A Formal Model-Based Approach to Engineering Systems-of-Systems

John Fitzgerald, Jeremy Bryans, Richard Payne

A Formal Model-based Approach to Engineering Systems-of-Systems

John Fitzgerald, Jeremy Bryans and Richard Payne

Centre for Software Reliability, Newcastle University, Newcastle upon Tyne NE1 7RU, United Kingdom
{John.Fitzgerald, Jeremy.Bryans, Richard.Payne}@ncl.ac.uk

Abstract. Systems-of-systems (SoS) are network-enabled synergistic collaborations between systems that are operationally and managerially independent, distributed, evolve dynamically and exhibit emergence. The design of dependable SoS requires model-based approaches that permit description of contracts between constituent systems at interfaces in a SoS architecture, including functionality and interaction behaviour, and that permit verification of global behaviours. We describe an approach to formal model-based SoS engineering using complementary notations for functional, interaction and architectural aspects. A case study in modelling information flow in an emergency response SoS demonstrates the viability of the proposed approach and highlights a need for common semantic foundations.

Keywords: **S**ystems-of-systems, Information flow, SysML, CSP, VDM, Analysis, Verification.

1 Introduction

Systems-of-systems (SoS) are network-enabled integrations of heterogeneous systems, delivering capabilities and services which cannot be achieved by the constituent systems alone. Examples include enterprise information systems, integrated manufacturing systems, and emergency response collaborations. SoS technology enables the provision of holistic services such as more efficient management and control, more agile response or efficient energy management.

SoS are distinguished from large monolithic systems by several characteristics [1]. The **managerial and operational independence** of the constituent systems means that it may be impossible to exercise centralised control over operation, or to ensure that goals are respected. SoS must cope with **evolution** caused by changes in the purposes and identity of constituent systems. Their geographically **distributed** character leads to a reliance on network/Internet technologies to ensure communication between constituents. **Emergence** is central to their functioning in that the SoS delivers a purpose that is not explicitly present in the constituent systems.

SoS can be viewed as Collaborative Network Organisations (CNOs) in the terms of the ARCON reference modelling framework [2]. SoS are classed as *Virtual*, *Collaborative*, *Acknowledged* or *Directed* [3,1] based on the strength of explicit

acknowledgement and subordination to centralized control. They thus exhibit a range of levels of joint endeavour [2], from simply networking to maintaining a joint identity.

The engineering of SoS is challenging because of the complexity of interactions of constituent systems, and the need for effective communication among diverse stakeholders. A consistent theme of SoS research has been the role of model-based techniques [4,5] as it has been for CNOs in general [6]. A precise model of SoS architecture, constituent systems, infrastructure and environment allows early exploration of design alternatives and the contracts that exist between constituent systems. This makes it possible to validate global properties such as resilience to faults or attacks, liveness, safety and security, that affect the reliance that can be placed on a SoS. If models are defined using languages with formal semantics, it becomes possible to perform machine-assisted analysis of global properties, providing early identification and elimination of errors. Formal methods thus offer a way to manage risk.

Although formal methods can be challenging to apply [6], advances in their automation have increased their viability, notably in software development [7]. However, these techniques have been applied only experimentally in SoS Engineering (e.g. [8]). The goal of our work, supported by the COMPASS project[1], is to develop modelling languages that are expressive enough to model the architecture and behaviour of candidate SoS structures, and sufficiently rigorously defined to permit trustworthy machine-assisted analysis of global properties.

This paper proposes an approach to formal model-based SoS engineering using complementary formalisms to describe functional and behavioural aspects of constituent systems, and verify global properties of the SoS. In Section 2 we describe this approach, and in Section 3 we describe its pilot application to a study of an emergency response SoS. Section 4 describes future research towards our goal.

2 A Formal Model-based Approach to SoS

Model-based engineering approaches are challenged by several characteristics of SoS. Independence means that there can only be limited knowledge about, and control over, constituent systems. This suggests that models should support the recording of contracts that bound constituents' behaviour without defining it completely and deterministically. Geographical distribution implies a need to model concurrency in terms of message passing between constituents. The need to manage evolution and structural change requires the ability to model architectural structures and particularly interfaces between constituents. The central role of emergence in SoS makes it imperative to support the verification of SoS-level properties. In addition to these requirements, experience in industry deployment of formal methods teaches us that it is necessary to provide strong links to an accepted architectural notation, and to have robust tools that support both simulation and static analysis [7].

