Certification of Transformation Algorithms in Model-Driven Software Development

Miguel Garcia, Ralf Möller

Institute for Software Systems (STS)
Hamburg University of Technology (TUHH), D-21073 Hamburg
{miguel.garcia, r.f.moeller}@tuhh.de

Abstract: The increasing reliance on Model-Driven Software Development calls for model compilers to assume the role of today's compilers, i.e., reliability of these components is of utmost importance. We describe how to certify model transformations in this context by bridging the gap between the languages in which such transformations are specified (e.g., Essential MOF, OCL, OO programs) and the decision procedures needed to verify properties expected of such transformations. Two major aspects are investigated in this paper: (i) valid output is obtained for each valid input, (ii) the output satisfies certain properties. Results from application projects validate our approach, which internally applies model-driven techniques to the certification process itself by mapping transformation specifications into the $^+$CAL model-checking language.

1 Introduction

Model-Driven Software Development (MDSD) is gaining consensus in the Software Engineering community as a viable technology to improve both productivity and quality. Metamodel-based definitions of Domain Specific Languages (DSLs) are routinely leveraged to automatically derive language processing tools (syntax-aware editors [JBK06], diagram editors [Ehr05], or software repositories [APM03]) using techniques that build upon a common infrastructure such as the Eclipse Modeling Framework (EMF) [Bud03]. The sketched productivity gains in jumpstarting a toolchain for MDSD can be traced back to the specification of the static semantics (also called well-formedness rules, WFRs, or validity-checking rules) of DSLs in terms of declarative OCL invariants [WK03] over a language metamodel expressed in Essential MOF (EMOF) [OMG06]. Such metamodels contain a wealth of machine-processable information, relieving tool implementors from manually hardwiring such specs into language processing tools (e.g., in the semantic analysis phase of a dedicated DSL compiler).

In order for MDSD to deliver on its full potential, commensurate progress is required to increase the quality of emerging model transformers and compilers. A method and its associated tooling is presented in this paper to reach that goal for a representative class of model transformations. In particular, we investigate languages for which an EMOF + OCL metamodel is available. This method involves the automatic translation of the input and output metamodels into a formalism for which a decision procedure is available to answer whether a given procedural transformation exhibits certain properties of interest. Two basic desirable guarantees for such tranformations are (a) that all output sentences belong to the target language [HZS05], and (b) that the transformation function covers the whole input language for which it was designed [WKC06]. Experiences with current model-driven tooling shows that these basic requirements are not always met. Beyond these general requirements, guarantees specific to a given transformation are also desirable. For example, an optimized implementation should produce the same result as the non-optimized version. Once these analyses have been performed, the follow-up problem is compiler correctness, i.e., ensuring that the transformations are semantics-preserving.

Nowadays, model compilers are in operation for various kinds of application tasks. For instance, transformations into Java Enterprise Edition (Java EE, defined in JSR-220) are very popular and include: DASL [Gol05], SecureUML [BDW06], and WebML [Cer06]. These DSLs allow for the specification of three-tier enterprise systems at a high level of abstraction and have metamodel-based descriptions. Similarly, platform-specific metamodels are available. For example, the persistent query language of Java EE (EJB3QL) has been metamodeled in [Gar06]: all the normative restrictions formulated in English in Ch. 4 of JSR-220 are recast as OCL invariants. Instantiations of the EJB3QL metamodel satisfying those invariants are valid abstract syntax trees (ASTs) for particular EJB3QL queries, and can be unparsed into text. Along the same lines, a complete metamodel of BPEL 1.1 is discussed in [Ake04].

In the current state of the practice, the WFRs on the input and output ASTs are evaluated at transformation-time, for each application of the transformation. We aim instead at certifying transformation algorithms at design-time, to make runtime-checks redundant, but moreover to get model compilers right early on, instead of patching them as new cases are discovered which were overlooked before widespread deployment.

As decision procedure for the task described above we adopt model-checking as described in [Lam06b]. A model-checker manipulates execution traces, which can be conceptualized as trees of states (each state also called a system snapshot), with root states derived to cover all initial conditions. Candidate successor states are computed from all actions enabled in the current state.

