An approach for managing semantic heterogeneity in Systems of Systems Engineering

Simon Foster, Alvaro Miyazawa,
Jim Woodcock, Ana Cavalcanti
University of York, UK
firstname.lastname@york.ac.uk

John Fitzgerald
Newcastle University, UK
john.fitzgerald@ncl.ac.uk

Peter Gorm Larsen
Aarhus University, Denmark
pgl@eng.au.dk

***Abstract*—Semantic heterogeneity is a significant challenge to integration in Systems of Systems Engineering (SoSE) due the large variety of languages, domains and tools which are used in their construction. In this paper we envision a strategy for managing this heterogeneity by decomposing domain specific languages into their "building block" theories which can be independently analysed, and used as a basis for linking with similar notations. This provides a systematic approach to building a tool-chain which integrates the different theories, methods and tools used in SoSE. Our approach has been piloted on the development of theories enabling machine-supported analysis of SysML models of SoSs. We conclude that the approach has further potential and identify lines of future research, notably in techniques for handling mixed discrete and continuous behaviour, timebands, mobility and model integration in SoSE.**

***Keywords* – systems of systems, modelling, integration, unifying theories, tool-chain, theorem proving.**

I. Introduction

Systems of Systems Engineering (SoSE) is a collection of techniques that support the development and maintenance of a potentially complex aggregate of independently owned and managed systems that are relied upon to provide an emergent service. The nature of the systems that form this aggregate vary in terms of the domain of application, level of independence, ownership, manageability, etc. This variance potentially leads to a plethora of seemingly incompatible models, methods and techniques, and the effective engineering of SoSs depends on their coordinated use.

These issues are observed at different levels of abstraction. For instance at a lower level, we observe different theories in play: integer and rational arithmetic, real and complex differential calculus, sequential and distributed computation, etc. On a different level, we observe different combinations of such theories being used to model systems. For instance, pure software systems are usually modelled by a combination of sequential and parallel computations, whilst the physical parts of cyber-physical systems are modelled in terms of systems of differential equations and more traditional computations.

At an even higher level, we observe variations in domain-specific languages (DSLs) used to represent both the theories and combinations of theories. For instance, programming languages such as C and Java have different syntaxes, and diagrammatic notations such as MATLAB Stateflow [1] and UML State Machines have some different elements and variations in their semantics. These variations at all levels of abstraction need to be managed in order to fully support the formal development of SoSs.

Moreover SoS engineering in large projects is complicated by the use of different tools for engineering the different constituents. The different DSLs are supported by a collection of tools that variously enable an engineer to describe, refine and analyse a system model at different stages of development. This makes it difficult to co-ordinate the tools to provide evidence that an SoS deployment fulfils its requirements. Although we might seek to enforce the use of a single tool in a large SoS consortium, experience tells us that this is not possible as different tools have unique contributions, and individual members will be have experience in a particular tool-set that cannot be abandoned without significant cost.

It is therefore necessary to face the challenge of semantic heterogeneity. We present a vision for an integrated semantic framework for SoS engineering that we believe will enable semantically heterogeneous notations and tools to be unified and co-ordinated. Our approach is to look at the individual notations involved and perform a semantic decomposition, which involves separating out the individual theoretical ideas in an effort to see how a notation fits with other similar notations. We achieve this using Hoare and He's *Unifying Theories of Programming* [2] semantic framework (UTP), which allows different theoretical aspects of a modelling language to be formally isolated, modelled and contrasted. By extension this means that SoS features, such as time, concurrency and mobility, can be considered as independent aspects that can then be composed to give a mathematical meaning to a system. Moreover, our semantic framework is *mechanised* meaning that we can reason mechanically about a DSL to prove both soundness properties and model correctness properties.

In the remainder we present our contributions. In Section II we explain how an integrated tool-chain can used to solve the problem of heterogeneity. In Section III we expand on this by introducing *theory engineering*, by which the constituent aspects of a DSL can be studied. In Section IV we exemplify our vision by analysing OMG's system modelling language *SysML* in terms of its constituent theories. In Section V we give some future directions for research, particularly in the area of cyber-physical systems. Finally in Section VI we conclude.

