Bryans J, Fitzgerald J, Payne R, Miyazawa A, Kristensen K.

SysML Contracts for Systems of Systems.

***In: IEEE 9th International Systems of Systems Engineering Conference (SoSE 2014).* 2014, Stamford Grand, Glenelg, Australia: IEEE.**

SysML Contracts for Systems of Systems

Jeremy Bryans, John Fitzgerald, Richard Payne
School of Computing Science
Newcastle University, UK
{jeremy.bryans, john.fitzgerald, richard.payne}@ncl.ac.uk

Alvaro Miyazawa
University of York, UK
alvaro.miyazawa@york.ac.uk

Klaus Kristensen
Bang & Olufsen, Denmark
krt@bang-olufsen.dk

***Abstract*—This paper proposes and demonstrates an architectural pattern for the contractual specification of interfaces between constituent systems within a System of Systems (SoS). We take a structured approach to the development of the pattern, which we call the Contract Pattern. It is developed and demonstrated in SysML using a case study from the Audio/Video domain. We also identify some of the obstacles in the way of checking the conformance of a constituent system to a contract, and discuss how these may be overcome.**

***Keywords* – systems of systems, modelling, SysML, architectural frameworks, contracts, interface specification.**

I. Introduction

This paper proposes and demonstrates an architectural pattern for the contractual specification of interfaces between constituent systems within a System of Systems (SoS).

There is considerable interest in the potential of model-based approaches as a means of addressing the distinctive challenges of SoS engineering. Such approaches encourage the disciplined development and use of abstract descriptions of SoSs, their requirements, architectures, and behaviours. These abstract descriptions (or *models*) permit analysis of (in)compatibilities between constituent systems, and of SoS emergent behaviours at relatively early stages, before deployment of solutions [1]. The use of model-based methods permits architectural frameworks to be developed that characterise the forms of model and views that have been found to be valuable in application domains, as well as specific modelling patterns that can be reused extensively and embody good practice. Where modelling notations have a well-defined semantics, many forms of analysis can be automated, and the delivery of such formal modelling and analysis techniques is the subject of considerable current research, including our own work on model-based engineering, which takes place as part of the EU COMPASS project[1].

However rich and formal the models, fundamental characteristics of SoSs still present significant challenges. In particular, the capacity of constituents to evolve independently means that integrators cannot justifiably rely on the behaviour of the constituent systems persisting through the life of the SoS. There is consequently a need in model-based SoS engineering to specify constituent system behaviour in a way that bounds the range of behaviours that can be relied upon without over-constraining them, while still allowing the analyses needed to promote desirable and limit undesirable emergent behaviours.

[1] www.compass-research.eu

To address the imprecision and uncertainty inherent in the description of constituent systems, we promote the *contractual description* of the architectural interfaces offered by the constituent systems. The constituent systems are free to choose the way in which they meet these contracts, as well as being free to choose the other contracts to which they adhere. The contribution of this paper is to present a definition of a contract as a SysML pattern, building on the work on contractual modelling reported in [2], [3]. Our pattern is expressed in terms of the views mandated by the COMPASS Architectural Framework Framework (CAFF) [4], which gives a disciplined way of defining architectural frameworks. Other architectural frameworks have also been presented in this way: for example a framework for modelling faults presented in [5]. We present the contract pattern using SysML [6], but the pattern itself is language-agnostic, and would permit representation in any sufficiently expressive modelling language.

The remainder of the paper is structured as follows. In Section II we outline relevant work on contracts and architectural modelling for SoSs. Section III describes the Contract Pattern within SysML using the CAFF, and in Section IV we demonstrate the pattern's use in a case study from the domain of audio/video networking. The potential exploitation of contracts described using the pattern is discussed in Section V. We draw conclusions and propose future work in Section VI.

II. Related Work

We propose the use of contracts to model the necessary internal behaviours in SoSs, alongside architectural interfaces within the widely used notation SysML [6], [7]. Several notations have been developed for defining system architectures. In [8] we survey this area more comprehensively, with particular emphasis on the support for the rigorous definition of interface specification. We use SysML in this paper primarily due to its increased use in industry.