Once a transformation algorithm has been specified, at algorithm-design time the model-checker can detect situations where the transformation does not terminate, or terminates without establishing the properties of interest. The coverage achieved by the admittedly finite analysis of a model-checker is much higher than testing because (a) states can be manipulated symbolically, and (b) several properties of interest depend on the shape of an object graph rather than on its size or concrete attribute values (the "small scope hypothesis" [Jac06]). For example, the condition "two lines intersect" can be manipulated without considering concrete crossing points. Model-checkers can detect these situations, taking a single state as representative of all those exhibiting such shape.

In summary, the contribution of this paper is twofold. First, we eliminate the laborious task of preparing a model-checkable specification for nontrivial MDSD transformations by reusing the EMOF + OCL WFRs for the static semantics contained in the metamodels of the languages participating in the transformation. Second, we demonstrate how to support the development of robust transformation algorithms by employing a model-checking engine. In particular, we investigate the language $^{+}$CAL and use the corresponding model-checker TLC. The significance of the approach is demonstrated using an application scenario concerning a well-known graph transformation problem (Schorr-Waite). This problem involves an in-place transformation, which are known to be harder to analyze than functional transformations.

The structure of this paper is as follows. Sec. 2 elaborates on static semantics and derives set-theoretic definitions for EMOF + OCL metamodels. Afterwards, it is exemplified how OCL pre- and postconditions are mapped to $^{+}$CAL. This is followed (in Sec. 3) by a discussion of the steps to follow when applying the proposed model-checking-based certification method, including an analysis of a sample transformation. Alternative and complementary approaches to model-based verification are discussed in Sec. 4. Knowledge is assumed from the reader about MDSD, set-theory and logic. The prototype discussed in this paper and accompanying examples can be downloaded from [Gar07].

2 Formalization of Essential MOF + OCL for model-checking

$^{+}$CAL [Lam06b] is a specification language designed to replace pseudo-code for writing high-level descriptions of algorithms. A $^{+}$CAL algorithm manipulates mathematical objects in a series of steps. The granularity of a step is chosen by the algorithm designer, ranging from a single built-in statement to a composite step inolving several $^{+}$CAL statements. A step exhibits transaction semantics: intermediate results are not visible to concurrently executing processes and system invariants are required to hold only at step boundaries.

$^{+}$CAL includes control-flow statements typical from block-structured programming (if-then-else, while-do, sequential composition), as well as constructs for expressing non-deterministic and concurrent algorithms. For the purposes of this work, we focus on the sequential subset of $^{+}$CAL. Mathematical expressions and logical formulae may appear in $^{+}$CAL programs whenever a construct calls for a value (e.g., in the condition part of an if-then-else, in an assert statement). In fact, the properties an algorithm should exhibit are routinely expressed as mathematical assertions on the input and output data (in our case: ASTs), with the model-checker being able to compute them for finite system snapshots.

The proposed certification method comprises the following steps:

1. Automatic translation of the definitions contained in the participating EMOF + OCL metamodels into $^{+}$CAL (these definitions allow the transformation algorithm to refer to well-formed ASTs).

2. Expressing a model transformation as a $^{+}$CAL algorithm operating on ASTs.

3. If appropriate, annotating the transformation with assumptions about its input, beyond the constraints expressed in the metamodels of input ASTs. The invoker of the transformation is responsible for satisfying these assumptions.

4. Annotating the transformation with assurances about the system state (in particular about the output ASTs) after every successful run of the transformation on valid input (i.e. on well-formed input satisfying the assumptions made in the previous item). The algorithm is responsible for satisfying these assurances.

Following Design-By-Contract [MMS98], the assumptions in (3) are called *transformation preconditions* and the assurances in (4) *transformation postconditions*. This terminology partly overlaps with that of Hoare logic, where a postcondition fully specifies the transformation between the pre and post states (no procedural statements are necessary). The checks that Design-By-Contract performs at runtime can be carried out at transformation-design time thanks to model-checking. As to the language for pre- and postconditions, besides $^{+}$CAL some or all of them can be expressed in OCL, with our prototype taking care of their translation, as described in the next two subsections. Using $^{+}$CAL directly may however prove beneficial given the growing number of third-party libraries of mathematical theories defined in $^{+}$CAL.

