Metro: An Analysis Toolkit for Template Semantics

Joanne M. Atlee[†], Nancy A. Day[†], Jianwei Niu[‡],
Eunsuk Kang[†], Yun Lu[†], David Fung[†], Leonard Wong[†]

Technical Report CS-2006-34
David R. Cheriton School of Computer Science
University of Waterloo

[†]School of Computer Science
University of Waterloo
Waterloo, Ontario
{jmatlee@,nday@,ekang@engmail,y4lu@,
d3fung@student,lkhwong@}.uwaterloo.ca

[‡]Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas
niu@cs.utsa.edu

22 September 2006

Abstract

We describe the Metro toolkit, which supports software modelling and analysis for requirements notations that have configurable semantics. Metro is based on a formalism, called template semantics, which structures the operational semantics of a family of notations as a predefined parameterized template that is instantiated with user-provided parameter values. Thus, the semantics of a single notation can be expressed succinctly as a set of parameter values to this template. The Metro toolkit takes as input a specification and a set of template-parameter values, and it produces an analyzable model. The toolkit can either translate the specification into the input language of an existing model checker (e.g., SMV), or compile the specification into a more primitive form (e.g., logic, BDDs) that is suitable for analysis. MagicDraw is used as a front-end for editing specifications and animating SMV-generated counterexamples.

1 Introduction

The Metro[1] project at the University of Waterloo is investigating methods and tools for modelling and analyzing software requirements that are written in notations whose semantics are configurable. The project is motivated by the observation that modelling notations evolve with use, and that seemingly small changes to a notation's semantics can result in time-consuming changes to associated analysis tools. Our goal is to develop a modelling environment in which users can configure the semantics of their software models and still have access to automated verification tools. Such an environment would support domain-specific and propriety modelling notations.

The Metro toolkit is a prototype of such a modelling environment. Metro uses **template semantics** [16] to support configurable semantics. In template semantics, a model is the composition of one or

[1]A metro is an environmentally friendly system for rapid transit between disparate places. By analogy, our goal is to ease the transit between specification notations and verification environments.

more nonconcurrent, **hierarchical state-transition systems (HTSs)**. Concurrency, synchronization, and communication among HTSs are achieved via compositions operators. The model's semantics is specified using a combination of predefined parameterized definitions (the **template**) and a set of parameters that customize the template. **Template parameters** encapsulate the semantics decisions that we have found are most likely to vary among modelling notations, such as

- Which of the generated events can enable transitions
- Which variable values are used to evaluate transition guards and assignment expressions
- How transitions are prioritized

We have used template semantics to express the semantics of a variety of model-based notations [17], including process algebras, statecharts [8] variants, and specification notations such as SCR [10] and parts of SDL [11]. The Metro toolkit is parameterized with the template-semantics parameters, so its tools conform to a model's specified semantics. As such, the toolkit provides formal-analysis support for a family of model-based requirements notations.

As with any attempt toward generality, there is a potential loss of efficiency in the analysis. We are currently exploring two approaches to providing semantically configurable analysis. In one approach, Metro produces an analyzable representation of the model's next-state relation. We call this approach **model compilation**, because it expands all definitions (of the template, the template-parameter values, and the model) into a predicate logic expression, which can be easily transformed into other analyzable representations, such as BDDs [3]. For most template parameters, model compilation can accept any predicate-logic expression as a parameter value because the expression is simply combined with the other logical definitions and expanded. In the second approach, Metro translates the original model directly into the input language of an existing model checker, such as SMV [14]. The translation approach works with a fixed set of template-parameter values. However, unlike in model compilation, the translation approach may be able to preserve the structure of the original model, which makes it easier to employ state-space optimizations.

In Section 2, we provide a brief overview of template semantics. Section 3 discusses the components of the toolkit, and Section 4 briefly overviews our experiences using the Metro toolkit.

2 Template Semantics

In this section, we briefly describe the syntax, execution semantics (parameterized by the template parameters), and composition operators used in template semantics. More details can be found in [16].

2.1 Syntax

A model is a collection of nonconcurrent, hierarchical transition systems (HTSs). An HTS is an extended state-machine, with a set of hierarchical control states, a set of transitions between control states, a set of events, and a set of typed variables. A transition is enabled by events and/or conditions. The execution of an enabled transition transforms an HTS from one set of states into another set of states, generates new events, and assigns new values to variables. Concurrency, synchronization, and communication among HTSs are achieved via compositions operators.