II. Tool Integration

In this section we introduce the idea of tool-chain integration through application of the UTP semantic framework, and

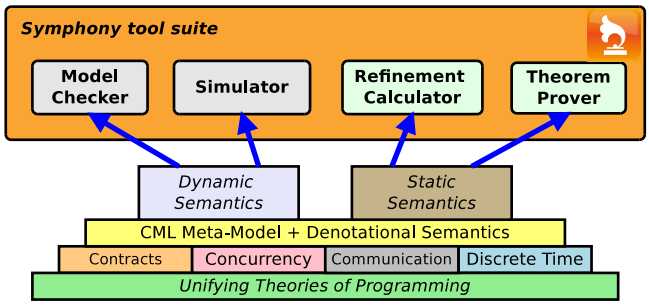

Fig. 1: The CML semantic stack

discuss the study of different DSLs through theory decomposition. In essence we seek to factor out common aspects of a DSL in an effort to relate it to other, similar DSLs.

Creation of an integrated tool-chain for SoSE requires that we give a unified semantic account of the artifacts and results from the various tools. This is the essence of integration: that the tools can be co-ordinated to produce coherent and dependable analysis results and evidence. Achieving this requires that the underlying DSLs of the different tools can be unified by providing them with unambiguous and compatible mathematical meaning. Different analysis tools are based on different notations, for example, a model simulator may work at the level of a transition relation described using structural operational semantics (SOS) rules, whilst a program verifier may use an axiomatic Hoare calculus. Though distinguishable, these formalisms are related in that they provide a particular abstract view of the modelling world. If we are to co-ordinate the tools, we also need to formally link the different semantic models to likewise ensure their compatibility.

The approach taken by UTP is to define *denotational models* for the different languages, taking input from standard meta-models. A denotational model allows us to give a language a semantics by assigning to each language construct a mathematical object. For example, the operators of an imperative programming language can be described using relational calculus. This then allows the application of the laws and proof procedures of the relational calculus to program verification. Denotational semantics for modelling languages are often much more complicated, combining a wide variety of theoretical notions. Once a suitably expressive model has been fixed, it can be used as a means to prove correspondence between the semantic models so that the associated tool evidence can be properly composed. This idea is illustrated in Figure 1 for a tool-chain consisting of a model checker and simulator, which are both based on operational semantics (dynamic), and a refinement calculator and program verifier, which are both based on axiomatic semantics (static). Though these four components are independent, their basis in a unified semantics means they can be co-ordinated during system development.

Within the context of the COMPASS project [3] this approach has been successfully applied to the development of the *Symphony* tool platform [4][1]. The Symphony tool provides syntax and type checking, interpretation/debugging, proof obligation generation, theorem proving, model checking, test automation and a connection to the Artisan Studio SysML tool where static fault analysis additionally is supplied. Symphony is based on an SoS modelling language called CML (The COMPASS Modelling Language), which combines a number of aspects required in SoS modelling such as discrete time, concurrency, processes, state and contracts [5]. A CML model consists of the following principle elements:

- *types*: such as numeric types, lists, sets, records, union types, and possible invariants;
- *functions*: map input types to output types, with possible pre- and post-conditions;
- *channels*: over which constituent systems can communicate messages;
- *processes*: model constituents, and in turn consist of:
  - *state variables*: private mutable state;
  - *operations*: acting on the state variables;
  - *actions*: specify reactive behaviour (operations calls, message passing, timeout...).