2.1 Translating EMOF into $^{+}$CAL

The translation of a user-specified OO class model (expressed in EMOF) results in definitions of sets, relations, and $^{+}$CAL [Lam06b] procedures to allocate instances and manipulate links and attributes, constituting a certified building block that simplifies the expression of the transformation algorithm. Our encoding of EMOF + OCL into $^{+}$CAL leverages previous work on set-theoretic semantics of UML and OCL [BKS02]. In the context of UML, a thorough analysis of the logical consistency of the MOF 2.0 specification is also reported in [AS06]. Our work focuses on the more recent EMOF. Due to space restrictions we cannot present the transformation of all EMOF language constructs, but as an example for the main ideas Figure 2 shows the invariant for the bidirectionality constraint stated in the class model in Figure 1

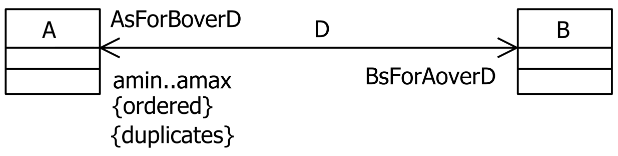

Figure 1: Sample EMOF model

In the example, there are only safety properties (the min-max bounds, lack of duplicates on the `BsForAoverD` association end, bidirectionality over `D`) which are trivial yet cumbersome to formalize by hand. In general, the generated specification contains logical predicates (such as *InvariantsDirectionality* in Figure 2) to check after every execution trace whether the constraints implicit in EMOF constructs and all OCL invariants are maintained. As a whole, these checks guarantee that the output sentence belongs to the output language, for all possible runs of the MDSD transformation algorithm.

```
for each instance $b$ of $B$, those instances of A
reachable over $D$ from it must in turn have $b$ among
their instances reachable over $D$

$InvariantBidirectionality\_AtoB \triangleq$
  $\lor\ AsForBoverD = \langle\rangle$
  $\lor\ \forall\, b \in$ DOMAIN $AsForBoverD$ :
        LET $AsReachable \triangleq AsForBoverD[b]$IN        a sequence
          $\forall\, i \in$ DOMAIN $AsReachable$ :
            LET $anA \triangleq AsReachable[i]$IN
              $b \in BsForAoverD[anA]$

counterpart of the above, this time for each a

$InvariantBidirectionality\_BtoA \triangleq$
  $\lor\ BsForAoverD = \langle\rangle$
  $\lor\ \forall\, a \in$ DOMAIN $BsForAoverD$ :
        LET $BsReachable \triangleq BsForAoverD[a]$IN        a set
          $\forall\, aB \in BsReachable$ :
            $ElemIsInSeq(a,\ AsForBoverD[aB])$

$InvariantsDirectionality \triangleq$  $\land\ InvariantBidirectionality\_AtoB$
                                       $\land\ InvariantBidirectionality\_BtoA$
```

Figure 2: Bidirectionality expressed in $^{+}$CAL

2.2 Translating OCL into $^{+}$CAL

OCL prescribes that, at the end of a transaction, each system snapshot should fulfill the specified invariants. Lacking transaction demarcation, it is generally agreed that invariants in an OO program should hold after object construction and after the execution of each public operation. The concept of transaction boundary is directly supported by $^{+}$CAL step granularity. The *efficient* evaluation of invariants can be challenging. Usually an operation involves a small number of updates which leave most invariants unaffected. Being $^{+}$CAL based on a temporal logic, there is no shortage of expressivity to encode OCL invariants, with the model-checker evaluating them in the background as execution traces are considered. Model-checkers can take into account the data-flow dependencies of formulae so as to approach non-redundant yet complete evaluation. This shortens the time elapsed from submitting an algorithm till counterexamples are found, thus increasing the productivity of the transformation designer. As already mentioned at the beginning of this section, OCL pre- and postconditions are translated as assertions into $^{+}$CAL, i.e., they are no substitute for the specification of the input-output transformation, which must be given as imperative statements.