2.2 Execution Semantics

The execution semantics of an HTS is defined in terms of allowable sequences of snapshots. A **snapshot** is a collection of information about an HTS's execution. Some snapshot elements reflect the current status of the HTS's execution, such as the current states, current events, and current variable values. Other snapshot elements store auxiliary data about states, events, and variables; these data are used to calculate allowable next snapshots. Examples of auxiliary data include states that enable transitions, past values of variables, and events that have triggered executed transitions.

An **execution step** moves an HTS from one snapshot to a successor snapshot. We use template semantics to specify the notion of an allowable execution step. Template semantics supports two types of steps: a

micro-step reflects the execution of a single transition, and a **macro-step** is a sequence of zero of more micro-steps. A macro-step always starts by sensing new inputs to the model. Notations differ on the conditions for starting a new macro-step (i.e., they differ on when to sense new inputs). Some notations sense new inputs after every micro-step; other notations wait until the HTS has reached a **stable snapshot**, in which no more transitions are enabled, before starting the next macro-step.

Template semantics structure the operational semantics of a family of notations into pre-defined definitions (the template) of common behaviour. The template includes parameterized definitions of *micro-step*, *macro-step*, and *stable snapshot*. Template parameters are the variation points in the template. Most parameters specify what information is stored in the snapshot elements, how snapshot data are used to identify enabled transitions, and how snapshot data are updated when a transition executes. Other template parameters specify when new inputs are sensed, how enabled transitions are prioritized, and what the model outputs.

2.3 Composition Operators

HTSs can be combined by composition operators to form **composed hierarchical transition systems (CHTSs)**. The template semantics of composition operators defines how multiple HTSs execute concurrently, how they communicate by exchanging events and data, and how they synchronize with each other. The definitions of these operators use the template parameters to ensure that the semantics of composition is consistent with the components' execution semantics. Metro supports variations of seven composition operators: parallel, interleaving, environmental synchronization, rendezvous, sequence, choice, and interrupt. These composition operators can be arbitrarily nested.

2.4 Simulating Other Modelling Notations

To use template semantics to describe an existing modelling notation, the user must

1. Map the notation's syntax to the syntax of HTSs. Because HTSs provide many features of state-based modelling notations, this step is usually straightforward.
2. Specify the notation's semantics as template-parameter values.
3. Map each of the notation's composition operators to one of the composition operators supported by Metro.

3 Tool Flow

In this section, we describe the components of the Metro toolkit, illustrated in Figure 1. Work in progress is indicated by dashed lines.

3.1 Specification Editor

The input to our toolkit is an XML representation of a CHTS model along with template-parameter values describing the semantics of the model's notation. We created this XML format to support interoperability among front-end tools. To make it easier to create these models, we use a UML editor called MagicDraw[2] and we created a plugin for MagicDraw that outputs the model in the XML format. There is a natural mapping between most CHTS constructs and MagicDraw's UML syntax. Composition operators are expressed using the UML concurrent-state operator, with the type of composition indicated in the operator's associated textual information.

Rather than write our own typechecker, we deeply embedded the CHTS syntactic constructs in the higher-order logic of the HOL theorem prover [7]. Metro parses a model's XML representation and converts it into CHTS terms in the HOL logic, which HOL then typechecks. Because higher-order logic is the

[2]MagicDraw is a trademark of No Magic Inc.