The semantics of CML is being formally constructed in its denotational, operational and axiomatic flavours [3], and these various bases have been used to implement independently the simulator, model checker and theorem prover. Moreover, we have mechanised semantic models for CML in the *Isabelle/HOL* [6] interactive theorem prover. Isabelle/HOL is a proof assistant for Higher-Order Logic, a kind of functional programming language in which one can also state and (dis)prove logical properties. Isabelle brings together a wide variety of automated proof tools [7], such as first-order automated theorem proving in the *sledgehammer* tool, and counter-example generation in the *nitpick* tool. It therefore provides an excellent basis for proving theorems about individual models, for example discharging consistency proof obligations for CML. Perhaps more importantly though it also allows us to formalise soundness proofs about the underlying meta-models themselves, which is the subject of the next Section.

III. Theory Engineering

Core to UTP is the idea of a *theory*: an isolated interesting problem domain that deserves independent study. A UTP theory consists of an *alphabet* describing observations that can be made, a *signature* consisting of constructors for theory objects, and *healthiness conditions* that define the conditions of theory membership. For example, consider a theory of reactive processes where behaviour is represented by event traces. The trace observations can be recorded by a variable $tr : Event^*$ whose values are lists of events. If we consider the trace before and after an action has executed, we need two such variables: tr and tr'. Then an obvious healthiness condition is that traces can only get longer, which can be formulated as $tr \leq tr'$. Additional healthiness conditions can then be used to formulate other aspects of concurrency, like time, as required by a denotational model. The signature of this theory consists of the usual operators for building concurrent processes, such as parallel composition and message passing. From a UTP theory we can derive laws of programming and concurrency which then act as the basis for various semantic models.

A variety of theoretical aspects have been formalised in UTP, including concurrency, discrete time, object-orientation, pointers and contracts. This is the theory layer of the semantic stack in Figure 1 which gives us the basis for performing semantic decomposition. When considering a particular language, we can link it to other languages by looking for

[1] See http://symphonytool.org/ for more information.

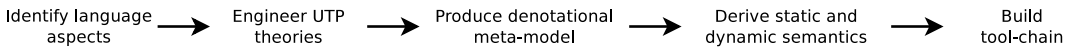

Fig. 2: Systematic approach to engineering semantic meta-models

common factors. In Figure 3 we consider the theoretical aspects present in nine languages, in terms of seven theories. Though incomplete, it nevertheless shows that there is both commonality and differences between them. If these are considered in terms of UTP theories, we are given a well-founded way to account for semantic heterogeneity.

Distinct UTP theories can also be formally linked, for example through *Galois connections* [2]. A Galois connection consists of a pair of functions which together formalise the best approximations of an object of one theory in another. This allows parts of a model in one language to be approximated and reproduced in another language, thus providing points of linkage. In CML, for example, a Galois connection is used to link processes that model time to those that do not (*untimed* processes), enabling composition of processes that are heterogeneous with respect to time. Therefore conquering semantic heterogeneity reduces to *theory engineering*: construction and analysis of constituent theories, formation of links between them, and their application to solve practical problems, such as SoSE. The overall approach is illustrated in Figure 2.

To mechanically support theory engineering we have created our own semantic embedding of UTP in Isabelle called *Isabelle/UTP* [8]. We have used this to mechanise a number of key theories underlying CML processes, such as imperative programs with total correctness, and concurrency in reactive processes. We have then combined these theories to form a mechanised denotational model for CML, which is in turn used as the basis for the theorem prover component of Symphony. This mechanisation increases confidence in the semantic model's soundness, in a similar way to how a pocket calculator can be used to verify a complicated calculation. Since we have a formal link between the different tools and underlying semantics we have an unbroken chain from engineering methodology to the underlying mathematics.

Moreover, from an practical standpoint, Isabelle provides a number of helpful features to aid in theory engineering. Isabelle is backed up by a large theory library, both bundled and in the associated *Archive of Formal Proofs*[2], to which theoreticians regularly contribute their mechanisation work. This gives the basis for importing existing theory mechanised by others into more expressive denotational models. For example, ordinary differential equations have been mechanised [9], and this can provide the basis for building a theory of continuous time for reasoning about hybrid and cyber-physical systems. This along with its powerful reasoning facilities and our implementation of UTP makes it an ideal environment in which to study semantic heterogeneity.