The conversion from OCL to $^{+}$CAL is performed by visitors over ASTs of OCL expressions, similar to the work of [BKS02]. A metamodel of $^{+}$CAL was prepared for this purpose. This AST-to-AST conversion could in principle be verified with the techniques described in this paper. An example of bootstraping the verification of a transformation is offered by the algorithm to translate from $^{+}$CAL to TLA^{+} (Temporal Logic of Actions [Lam02], the logic underlying $^{+}$CAL), which is itself specified in TLA^{+}.

3 Certification process

Certification of an algorithm is an iterative process. Whenever it can be shown at design time that an algorithm is bound to fail at runtime (i.e., for some inputs does not terminate, breaks metamodel invariants, or does not fulfill its part of the contract in establishing postconditions) the model-checker not only indicates failure but presents an execution trace leading to that situation (a counterexample). The algorithm designer may apply a combination of (i) reformulating the algorithm to handle the situation that caused the failure, (ii) strengthening the preconditions (making the algorithm applicable to a subset smaller than well-formed ASTs), or (iii) weakening the postconditions. The practical limit to postcondition weakening is that the output must still be well-formed, as demanded by metamodel invariants.

3.1 Directly specifying transformation algorithms in terms of EMOF

As a further means to increase certification productivity, a textual syntax for an object-oriented programming language ("Executable EMOF", or xEMOF for short) could define statements to manipulate instances of metamodels, as a high-level notation to express model transformation algorithms. The rationale for this is the large number of transformations already expressed in terms of the Visitor design pattern [Gam98]. In a green field scenario, a language such as xEMOF would also prove useful by reducing the conceptual distance between a transformation algorithm and its implementation, thus shortening the certification process.

A language such as xEMOF is not just a thin layer of syntactic sugar over $^{+}$CAL. Instead, the translation is non-trivial because (a) method-dispatch in an object-oriented language depends on runtime-types instead of only declared types, (b) method overloading similarly complicates method selection, and (c) the interplay between inheritance and object initialization has to be taken into account. A more realistic modeling of Java, including for examples exceptions (which introduce alternative return paths and require bookkeeping to correctly unfold the activation frames in the call-stack) would not add expressive power yet complicate the translator.

Besides those transformations expressed in terms of visitors, another large group of existing transformations relies on pattern-matching mechanisms followed by in-place transformations (ATL [JK06], QVT [OMG05], graph-grammars [Ame06]). Provided that they manipulate EMOF-based ASTs, their execution engines can similarly be formulated in terms of $^{+}$CAL reusing the translation performed by our prototype (see Sec. 2).

3.2 Certifying a non-trivial in-place transformation: Schorr-Waite

As to the readability of $^{+}$CAL, Figure 3 shows the encoding of the Schorr-Waite graph-marking algorithm. It is complex enough to serve as a reference case in source code verification [HM05, MN05], as it involves modifying in-place an AST-like pointer-rich data structure, yet intuitive enough that its operation can be explained succinctly. To our knowledge, this is the first account on model-checking Schorr-Waite. Although we manually crafted in $^+$CAL the Schorr-Waite algorithm, the discussion in the previous subsection and also, e.g., the results of [BB06] indicate that this manual effort can be reduced if not eliminated in the near future.

The Schorr-Waite algorithm performs a depth-first traversal of a directed graph, starting from a specific node called the root. Given that memory is at a premium during garbage collection, Schorr-Waite offers a constant upper bound on memory usage by avoiding keeping a stack with the nodes in the current path. Instead, as new nodes are visited, the link that was followed last is reversed in place. Upon going back along the current path, the algorithm reconstructs the original topology. Pointer reversal avoids thus the introduction of a stack.

