GRAPH TRIGGERS AND INCREMENTALLY EVALUATED QUERIES OVER EMF MODELS

Gábor BERGMANN
Advisor: Dániel VARRÓ

I. Introduction

As model management platforms are gaining more and more industrial attraction, the importance of techniques for processing large models is also increasing. A further challenge is coping efficiently (in terms of computational performance and required manual effort) with the constantly evolving nature of models. A model management platform can greatly support the declarative processing of models with the following three features:

- efficiently evaluated declarative model queries,
- change-driven [1] reactive behaviour triggered by changes captured using model queries,
- and declaratively defined model manipulation operations.

Scenarios where such features offer great benefits include model transformation (especially live synchronization [2]), code generation, domain specific behaviour simulation and model validation.

A leading industrial modeling ecosystem, the Eclipse Modeling Framework (EMF [3]), provides different ways to query the contents of models. These approaches range from (1) the use of high-level declarative constraint languages (like OCL [4]) to (2) a dedicated query language [5] resembling SQL, or, in the most basic case, (3) manually programmed model traversal using the generic model manipulation API of EMF. However, industrial experience shows scalability problems of complex query evaluation over large EMF models, taken e.g. from the automotive domain, especially when confronted with evolving models.

To address these issues in the context of EMF, a novel solution for model queries was developed in cooperation with several colleagues. EMF-INCQUERY [6] automatically provides efficient incremental evaluation, without manual coding, to cope with the evolving nature of models. This paper presents an overview of EMF-INCQUERY, then investigates the possibility of applying similar techniques to provide declarative model manipulation and a triggering mechanism.

II. Background: EMF and GT

The Eclipse Modeling Framework (EMF [3]) provides automated code generation and tooling (e.g. notification, persistence, editor) for Java representation of models. EMF models consist of an (acyclic) containment hierarchy of model elements (EObjects) with crossreferences – some of which may only be traversed by programs in one direction (unidirectional references). Additionally, each object has a number of attributes (primitive data values).

EMF uses Ecore metamodels to describe the abstract syntax of a modeling language. The main elements of Ecore are the following: EClass, EAttribute and EReference. EClasses define the types of EObjects, enumerating EAttributes to specify attribute types of class instances and (unidirectional) EReferences to define association types to other EObjects. Some EReferences additionally imply containment.

Example: Figure 1(a) shows a simple example metamodel for a State Machine. The two EClasses are “Machine” and “State”. Each Machine contains States through a containment EReference called “states”, and one of them is marked by the “current” EReference to denote the current State of the Machine. Possible transitions are indicated by “next” EReferences between States.

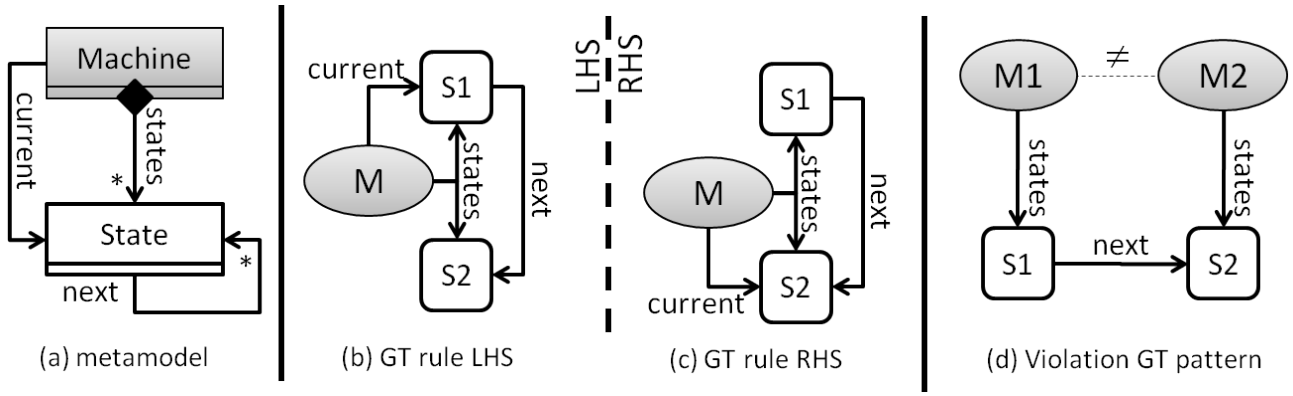

Figure 1: State Machine example

Graph patterns [7] constitute an expressive formalism used for various purposes in Model Driven Development, such as defining declarative model transformation rules, defining the behavioral semantics of dynamic domain specific languages, or capturing general purpose model queries including model validation constraints. A graph pattern (GP) represents conditions (or constraints) that have to be fulfilled by a part of the instance model. A basic graph pattern consists of structural constraints prescribing the existence of nodes and edges of a given type (or subtypes, subject to polymorphism). Some languages (e.g. VIATRA2 [7]) include a way to express negation, resulting in an expressive power equivalent to first order logic [8]. Attribute constraints are also a typical feature. A match of a graph pattern is a group of model elements that have the exact same configuration as the pattern, satisfying all the constraints.

