Analyzing Dynamic Models using a Data-flow based Approach

Christian Saad, Bernhard Bauer (Supervisor)
christian.saad@informatik.uni-augsburg.de

University of Augsburg

Abstract. Meta modeling as a method allows to devise languages suited for specific application domains, e.g. for describing the structural or behavioral aspects of software systems. As meta models constitute an abstract syntax (often enriched with static semantics) there exist obvious similarities to the area of formal languages. The research effort described in this paper is intended to examine to what extent and to what benefit compiler construction concepts, namely the data-flow analysis method, can be transferred to the modeling domain in order to validate static semantics and perform abstract interpretations on models.

1 Motivation

In the last decade, the use of meta models has evolved from a scientific approach to a widely used technique in areas like software development and business process modeling (BPM). This advancement has been greatly endorsed by industry standards like the OMG's Meta-Object Facility (MOF) framework that forms the basis for several prominent modeling languages like the the Unified Modeling Language or the Business Process Modeling Notation (BPMN) and even a model-centric development approach called Model-driven Architecture (MDA)[1].

Nevertheless, when compared to formal languages, an inherent flaw of the modeling technique becomes obvious: The restrictions on a language's structure which are given by its abstract syntax, i.e. a context-free grammar (CFG) or a meta model, are often not enough to ensure correct expressions - a check of the static semantics is also in order. For this purpose, CFGs are often extended with attributes, forming attribute grammars (AG), which enable compilers to validate statements based on the context in which they appear (cf. [1]).

A limited amount of static analysis can be performed using the OMG's Object Constraint Language (OCL, [7]) which enables to phrase constraints on the structure of MOF and UML-based meta models. However, it has several drawbacks: Through navigation statements, OCL constraints are tightly tied to the structure of the meta model. This can lead to difficulties if constraints require the consideration of a model element's context, e.g. the position of an action in an UML activity diagram relative to its preceeding/succeeding actions. As a

[1] http://www.omg.org/technology/documents/modeling_spec_catalog.htm

language which is mainly intended to be used to validate constraints by performing static queries on a model's elements, OCL does not contain semantics which would enable a fixed-point analysis since, in the case of cyclic dependencies this would require a continuous reevaluation the DFA's equation system. Finally, complex navigation statements make OCL rules very prone to be invalidated if the structure of the underlying meta model changes, often requiring extensive adjustments.

Since models comprise a graph structure, defining and calculating information flow enables an advanced analysis of a model's properties. These may either be properties of the graph structure itself, e.g. strongly connected component regions, or semantic attributes, e.g. the availability of a resource created at one point in a control-flow model in some other part of the model, This research is dedicated to adapting the data-flow analysis (DFA) approach commonly used in compiler construction for deriving optimizations from a program's control-flow graph, to modeling, thus enabling a fixed-point analysis on (meta) models.

2 Related Work

Since the definition of static semantics is a vital step when developing formal languages, many different techniques have been considered for this purpose.

The OCL can be considered a comparatively simple language for requirements exceeding syntactic expressiveness and by design is very well-integrated into the modeling domain. A critical evaluation of its expressive power can be found in [5], with emphasis on difficulties in calculating transitive closures.

Approaches considering the use of formal semantics include graph transformations, abstract state machines or first order logic (cf. [8] [6] [3]).

Several authors use DFA-based methods to derive information from models. In [9], a fixed-point calculation is used on executable models to derive *def/use* relationships between actions while the authors of [2] propose the use of control-flow information to improve software testing.

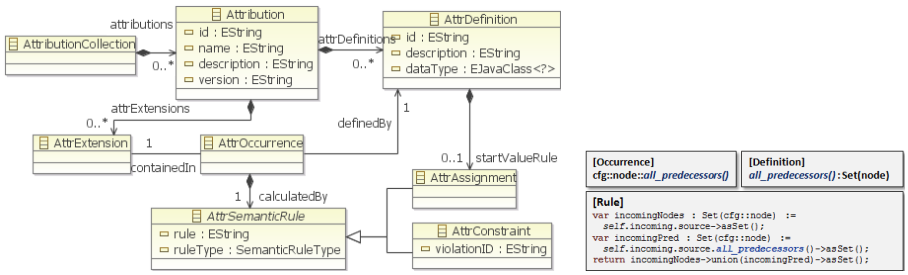
The focus in these cases is, however, directed towards implementing a specific case study. To the best of our knowledge, there is no other research effort with the goal of applying the concept of a fixed-point based DFA calculation to models.

3 Approach for Model-based Data-flow Analysis

Both meta models and (context-free) grammars operate on different layers of abstraction. For grammars, these usually consist of EBNF, CFGs and language expressions. In the widely-used MOF one distinguishes between meta-meta (M3), meta (M2) and model (M1) layer. Because of conceptual similarity, both techniques can be aligned according to their levels of abstraction. This alignment requires that data-flow equations are attached to language artefacts (i.e. M2 meta classes) and instantiated/executed for expressions, i.e. M1 objects.

