Partial Behaviour Modelling: Foundations for Incremental and Iterative Model-Based Software Engineering

Sebastian Uchitel[1,2]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London, SW7 2RH, UK
[2] Department of Computer Science, FCEN, Universidad de Buenos Aires,
Intendente Güiraldes 2160, C1428EGA, Argentina
suchitel@dc.uba.ar, s.uchitel@doc.ic.ac.uk

Abstract. Rigorous modelling of the intended behaviour of software intensive systems has been shown to be successfull in uncovering requirements and design flaws. However, the impact that behaviour modelling has had among practitioners is limited. The construction of behaviour models remains a difficult and laborious task that requires significant expertise. In addition, traditional approaches to behaviour models require complete descriptions of the system behaviour up to some level of abstraction. This completeness assumption is limiting in the context of software development process best practices which include iterative development, adoption of use-case and scenario-based techniques and viewpoint- or stakeholder-based analysis; practices which require modelling and analysis in the presence of partial information about system behaviour. Our aim is to support the iterative and incremental construction of behaviour models by means of construction, composition and analysis of partial, heterogeneous, yet formal, descriptions of behaviour. In this talk we discuss how modal transitions systems can provide the basis for such support and present some of the model synthesis and composition techniques we have developed.

1 Introduction

Software systems are amenable to analysis through the construction of behaviour models. This corresponds to the traditional engineering approach to construction of complex systems. Models can be studied to increase confidence on the adequacy of the product to be built. The advantage of using behaviour models to describe systems is that they are cheaper to develop than the actual system. Consequently, they can be analysed and mechanically checked for properties in order to detect design errors early in the development process and allow cheaper fixes.

Although behaviour modelling and analysis has been shown to be successful in uncovering subtle requirements and design errors, adoption by practitioners has been slow. Partly, this is due to the complexity of building behavioural models in the first place – behaviour modelling remains a difficult, labour-intensive task that requires considerable expertise. To address this, a wide range of techniques for supporting automated and semi-automated synthesis of behaviour models have been investigated. In particular, synthesis from scenarios and use cases (e.g., [24, 10, 3, 19]), has been studied extensively.

A current limitation of synthesis approaches is that the models being synthesized, e.g., labeled transition systems (LTSs) [14], are typically assumed to be complete descriptions of the system behaviour. That is, that they completely classify all behaviours with respect to some fixed alphabet as either behaviour that the system-to-be is required to exhibit or behaviour that the system-to-be is prohibited from exhibiting. The required behaviour is decribed by the transitions that appear in the behaviour model. The proscribed behaviour is defined as anything that is not described by the model's transitions. This completeness assumption that usually is attached to behaviour models is problematic if these models is to be built from a scenario based-specifications which is inherently partial as synthesis procedures are left to cope with completing the specification automatically, or the engineer is required to put in more information before any meaningful analysis can be performed. Utlimately, this completeness assumption is limiting in the context of software development process best practices which include iterative development, adoption of use- case and scenario-based techniques and viewpoint- or stakeholder-based analysis; practices which require modelling and analysis in the presence of partial information about system behaviour.

A workaround to the completeness assumption is to reinterpret the two sets of behaviours that a behaviour model describes. Rather than interpreting the behaviour that cannot be reproduced by the transitions of a model as proscribed behaviour, it can be interpreted as being "yet to be determined". This interpretation works for scenario-based specifications that have an existential semantics (e.g. MSCs [13]) as these specifications provide examples of what the system must do, but do not say anything about what it must not do. Consequently, a behaviour model synthesized from scenarios provides a *lower bound* from which to identify the behaviours that the system will provide but that have not been explicitly captured by the scenarios. As these new behaviours are identified, they are added to the scenario specification which is then used to synthesis a new behaviour model that includes these new behaviours. This elaboration process can be formalised at the behaviour level with some notion of refinements such as trace inclusion or *simulation* [20].