[2]http://afp.sourceforge.net/

IV. Theoretical decomposition of SysML

SysML is a graphical notation aimed at modelling systems and as such has been adopted as a base notation for the COMPASS project along with CML. SysML provides a number of diagrams that help construct a model in a manner akin to the weaving process found in the aspect-oriented programming paradigm. Four of these are structural diagrams: block definition, internal block, package and parametric diagrams [10]. Block diagrams support the definition of the blocks that form the model as well as their components and relations, internal block diagrams support the description of the internal structure of composite blocks, package diagrams represent the interdependencies between sets of elements of the model, and parametric diagrams allow the statement of constraints over the properties of the model.

SysML also provides four behavioural diagrams: use case, sequence, activity and state machine diagrams. These support the description of the behaviours of the system, often at different levels of abstraction. For instance, use-case diagrams are often used to model high-level interactions with the system, whilst state-machine diagrams model at a lower level of abstraction how the individual components of the system behave.

State-machine diagrams describe the system in terms of its configurations (states), and activity diagrams provide the means of describing workflows and an alternative perspective in the specification of the behaviours of systems. Sequence diagrams support the specification of scenarios, which describe particular ways in which the elements of the system can interact by means of message exchanges. Finally, requirement diagrams provide support for structuring requirements in terms of decomposition and derivation as well as traceability.

Whilst CML has a formal foundation based on the already mentioned UTP, SysML lacks a formal account beyond syntactic and basic consistency properties. This limitation has been tackled by an integration of SysML in the formal setting of

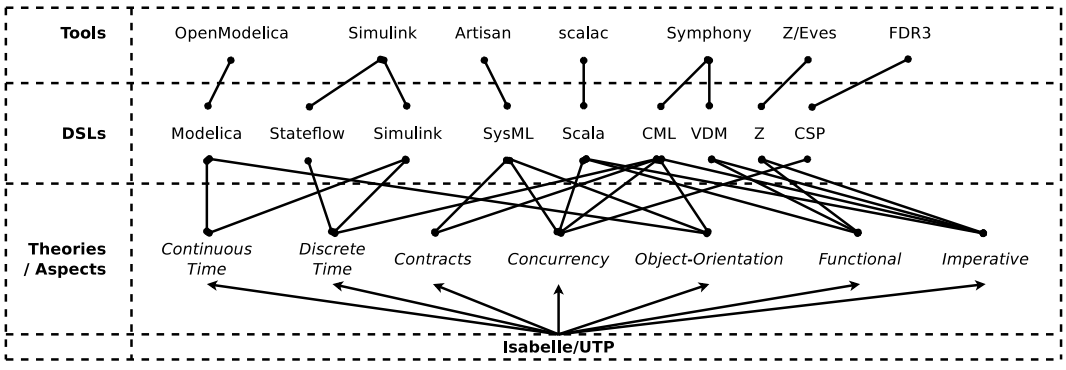

Fig. 3: Vision for links between tools, languages and theories

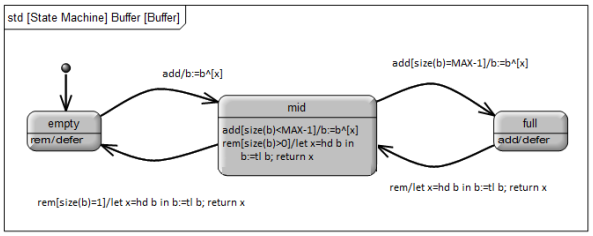

Fig. 4: State machine specifying the behaviour of a buffer.

CML	SysML
Formal	Semi-Formal
Synchronous communication	Asynchronous communication
Single notation	Multiple notation
Textual	Diagrammatic
Complete static semantics	Partial static semantics
Complete dynamic semantics	No dynamic semantics
No support for views	Support for views
Academic	Industrial

Fig. 5: Differences between CML and SysML.