1 Comprehensive Modelling for Advanced Systems-of-Systems, EC FP7 Project 287829, http://www.compass-research.eu

No single formalism meets all of the demanding requirements. As the ECOLEAD project concluded, there is no "universal language" for modelling problems for CNOs in general [2]. We therefore aim to define combinations of interoperable modelling techniques and extend them for SoS development to allow trade-off analysis and verification of SoS-level properties.

Many formal languages have been developed for expressing and analysing particular system characteristics [9, 10, 11]. However, for a SoS, we need to cover functionality, concurrency, communication, inheritance, time, sharing, and mobility. Some languages cover a few of these features, and there are integrations of formalisms that cater for data, concurrency, and time [12, 13, 14, 15, 16]. The verification of global properties in design also suggests a need for a theory that covers refinement.

Given the requirements above, our baseline technologies are SysML for architectural description, CSP [10] for describing concurrency and communication and VDM [17] for data and functionality. We extend SysML [18] with the ability to express rigorous interface contracts [19], giving SoS engineers the ability to experiment with consequences of different architectural design decisions. SoS engineers may also define expected interfaces of the constituent systems, which may in turn be provided to developers/operators of constituent systems, or used as a basis of their assessment, providing greater confidence that constituent systems adhere to the expected properties on interfaces. We aim to allow engineers to operate either at the SysML graphical level or at the textual level, or at a combination of these, since there will be support for moving between these views. In Section 3, we explore the feasibility of this combination of formalisms for model-based SoS engineering via a case study.

3 A Case Study in Emergency Service Co-ordination

In this section we present a study in emergency coordination in order to evaluate the modelling approach proposed in Section 2. Our study is based on the London Emergency Services Major Incident Procedure Manual [20] which documents the process for identifying a major incident, initiating appropriate services (fire, police, ambulance etc.), and the roles and responsibilities of service members involved. The coalition of services forms a SoS: the constituent systems are normally independent services; there is mobility and geographic distribution, and a need to evolve rapidly as goals or volatile conditions change. Ultimately, the SoS must provide an emergent service to stakeholders ranging from people involved in the incident to the media and authorities. This coalition was previously explored using the Event-B formalism [21]. However, that model does not address interaction between participants, and does not provide an accessible representation of the SoS architecture.

We first introduce the application and then the formalisms SysML, CSP and VDM. We present models of the complementary architectural, behavioural and functional aspects of the SoS (Sections 3.1-3.3). For brevity, we omit some details of the formal models, but give a flavour of them. In Section 4, we draw conclusions about the research required to develop a more integrated modelling and analytic framework.

The response to all major incidents follows a broadly similar structure. Members of each service attending the scene form *Bronze* (operational) command. For more severe incidents, a *Silver* (tactical) command is formed containing representatives of all the services involved. For long-running incidents, a *Gold* (strategic) command may be formed at a geographically distant point. Each service has members working at each level, e.g. *Silver Police*. Each level and service has different responsibilities.

There is a strict information flow policy in the Bronze/Silver/Gold structure, illustrated in Fig. 1. The members of the coalition in a given service and level are permitted to communicate with other members and the same level, for example Bronze Police may communicate with other Bronze officers. The services also have their own communication structure which may be used between adjacent levels. For example, Bronze Police may communicate with Silver Police, but not directly with Gold Police. Communication with the media is (in this example) the sole responsibility of Gold Police.

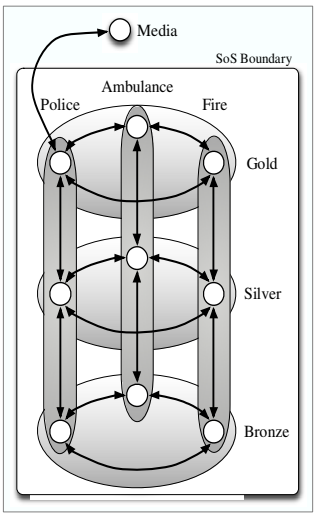

Fig. 1: Permitted information flows between coalition members of different levels

We focus on the rules [20,22] for releasing casualty information to the media, and the requirements that these rules place on the interfaces between the emergency services (constituent systems). Confusion can arise if the media aggregate casualty figures from multiple sources ("double-counting"), leading to overestimation of the incident's severity. To avoid this, all casualty details must be given to Gold command, which is then responsible for producing a more reliable estimate and passing this to the media. The previous Event-B study [21] considered the passage of information through the emergency response system, and sought to ensure that information was not released to the media without first being cleared by Gold.