Graph patterns can form the basis of declarative model queries, where the pattern spacifies what arrangement of elements is sought after, not how or where to find them. In an EMF context, each node in the pattern represents an EObject (EMF instance object), and the type of the node identifies the EClass of the object. This feature is useful to select only those model elements that conform to a certain type. Furthermore, the pattern nodes are connected by directed edges, annotated by an EReference type (or containment), to express how these elements reference each other. Finally, attribute constraints filtering and comparing the attributes of these elements can also be added.

Example: Figure 1(b-d) show three simple graph patterns over graph models that conform to the metamodel depicted in Figure 1(a). In particular, the pattern in Figure 1(b) identifies a Machine M and two of its States $S1$ and $S2$, where $S1$ is the current state and $S2$ is one of the possbile successor states. The pattern in Figure 1(c) is similar, but $S2$ is the current state and $S1$ is one of its predecessors. Finally, the pattern in Figure 1(d) captures two different Machines $M1$ and $M2$, with states $S1$ and $S2$ respectively, where $S2$ is marked as a successor of $S1$.

Graph transformation (GT) [9] provides a high-level rule and pattern-based manipulation language for graph models. Graph transformation rules can be specified by using a left-hand side (LHS) graph pattern determining the applicability of the rule, and a right-hand side (RHS) graph pattern which declaratively specifies the result model after rule application. Elements that are present only in (the image of) the LHS are deleted, elements that are present only in the RHS are created, and other model elements remain unchanged.

Example: The GT rule formed from Figure 1(b) as LHS and Figure 1(c) as RHS describes how the state machine can advance its state. The LHS captures the current state and a successor state, which are needed to fire the rule. When the rule is applied on a match of the LHS, it is substituted with the image of the RHS, meaning that $S2$ is marked as the current state from that time on.

An important performance issue associated with large models is that they have to be reconsidered again and again after each small change, causing a significant overhead. Incrementality is therefore a

valuable property of model processing mechanisms in the context of evolving models. In case of GT-based approaches, most of the computational complexity is associated with graph pattern matching, thus incremental pattern matching (INC) [10] is a promising way to address the performance problems.

INC techniques rely on a cache which stores the results of a query explicitly. The result set is readily available from the cache at any time without additional search, and the cache is incrementally updated whenever (elementary or transactional) changes are made to the model. As results are stored, they can be retrieved in constant time, making query evaluation extremely fast. The trade-off is increased memory consumption, and increased update costs (due to continuous cache updates).

III. EMF-INCQUERY

The aim of the EMF-INCQUERY approach is to bring the benefits of graph pattern based declarative queries and incremental pattern matching to the EMF domain. The advantage of declarative query specification is that it achieves (efficient) pattern matching without time-consuming, manual coding effort associated to ad-hoc model traversal. While EMF-INCQUERY is not the only technology for defining declarative queries over EMF [4, 5], its distinctive feature is incremental pattern matching. Thanks to this matching technique, EMF-INCQUERY has special performance characteristics suitable for scenarios such as on-the-fly well-formedness validation. See [6] for measurements revealing how EMF-INCQUERY can be several orders of magnitude more efficient than other approaches.

Additionally, some shortcomings of EMF are mitigated by the capabilities of EMF-INCQUERY, such as cheap enumeration of all instances of a certain type, regardless of where they are located in the resource tree. Another such use is the fast navigation of EReferences in the reverse direction, without having to augment the metamodel with an EOpposite (which is problematic if the metamodel is fixed, or beyond the control of the developer).

EMF-INCQUERY provides an interface for each declared pattern for (i) retrieving all matches of the pattern, or (ii) retrieving only a restricted set of matches, by binding (a-priori fixing) the value of one or more pattern elements (parameters).

In both cases, the query can be considered instantaneous, since the set of matches of the queried patterns (and certain subpatterns) are automatically cached, and remain available for immediate retrieval throughout the lifetime of the EMF ResourceSet. Even when the EMF model is modified, these caches are continuously and automatically kept up-to-date using the EMF Notification API. This maintenance happens without additional coding, and works regardless how the model was modified (graphical editor, programmatic manipulation, loading a new EMF resource, etc.).