COMPASS based on the informal semantics described in [11] and [12]. We first identified the elements of the notation for which formal support in the form of UTP theories already existed. For instance, state machines describe a subset of reactive processes that involve communication, parallelism and data operations in very specific patterns. Flow ports and parametric diagrams, on the other hand, potentially require the availability of a continuous-time theory as they can specify physical aspects of the system that are often modelled by systems of differential equations. Table 5 summarises some of the differences between CML and SysML that are further discussed in this section.

In the particular case of COMPASS, it has been observed that the subset of SysML that can be formalised within the currently available theories is the subset that can be specified in CML if we add extra abstractions. For instance, basic SysML constructs (such as transitions) often specify behaviours that are not available as primitives, but can be specified using CML. For this reason, the semantics of SysML has been defined in terms of CML, which then provides the link to the more basic theories of discrete time, concurrent state, object orientation, designs and refinement.

As an example, the semantics of the state-machine diagram shown in Figure 4 is a CML process whose behaviour is described by a number of parallel actions: one for the state machine, and one for each state and transition. There are eight parallel actions: `stm_Buffer`, `s_empty`, `s_mid`, `s_full`, `t_empty_mid`, `t_mid_empty`, `t_mid_full` and `t_full_mid`. The CML model of the state machine in Figure 4 is then used to model the behaviour of the block `Buffer` that contains that state machine, which in turn is used to specify the model of the overall system.

A particularly interesting point is related to the communication patterns in SysML and CML. Whilst in CML communication is strictly synchronous, in SysML it is predominantly asynchronous. As a consequence, the semantics of SysML must provide an account for asynchronous communication in terms of synchronous communications. This is achieved by the introduction of two CML communications for each SysML communication – one for sending a value and another for receiving a value – as well as a buffer that allows values sent through a communication channel to be queued.

Another aspect in which SysML and CML differ in spite of similar terminologies is the use of operations. In CML, a class operation only modifies data, while in SysML, a block operation may also contain reactive behaviour (e.g., sending an event). The consequence of this mismatch is that SysML operations are modelled as CML actions of a process (that models a block). However, the actions of a process in CML are encapsulated and therefore cannot be called by other processes. This differs from the semantics of operations in SysML, and for this reason the actions that model the SysML operations must be made accessible by the only means a CML process has for interaction with other process: communication.

Now, if we consider a scenario where the design of a SoS involves multiple notations, provided these notations have a common foundation and are compatible, we should be able to analyse the collective behaviour of the SoS. This is, however, rarely the case. An an example, we consider a model where the most abstract specification of the SoS is described in CML, and its design is specified in such a way that some of the constituent systems are modelled in SysML and others are modelled in MATLAB's Simulink/Stateflow[3]. Since [13] and [14] provide an account for a subset of discrete time Simulink/Stateflow in *Circus* [15], which is a state-rich process algebra that shares a similar semantic foundation as CML, it would be desirable to have these models integrated and analysed. For this to be possible, the different models need to be made compatible, that is, an operation call in SysML must be translated into an interaction that a Stateflow diagram or a CML process can understand. If such compatibility can be achieved, the models can be integrated (provided they are all based on discrete time) and their interactions analysed. In particular, it should be possible to compare the abstract specification of the SoS in CML and the actual design modelled in the different formalisms by means of a theory of refinement.

V. Future Directions

In this Section we sketch out some future directions for research to address semantic heterogeneity in SoS Engineering, including: mixed discrete and continuous behaviour, time-bands, mobility and model integration.

Continuous Time. One of the most challenging areas for semantic heterogeneity in SoSE is the link between continuous time models of environmental and controlled phenomena, and the discrete time models of digital systems that interact with them. There are many possible SoSs in which one would wish to verify the presence or absence of an emergent behaviour that requires both cyber and physical models. For example, the integrator of a smart grid SoS may need to verify that feature interaction between existing independent power distribution systems will not lead to overloading of physical storage media such as batteries, or "brown outs" in the network. This requires

[3]Simulink/Stateflow is graphical notation that supports the specification of cyber-physical systems in terms of both discrete and continuous time constructs.