The correctness guarantees we expect are: (a) termination, (b) that all nodes reachable from the root (and only those) are marked, and (c) that the algorithm leaves the topology unchanged. These guarantees are encoded as assertions: the results computed by the implementation are compared with those resulting from mathematical definitions which are executable in $^+$CAL. Two assertions are found at the end of Figure 3: *Topology(pointsTo)* returns (as a set of edges) the reachability information which was updated in-place, for comparison with the original topology (a discrepancy would result in a counterexample). Similary, the last *assert* compares the set of reachable nodes (as computed by Schorr-Waite, i.e. those nodes having *markBit* set to true) with the set obtained as the transitive closure of the *g.edge* (mathematical) relationship applied to the root. The complete source code can be downloaded from [Gar07].

We model-checked the $^+$CAL algorithm in Figure 3 considering graphs with up to 10 nodes, finding that acceptable runtimes can be achieved (see also [Lam06a] for the empirical behavior of TLC). Thus, we conclude that the proposed method is practically significant for MDSD. As mentioned before, due to the small scope hypothesis [Jac06], many, if not all, problems will be found using this problem size.

4 Related work: Alternative and complementary approaches

The method reported in this paper allows for the validation of model transformation algorithms. If implemented carefully, the assertions made for an algorithm carry over to its implementation. This manual coding step in a language such as Java introduces the possibility of a non-conforming implementation. We argue that validation of transformation algorithms is still necessary: a faulty transformation algorithm, however correctly implemented, will not improve quality. In addition, metamodel-based approaches (involving OCL specifications) allow for a higher expressivity of constraints to be validated than, e.g., approaches based on XML-Schema.

A straightforward solution to the manual implementation problem consists in devising a translator from the language in which the transformation was certified ($^+$CAL or xEMOF) into Java, and certifying this translator. Assuming that verification of a (manually or automatically derived) Java implementation is required, the OCL invariants contained in the metamodel can still be reused by translating them to JML [Bur05] as discussed in [Ham04]. As with all source-code level verification approaches, a larger state space has to be explored, thus reaching practical limits more quickly than for the model-level counterpart as a result of the faithful representation of Java Virtual Machine abstractions. For example, expressions in Java may have side-effects while OCL expressions are guaranteed to be read-only. Adopting EMOF models as the only mechanism to define state allows us to consider only those state evolutions allowed by EMOF, reducing the state space to explore. This is in tune with the principle of reasoning at the highest-level of abstraction possible, because it's more efficient.

The application of TLA$^+$ has been investigated also in other software engineering contexts. In the field of enterprise software architectures, model-checking of web service protocols is reported in [Joh04]. An Eclipse-based text editor to support editing $^+$CAL and TLA$^+$ specifications is described in [GVZ05].

Expressing behavior at the level of object-oriented models is also the aim of approaches under the Executable UML umbrella [Rai04]. Most products in this category target the embedded systems or telecommunications market and are heavily focused on statecharts.

Another formalization of OCL with tool support for verification is KeY [BHS07] which targets different verification use cases from the ones addressed in this paper. Its execution language is JavaCard, with both JML and a dedicated Dynamic Logic as verification backends. KeY is used in tandem with a commercial UML tool.

Brucker et. al. [BW06a] describe in detail a tool to transform UML+OCL into a formalization processable by a theorem prover. The same team has also mechanized a Hoare-calculus for an idealized object-oriented programming language [BW06b]. In principle, both tools can evolve into an integrated proof environment for object-oriented programs. As of now, verification based on interactive theorem-provers is not fully automatic (that's where the *interactive* comes in), requiring assistance from the user who has to understand the underlying logic and the deduction rules. Once a language-processing algorithm has been formulated in an imperative language amenable for Hoare-analysis, those verification conditions that cannot be automatically derived by the tool have to be specified manually. Proof tactics are then to be applied (automatically or assisted by the user) to discharge the verification conditions and therefore the (Hoare) pre- and postconditions for the algorithm as a whole. $^+$CAL is translated to TLA$^+$, Temporal Logic of Actions [Lam02]. There is as of now no mechanized Hoare-calculus (theorem-prover supported) for $^+$CAL. However, Merz [Mer03] has made progress on mechanizing TLA$^+$ in Isabelle [NPW02].

