviors for the SoS from a global point of view. A model checker can be used to test whether the behaviors of the composite model of the SoS fall within this range.

We discussed earlier integration as a SoS problem. The contractual style of interface definition that we develop here enables us to use refinement as an approach to integration of a new CS into a SoS. If we have a CML model of a new device we wish to add to the network, then we can use refinement to check that the device meets the interface contract, and therefore that it will be able to participate successfully in the election process. Developing a refinement checker is ongoing work in the COMPASS project. The ability to integrate the device into the SoS will also depend on the openness of the device, as mentioned earlier, as changes may need to be made to the internal architecture in order to meet the interface contract. The COMPASS architectural guidelines provide directions on addressing such integration issues, and work is ongoing in this area.

7. Conclusions

We have proposed a model-based approach to assist in the integration of new or modified constituent systems into an SoS. We leverage SysML and formal methods to enrich interface specifications with contracts that describe the expected and permitted behavior of constituents. Translation to the formal CML language enables the verification of emergent properties.

We have demonstrated the feasibility of the approach in an industrial case study. The description of the ordering of interface events is not adequate to describe the requirements that we must place on potential members of our AV SoS in order to ensure that emergent leader election behavior is maintained. We show that interface specifications must be augmented with a contractual description that places further requirements on the behavior of a Device. We further show how analysis of the full AV SoS in a formal modeling language can contribute to our understanding of the emergent behavior, clarifying the requirements to be satisfied by suppliers of new devices to be integrated to the network, or the requirements

to be demonstrably maintained when such devices are modified or `upgraded'.

We plan several areas of future work. The definition of contract specification could be presented in terms of an architectural framework (Holt and Perry ed. 2013), permitting adaptations of the approach to a range of domains. We intend to explore the potential for extending the contract notion to allow CML features, such as pre/postconditions and invariants, to be added at the SysML level, and extending the expressiveness of contract specifications to include non-functional properties. We have demonstrated analysis of the SoS model through simulation. CML gives us the potential to use other forms of analysis including theorem proving (Foster and Payne 2013), as well as conventional model-checking in which the property specification is given as a formula in a temporal logic (Lowe 2008). The approach has the potential to help in resolving the design of constituents that must conform to multiple, possibly conflicting contractual specifications. As we have noted, the contract description we have presented acts as a specification for a constituent system; we can explore the potential of using such contracts as a basis for negotiating the terms of procurement of constituent systems, as well as the run-time monitoring of adherence to contracts.

Acknowledgments

The work presented here is supported by the EU Framework 7 Integrated Project ``Comprehensive Modeling for Advanced Systems of Systems'' (COMPASS, Grant Agreement 287829). For more information see http://www.compass-research.eu.

References

Antonino, P.R.G., M.V.M. Sampaio, A.C.A. Oliveria, K.E. Kristensen, and J.W. Bryans. 2014. "Leadership Election: an Industrial SoS Application of Compositional Deadlock Verification." Submitted to NASA Formal Methods Symposium. Available at: http://www.compass-research.eu/.

Baldwin W. Clifton, Stephanie Hostetler and Wilson N. Felder. 2013. "Mathematical Models of Emergence in Complex Systems-of-systems." In Proc. 23rd INCOSE International Symposium.

Boardman, John and Brian Sauser. 2006. "System of Systems –the meaning of "of"." In Proc. of the IEEE/SMC Intl. Conf. on System of Systems Engineering, Los Angeles, CA. IEEE.

Bryans, Jeremy, Richard Payne, Jon Holt, and Simon Perry. 2013. "Semi-Formal and Formal Interface Specification for System of Systems Architecture." In Proc. IEEE SysCon.

Bryans, Jeremy, Samuel Canham, and Jim Woodcock. 2013. "CML Definition 3 - Denotational Semantics." COMPASS Deliverable D31.4a. Available at: http://www.compass-research.eu/.

Cantot, Pascal and Dominique Luzeaux. 2011. *Simulation and Modeling of Systems of Systems*. Wiley.

Dwork, Cynthia, Nancy Lynch, and Larry Stockmeyer. 1998. "Consensus in the presence of partial synchrony." J. ACM 35(2): 288-323.

Fitzgerald, John, Simon Foster, Claire Ingram, Peter Gorm Larsen, and Jim Woodcock. 2013 "Model-based Engineering for Systems of Systems: the COMPASS Manifesto." Available at: http://www.compass-research.eu/Project/Publications/MBESoS.pdf.