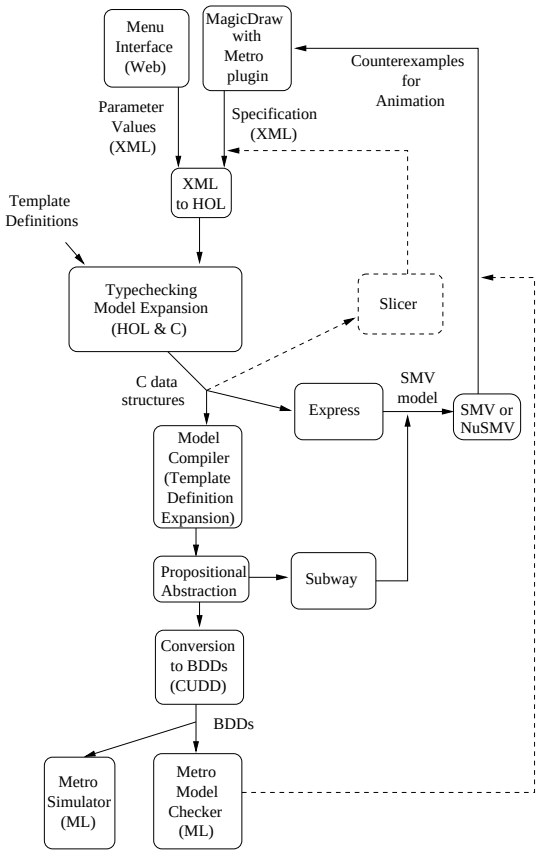

Figure 1: Components of the Metro Toolkit

underlying substrate, the user can parameterize the specification (e.g., for replicated components) and can introduce functions (similar to macros) to abbreviate the specification. Metro expands these abbreviations, in a phase we call **model expansion**, to produce an unabbreviated model. Most importantly, embedding CHTSs in higher-order logic means that the Metro modelling notation is easily extensible (e.g., more complex expression languages), and that we can do theorem-proving analysis in the future.

3.2 Semantics Input

Template-parameter values specify the semantics of the modelling notation. The user can either specify parameter values as formulae in higher-order logic, or select from a menu of predefined parameter values. Many of the values are set-based expressions that refer to elements of the snapshot.

3.3 Translation

Given an unabbreviated model and its semantics, expressed as template-parameter values, Metro can translate the model into its equivalent representation in the input language of an existing analyzer. A translator can optimize models for the target analyzer, but it can support only a fixed set of template-parameter values. We have implemented one translator, called Express [12], which translates Metro models into SMV models.

3.4 Model Compiler

Alternatively, Metro can **compile** a specification into a more primitive representation. We have written the template-semantics' template as definitions in higher-order logic. During model compilation, Metro expands these definitions for a particular CHTS with the user-provided template-parameter values, to produce a predicate-logic expression[3] of the model's transition relation (from snapshot to allowable successor snapshot). This compilation is performed using a method called **symbolic functional evaluation (SFE)** [6]. SFE can expand any term expressed in higher-order logic, so users can create ad-hoc template-parameter values in higher-order logic. For efficiency, SFE stores and manipulates terms as pointer-based data structures written in C; the data structures share terms, such that replicated terms are stored and expanded exactly once.

3.5 Analysis of Compiled Models

We have implemented two methods for model checking a compiled model. In one approach, Metro abstracts the compiled model to propositional logic by representing each unique atomic sentence in the predicate-logic expression as a distinct Boolean variable. The resulting expression is converted to a BDD in the CUDD BDD package [19]. We have built a basic model checker in ML, and have created an ML foreign-function interface to CUDD (written in C). A simulator, also written in ML, uses the BDD representation to generate the next possible snapshots. We have found that the combined ML and C programming environment makes it easier to develop analysis algorithms and still produce efficient code.

In the second approach, which we call Subway, Metro abstracts the compiled model to propositional logic, as described above, and translates the resulting expression into SMV. The SMV model consists primarily of a very large TRANS section and is not readable. However, this approach has the advantage of using a more advanced model checker with built-in optimizations.

3.6 Counterexample Animator

Counterexamples that are produced by NuSMV [4] for Metro-generated SMV models can be animated in MagicDraw using our plugin. The animation steps through the counterexample, highlighting in each step of the scenario the current states of the model's execution.

3.7 Future Components

We are currently investigating how to apply model-checking optimizations to models that have configurable semantics. For example, we are creating a slicer that operates on CHTS models and abstracts away information that is irrelevant for a particular query. We also plan to investigate translations to other analyzers and theorem provers.

4 Experience

So far, Metro has been used only in-house. Metro researchers have used Metro, SMV, and NuSMV to build and model check several variations of two specifications: a heating system [5] and a single-lane bridge [13]. Both specifications are relatively small, and were built primarily to exercise all of the composition operators that Metro currently supports. In addition, students taking a graduate course on computer-aided verification used Metro and SMV to model and verify specifications of elevators, the Kaiserslautern light control system [2], and a hotel-room safe. In most cases, the chosen semantics of a specification deviate slightly from the classic semantics of a popular notation. For example, many specifications have a statecharts-like semantics, but use rendezvous for a critical synchronization. Other specifications have a CCS-like semantics, but use global shared variables (c.f., data-passing CCS [15], which uses local variables and data-passing events).

[3]The result is in predicate logic because all higher-order features are expanded.

The specification of the Kaiserslauten lighting system is based on the SCR version of this problem [9] and simulates SCR semantics [10].