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. Our approach builds on the Design by Contract (DbC) software engineering technique [9], used to constrain software operations. Previous work has considered nonfunctional properties and DbC in architectural interfaces [10], how interfaces in SysML can be translated into the formal notation COMPASS Modelling Language (CML) [8] and in [11] we proposed a method of extending SysML interface descriptions with a contractual pattern. The current work extends the pattern identified in [11] and takes a structured and rigorous approach to its formalisation.

Modelling patterns have been used extensively in the object-oriented software engineering community for many years. A simple outline to describing software patterns was used in [12], which adopted pattern concepts from [13]. In [14] we describe several modelling patterns for SoS. We identify a collection of existing system engineering *architectural patterns*[2] relevant to SoS including the Service-Oriented Architecture, Centralised and Publish-Subscribe patterns – described using a similar approach to [12]. We also define several *enabling patterns*[3] such as the Interface and Traceability patterns. The approach to defining enabling patterns was modified, prescribing the identification of the relevant concepts and related viewpoints on the SoS architecture. This approach was further expanded in [4] as a means to consistently define both enabling patterns and architectural frameworks – the CAFF, which we use in this paper to define the Contract Pattern. The Contract Pattern may be considered as an enabling pattern; it may be applied in architectural frameworks and across different architectural patterns.

III. Contract Pattern

As mentioned in Section I, there is a need to model the constituent systems composing a SoS in order to allow the analysis of emergent behaviours at the SoS boundary. The Contract Pattern aims to allow a modeller to define this behaviour in such a way that it does not over-constrain the implementation of the constituent systems, and also enables the required analyses. It provides a collection of viewpoints which aid the SoS integrator in modelling and defining the contracts of a SoS in a consistent manner. The pattern is described using SysML and implemented as a SysML profile. It is important to note, however, that the pattern is notation agnostic – it does not require that the views defined in the profile should be given using SysML. The Contract Pattern extends the informal description given previously [3], and complies with views advocated by the CAFF [4].

The CAFF approach requires that viewpoints are identified for a pattern or architectural framework, relationships between those viewpoints are described, and the syntax (or permitted modelling elements) of the viewpoints are defined. The CAFF approach also advocates recording the context of the pattern and its viewpoints, typically through use cases, and a collection of rules which constrain the viewpoints.

In Figure 1, we identify the needs of SoS engineers who we envisage to be the users of the Contract Pattern. The figure, a CAFF Context View for the Contract Pattern, states that the main need to be addressed is to enable analysis of SoS emergent behaviour. One way to do this is to model the SoS in terms of contracts. The subsequent needs, including *identify contracts in the SoS* and *define contract functionality*, are met by the different viewpoints which constitute the Contract Pattern.

The ontology is the collection of terms in the problem domain and the relationships of these terms to each other. The

[2]Architectural patterns describe specific, constrained, system architectures; both in terms of structure and behaviour.

[3]Enabling patterns are specific constructs of modelling elements whose combination and subsequent use enables a number of systems engineering applications.

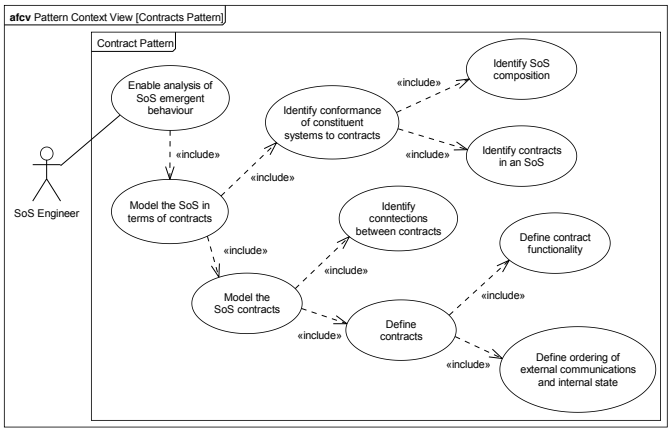

Fig. 1. Architectural Framework Context View identifying the needs met by the Contract Pattern

Ontology Description View provides a means of recording the ontology. In the ODV for the Contract Pattern in Figure 2 we state that an *SoS* is composed of *Constituent Systems* – as is commonly accepted in literature. We introduce the term *Contractual SoS* to refer to the collection of *Contracts* to which the SoS must conform. Constituent systems, in turn, conform to contracts. Contracts and constituents own ports which in turn expose interfaces. We use the *Interface Pattern* presented in [14] to model the connections between contracts and constituents.