5 Conclusions and Further Work

Given the remarkable progress during the last decade in the areas of model-checking and in anchoring the semantics of metamodeling, there is no reason preventing combining their strengths to increase the reliability of model compilers. Our findings confirm that language metamodeling techniques contribute not only to the productivity of MDSD but also to its quality. Our prototype aims at enabling the interchange of standard metamodels and certified transformations within the software engineering community, reaping the benefits of network effects. Integrated model-driven toolchains for enterprise-scale projects involve metamodels for several languages, whose development costs would be prohibitive if done from scratch and in isolation by separate teams. We foresee the existence of peer-reviewed, public repositories of machine-checked metamodels and transformations in the near future.

Software production exhibits two trustworthiness gaps: the transition from (a) software design to high level source code, and (b) that from source code to binary code. Step (b), i.e., *compiler correctness*, was addressed in the Verifix project [GGZ04]. Being OCL used in software designs, another example of (a) is answering whether an OCL query retrieves the same resultset as its translated SQL'92 formulation. In the example, the formal problem consists in determining whether the membership conditions defined over their respective universes by the OCL and the SQL'92 queries are logically equivalent. Research in this field for model compilers targeting an enterprise-class software architecture is at its early stages.

Our work on certifying model transformations will be continued with the development of extensions to the Eclipse Modeling infrastructure (supported by a 2006 Eclipse Innovation Grant). This project involves implementing transformations so that generated Java code exhibits (a) efficient detection of broken OCL invariants, (b) reactive rules, and (c) statechart-related design patterns. There is an interplay between these R&D activities. The listed capabilities were motivated by our work on declarative specifications of behavior, as necessary for example when defining consistency-enforcing algorithms for use in DSL editors. In turn, the experience gained with case studies on model-checking transformations is directly applicable to the language design of the modeling infrastructure extensions mentioned above.

References

[Ake04] Akehurst, D. H.: Validating BPEL Specifications using OCL. Tech. Report 15–04, University of Kent at Canterbury, August 2004.

[Ame06] Amelunxen, C. et. al.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In (Rensink, A.; Warmer, J. Eds.) Model Driven Architecture – Foundations and Applications: 2nd European Conf., LNCS 4066, pp. 361-375, Heidelberg, 2006. Springer Verlag.

[APM03] Antoniol, G.; Di Penta, M.; Merlo, E.: YAAB (Yet Another AST Browser): Using OCL to Navigate ASTs. In IWPC'03: Proc. of the 11th IEEE Int. Workshop on Program Comprehension, pp. 13–22, Washington, DC, USA, 2003. IEEE Computer Society.

[AS06] Amelunxen, C.; Schürr, A.: On OCL as part of the Metamodeling Framework MOFLON. In (Demuth, B. et. al. Eds.) Proc. Workshop on OCL for (Meta-)Models in Multiple Application Domains, Tech. Report TUDFI06– 04–Sept. 2006, pp. 182-193. Technische Universität Dresden, 2006.

[BB06] Büttner, F.; Bauerdick, H.: Realizing UML Model Transformations with USE. In (Chiorean, D. et. al. Eds.) UML/MoDELS Workshop on OCL (OCLApps'2006), pp. 96-110. Technical University of Dresden, Tech. Report TUD–FI06, 2006.

[BDW06] Brucker, A. D.; Doser, J.; Wolff, B.: A Model Transformation Semantics and Analysis Methodology for SecureUML. Tech. Report 524, ETH Zürich, 2006.

[BHS07] Beckert, B.; Hähnle, R.; Schmitt, P. H. (Eds.): Verification of Object-Oriented Software: The KeY Approach. LNCS 4334, Springer-Verlag, 2007.

[BKS02] Beckert, B.; Keller, U.; Schmitt, P. H.: Translating the Object Constraint Language into First-order Predicate Logic. In Proc. VERIFY Workshop at Federated Logic Conf. (FLoC), Copenhagen, Denmark, 2002.

[Bud03] Budinsky, F. et. al.: Eclipse Modeling Framework. Addison-Wesley Professional, Boston, MA, USA, 2003.

[Bur05] Burdy, L. et. al: An overview of JML tools and applications. Software Tools for Technology Transfer, 7(3):212-232, June 2005.

[BW06a] Brucker, A. D.; Wolff, B.: The HOL-OCL Book. Tech. Report 525, ETH Zürich, 2006.