To evaluate how well the components of Metro scale to larger specifications, we are working on a template-semantics version of the A-7E specification [1]. We plan to compare the generated SMV models against each other and against a manually created model; the basis of the comparison will be the time it takes to model check the specification's mode-combination properties. In a separate project, we are working on the template semantics for BoxTalk [20], which is a model-based programming language developed by AT&T for programming CallVantage® features.

5 Summary

The Metro toolkit is a prototype modelling and analysis environment that supports specification notations whose semantics are configurable. Metro uses MagicDraw as a frontend for editing graphical models, and supports an XML textual representation of the model. Additional features of the Metro toolkit are

- Support for multiple approaches to model checking, including translation and model compilation
- General-purpose definition expansion, in which any part of the model can be abbreviated or parameterized
- Extensibility to modelling-notation features that can be represented in higher-order logic, and to additional analysis tools

Many of the tools in the Metro toolkit (the model expander, model compiler, Express translator) can run as web services and have web interfaces. This means that we will be able to make Metro available to others over the Internet. For example, users would be able to enter a model's XML representation and its semantics, in terms of template-parameter values, and use Express to generate SMV code.

In the future, we plan to work on additional case studies, compilation to different target analyzers, implementations of model-checking optimizations, and methodology. We are also working on a code generator that is configurable with respect to the template-semantics parameters [18].

References

[1] T. Alspaugh et al. Software Requirements for the A-7E Aircraft. Technical report, Naval Research Laboratory, March 1988.

[2] E. Börger and R. Gotzhein. J.UCS Special Issue on Requirements Engineering - The Light Control Case Study. *Jour. of Universal Comp. Sci.*, 6(7):580–581, July 2000.

[3] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, August 1986.

[4] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *Int. Journal on Soft. Tools for Technology Transfer*, 2(4):410–425, 2000.

[5] N. Day. *A Framework for Multi-Notation, Model-Oriented Requirements Analysis.* PhD thesis, University of British Columbia, October 1998.

[6] N. A. Day and J. J. Joyce. Symbolic functional evaluation. In *TPHOLs*, volume 1690 of *LNCS*, pages 341–358. Springer, 1999.

[7] M. Gordon and T. Melham, editors. *Introduction to HOL.* Cambridge University Press, 1993.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. of Comp. Prog.*, 8:231–274, 1987.

[9] C. Heitmeyer and R. Bharadwaj. Applying the SCR requirements method to the light control case study. *Jour. of Universal Comp. Sci*, 6(7):650–678, July 2000.

[10] C. L. Heitmeyer and R. D. Jeffords. The SCR tabular notation: A formal foundation. Technical Report NLR/MR/5546-03-8678, Naval Research Lab, 2003.

[11] ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union - Standardization Sector, 1999.

[12] Y. Lu, J. M. Atlee, N. A. Day, and J. Niu. Mapping template semantics to SMV. In *ASE*, pages 320–325. IEEE Computer Society, 2004.

[13] J. Magee and J. Kramer. *Concurrency, State Models & Java Programs*. John Wiley & Sons, UK, 1999.

[14] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.

[15] R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[16] J. Niu, J. M. Atlee, and N. A. Day. Template semantics for model-based notations. *IEEE Trans. on Soft. Eng.*, 20(10):866–882, Oct. 2003.

[17] J. Niu, J. M. Atlee, and N. A. Day. Understanding and comparing model-based specification notations. In *RE*, pages 188–199. IEEE Comp. Soc. Press, Washington D.C. 2003.

[18] A. Prout, J. M. Atlee, and N. A. Day. Configurable model-driven development. *(submitted to ICSE'06)*, September 2005.

[19] F. Somenzi. CUDD: CU decision diagram package, 2005.

[20] P. Zave and M. Jackson. A call abstraction for component coordination. In *Int. Coll. on Automata, Languages, and Programming: Workshop on Formal Methods and Component Interaction*, June 2002.

Demonstration

In this demonstration of the Metro toolkit, we will use Express to show how different template-parameter choices affect the behaviour of a model. We will use the simple specification of cars crossing a single-lane bridge, based on the example found in [13]. We choose to demonstrate this model because it uses a variety of composition operators (environmental and rendezvous synchronization, and interleaving). A part of this specification, drawn in MagicDraw, is given in Figure 2. Using our plugin for MagicDraw, the model is turned into an XML representation (Figure 3.)