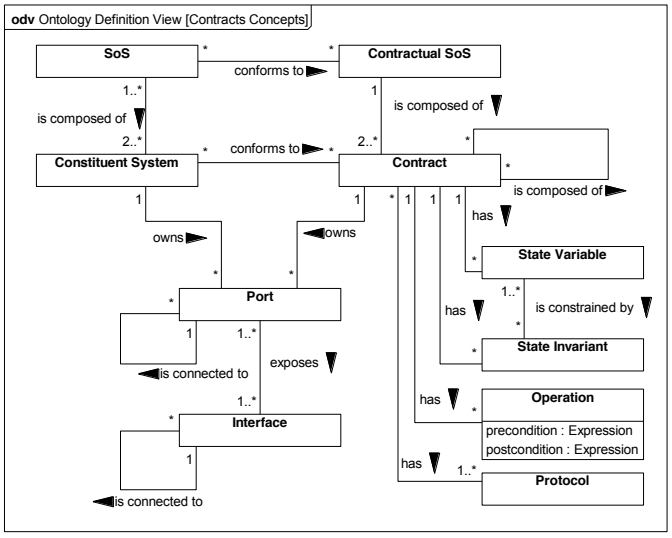

Fig. 2. Ontology Description View relating the key SoS contracts concepts

The Contract Pattern requires five viewpoints: the *Contractual SoS Definition Viewpoint*, the *Contract Conformance Viewpoint*, the *Contract Definition Viewpoint*, the *Contract Connections Viewpoint* and the *Contract Protocol Viewpoint*. Their individual purposes are identified in Table I, and meet the needs of the Contract Pattern, as identified in Figure 1.

As is typical with an SysML model, we consider a model defined using the Contract Pattern to consist of a collection of related views[4]. As such, model elements identified in one view

[4]We use the term view to correspond to an instance of a viewpoint.

TABLE I. INFORMAL DESCRIPTION OF THE CONTRACT PATTERN VIEWPOINTS

Name	Purpose of View
Contractual SoS Definition Viewpoint	Identifies the contracts which comprise the Contractual SoS.
Contract Conformance Viewpoint	Identifies the constituent systems which make up the SoS and denotes the contracts to which those constituent systems conform. Includes all the contracts identified in the *Contractual SoS Definition Viewpoint*.
Contract Connections Viewpoint	Shows connections and interfaces between contracts of the Contractual SoS. Includes all the contracts identified in the *Contractual SoS Definition Viewpoint*.
Contract Definition Viewpoint	Defines the operations, state variables and state invariants for a single contract identified in the *Contractual SoS Definition Viewpoint*.
Contract Protocol Viewpoint	Defines the behaviour of a contract identified in the *Contractual SoS Definition Viewpoint* in terms of the ordering of messages between other members of the SoS and calls to the contract operations.

may be referenced or further defined in other views. In Figure 3, we define the relationships between the viewpoints of the Contract Pattern using a Viewpoint Relationship View (VRV).