[BW06b] Brucker, A. D.; Wolff, B.: A Package for Extensible Object-Oriented Data Models with an Application to IMP++. In (Roychoudhury A.; Yang, Z. Eds.) Int. Workshop on Software Verification and Validation (SVV 2006), Computing Research Repository (CoRR). Seattle, USA, August 2006.

[Cer06] Ceri, S.: Process Modeling in Web Applications. ACM Trans. on Softw. Eng. and Methodology, 15(4):360–409, 2006.

[Ehr05] Ehrig, K. et. al.: Generation of Visual Editors as Eclipse Plug-ins. In ASE'05: Proc. of the 20th IEEE/ACM Int. Conf. on Automated Software Engineering, pp. 134-143, New York, NY, USA, 2005. ACM Press.

[Gam98] Gamma, E.; Vlissides, J.; Johnson, R.; Helm, R.. Design Patterns CD: Elements of Reusable Object-Oriented Software, (CD-ROM). Addison-Wesley Longman, Boston, MA, USA, 1998.

[Gar06] Garcia, M.: Formalizing the Well-formedness Rules of EJB3QL in UML + OCL. In (Kühne, T. Ed.) Reports and Revised Selected Papers, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October, 2006. LNCS 4364, pp. 66–75.

[Gar07] Garcia, M.: Accompanying materials to this paper. http://www.sts.tu-harburg.de/~mi.garcia/pubs/2007/se2007. Last visited on 2007-02-07.

[GGZ04] Glesner, S.; Goos, G.; Zimmermann, W.: Verifix: Konstruktion und Architektur verifizierender Übersetzer. it – Information Technology, pp. 265-276, 2004.

[Gol05] Goldberg, B.: The DASL Language: Programmer's Guide and Reference Manual, Tech. Report TR- 2005-128, Sun Microsystems Research Labs, 2005.

[GVZ05] Gruschko, B.; Vogt, F. H.; Zambrovski, S.: The Use of TLA+ and Model Checking Tools in the Eclipse Environment. In 2nd Int. Workshop on Web Services and Formal Methods, Versailles, France, September 2005.

[Ham04] Hamie, A.: Translating the Object Constraint Language into the Java Modelling Language. In SAC'04: Proc. of the 2004 ACM Symp. on Applied Computing, pp. 1531-1535, New York, NY, USA, 2004. ACM Press.

[HM05] Hubert, T.; Marche, C.: A case study of C Source Code Verification: the Schorr-Waite Algorithm. In SEFM'05: Proc. of the 3rd IEEE Int. Conf. on Softw. Eng. and Formal Methods, pp. 190-199, Washington, DC, USA, 2005. IEEE Computer Society.

[HZS05] Huang, S. S.; Zook, D.; Smaragdakis, Y.: Statically Safe Program Generation with SafeGen. In (Glück, R.; Lowry, M. R. Eds.) GPCE'05: Proc. of the 4th Int. Conf. on Generative Programming and Component Eng., LNCS 3676, pp. 309-326. Springer, 2005.

[Jac06] Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press, 2006.

[JBK06] Jouault, F.; Bézivin, J.; Kurtev, I.: TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In GPCE'06: Proc. of the 5th Int. Conf. on Generative Programming and Component Eng., pp. 249-254, New York, NY, USA, 2006. ACM Press.

[JK06] Jouault, F.; Kurtev, I.: On the Architectural Alignment of ATL and QVT. In Proc. of the 2006 ACM Symp. on Applied Computing (SAC 06), pp. 1188-1195, Dijon, France, 2006. ACM Press.

[Joh04] Johnson, J. E.; Langworthy, D. E.; Lamport, L.; Vogt, F. H.: Formal Specification of a Web Services Protocol. Electr. Notes Theor. Comput. Sci., 105:147-158, 2004.

[Lam02] Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman, Boston, MA, USA, 2002.

[Lam06a] Lamport, L.: Checking a Multithreaded Algorithm with $^+$CAL. In (Dolev, S. Ed.) Proc. of the 20th Int. Symp. on Distributed Sys., DISC 2006, Stockholm, Sweden, September 2006, LNCS 4167, pp. 151–163.