3.1 Architectural Model in SysML

SysML [18] is a profile for UML 2.0, developed for system engineering, but also supporting the modelling of SoS architectural definitions. It has wide industrial support and a sound tool base. SysML provides several diagram types, with "precise natural language" semantics, to support the description of SoS architectural structure, behaviour and requirements.

A detailed SysML architectural definition of the case study is given in [22]. For brevity, we omit the general SoS structure. However, Fig. 2, a SysML Internal Block Diagram, details the points of interaction between the SoS constituent systems relevant to casualty information clearance. Contracts between constituent systems are given as provided and required interfaces, containing collections of operation signatures. For example, in Fig. 2, the `order_to_collect_info` interface is provided by Bronze officers and required by Gold command. The interface contains a single operation (given in the full interface definition [22], with the signature `collectCasualtyDetails(loc:Location)`, where location is an abstract data type) to order Bronze officers to collect casualty information. The inclusion of pre/postconditions on operations is optional in SysML, and rarely used in practice. However, in order to accommodate interface specifications rich enough for formal analysis of SoS, extensions to interfaces have been proposed, including more rigorous operation definitions, state machine diagrams defining communication protocols, and the means to record the rationale for contract agreement between interfaces [19].

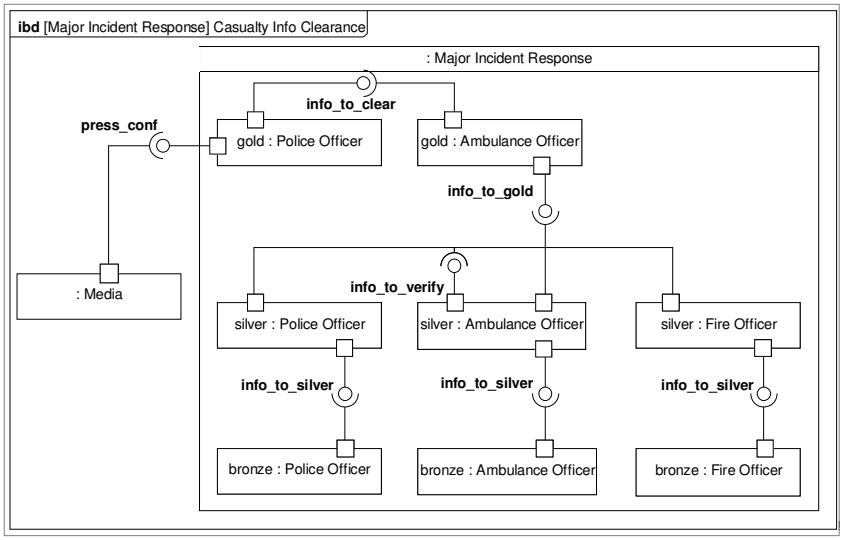

Fig. 2: SysML Internal Block Diagram of Major Incident Response connections

Given a SysML architectural model, complementary CSP and VDM models may be defined covering the interaction behaviour and data and functionality aspects respectively. The SysML model provides a basis for ensuring consistency between the defined SoS internal structure and the interface definitions.

3.2 Modelling Interaction Behaviour using CSP

The CSP [10] formalism allows the interaction behaviour of the SoS to be modelled as a set of processes. A process is made up of sequences of actions (or *events*). Process combinators make explicit the events shared between processes. An abstract model serves to specify the permitted SoS behaviours; more concrete models include communication and structural detail, and may be checked for conformance with the abstract model.