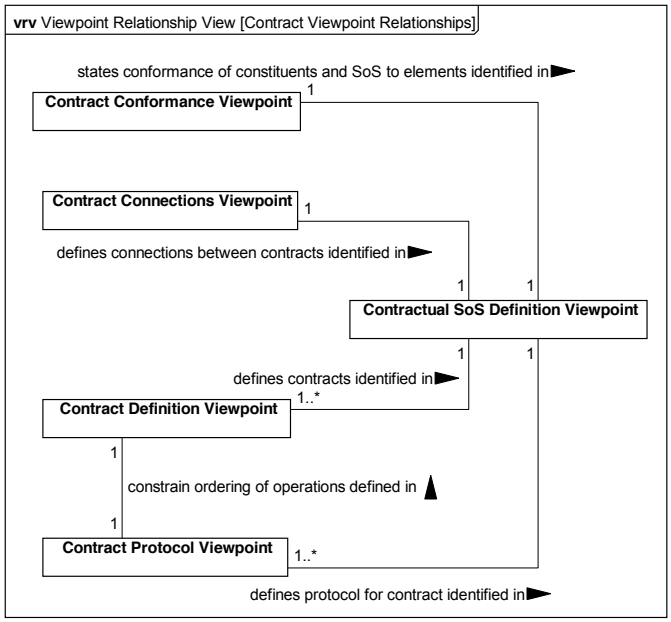

Fig. 3. Viewpoint Relationship View relating the contract viewpoints

The VRV in Figure 3, for example, states that a contract identified in a *Contractual SoS Definition Viewpoint* may be defined in a *Contract Definition Viewpoint*, connected to other contracts in a *Contract Connections Viewpoint* and that the conformance relations between contracts and constituent systems are identified in a *Contract Conformance Viewpoint*.

The different model elements of a given viewpoint and their relationships are defined in Viewpoint Definition Views (VDVs). A VDV for the *Contract Conformance Viewpoint* is given in Figure 4 VDVs are equivalent to defining the syntax for each of the viewpoints – they determine which syntactic elements may be included on a viewpoint. In the *Contract Conformance Viewpoint*, for example, the model elements which may be included comprise a single *SoS* element, which is composed of two or more *Constituent System* elements, one *Contractual SoS*, which is composed of several *Contract* elements, and finally there are several *conformsTo* elements which link *SoS* and *Contractual SoS* elements, and *Constituent System* and *Contract* elements.

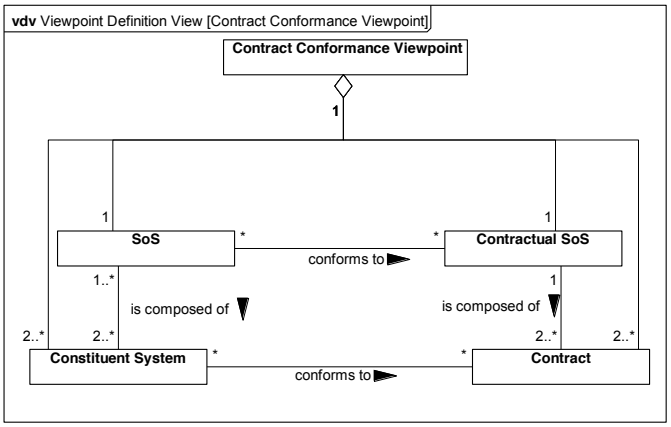

Fig. 4. Viewpoint Definition View for the Contract Conformance Viewpoint

IV. ILLUSTRATIVE CASE STUDY

We illustrate the Contract Pattern with a case study based on a Bang & Olufsen (B&O) home Audio Visual (AV) network linking multiple AV devices. The network exhibits the characteristic properties of a SoS; for example, constituent systems are heterogeneous and may evolve (through software or firmware upgrades). The SoS may display emergent behaviour, and, although all constituent systems operate at the behest of the user, they may in fact be legacy or non-B&O systems, potentially limiting their controllability within the SoS.

Following the Contract Pattern we first model a Contractual SoS – that is a SoS which may be modelled in terms of the contracts being offered, rather than models of the constituent systems themselves. In the *Contractual SoS Definition View* in Figure 5, we consider an *AV Contractual SoS*, comprising multiple *AV Device* contracts and one *Transport Layer* contract. The *AV Device* contract comprises a *LE Device*, a *Browsing Device* and a *Streaming Device* contract.

The *AV Device* contract is composed of three contracts. The *LE Device* contract states the functionality of an AV device necessary to guarantee a single leader, and this will be the main contract we concentrate on in this paper. The other contracts that form the *AV Device* contract ensure correctness of the media browsing and streaming functionalities of AV constituent systems (the *Browsing Device* and *Streaming Device* contracts respectively). The correct communication of data between contracts is ensured by the *Transport Layer* contract.

