Rigorous Development with UML-RT

Rodrigo Ramos[1], **Augusto Sampaio**[1], **Alexandre Mota**[1]

[1]Centre for Informatics, Federal University of Pernambuco
P.O.Box 7851, CEP 50740-540, Recife-PE, Brazil

{rtr,acas,acm}@cin.ufpe.br

Abstract. *With model-driven development being on the verge of becoming an industrial standard, systematic development strategies based on safe model transformations is a demand. However, the lack of a formal semantics makes the application of such transformations inadequate, especially transformations that must take into account changes in both behavioural and structural diagrams. As part of the Master thesis briefly summarised in this paper [Ramos, 2005], we have defined a semantics for UML-RT through a mapping into a formal notation, called Circus. Based on the semantics, we have proposed a set of transformation laws that aims to systematise the evolution of UML-RT models during development, with preservation of both static and dynamic aspects. Soundness and completeness of the laws are also addressed, and a case study has been developed to illustrate the overall approach.*

1. Introduction

Recently, Model Driven software Engineering (MDE) has influenced Software Engineering with a development process driven by the activity of modelling, as well as abstraction and automation of several tasks of a software development. Despite its strengths, the rigorous development of non-trivial applications (which need to emphasise the specification and verification of components) do not seem feasible without the assignment of formal semantics to the models. The reason is that well-known model transformations, which are important artifacts for model evolution in MDE, might not preserve behaviour.

In the literature, several efforts address the problem through the integration of object-oriented models with formal languages, proposing semantics and transformations for the Unified Modeling Language (UML), as well as its extensions. In our work [Ramos, 2005], we emphasise the use of UML-RT [Selic and Rumbaugh, 1998], a UML profile that has a clear definition for reactive components and protocols, and is useful to describe concurrent and distributed applications. Our approach mainly differentiate from related work (for instance, [Fischer et al., 2001, Engels et al., 2001]) for proposing a unified formal semantics for the several views of the model [Ramos et al., 2005] via mapping into *Circus* [Woodcock and Cavalcanti, 2002], a formal language that combines CSP and Z, and also support Morgan's refinement calculus. We also propose a comprehensive set of transformation laws, simultaneously taking in account the static and dynamic views of the model [Ramos et al., 2006]. These transformations are algebraically presented based on (and provably correct according to) the proposed semantics, focusing on the new elements that UML-RT adds to UML, and showing the necessary provisos for their application. Other works [Sandner, 2000, Engels et al., 2002] have taken in account architectural refactoring in UML-RT like ours. However, these works fail to make side conditions of laws explicit, and do not present a comprehensive set of algebraic laws, nor systematise a strategy for algebraic-based model transformations, as we have done.

In our approach, two groups of laws are identified: the first one embodies a comprehensive set of laws that govern small changes in the main model views, like introducing or removing a model element; the second group presents more elaborate laws derived from the composition of these basic laws, like decomposing a capsule into parallel component capsules. The derived laws can be considered as precise model refactorings that are easily applied in a rigorous development. The formalisation of these large grain transformations is built on ideas informally presented in [Sampaio et al., 2004].

The next section presents an overview of the UML-RT semantics and Section 3 briefly discuss the laws, considering soundness and completeness. The final section summarises our contributions.

2. UML-RT Semantics Overview

UML-RT, like other architectural description languages, models reactive systems with active architectural components working concurrently and communicating among themselves. Communication is modelled by means of input and output message exchange, which can be synchronous or asynchronous. These concepts have been introduced to UML-RT via four new design elements: capsule, protocol, port and connector. Capsules (active classes) describe architectural components whose unique points of interaction are called ports, which are assembled by connectors and realise communication signals previously declared in a protocol.

To clarify the main concepts of UML-RT, we illustrate the UML-RT notation and semantics using an example of a simplified industrial machine (Figure 1). This machine is responsible for receiving work pieces and, after processing, making them available for client requests. It is just a small component of major system, where it can be placed to form a larger architectural pattern, such as a pipeline. In the structure diagram Str_M (bottom of Figure 1), it is possible to see that a capsule *Machine* is actually composed of two other capsules instances *proc* and *son* of types *Processor* and *Storage*, respectively. In this specific diagram, *Machine* delegates all messages that it communicates through ports *mi* and *mo* to *proc* and *son*. Their types are presented in the class diagram Cls_M (top-left of Figure 1), showing the relationship among capsules, protocols, and classes. For instance, a relationship *son* between *Machine* and *Storage* represents that there is an instance *son* inside the structure of *Machine*, and a relationship *so* between *Storage* and the protocol *STO* represents a port *so* of this protocol type in *Storage*. Another important part of the system behaviour is described through statecharts (right-hand side of Figure 1). Via this diagram, protocols might establish specific sequences of interaction, such as in *STO*, and capsules might define how their reactive behaviour is triggered by external signals.