In the abstract specification, the passage of information through the SoS is modelled as a process `INFO(i)`. The variable `i` is parameterised over the set of all information `Inf`. A process `INFO(i)` is made up of three events: each information item `i` can first be learned, then cleared, and finally released to the media. No further activity is then possible for that process. The specification `ACOAL` is the combination (`|||`) of these processes for all possible values of parameter `i`. The interleaving combinator (`|||`) indicates that the individual processes `INFO(i)` do not interact with each other.

```
INFO(i) = learn.i -> clear.i -> release.i -> STOP
ACOAL = ||| i:Inf @ INFO(i)
```

The more concrete model below identifies the coalition levels and the communication events between them. The Bronze process `BR(i)` begins by learning the information item `i` and then describes the passing of `i` to Silver. The synchronisation event `bscomm.i` describes the passing of information item `i` along the channel `bscomm`. The Silver process `SI(i)` begins with the synchronisation event `bscomm.i`, through which it learns about information item `i`. Silver then passes the item to Gold, which clears and releases the item.

```
BR(i) = learn.i -> bscomm.i -> STOP
BRONZE = ||| i:Inf @ BR(i)
SI(i) =  bscomm.i -> sgcomm.i ->  STOP
SILVER = ||| i:Inf @ SI(i)
GD(i) =  sgcomm.i -> clear.i -> release.i -> STOP
GOLD = ||| i:Inf @ GD(i)
```

The three processes are combined in the concrete specification as `CCOAL`, communicating on the events `bscomm` and `sgcomm`.

```
CCOAL = BRONZE [|{|bscomm|}|] (SILVER [|{|sgcomm|}|] GOLD)
        \ {| bscomm, sgcomm |}
```

A model checker such as FDR can be used to check that this concrete SoS description admits only behaviours permitted by the abstract specification. The next step is to decompose GOLD into processes representing the emergency services. For example:

```
P(i) =  sgcomm.i -> pclear.i -> release.i -> STOP
Police = ||| i:Inf @ P(i)
```

The distributed GOLD command is the combination of these processes:

```
GOLDdist = ((Police [|{| sgcomm, release |}|] Fire)
           [|{| sgcomm, release |}|] Amb)
```

The distributed SoS combines the distributed GOLD with the previous processes.

```
COALdist = (BRONZE [| {|bscomm|} |] SILVER)
           [| {|sgcomm|} |] GOLDdist
           \ {|bscomm, sgcomm, pclear, fclear, aclear|}
```

The requirement that the distributed coalition model (`COALdist`) respects (denoted by "`[T=`") the behavioural constraints specified in the abstract model (`ACOAL`) can be asserted formally as follows and checked with tool support:

```
assert ACOAL \ {|clear|} [T= COALdist
```

3.3 Modelling Functionality using VDM

The VDM formal method [17] supports the description of functionality in terms of executable code or in terms of abstract contracts. Tool support is particularly strong for simulation, and there is an established coupling to UML.

A model of the emergency response SoS is given in this section. The model contains two model-specific data types: `Info` is an abstract token type and `CType` is an enumerated type representing the coalition levels (Bronze, Silver, and Gold). The model focuses on recording its state in terms of the information in each state (`known`, `cleared` or `released`), and the level at which that information is known (`coal_known`). A data type *invariant* records consistency restrictions on the allowable state. In this model, the invariant ensures that released information must have been cleared (`released subset cleared`), and all known information is known by allowed coalition levels (`dom coal_known = coalition`).

```
types
Info = token;
CType = <Bronze> | <Silver> | <Gold>

state Coal of
      known: set of Info
      cleared: set of Info
      released: set of Info
      coalition: set of CType
      coal_known: map CType to set of Info
inv mk_Coal(-,cleared, released, coalition, coal_known) ==
       released subset cleared and
       dom coal_known = coalition
```

State-changing functionality is defined in terms of operations that are specified contractually by means of preconditions and postconditions. Consider, for example, Gold command's `clear` and `release` events, the interaction behaviours of which are specified in CSP in process `GD(i)` in Section 3.2. In VDM, the functionality is specified as operations parameterised over information `i:Info`. The `ClearGold` operation assumes in the precondition, `pre i in set coal_known(<Gold>)`, that `i` is known to Gold command. If this assumption is satisfied, the operation guarantees in the postcondition, `post cleared = cleared~ union {i}`, to

add it to the cleared set (where `cleared~` refers to the initial value of the `cleared` state variable). The `ReleaseGold` operation is similar.

```
ClearGold(i:Info)
pre i in set coal_known(<Gold>)
post cleared = cleared~ union {i};

ReleaseGold(i:Info)
pre i in set coal_known(<Gold>) and
    i in set cleared
post released = released~ union {i};
```

Both operations give rise to proof obligations to ensure preservation of the state invariant, including that of ensuring that released information must have previously been cleared, and that both operations preserve this. Such obligations can be generated automatically, and may be discharged by inspection, testing or formal proof.