The *Contracts Connection View* in Figure 6 depicts the connections between the contracts comprising the SoS, and their provided and required interfaces. The figure shows that AV Device contracts are all connected to the Transport Layer

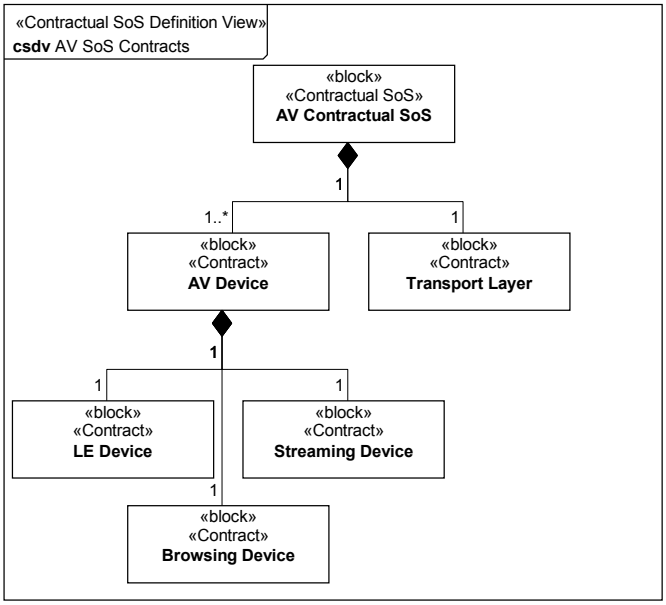

Fig. 5. Contractual SoS Definition View for AV SoS

contract through the same two interfaces (*rec* and *send*). These interfaces, not given here, may be defined using the Interfaces Pattern[5]. These interfaces comprise simple sending and receiving operations – and are data agnostic. We also show that the LE Device, Browsing Device and Streaming Device contracts connect to the transport layer through the *rec* and *send* interfaces. In this figure, we give only two AV Device contracts; there may be many more.

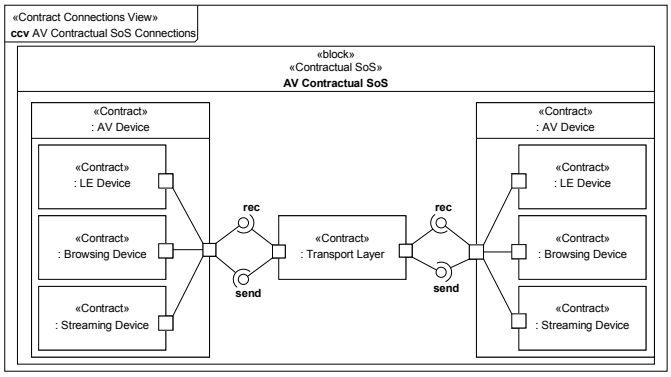

Fig. 6. Contract Connections View for AV SoS

The *Contract Conformance View* in Figure 7 shows how the constituent systems in an example AV SoS conform to those contracts. The network conforms to the Transport Layer contract, and the TV, Hifi and Content Provider all conform to the AV Device contract.

Next we produce a *Contract Definition View* and *Contract Protocol View* for each of the identified contracts. These two views allow the description of a contract in terms of their internal state, internal and external operations and the ordering

[5]The Interface Pattern, defined using the CAFF and given in [14], identifies the publicly accessible operations and their protocols.

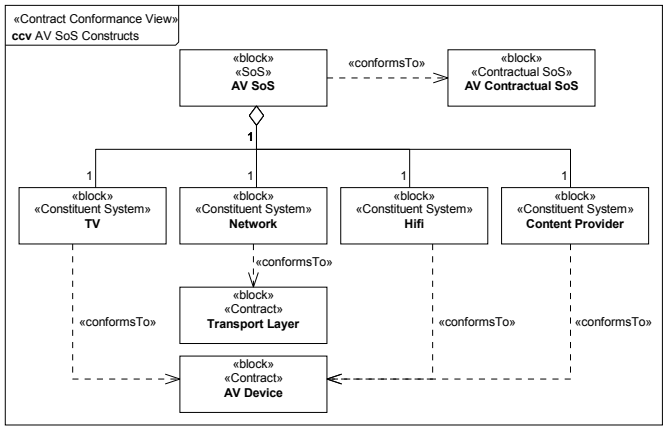

Fig. 7. Contract Conformance View for AV SoS and constituent systems

of the operations and communications with the contract's environment. By defining the contract in this manner, we enable the analysis of data and behavioural correctness including simulation, model checking and theorem proving, discussed in earlier work in [3]. Another form of analysis – ensuring contract conformance – is considered in Section V. For space reasons, in this paper we consider only the LE Device contract.