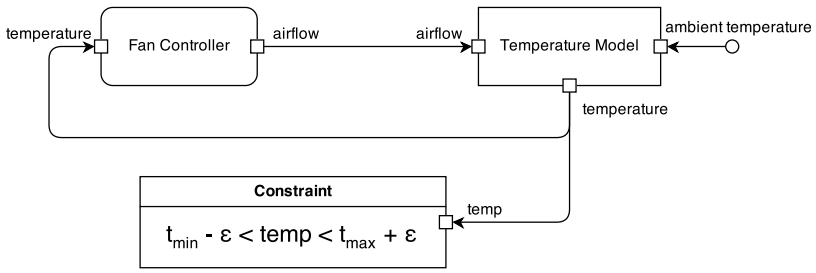

Fig. 6: A hybrid model of a warehouse cooling system

modelling of both the (discrete) computing systems providing control, as well as the physics of electrical storage and distribution. The integrated computing elements of cyber-physical systems are likely to be complex, given the need to handle faults originating in the independent constituent systems.

In order to illustrate the research questions posed by the need to handle discrete and continuous models, and the extent to which our theory decomposition and UTP-based approach can help, we consider an admittedly very simple control example inspired by a Simulink model [1]. In the scenario that we consider, the owner of a warehouse storing a temperature-sensitive product wishes to install a cooling system that ensures the warehouse never exceeds a particular temperature, but is also cost effective. We model this by the hybrid system shown in Figure 6. The physical temperature of the warehouse is modelled by a continuous time model, with two parameters: the *ambient temperature* and *airflow* from the fan. The fan controller is a discrete component, which is connected to a temperature model via the port *temperature*. The controller starts in an off state, but when the ambient temperature reported exceeds t_{max}, the fan is activated to ensure the temperature does not exceed $t_{max} + \epsilon$. This in turn contributes to an increase in airflow that lowers the temperature according to the continuous-time model. Once the temperature reaches t_{min}, the fan is turned off. From the design perspective we would like to understand two main variables of this system: (1) the minimal ϵ we can have and (2) the lowest sampling rate needed to respond sufficiently quickly. Both of these questions are important to the modelling and implementation of a correct discrete controller.

Co-modelling. Modelling such systems requires that we consider both the discrete and continuous models. This task falls in the domain of *co-modelling*, where both aspects are considered in the engineering of the model [16]. Co-modelling includes techniques such as co-simulation, where a discrete and a continuous model are simulated in parallel, and hardware in the loop simulation (HiL), where the continuous model is swapped for a real hardware component. These techniques though informative, only provide one part of the engineering toolbox. As in formal methods we would like access to other tools, such as model checking and theorem proving, which together can provide greater assurance of correctness.

By formally constructing these models and applying such tools we may be able to verify that for a particular ϵ and sampling rate, the temperature bounds are respected. Whilst there are specialised tools that target hybrid systems, the majority of existing research into analysis tools focuses on the discrete domain. If we are to apply methods from the discrete domain to hybrid systems, we need to understand the theories behind both domains and formalise how they are linked. One step towards this is the development of continuous-time models within the context of the UTP, and linking them to existing models of discrete-time such as *Circus Time* [17] and CML.

There are a number of interesting research directions in this area. In particular the *timebands* frameworks [18] could be applied to specify and analyse hybrid systems through considering time at different levels of granularity. A timeband is an abstraction of the real-time continuum to a particular time unit, which defines the minimal interval at which *events* can be distinguished. Within a particular timeband events are instantaneous, whilst in a finer band an event can be associated to an *activity* which may have duration. For example, the minute timeband can be used to distinguish the occurrence of events separated by minutes, whilst the second timeband can further distinguish events occurring in the same minute. Furthermore, an event in the minute timeband can be associated with an activity in the second timeband, which is itself decomposed into individual events. This can be applied to compare different levels of abstraction of the time domain, and can therefore be used to link discretisations of a continuous model.