Comments on the Case Study. Compared to Bryans et al.'s model in Event-B [21], this multi-paradigm approach more clearly shows the interfaces between constituent systems, and hence the points at which structural change is possible, as well as the interaction behaviour, which is here explicit in the CSP rather than "hidden" in event guards. Most importantly, it permits the verification of SoS-level properties that cut across multiple aspects. For example, extending the model to encompass communications errors entails alterations to interaction behaviour (in CSP) and functionality (recording "lost" messages) in the VDM model. Adding redundancy to manage such error would require a modification to the architectural model as well.

Although Bryans et al.'s previous model is less transparent with respect to the SoS architecture and interaction behaviour, it does benefit from the specialist automated verification tools that can be developed for a single formalism. Currently our multiple formalisms do not benefit from a completely consistent semantic base, so that we are not yet able to automate analysis to the same extent.

4. Conclusions and Future Work

We have proposed a multi-paradigm modelling approach to address the particular challenges of SoS engineering, using baseline formalisms that cover architectural modelling, communication and concurrency, data and functionality. Our case study suggests that it is possible to produce consistent models in such formalisms that describe features of a SoS sufficient to verify global properties of interest.

Although multiple modelling techniques are required to cover the full range of aspects of a SoS, we note that researchers and practitioners in CNO modelling tend to stick with one approach even though it might not be the most appropriate for all or a part of the modelling effort [6]. We aim to develop a unified framework that integrates architectural, behavioural and functional models. Semantic interoperability between models is needed for verification of properties that cross aspects. A promising starting point is Hoare & He's Unifying Theories of Programming [23]. This will be developed in a series of definitions starting with basic modelling features and extending these with time and object-orientation.

While formal model-based methods are valuable in describing and verifying the properties of SoS configurations, the capacity exists to restructure or reconfigure during operation in response to faults or attacks. Indeed, a SoS architecture has been proposed to manage such reconfiguration [24]. The semantics and pragmatics of policy languages for dynamic reconfiguration remain open, including the definition and acquisition of metadata, and the expression and verification of policies [25].

Feedback from practice is required in any attempt to develop any formal modelling framework. In the COMPASS project, our emerging methods will be evaluated through several industry case studies. For example, in a home audio-video ecosystem, networked systems such as TV, home cinema, DVD and MP3 players deliver digital content from internal or external sources to multiple users. Providers and integrators of constituent systems require the ability to verify overall performance and that the SoS will respect digital rights management (DRM) contracts on the content. A second example is dynamic coordination of healthcare services in response to an accident (call management, dispatching, triage, hospital management systems, etc.). As with the audio-video ecosystem, global properties such as confidentiality need to be analysed. In both cases, the ability to perform such verification is complicated by the need to cope with failures in infrastructure or constituents.

In spite of the emerging potential of formal techniques, there naturally remains a gap between the formal "supply-side" models of SoS compositions and the users' "demand-side" experience [26]. As with collaborative networked organisations more generally, the development of dependable SoS requires a wide range of disciplines and skills, both socio-technical and formal.