Figure 8 is a partial *Contract Definition View* for the LE Device contract. This partial definition names the private attributes (or values) and operations for the contract, then identifies two invariants over the values and gives three of the operations in more detail. The remaining operations are omitting for legibility reasons. The two invariants, *inv1* and *inv2*, constrain the LE Device's *mem* and *otherLeaders* state variables respectively. The definition of these invariants can be given in natural text, but in this example we use structured expressions defined in CML, which is the formal modelling notation developed specifically for modelling SoSs.

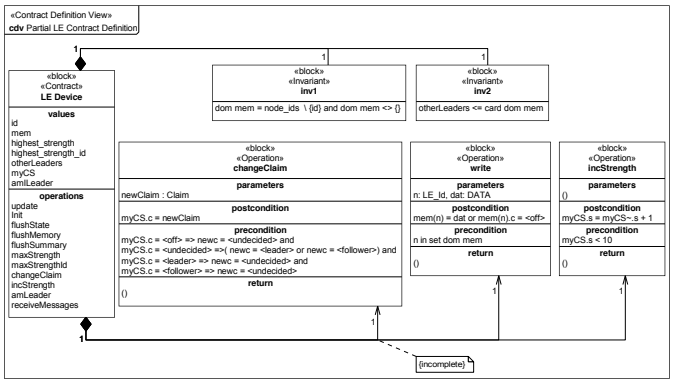

Fig. 8. Partial Contract Definition View for LE Device contract

For example, within the leadership election, a device may choose to take the role of a leader, a follower, or to be undecided. The *changeClaim* operation allows the device to change the role that it is going to take. It requires the new role (or *claim*) as a parameter. The precondition constrains the legal choice of new claims depending upon the current state of the LE Device. For example, the first clause of the precondition states that if the claim prior to operation invocation

is set to $\langle off \rangle$, then the new claim must be $\langle undecided \rangle$. If the precondition holds, then the postcondition states that the *changeClaim* operation will result in the LE Device's claim to be the one given as the parameter. The pre- and postconditions are, as with the state invariants, defined using CML.

Figure 9 is the *Contract Protocol View* for the LE Device. The protocol for this contract dictates the permitted ordering of operation calls for constituent system conforming to the contract. Briefly, the LE Device begins in an *Off* state, and may transition only to the *On* state. When in this state, the LE Device concurrently behaves in the *Election* state (transitioning between acting as $\langle leader \rangle$, $\langle follower \rangle$ and $\langle undecided \rangle$) and the *Listener* state, (in which it updates its view of the states of the other devices according to the messages it receives).

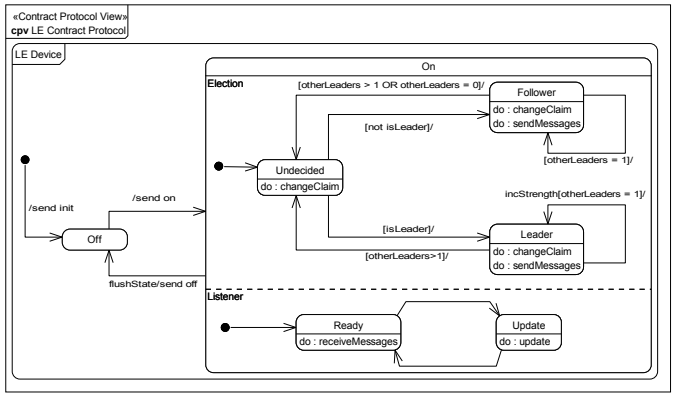

Fig. 9. Contract Protocol View for LE Device contract

In this section we have shown an example contract – the *LE Device* contract – presented following the Contract Pattern. An AV constituent system in the SoS must conform to this (and other) contracts. In the following section, we consider the question of verifying this conformance.