An alternative workaround is to consider the behaviour explicitly described by the transitions of a behaviour model as unclassified and to assume that the rest of the behaviour is known to be proscribed. This is the interpretation taken for senario-based specifications that have a universal semantics such as Constant LSCs [10]. In such approaches, as with approaches that do synthesis from declarative specifications such as goal models [19]. The specification prunes the acceptable space of behaviours as more universal properties are added to the specification. The fact that a behaviour satisfies a universal statement does not mean that the system is required to provide that trace; the trace could be violating another property, possibly one yet to be elicited. Consequently, a behaviour model synthesized from properties should characterize all possible behaviours that do not violate the properties. Such a model provides an *upper bound* on all the behaviours that the system will actually provide, once implemented. Validation of behaviour models synthesized from properties can prompt the elicitation of more properties, which in turn will further approximate from above the intended behaviour of the system to be. In other words, as new properties are elicited, the resulting synthesized model will be able to do *less* (notion that can be formally captured using a traditional notion of refinement such as simulation), describing behaviour that is closer to that of the system to be.

The problem is that if behaviour models are to be synthesised from rich scenario based languages that use combine existential and universal scenarios as first envisioned in [10], the target synthesis formalism cannot be in the form of traditional behaviour models such as LTS because these are not capable of capturing simultaneously both the upper and lower bounds [22] that universal and existential statements provide.

2 Partial Behaviour Models

Partial behaviour models, such as Modal Transition Systems (MTS) [17], disinguish between three kinds of behaviour, required, proscribed and unknown, and therefore can describe *both* an upper and a lower bound to the intended system behaviour, allowing both bounds to be refined simultaneously. For instance, MTS are equipped with two kinds of transitions *required* transitions and *possible* transitions. The former provide a lower bound to system behaviour, while the latter provide the lower bound to system behaviour.

The semantics of a partial behaviour model can be thought of as a set of traditional behaviour models. For instance, MTS semantics can be given in terms of sets of LTSs that provide all of the behaviour required by the MTS, do not provide any of the behaviour proscribed by the MTS, and make arbitrary decisions on the MTS's unknown behaviour. Intuitively, as more information becomes available, unknown or unclassified behaviour gets changed into either required or proscribed behaviour. The notion of refinement between MTSs capture this intuition formally and provides an elegant way of describing the process of behaviour model elaboration as one in which behaviour information is acquired and introduced into the behaviour model incrementally, gradually refining an MTS until it characterizes a single LTS.

The original notion of refinement was aimed at comparing MTS models with the same alphabet and no unobservable transitions and is referred to as strong refinement [17]. Although in [17] a notion of weak refinement that allows for unobservable actions was defined, this notion was then extended to account for models different alphabets [23]. More recently, an alternative, possibly more appropriate observational refinement, based on branching equivalence [25] has also been proposed [7].

A particularly useful notion in the context of software and requirements engineering is that of *merge*. Merging two consistent models is a process that should result in a minimal common refinement of both models where *consistency* is defined as the existence of one common refinement. Intuitively, merging builds a model that characterises the intersection of the LTS characterised by the models being merged. In other words, the merge characterises the LTSs that provide all the required behaviour of the MTS being merged, and that do not provide any of the proscribed behaviour of the MTS being merged.

MTS merging can be used as the conjunction of multiple partial operational descriptions. The original formulation of was done by Larsen in [16] where an incomplete merge algorithm was proposed for MTS under strong refinement, recently we have presented a correct and complete version [8]. The problem of merge under observational refinements is still open, a partial result can be found in [23] where incomplete algorithm for merging models with different alphabets under weak refinement is presented.

We have revisted the problem of behaviour model synthesis in the context of MTS. We have provided a generic extension of synthesis approaches that start from existential scenario-based specifications and build LTS models [22]. The extension, produces an MTS model instead of an LTS which captures appropriately the lower bound to intended system behaviour provided by such specifications. However, given that MTS are more expressive than LTS, we have explored opportunities for the defining novel synthesis approaches that start from more expressive scenarios notations. In particular, we have investigated triggered existential scenarios [21] which have been neglected in existing scenario description languages as it is impossible to adequately capture their semantics using traditional behaviour models.

3 Conclusions

In this talk we discuss Modal Transition Systems [17] and some of their theoretical foundations and semantics. We discuss how such models can support iterative and incremental behaviour modelling based on a notion of refinement that prunes the space of acceptable implementations of the system-to-be and based on model merging. We also discuss how merge and synthesis of Modal Transition Systems can aide in the analysis and elaboration of system behaviour from multiple, partial and heterogeneous descriptions of behaviour and demonstrate some of these ideas using the Modal Transition System Analyser, a tool that aims to support incremental elaboration of partial models [5] and that is available, open source, at `http://sourceforge.net/projects/mtsa/` . We finalise with a number of open problems and directions of future work.