Foster, Simon and Richard Payne. 2013. "Theorem Proving Support – Developers Manual." COMPASS Deliverable D33.2b. Available at: http://www.compass-research.eu/.

Hoare, Tony. 1985. *Communicating Sequential Processes*. Prentice-Hall.

Hoare, Tony, and Hi Jifeng. 1998. *Unifying Theories of Programming*. Prentice Hall.

Holt, Jon and Simon Perry, ed. 2012. “Report on Guidelines for SoS Requirements.” COMPASS Deliverable D21.1. Available at: http://www.compass-research.eu/.
Holt, Jon and Simon Perry, ed. 2013. “Initial Report on Guidelines for Systems Engineering for SoS.” COMPASS Deliverable D21.3. Available at: http://www.compass-research.eu/.
INCOSE 2011. “Systems Engineering Handbook.” Version 3.2.2.
Jamshidi, Mo. 2008. *System of Systems Engineering: Innovations for the Twenty- First Century*. Wiley, 1st edition.
Kemp, Duncan, Dave Camm, Rhienne Evans and Jon Elphick. 2013. “Steampunk System of Systems Engineering: A case study of successful System of Systems engineering in 19th century Britain.“ In 23rd INCOSE International Symposium.
Lowe, Gavin. 2008.“Specification of communicating processes: temporal logic versus refusals-based refinement.” Formal Asp. Comput. 20,(3) 77-294.
Maier, Mark W. 1996. “Architecting Principles for Systems-of-Systems”. In Sixth International Symposium of the International Council on Systems Engineering, INCOSE.
Meyer, Bertrand. 1998. Object-Oriented Software Construction (2nd ed.). Prentice-Hall.
Mordecai, Yaniv and Dov Dori. 2013. “A Model-Based Framework for Architecting System-of-Systems Interoperability, Interconnectivity, Interfacing, Integration, and Interaction”, In 23rd INCOSE International Symposium.
Object Management Group OMG. 2012. “Systems Modeling Language version 1.3.”. http://www.omg.org/spec/SysML/1.3 (accessed 2013).
Payne, Richard and John Fitzgerald. 2010. “Evaluation of Architectural Frameworks Supporting Contract-based Specification.” Technical Report CS-TR-1233, School of Computing Science, Newcastle University.
Perry, Simon, ed. 2013. “Report on Modeling Patterns for SoS Architectures.” COMPASS Deliverable D32.3, 2013. Available at: http://www.compass-research.eu/.
Roscoe, A. W. 2010. Understanding Concurrent Systems. 1st Edition. New York: Springer-Verlag.
Woodcock, J., A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. 2012 “Features of CML: a formal modeling language for Systems of Systems” Proc. 7th Intl. IEEE Conf. on System of Systems Engineering.

Biography

Jeremy Bryans is a Senior Research Associate in the School of Computing Science at Newcastle University. His research concerns the modeling and analysis of collaborating systems, and in the development of trustworthy policies for their interaction. He is currently a Co-Investigator on the EPSRC project Trusting Dynamic Coalitions, on which he works on designing and composing provenance policies for coalition members. Part of his time is spent on the EU project COMPASS, on which is developing semantic foundations for a modeling language for systems of systems.

John Fitzgerald is Professor of Computing Science and Director of the Centre for Software Reliability (CSR) at Newcastle University, UK, where he leads the international COMPASS project on systems of systems engineering. A specialist in rigorous model-based analysis and design of resilient systems, he has contributed to methods and tools applied commercially in areas as diverse as chip design and options trading. Fitzgerald studied verification technology (PhD, Manchester Univ., 1991), before working on design techniques for avionic systems with British Aerospace as a SERC Fellow and Lecturer at Newcastle. He is Chairman of Formal Methods Europe, a member of INCOSE, the ACM, and IEEE, and a Fellow of the

BCS.

Richard Payne is a Research Associate in the School of Computing Science at Newcastle University. His research interests include architectural modeling, 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. He is now working on the COMPASS project, designing advanced modeling and verification techniques.

Klaus Kristensen (MSc cum laude Computer Science with Mathematics, University of Aalborg, Denmark) has over 18 years' experience working as a software engineer and software architect in companies including Bang & Olufsen, Motorola and LEGO, and is an expert in Audio/Video systems engineering. In the COMPASS project, he leads work on the instantiation and evaluation of model-based SoS engineering methods in the AV domain.