V. Contract Conformance

In the *Contract Conformance Viewpoint* a SoS engineer may state informally that a constituent system of the SoS conforms to a defined contract. In previous work, we considered how our earlier version of contracts for SysML may be translated to the formal notation CML, and suggested that CML refinement could be used as means of checking conformance. This idea is outlined in Figure 10. In this paper we do not discuss the process of translation, rather we consider some of the issues involved in using the results of this translation to provide evidence for contract conformance. A complete set of formal translation rules is defined in [15].

The contract and the constituent system will often have different interfaces, that is, offer different means of interaction. This is because a contract does not specify the entirety of system behaviour, but allows the designer of a constituent system some freedom to design operations of the constituent system as they choose, provided they meet the contract.

It seems clear that these additional operations and signals should be ignored when checking conformance. For this reason, we propose the use of a new feature called *hiding*. This is an annotation of a contractual SoS that indicates which of

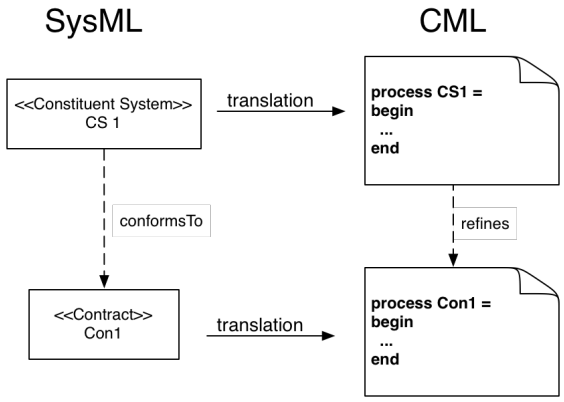

Fig. 10. Outline of relationship between contract conformance and CML refinement

its operations and signals are internal, and should be ignored when checking conformance. The result of this annotation is that during an execution of the contractual SoS, occurrences of the hidden operations and signals are not visible and do not require any external stimulus to take place. Instead, they take place when they become available.

Although hiding solves the issue of comparing models with different means of interaction, some care must be taken when using it to ensure that unused operations and signals are not hidden. This problem can occur in two main situations. The first is when an interface in a constituent system provides an operation which is not required by any of its connected interfaces. If the provided but not required operation is hidden it can take place without external stimulus. This leads to a problem because the SysML semantics of ports means that they are always ready to receive operation calls, and that calls are stored for the block to deal with. The operation can then take place infinitely often. Allowing state machines to be defined at ports would allow us to avoid this problem, provided the state machine placed appropriate limits on the permitted calls to the operation. Note that the Interface Pattern allows the use of state machines to be defined at ports [14].

The second way in which the problem can occur is through associations. SysML permits associations between blocks, and therefore allows the possibility that operations are called directly, rather than though an interface. We avoid this in the Contract Pattern by stipulating that connections between constituent systems may only be through interfaces at ports.

In summary, in order to support the verification of refinement between SysML elements, they must be made compatible. Hiding makes this compatibility possible, but must be used with care, to ensure that divergent behaviours are avoided. When the contracts are compatible, we are in a position to translate them to CML, and to carry out a refinement check between the two models (Figure 10). The results of this may be reported to the engineer, and recorded at the SysML level. For example, success may be included on a *Contract Conformance Viewpoint* as an *evidence* model element, as in Figure 11. Failure may be reported as a trace of the system which is not permitted by the contract (or vice versa). This trace may be translated into a SysML sequence diagram which may be more readily understood by the SoS engineer.

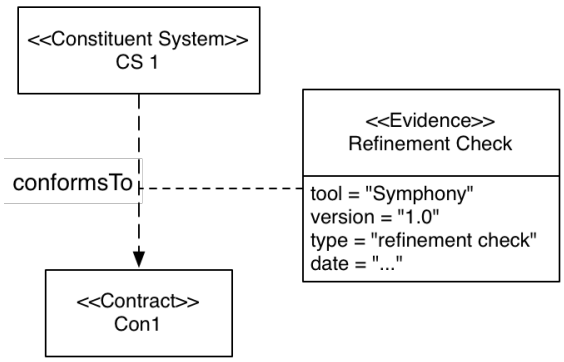

Fig. 11. Concept for recording refinement results in CCV

VI. Conclusion

We have proposed and demonstrated the Contract Pattern: an architectural pattern for the contractual specification of interfaces between constituent systems of an SoS. Included informally in the pattern is the concept of the *conformance* of a constituent system to its contract. In Section V we investigated some of the issues involved in using CML refinement to make the meaning of conformance more precise. Contractual specification is valuable during design in order to reduce the uncertainty associated with emergence. We anticipate being able to monitor contracts during the operational phases of an SoS, in order to check conformance and identify non-conformance of constituent systems.