[Lam06b] Lamport, L.: The $^+$CAL Algorithm Language. Fifth IEEE Int. Symp. on Network Computing and Applications (NCA'06), 2006. http://research.microsoft.com/users/lamport/pubs/pluscal.pdf. Last visited on 2007-02-07.

[Mer03] Merz, S.: On the Logic of TLA$^+$. Computers and Informatics, 22:351-379, 2003.

[MMS98] Meyer, B.; Mingins, C.; Schmidt. H.: Providing Trusted Components to the Industry. Computer, 31(5):104-105, 1998.

[MN05] Mehta, F.; Nipkow, T.: Proving Pointer Programs in Higher-Order Logic. Inf. Comput., 199(1-2):200-227, 2005.

[NPW02] Nipkow, T.; Paulson, L. C.; Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higher-Order Logic, LNCS 2283. Springer, 2002.

[OMG05] Object Management Group. MOF QVT Final Adopted Specification, formal/05-11-01, November 2005.

[OMG06] Object Management Group. Meta Object Facility (MOF) Core Specification, formal/06-01-01, January 2006.

[Rai04] Raistrick, C.; Francis, P.; Wright, J.; Carter, C.; Wilkie, I.: Model Driven Architecture with Executable UML. Cambridge University Press, Cambridge, UK, 2004.

[WK03] Warmer, J.; Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman, Boston, MA, USA, 2003.

[WKC06] Wang, J.; Kim, S-K.; Carrington, D.: Verifying Metamodel Coverage of Model Transformations. In ASWEC'06: Proc. of the Australian Softw. Eng. Conf., pp. 270-282, Washington, DC, USA, 2006. IEEE Comp. Soc.

```
algorithm test {
variables
  $g \in VertexGraph$; $root \in g.node$;
  $alloc = [v \in g.node \mapsto MemnodeForVertex[v]]$;
  $pointsTo = [v \in g.node \mapsto SetToSequence(Targets[v, g])]$;
  $current = root$; $next = Null$; $prev = Null$; $i = 1$; $backref = Null$; $bQuit = \text{FALSE}$;
{$while(\neg bQuit)\{$
   $\backslash *$ go down the leftmost branch
   $while((current \neq Null) \wedge (alloc[current].markBit = \text{FALSE}))\{$
      $alloc[current].markBit := \text{TRUE}$; $alloc[current].flag := 1$;
         $if(Len(pointsTo[current]) > 0)\{$
           $i := alloc[current].flag$; $next := pointsTo[current][i]$;
           $pointsTo[current][i] := prev$; $prev := current$;
           $alloc[current].flag := alloc[current].flag + 1$; $current := next$;
         };
      }; $\backslash *$ end of while current
      $\backslash *$ retreat, all objects pointed from current have been visited
   $while( \wedge (prev \neq Null)$
         $\wedge (alloc[prev].flag = Len(pointsTo[prev]) + 1))\{$
      $i := Len(pointsTo[prev])$; $next := pointsTo[prev][i]$;
      $pointsTo[prev][i] := current$; $current := prev$; $prev := next$;
      }; $\backslash *$ end of while prev
      $\backslash *$ retreated back to the starting point
      $if(prev = Null)\{bQuit := \text{TRUE};\}$;
      $if(\neg bQuit)\{$ $\backslash *$ visit subgraph to the right of prev
        $i := alloc[prev].flag - 1$; $backref := pointsTo[prev][i]$;
        $pointsTo[prev][i] := current$; $next := pointsTo[prev][alloc[prev].flag]$;
        $pointsTo[prev][alloc[prev].flag] := backref$;
        $current := next$; $alloc[prev].flag := alloc[prev].flag + 1$;
      };
};
$assert(Topology(pointsTo) = g.edge)$;
$assert(ReachableFrom[g, root] = \{v \in g.node : alloc[v].markBit\})$;
}}
```

Figure 3: Schorr-Waite expressed in $^+$CAL. Preconditions concern valid input (not shown), postconditions (see the assert statements) verify certain topological invariants.