References

1. Maier, M.W.: Architecting Principles for Systems-of-Systems. Systems Engineering 1(4):267--284 (1998)
2. Camarinha-Matos, L.M., Afsarmanesh, H. (Eds.): Collaborative Networks: Reference Modeling, Springer (2008)
3. Dahmann, J.S., Rebovich G., Lane, J.A.: Systems Engineering for Capabilities, CrossTalk Journal 21(11):4—9 (2008)
4. Maier, M.W.: Research Challenges for Systems-of-Systems, Intl. Conf. on Systems, Man and Cybernetics, IEEE (2005)
5. Valerdi, R., Axelbrand, E., Baehren, T., Boehm B., et al.: A Research Agenda for System-of-Systems Architecting. Intl. Jnl. System of Systems Engineering 1(1--2): 171-188, Inderscience (2008)
6. Camarinha-Matos, L.M. Afsarmanesh, H.: A comprehensive modelling framework for collaborative networked organizations, J. Intell. Manuf. 18:529--542 (2007)
7. Woodcock, J.C.P., Larsen, P.G., Bicarregui, J.C., Fitzgerald, J.S.: Formal Methods: Practice and Experience, ACM Computing Surveys 41(4):1--36 (2009)
8. Caffall, D.S., Michael, J.B.: Formal methods in a system-of-systems development, IEEE Intl. Conf. Systems, Man and Cybernetics, pp. 1856--1863 (2005)
9. Woodcock, J.C.P, Davies, J.: Using Z Specification, Refinement, and Proof, Prentice-Hall (1996).
10. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall Intl., 1st edn 1985, new edn. Jim Davies (Ed.) (2004).

11. Pnueli, A.: The temporal logic of programs. In 18th IEEE Symp. Foundations of Computer Science, pp. 46-57 (1977).
12. Treharne, H., Schneider, S.: Using a process algebra to control B OPERATIONS. In 1st International Conference on Integrated Formal Methods IFM'99, LNCS 437-457. Springer-Verlag, 1999.
13. Fischer, C.: Combination and Implementation of Processes and Data: from CSP-OZ to Java. PhD thesis, Fachbereich Informatik Universität Oldenburg (2000).
14. Dong, J.S., Hao, P., Qin, S., Sun, J., Wang, Y.: Timed Patterns: TCOZ to Timed Automata, International Conference on Formal Engineering Methods, LNCS 3308:483-498, Springer (2004).
15. Leavens, G.T., Leino, K.R.M., Poll, E., Ruby, C., Jacobs, B.: JML: notations and tools supporting detailed design in Java, OOPSLA 2000, pp. 105-106 (2000).
16. Beckert, B., Hähnle, R, Schmitt, P.H. (Eds.): *Verification of Object-Oriented Software: The KeY Approach*. LNAI 4334. Springer (2007).
17. Fitzgerald, J.S., Larsen, P.G., Mukherjee, P. Plat, N Verhoef, M.: Validated Designs for Object-oriented Systems, Springer (2005)
18. Object Management Group: OMG Systems Modeling Language (OMG SysML) v1.2, OMG Document Reference: formal/2010-06-02 (2010)
19. Payne, R.J., Fitzgerald, J.S.: Interface Contracts for Architectural Specification and Assessment: a SysML Extension, Proc. Workshop on Dependable Systems of Systems (WDSoS '11), University of York, UK (2011)
20. London Emergency Services Liaison Panel: Major Incident Procedure Manual, 7th edn., TSO (The Stationery Office) (2007)
21. Bryans, J. W, Fitzgerald, J.S. McCutcheon, T.: Refinement-Based Techniques in the Analysis of Information Flow Policies for Dynamic Virtual Organisations. In: Camarinha-Matos, L.M., Afsarmanesh, H. (Eds.) Adaptation and Value Creating Collaborative Networks IFIP AICT 362: 314-321, Springer (2011)
22. Payne, R.J. and Bryans, J.W.: Modelling the Major Incident Procedure Manual: A Systems of Systems Case Study, Tech. Rep. CS-TR-1320, School of Computing Science, Newcastle University, UK, (2012).
23. Hoare, C.A.R., He Jifeng: Unifying Theories of Programming, Prentice-Hall (1998)
24. Calinescu, R., Kwiatkowska, M.: Software Engineering Techniques for the Development of Systems of Systems. In: Choppy, C., Sokolsky, O. (Eds.) Foundations of Computer Software, LNCS 6028:59-82, Springer (2010)
25. Payne, R.J.: Verifiable Resilience in Architectural Reconfiguration, PhD Thesis, School of Computing Science, Newcastle University, UK, (2012)
26. Cohen, B., Boxer, P.: Why Critical Systems Need Help to Evolve, IEEE Computer 43(3):56-63 (2010).