The description of the pattern has followed the guidelines for the definition of architectural frameworks and patterns used within the COMPASS project. We anticipate that this will assist us in integrating the Contract Pattern with other patterns and frameworks. Further work will include integrating the Fault Modelling Architectural Framework [5] to allow us to develop contracts which are tolerant to faults. Other possible extensions to the pattern include non-functional properties and security features explicitly in the definition of a contract.

Other areas of further work include: defining static semantic rules that can be checked to ensure that a contract has used the Contract Pattern correctly; exploring the link between conformance and CML refinement further, including support for the definition and refinement of SoS-level contracts; exploring the benefits and consequences of contract composition; and to define a pattern for representing refinement at the SysML level (including concepts such as hiding as described in Section V) which may be combined with the Contract Pattern.

The case study presented here has been sufficient to highlight areas of future development of the Contract Pattern. Future work is evaluate the applicability of the Contract Pattern in the design of an industrial scale system.

Acknowledgment

The work presented here is supported by the UK EPSRC platform grant on Trustworthy Ambient Systems (TrAmS-2), and by the EU Framework 7 Integrated Project "Comprehensive Modelling for Advanced Systems of Systems" (COMPASS, Grant Agreement 287829). For more information see http://www.compass-research.eu.

References

[1] M. Jamshidi, Ed., *System of Systems Engineering: Innovations for the Twenty-First Century*. Wiley Series in Systems Engineering and Management, 2008.

[2] R. Payne, J. Bryans, J. S. Fitzgerald, and S. Riddle, "Interface specification for system-of-systems architectures," in *SoSE*, 2012, pp. 567–572.

[3] J. W. Bryans, J. S. Fitzgerald, R. J. Payne, and K. Kristensen, "Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1406, January 2014.

[4] J. Holt, S. Perry, F. O. Hansen, S. Hallerstede, and K. Kristensen, "Initial Report on Guidelines for Architectural Level SoS Modelling," COMPASS Deliverable, D21.2, Tech. Rep., 2013, available at http://www.compass-research.eu/.

[5] Z. Andrews, J. Holt, C. Ingram, R. Payne, S. Perry, and A. Romanovsky, "Traceable Engineering of Fault-Tolerant SoSs," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1391, 2014.

[6] "OMG Systems Modeling Language (OMG SysML$^{\mathrm{TM}}$)," SysML Modelling team, Tech. Rep. Version 1.2, June 2010, http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

[7] J. Holt and S. Perry, *SysML for Systems Engineering*. IET, 2008.

[8] J. Bryans, R. Payne, J. Holt, and S. Perry, "Semi-formal and formal interface specification for system of systems architecture," in *7th International Systems Conference, IEEE SysCon*. IEEE, April 2013.

[9] B. Meyer, *Object-Oriented Software Construction, 1st edition*. Prentice-Hall, 1988.

[10] R. J. Payne and J. S. Fitzgerald, "Evaluation of Architectural Frameworks Supporting Contract-based Specification," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1233, December 2010.

[11] J. Bryans, J. S. Fitzgerald, R. Payne, and K. Kristensen, "Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML," in *submitted to Annual International Symposium of the International Council on Systems Engineering*. INCOSE, 2014.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object Oriented Software*. Addison Wesley, 1995.

[13] C. Alexander, M. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-Ling, and S. Angel, *A Pattern Language*. Oxford University Press, 1977.

[14] S. Perry, "Report on Modelling Patterns for SoS Architectures," COMPASS Deliverable, D22.3, Tech. Rep., February 2013. [Online]. Available: http://www.compass-research.eu/deliverables.html

[15] A. Miyazawa, "Final Report on Combining SysML and CML," COMPASS Deliverable, D22.4, Tech. Rep., March 2013, available at http://www.compass-research.eu/.