Acknowledgments

Dario Fischbein, Nicolas D'Ippolito, German Sibay, Greg Brunet, Mathieu Sassoulas, Victor Braberman, and Marsha Chechik have all collaborated on parts of the work we present. The work we present has been funded in part by CONICET and grant ERC 204853/PBM.

References

1. A. Antonik, M. Huth, K. Larsen, U. Nyman, and A. Wasowski. EXPTIME-complete Decision Problems for Mixed and Modal Specifications. In *15th International Workshop on Expressiveness in Concurrency*, August 2008.
2. A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wasowski. Complexity of decision problems for mixed and modal specifications. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *LNCS*. Springer, 2008.
3. Y. Bontemps, P. Heymans, and P.-Y. Schobbens. From live sequence charts to state machines and back: A guided tour. *IEEE Transactions on Software Engineering*, 31(12):999–1014, 2005.
4. G. Brunet, M. Chechik, and S. Uchitel. Properties of behavioural model merging. In *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, volume 4085 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2006.
5. N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel. Mtsa: The modal transition system analyser. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*, pages 475–476. IEEE, 2008. Available at: `http://sourceforge.net/projects/mtsa/`.
6. A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings*, pages 193–202. IEEE Computer Society, 2008.
7. D. Fischbein, V. Braberman, and S. Uchitel. A sound observational semantics for modal transition systems. In M. Leucker and C. Morgan, editors, *Theoretical Aspects of Computing - ICTAC 2009, 6th International Colloquium, Kuala Lumpur, Malaysia, August 16-20, 2009. Proceedings*, Lecture Notes in Computer Science. Springer, 2009.
8. D. Fischbein and S. Uchitel. On correct and complete strong merging of partial behaviour models. In M. J. Harrold and G. C. Murphy, editors, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, pages 297–307. ACM, 2008.
9. D. Fischbein, S. Uchitel, and V. A. Braberman. A foundation for behavioural conformance in software product line architectures. In R. M. Hierons and H. Muccini, editors, *ROSATEA*, pages 39–48. ACM, 2006.
10. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.
11. M. Huth. Refinement is complete for implementations. *Formal Aspects of Computing*, 17(2):113–137, 2005.
12. H. Hüttel and K. G. Larsen. The use of static constructs in a modal process logic. In A. R. Meyer and M. A. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8,*

1989, Proceedings, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.

13. ITU. Recommendation z.120: Message sequence charts. *ITU*, 2000.
14. R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 1976.
15. I. Krka, Y. Brun, G. Edwards, and N. Medvidovic. Synthesizing partial component-level behavior models from system specifications. In H. van Vliet and V. Issarny, editors, *ESEC/SIGSOFT FSE*, pages 305–314. ACM, 2009.
16. K. G. Larsen, B. Steffen, and C. Weise. A constraint oriented proof methodology based on modal transition systems. In *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*. Springer, 1995.
17. K. G. Larsen and B. Thomsen. A modal process logic. In *Proceedings, Third Annual Symposium on Logic in Computer Science, 5-8 July 1988, Edinburgh, Scotland, UK*. IEEE Computer Society, 1988.
18. K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*, pages 108–117. IEEE Computer Society, 1990.
19. E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering Journal*, 15(2):175–206, 2008.
20. R. Milner. *Communication and Concurrency.* Prentice-Hall, New York, 1989.
21. G. Sibay, S. Uchitel, and V. A. Braberman. Existential live sequence charts revisited. In W. Schäfer, M. B. Dwyer, and V. Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 41–50. ACM, 2008.
22. S. Uchitel, G. Brunet, and M. Chechik. Synthesis of partial behavior models from properties and scenarios. *IEEE Transactions on Software Engineering*, 35(3):384–406, 2009.
23. S. Uchitel and M. Chechik. Merging partial behavioural models. In R. N. Taylor and M. B. Dwyer, editors, *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004, Newport Beach, CA, USA, October 31 - November 6, 2004*, pages 43–52. ACM, 2004.
24. S. Uchitel, J. Kramer, and J. Magee. "Incremental Elaboration of Scenario-Based Specifications and Behaviour Models using Implied Scenarios". *ACM TOSEM*, 13(1), 2004.
25. R. J. van Gabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.