We assume that events, guards and actions in a statechart are expressed using the *Circus* notation. For example, in the statechart of Storage, there are two transitions from state Sa. The one on the right triggers if the req signal arrives through port so and the buffer is non-empty. The corresponding action declares a variable x to capture the result of the method remove. This is the way it is done in *Circus*, since remove is actually interpreted as a Z Schema. The value of x is then sent through port so. The syntax for writing these actions of communication are as in CSP.

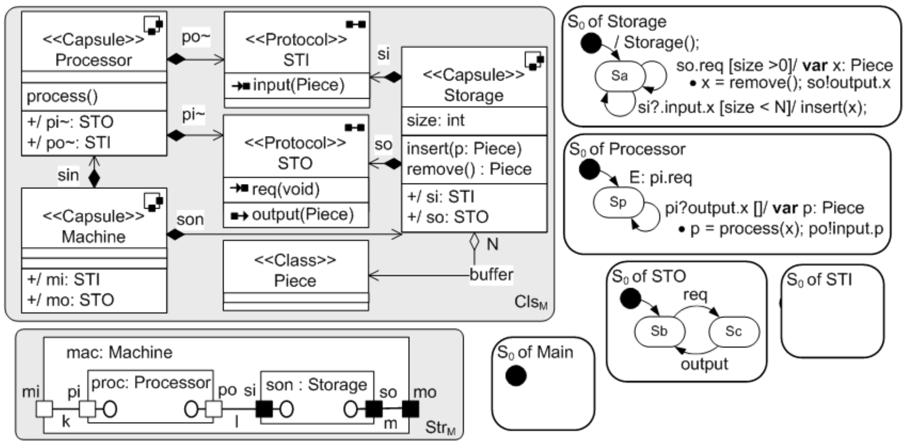


Figure 1. A simplified industrial machine

Language constructs similar to those of UML-RT concepts are also available in *Circus*, where concurrent components are represented by processes that interact via channels. Therefore, capsules and protocols are semantically mapped into processes, ports into channels, and classes into Z paragraphs, which act as passive data registers. Furthermore, connections are represented by means of using shared channels. As an example, consider the following mapping of the capsule Storage into *Circus*, which was obtained using a sequence of mapping rules presented in [Ramos, 2005, Ramos et al., 2005]. The process declaration body of Storage (delimited by the **begin** and **end** keywords) is composed of a Z state schema, action paragraphs and a main action (delimited after the • symbol), which defines the process behaviour; action paragraphs are used to structure the behaviour of a main action and to express data operations. Each method is a Z schema, specified with its pre- and postcondition.