IV. Graph Triggers over EMF

EMF-INCQUERY is useful in itself to provide an expressive and efficiently evaluated query language. However, the incrementality of the result set opens up the possibility of considering graph changes as events. As proposed with co-authors in previous work [2] outside the context of EMF, changes of the model graph, as captured on an arbitrary granularity defined by a graph pattern, can drive the execution of Graph Triggers. There are two basic kinds of triggers: those that are activated by the appearance of a match of a graph pattern, and those that are fired on a disappearance (a language for capturing more precise conditions of trigger guards is proposed in [1]). A trigger is thus specified by this guard condition and a reaction that can be defined by an arbitrary sequence of Java code.

Example: The graph pattern in Figure 1(d) identifies a violation of the well-formedness of the state machine model, in which a “next” edge leads to a different Machine. By registering a Graph Trigger that is activated on the appearance of this graph pattern and notifies the modeling expert of the mistake, the domain-specific modeling environment can benefit from on-the-fly well-formedness checking.

To reflect the changes in the match set during a sequence of model manipulation, a device called delta

monitor continuously and incrementally maintains such a difference. This makes the implementation of the trigger engine a matter of regularly checking the delta monitors and firing triggers when necessary. The timing of this check must be chosen such that the incremental pattern matcher already reflects a consistent state of the model. In case of EMF Transactions, the triggers are fired at the pre-commit phase, so that they can modify the model.

The usefulness of such a service was demonstrated by applying it in multiple ways. As a demonstration of live transformation (pioneered in [2]) over EMF, triggers facilitated an on-the-fly bidirectional synchronization between two views of the same model in two different modeling formalisms of the SecureChange EU research project. A second likely usage, similar to the example given above, is on-the-fly checking of structural well-formedness properties, as demonstrated in [6]. As a third usage, change itself was used as a source of information (in addition to the current state of the model) in the same SecureChange case study [1].

Finally, the possibility of declarative model manipulation over EMF was also investigated, although the implementation on top of EMF-INCQUERY remains future work. The natural choice of formalism to apply on EMF would be GT rules, investigated in e.g. [11]. The main challenge is the hierarchical nature of the EMF Resources, implying that new elements can only be created in a container element, and that element deletions cause the removal of the entire containment sub-tree, which can cause inconsistency if e.g. the same GT rule connects a new edge to a deleted element. To address these issues, [11] imposes some consistency restrictions on GT rules, and these conditions assume complete knowledge of the Ecore metamodels, which may be unrealistic.

V. Conclusion

The proposed technology applies academically researched model transformation features (namely incremental pattern matching and graph triggers) in the industrial platform of EMF; benefits manifest in use cases including live model synchronization and structural well-formedness checking.

References

[1] G. Bergmann, I. Ráth, G. Varró, and D. Varró, "Change-driven model transformations: Taxonomy and language," *Journal of Software and Systems Modeling*, 2010, Submitted.

[2] I. Ráth, G. Bergmann, A. Ökrös, and D. Varró, "Live model transformations driven by incremental pattern matching," in *Theory and Practice of Model Transformations*, vol. 5063/2008 of *Lecture Notes in Computer Science*, pp. 107–121. Springer Berlin / Heidelberg, 2008.

[3] The Eclipse Project, *Eclipse Modeling Framework*, http://www.eclipse.org/emf.

[4] The Eclipse Project, *MDT OCL*, http://www.eclipse.org/modeling/mdt/?project=ocl.

[5] The Eclipse Project, *EMF Model Query*, http://www.eclipse.org/modeling/emf/?project=query.

[6] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, "Incremental evaluation of model queries over EMF models," in *Model Driven Engineering Languages and Systems, 13th International Conference, MODELS'10*. Springer, Springer, 10/2010 2010, Acceptance rate: 21%.

[7] D. Varró and A. Balogh, "The Model Transformation Language of the VIATRA2 Framework," *Science of Computer Programming*, 68(3):214–234, October 2007.

[8] A. Rensink, "Representing first-order logic using graphs," in *International Conference on Graph Transformations (ICGT), LNCS 3256*, pp. 319–335. Springer, 2004.

[9] H. Ehrig and et al., Eds., *Handbook on Graph Grammars and Computing by Graph Transformation*, vol. 2, World Scientific, 1999.

[10] G. Bergmann, A. Ökrös, I. Ráth, D. Varró, and G. Varró, "Incremental pattern matching in the VIATRA model transformation system," in *Graph and Model Transformation (GraMoT 2008)*, G. Karsai and G. Taentzer, Eds. ACM, 2008.

[11] E. Biermann, C. Ermel, and G. Taentzer, "Precise semantics of EMF model transformations by graph transformation," in *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, pp. 53–67. Springer-Verlag, 2008.