For instance we could start from a continuous time model that corresponds to the constraint in Figure 6. Next we would specify and verify a set of timebands that model a controller guaranteeing the constraint using a high sampling rate. In subsequent steps we can refine these timebands to use lower and lower sampling rates without violating the required temperature bounds. This would ensure that the final discrete controller implementation guarantees the initial property.

Formalisation in the UTP can be given by two theories for discrete and continuous time. One possibility for a discrete time UTP theory, based on timebands, is to model the time continuum as a function $cont : \mathbb{N} \to Event$ and a variable $quant : \mathbb{R}$ to represent the time unit. A particular value of *quant* then corresponds to a particular timeband. A theory of continuous time in contrast can have $rcont : \mathbb{R} \to Event$ and behaviours can be modelled via a system of ordinary differential equations. We can then formalise Galois connections between different granularities of time in a discrete time theory within the continuous domain. This in turn allows us to transfer results proved in the continuous domain to a discretisation. Moreover, the two theories could provide models for related calculi, such as the *duration calculus*, which allow the formulation of temporal properties over a continuum.

Mobility. Aside from hybrid and cyber-physical models, a number of other aspects are important to SoSE. When considering evolution and reconfiguration of an SoS, we need to consider the modelling of *mobility*. Mobility, broadly speaking,

allows the representation of systems whose topology and architecture can change dynamically at runtime. For instance, in an emergency response system the communication system needs to reconfigure to take advantage of the best medium available. This may involve switching from radio, to cellular network, to satellite communication, depending on the circumstances. There are two main approaches to mobility, *process mobility* where individual processes can change location (e.g. Ambient Calculus [19]), and *channel mobility* where the processes are fixed, but the communicating topology can change (e.g. π-calculus [20]). Both models can be incorporated into the UTP possibly involving suitable higher-order models [21]. When used in conjunction with timebands, we may also be able to specify situations in which the loss of service due to hardware swapping has no significant impact on the availability of services. This kind of combination again can be supported by UTP theory composition.

Integration. We can then go one step further and consider the question of integration of existing models. If we already have access to a pre-existing temperature model in a diagrammatic notation such as Modelica [22] and wish to specify the controller in Stateflow, instead of translating the Modelica model into a compatible Simulink model we may wish to integrate them directly. Two possible solutions to this problem exist. Firstly, we can formalise the primitive notions in the languages as UTP theories, and construct semantic models for Modelica and Stateflow linked by their common factors. Secondly, we can specify the semantics independently and then provide formal adaptors that mediate the interaction between the two models. These adaptors can be modelled in the UTP by Galois connections. In this case we would like to access the *airflow* port interface in the Modelica temperature model, and the *temperature* port in the Stateflow controller model, both of which must be approximated in the adjacent model.

VI. Conclusion

The need to deal with diversity is one of the distinguishing characteristics of SoS Engineering [23]. In our work, we focus on the need to address the semantic diversity of models that make up a SoS description. Our systematic approach exploits the UTP to provide "building block" theories that can be composed via Galois connections to provide reasoning systems able to compose results from previously separate models.

This is still at the level of a vision. However, first steps have been realised in CML and the benefits are to be seen in the integrated tool-chain that is now emerging from this work. We have demonstrated how the approach plays out in the integration of SysML with CML. The practical costs of providing a semantic integration are yet to be evaluated, but it is important to emphasise that they are "one-off" costs in the sense that, once a UTP-based integration has been achieved, it serves for any use of the constituent model types.

Our experience in applying theory engineering in COMPASS suggests that there is potential in this approach for providing a sound basis for reasoning about global emergent SoS behaviours from heterogeneous component models, and so we have identified promising next steps in research. The approach works at the foundations of constituent theories, but has a direct bearing on the feasibility of sound automated tool support that is able more fully to realise the value of model-based SoS Engineering.