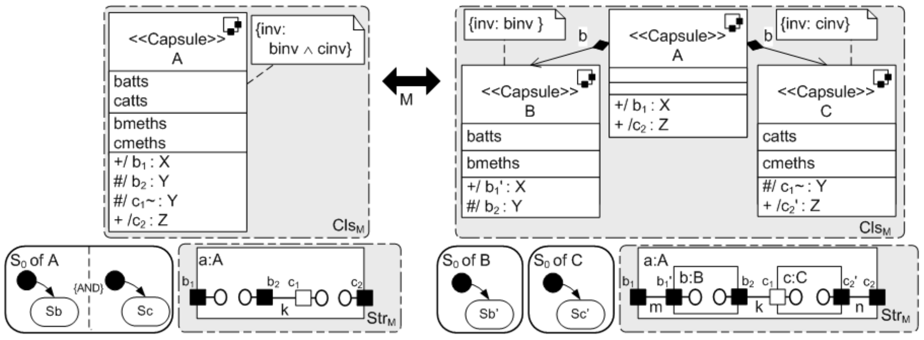
$| \; N : \mathbb{N}$
$T_{\mathsf{STI}} ::= \mathsf{input} \ll \mathsf{Piece} \gg$
$T_{\mathsf{STO}} ::= \mathsf{req} \mid \mathsf{output} \ll \mathsf{Piece} \gg$
$\mathbf{channel}\ \mathsf{si} : T_{\mathsf{STI}},\ \mathsf{so} : T_{\mathsf{STO}}$
$\mathbf{process}\ \mathsf{Storage} \mathrel{\widehat=} \mathbf{begin}$
$\quad \mathbf{state}\ \mathit{StorageState} \mathrel{\widehat=} [\mathsf{buffer} : \mathrm{seq}\,\mathsf{Piece};\ \mathsf{size} : 0..N \mid \mathsf{size} = \#\mathsf{buffer} \leq N]$
$\quad \mathbf{initial}\ \mathit{StorageInit} \mathrel{\widehat=} [\mathit{StorageState}' \mid \mathsf{buffer}' = \langle\rangle \land \mathsf{size}' = 0]$
$\quad \mathsf{insert} \mathrel{\widehat=} [\Delta \mathit{StorageState};\ x? : \mathsf{Piece} \mid \mathsf{size} < N \land$
$\qquad \mathsf{buffer}' = \mathsf{buffer} \frown \langle x? \rangle \land \mathsf{size}' = \mathsf{size} + 1]$
$\quad \mathsf{remove} \mathrel{\widehat=} [\Delta \mathit{StorageState};\ x! : \mathsf{Piece} \mid \mathsf{size} > 0 \land x! = \mathit{head}\,\mathsf{buffer} \land$
$\qquad \mathsf{buffer}' = \mathit{tail}\,\mathsf{buffer} \land \mathsf{size}' = \mathsf{size} - 1]$
$\quad \mathsf{Sa} \mathrel{\widehat=} (\mathsf{size} < N \mathbin{\&} \mathsf{si?input.x} \rightarrow \mathsf{insert};\ \mathsf{Sa})$
$\qquad \Box\ (\mathsf{size} > 0 \mathbin{\&} \mathsf{so.req} \rightarrow (\mathbf{var}\,\mathsf{x} : \mathsf{Piece} \bullet \mathsf{remove};\ \mathsf{so!output.x});\ \mathsf{Sa})$
$\bullet\ \mathit{StorageInit};\ \mathit{Sa}$
$\mathbf{end}$

3. Transformation Laws

Based on the formal semantics briefly discussed in the previous section, as well as on the refinement notions and laws of *Circus* [Sampaio et al., 2002], we propose some transformation laws for UML-RT. The laws deal with static and dynamic model aspects represented by the three most important diagrams of UML-RT: statechart, class and structure diagrams. The complete set of laws and our approach to their formal proofs can be found in [Ramos, 2005].

As illustration, we present an elaborate transformation law that decomposes a capsule A into parallel component capsules (B and C) in order to tackle design complexity and to potentially improve reuse.

Law 1 *Capsule Decomposition*



provided

($\rightarrow$) $\langle \mathsf{batts}, \mathsf{binv}, \mathsf{bmeths}, (\mathsf{b}_1, \mathsf{b}_2), \mathsf{Sb} \rangle$ *and* $\langle \mathsf{catts}, \mathsf{cinv}, \mathsf{cmeths}, (\mathsf{c}_1, \mathsf{c}_2), \mathsf{Sc} \rangle$ *partition* A.

($\leftrightarrow$) *The statecharts of the protocols* X *and* Z *are deterministic.*

Protocols X and Z are not illustrated because any deterministic machine can be used. On the left-hand side of Law 1, the side condition requires that A must be partitioned, a concept formalised in [Ramos, 2005]. Informally, the capsule must be formed by two disjoint groups of variables, methods, ports and statechart regions, which communicates only by internal ports of the capsule; each partition is forbidden to access methods and variables of the other part.

The effect of the decomposition is to create two new component capsules, B and C, one for each partition, and redesign the original capsule A to act as a mediator. The external signature and observable behaviour of A is preserve. In general, the new internal behaviour of A might depend on the particular form of decomposition. Law 1 captures a parallel decomposition. On the right-hand side of the law, A has no state machine. It completely delegates its original behaviour to an instance of B and C through the structure diagram. When A is created, it automatically creates the instances of B and C, which execute concurrently and play the same roles of the partitions that originated them.