Since attribute grammars are a well-proven extension of this design, it was decided that this strategy will also be employed in the definition of data-flow

equations: An *attribute definition* consisting of an identifier and a data-type can be bound to several meta model classes through the use of *attribute occurrences*. *Semantic rules* (corresponding to data-flow equations) are assigned to *attribute definitions* and *attribute occurrences* to calculate initialization and iteration values, respectively. Data-flow between attributes occurs if a *semantic rule* requests the value of another attribute as input.



(a) Data-flow analysis meta model (b) Calculate predecessors

Fig. 1. Model-based DFA definition in the notion of attribute grammars

To stay consistent with the notion of modeling and to minimize frictional losses between different techniques, it is desirable to represent DFAs themselves as models, i.e. to provide a meta model which in the described alignment hierarchy acts as an extension of the M3 layer (cf. Figure 1(a)).

To calculate a DFA for a given model, attribute occurrences assigned to meta classes need to be instantiated for model elements derived from these classes. While the instantiation process is straightforward, it has to be noted that this must happen in compliance with principles like generalization, i.e. if defined for a super class, attributes must be inherited by instances of sub classes.

An example is shown in Figure 1(b) which calculates the transitive closure of predecessor nodes in a simple control-flow graph. The class *node* has an assigned attribute occurrence of the type *all_predecessors*. The associated rule, which is repeatedly executed for all *nodes*, recursively creates the union of direct predecessors and the value of *all_predecessors* at preceding nodes.

The fixed-point calculation significant differs from traditional DFA techniques: Since semantic rules may request the value of arbitrary attributes as input when they are executed, there exists neither an inherent flow direction nor any prior knowledge about output relationships between attribute instances. Since the commonly used worklist algorithm depends on this information, it is not applicable in this context. To accomodate for this, an algorithm has been developed that dynamically records input/output relationships by executing the rules recursively to create a dependency graph which can then be used as a basis to derive a nearly optimal execution order, thus minimizing the amount of rule executions.

4 Research

The research effort described in this abstract has to cover the following issues:

Definition and Alignment To transfer the method of DFA to the modeling domain, similarities and differences between both application areas must be identified and a new definition language has to be devised and aligned with the modeling techniques.

DFA Algorithms Suitable algorithms for calculating DFA equation systems have to be devised and proven to be correct.

Complexity and Performance The practical and theoretical performance of different DFA solving algorithms has to be evaluated.

Tooling and Use Cases To prove the feasability of this approach, a tooling environment has to be provided alongside the implementation of several use cases which need to be evaluated against implementations based on alternative theoretical foundations.

While efforts for DFA definition and integration into modeling are well advanced (as described in the previous section), a formalization of these results is still in order. The same goes for the DFA solving algorithm which has proven to perform very well in comparison to adaptions of conventional algorithms like the work list method.

The Model Analysis Framework (MAF[2]) is a fully functional prototype providing tooling support for the described concepts. It is built upon Eclipse modeling techniques, namely the Eclipse Modeling Framework (EMF), an implementation of the MOF standard. Once all artefacts required for an analysis (i.e. meta models, models and attributions based on the meta model shown in Figure 1(a)) are loaded into the respective repositories, the defined attributes are instantiated for the model's elements and handed to the selected evaluation algorithm. First experiments have shown that the developed algorithm preforms much better in calculating the result values than a traditional worklist algorithm that has been enhanced with the ability to handle dynamically discovered output dependencies.

Currently implemented case studies (which are available from the code repository) include: Calculating properties of control-flow graphs like transitive predecessor/successor sets and strongly connected components (SCC), definition/use relationships in workflows and the hierarchical subdivision of control-flows into its Single-Entry-Single-Exit (SESE) components using a token-flow algorithm (cf. [4]) which has been reimplemented using DFA. In the future, additional applications areas are planned like clone detection, validating modeling guidelines and generating test models for model-based testing approaches.

[2] `http://code.google.com/p/model-analysis-framework/`

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley (August 2006), http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0321486811
2. Garousi, V., Bri, L., Labiche, Y.: Control Flow Analysis of UML 2.0 Sequence Diagrams (2005)
3. Georg, G., Bieman, J., France, R.: Using Alloy and UML/OCL to specify run-time configuration management: a case study. Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremists 7 (2001)
4. Götz, M., Roser, S., Lautenbacher, F., Bauer, B.: Token Analysis of Graph-Oriented Process Models. New Zealand Second International Workshop on Dynamic and Declarative Business Processes (DDBP), in conjunction with the 13th IEEE International EDOC Conference (EDOC 2009) (September 2009)
5. Mandel, L., Cengarle, M.: On the expressive power of the Object Constraint Language OCL. Available on the World Wide Web: http://www. fast. de/projeckte/forsoft/ocl (1999)
6. Ober, I.: An ASM Semantics of UML Derived from the Meta-Model and Incorporating Actions pp. 356–371 (2003)
7. Object Management Group: Object Constraint Language. http://www.omg.org/spec/OCL/2.0/ (Mai 2006)
8. Varró, D.: A formal semantics of UML Statecharts by model transition systems. Graph Transformation pp. 378–392 (2002)
9. Waheed, T., Iqbal, M., Malik, Z.: Data Flow Analysis of UML Action Semantics for Executable Models. Lecture Notes in Computer Science 5095, 79–93 (2008)