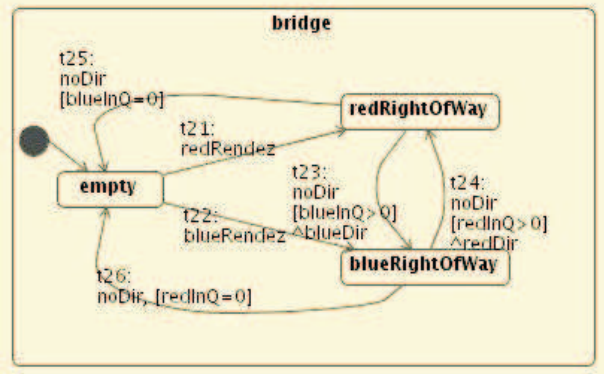

Figure 2: Part of Bridge Specification

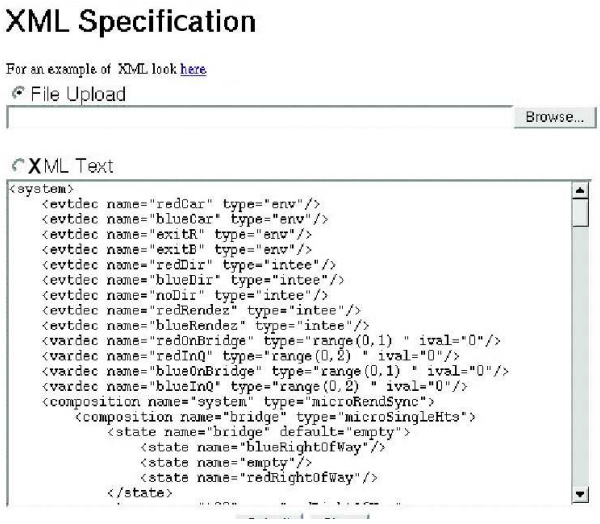

Figure 3: XML of Bridge Specification

The user specifies the model's semantics as a set of template-parameter values. Express supports a fixed set of parameter values, which can be chosen from a web-based menu. Via the menu interface, the user can select the parameter values for a particular notation (as shown at the top of Figure 4), or the user can select from drop-down menus for individual template parameters (Figure 4).

Express translates the model to SMV for input to the NuSMV model checker. Figure 5 shows a counterex-

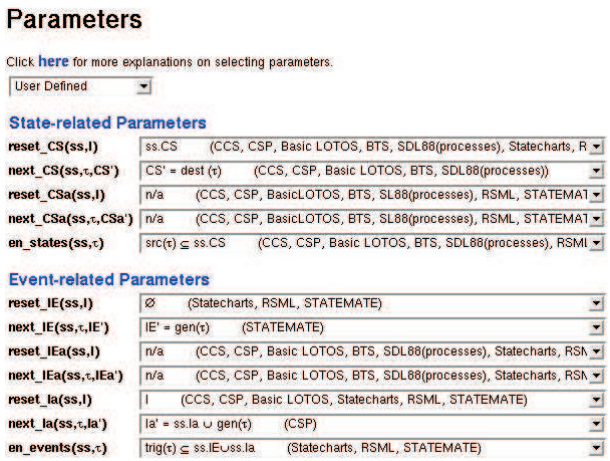

Figure 4: Entry of Template Parameter Values

ample from NuSMV. The variable names are prefixed to indicate snapshot elements (e.g., "pss" indicates previous snapshot and "CS" refers to the control states snapshot element).

In the research-tool demonstration, we will explain why problems are sometimes easier to model in notation variants than in classic modelling notations. We will show how easy it is to create new modelling notations and variants, via template-semantics parameters, and how the Metro toolkit provides analysis support for these new notations.

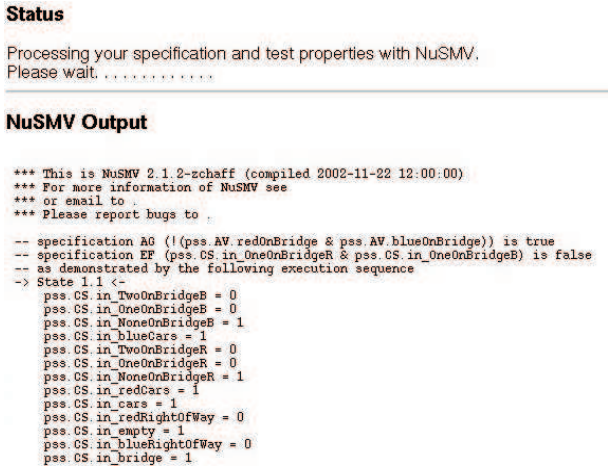

Figure 5: NuSMV Output