Acknowledgment

This work is supported by EU FP7 Integrated Project "Comprehensive Modelling for Advanced Systems of Systems" (COMPASS, Grant Agreement 287829). For more information see http://www.compass-research.eu.

References

[1] *Stateflow and Stateflow Coder 7 User's Guide*, The MathWorks,Inc., `www.mathworks.com/products`.

[2] T. Hoare and J. He, *Unifying Theories of Programming*. Prentice Hall, 1998.

[3] J. Fitzgerald, P. Larsen, and J. Woodcock, "Foundations for Model-based Engineering of Systems of Systems," in *Complex Systems Design and Management*, M. A. et al., Ed. Springer, January 2014, pp. 1–19.

[4] J. W. Coleman, A. K. Malmos, P. G. Larsen, J. Peleska, R. Hains, Z. Andrews, R. Payne, S. Foster, A. Miyazawa, C. Bertolini, and A. Didier, "COMPASS Tool Vision for a System of Systems Collaborative Development Environment," in *IEEE SoSE*, July 2012, pp. 451–456.

[5] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, "Features of CML: a Formal Modelling Language for Systems of Systems," in *IEEE SoSE*, 2012.

[6] T. Nipkow, M. Wenzel, and L. Paulson, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer, 2002, vol. 2283.

[7] J. C. Blanchette, L. Bulwahn, and T. Nipkow, "Automatic proof and disproof in Isabelle/HOL," in *FroCoS*, ser. LNCS, vol. 6989. Springer, 2011, pp. 12–27.

[8] S. Foster, F. Zeyda, and J. Woodcock, "Isabelle/UTP: A mechanised theory engineering framework," in *5th Intl. Symposium on Unifying Theories of Programming*, 2014.

[9] F. Immler and J. Hölzl, "Numerical analysis of ordinary differential equations in Isabelle/HOL," in *ITP*, ser. LNCS, vol. 7406. Springer, 2012, pp. 377–392.

[10] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML*. Morgan Kaufman OMG Press, 2008.

[11] "OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1," OMG, Tech. Rep., 2011.

[12] "OMG Systems Modeling Language (OMG SysML$^{\mathrm{TM}}$)," SysML Modelling team, Tech. Rep. Version 1.2, June 2010.

[13] A. Cavalcanti, P. Clayton, and C. O'Halloran, "Control Law Diagrams in *Circus*," in *FM*, ser. LNCS, vol. 3582. Springer, 2005, pp. 253–268.

[14] A. Miyazawa and A. Cavalcanti, "Refinement-oriented models of stateflow charts," *Sci. Comp. Prog.*, vol. 77, no. 10-11, pp. 1151–1177, 2012.

[15] A. Cavalcanti, A. Sampaio, and J. Woodcock, "A Refinement Strategy for Circus," *Formal Aspects of Computing*, vol. 15, no. 2-3, pp. 146–181, 2003.

[16] J. Fitzgerald, P. Larsen, and M. Verhoef, Eds., *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer, 2014.

[17] A. Sherif, A. Cavalcanti, J. He, and A. Sampaio, "A process algebraic framework for specification and validation of real-time systems," *Formal Aspects of Computing*, vol. 22, no. 2, pp. 153–191, 2010.

[18] A. Burns and I. J. Hayes, "A Timeband Framework for Modelling Real-Time Systems," *Real-Time Systems*, vol. 45, no. 1–2, pp. 106–142, 2010.

[19] L. Cardelli and A. Gordon, "Mobile ambients," *Theoretical Computer Science*, vol. 240, no. 1, pp. 177–213, 2000.

[20] R. Milner, *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[21] F. Zeyda and A. Cavalcanti, "Higher-order UTP for a theory of methods," in *UTP*, ser. LNCS, vol. 7681. Springer, 2012, pp. 204–223.

[22] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. Wiley-Blackwell, 2010.

[23] J. Boardman and B. Sauser, "System of Systems – the meaning of "of"," in *IEEE SoSE*, 2006, pp. 118–123.