Completeness The comprehensiveness of a set of algebraic laws is usually studied through a normal form reduction process. In our case, the expressiveness of our set of laws can be asserted by a reduction strategy that transforms an arbitrary UML-RT models into a UML model extended with a single capsule responsible for all the interactions with the environment; this capsule is also responsible for maintaining the active behaviour of the entire modeled system. In [Ramos, 2005, Ramos et al., 2006], we show that, collectively, our set of laws is power enough to carry out this reduction. This target UML model can be regarded as a *normal form*, and, therefore, our strategy can be regarded as a contribution to a completeness strategy captured by *normal form* reduction.

Case Study The focus of our approach is to support a formal transition from analysis into design. The transformation laws we have proposed may be useful to formalise informal analysis and design guidelines widely adopted by development processes as, for instance, the Rational Unified Process (RUP). The analysis and design disciplines of RUP include several activities that, broadly, identify abstractions, develop an abstract (analysis) model, and progressively refine it into a concrete design model. This is illustrated through the development of two case studies: a simple manufacturing system and a simple operational system [Ramos, 2005, Ramos et al., 2006, Sampaio et al., 2004].

4. Summary of Contributions

Our contributions can be summarised as follows.

- Assignment of a formal semantics for UML-RT via mapping into *Circus*, which acts as a useful hidden formalism for software engineering practise.
- Proposition and proof of a comprehensive set of basic algebraic laws for UML-RT that preserve the system behaviour on both static and dynamic views.
- Presentation of larger grain laws, derived from basic laws, that might be considered as precise model refactorings and easily applied in a rigorous development.
- A notion of relative completeness, briefly presented through a strategy of reducing an arbitrary UML-RT model to a UML model, entirely based on the laws.
- Seamless application of the laws through design activities of the Rational Unified Process in the development of a case study.

As far as we are aware, an entirely formal approach to model transformations using UML-RT, including a unified semantic mapping, is an original contribution.

References

Engels, G., Heckel, R., Küster, J. M., and Groenewegen, L. (2002). Consistency-Preserving Model Evolution Through Transformations. In *Proc. of the UML Conference*, volume 2460 of *Lecture Notes in Computer Science*, pages 212–226. Springer-Verlag.

Engels, G., Küster, J. M., Heckel, R., and Groenewegen, L. (2001). A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *Proc.of the ACM ESEC Conference*, pages 186–195. ACM Press.

Fischer, C., Olderog, E.-R., and Wehrheim, H. (2001). A CSP View on UML-RT Structure Diagrams. In *Proc. of the FASE Conference*, pages 91–108. Springer-Verlag.

Ramos, R. (2005). Desenvolvimento Rigoroso com UML-RT. Master's thesis, Federal University of Pernambuco, Recife, Brazil.

Ramos, R., Sampaio, A., and Mota, A. (2005). A Semantics for UML-RT Active Classes via Mapping into *Circus*. In *Proc. of the 7th IFIP FMOODS Conference*, volume 3535 of *Lecture Notes in Computer Science*, pages 99–114. Springer-Verlag.

Ramos, R., Sampaio, A., and Mota, A. (2006). Transformation Laws for UML-RT. In *Proc. of the 8th IFIP FMOODS Conference*, Italy. To appear in Lecture Notes in Computer Science.

Sampaio, A., Mota, A., and Ramos, R. (2004). Class and Capsule Refinement in UML For Real Time. In *Proc. WMF'03*, volume 95 of *ENTCS*, pages 23–51. Elsevier.

Sampaio, A., Woodcock, J., and Cavalcanti, A. (2002). Refinement in *Circus*. In *Proc. of the FME Symposium*, volume 2391 of *Lecture Notes in Computer Science*, pages 451–470. Springer.

Sandner, R. (2000). Developing Distributed Systems Step By Step With UML-RT. In *Proc. of the VVVNS Workshop*. Universität Münster.

Selic, B. and Rumbaugh, J. (1998). Using UML For Modeling Complex RealTime Systems. Rational Software Corporation. available at http://www. rational.com.

Woodcock, J. and Cavalcanti, A. (2002). Semantics of *circus*, the. In Bert, D., Bowen, J. P., Henson, M. C., and Robinson, K., editors, *ZB 2002: Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer-Verlag.