Implementing Efficient Model Validation in EMF Tools

Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
{bergmann,hegedusa,ahorvath,rath,ujhelyiz,varro}@mit.bme.hu

***Abstract*—Model-driven development tools built on industry standard platforms, such as the Eclipse Modeling Framework (EMF), heavily use model queries in various use cases, such as model transformation, well-formedness constraint validation and domain-specific model execution. As these queries are executed rather frequently in interactive modeling applications, they have a significant impact on the runtime performance of the tool, and also on the end user experience. However, due to their complexity, they can be time consuming to implement and optimize on a case-by-case basis. To address these shortcomings, we developed the EMF-INCQUERY framework for defining declarative queries over EMF models and executing them effectively using a caching mechanism.**

In the current paper, we demonstrate how our framework can be easily integrated with other EMF tools. We describe a case study in which EMF-INCQUERY is integrated into the open source Papyrus UML environment to provide on-the-fly validation of well-formedness criteria in UML models.

***Index Terms*—EMF; model query; incremental evaluation; model validation**

I. INTRODUCTION

As model management platforms are gaining more and more industrial attraction, the importance of automated model querying techniques is also increasing. Queries form the underpinning of various technologies such as model transformation, code generation, domain-specific behaviour simulation and well-formedness validation.

The leading industrial modeling ecosystem, the Eclipse Modeling Framework (EMF [1]), provides different ways for querying the contents of models. These approaches range from manually coded model traversal to high-level declarative constraint languages such as MDT-OCL [2]. However, industrial experience [3] shows scalability problems of complex query evaluation over large EMF models, taken from the various modeling domains; and manual query optimization is time consuming to implement on a case-by-case basis.

In order to overcome this limitation we implemented the EMF-INCQUERY[1] framework [3] for declaratively defining queries over EMF models, and executing them efficiently *without manual coding* using incremental pattern matching techniques [4]. The benefits of EMF-INCQUERY with respect to the state-of-the-art of querying EMF models include: (i) high performance querying of models in the range of millions of elements, (ii) efficient addressing of instance enumeration and backward navigation both frequently encountered shortcomings of EMF's programming interface and (iii) user friendly declarative graph pattern based formalism for easier query definition.

In the current tool paper, our aim is to demonstrate how easily EMF-INCQUERY can be integrated with EMF-based tools. As a complex case study[2], we show how EMF-INCQUERY can be integrated non-intrusively into standard tools such as the the Papyrus UML [5] modeling environment, to provide support for on-the-fly well-formedness validation of UML models.

The paper is structured as follows: first, Section II gives a brief introduction to the basics of EMF-INCQUERY. Section III shows how incremental model validation using EMF-INCQUERY can be integrated into an EMF application. Section IV gives an overview of some related technologies, and Section V concludes the paper.

II. BACKGROUND

The basics of EMF-INCQUERY are briefly outlined in this Section.

A. Model Queries by Graph Patterns

Graph patterns [6] constitute an expressive formalism used for various purposes in model-driven development, such as defining declarative model transformation rules, capturing general-purpose model queries including model validation constraints, or defining the behavioral semantics of dynamic domain-specific languages. 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. Languages usually include a way to express *attribute constraints*. A *negative application condition* (NAC) defines cases when the original pattern is *not* valid (even if all other constraints are met), in the form of a negative sub-pattern. 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 (except for NACs, which must be made unsatisfiable). The complete query language of the EMF-INCQUERY framework is described in [7].

[1] http://viatra.inf.mit.bme.hu/incquery

[2] http://viatra.inf.mit.bme.hu/incquery/examples#Papyrus

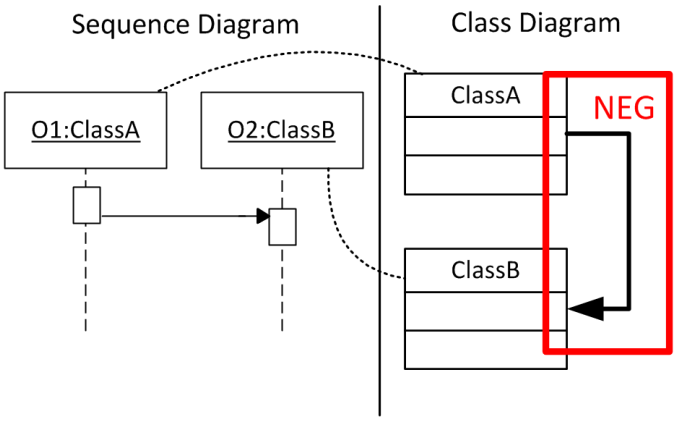

Figure 1. Graph Pattern for Message without Association

Figure 1 depicts a simplified graph pattern that identifies a violation of a UML design guideline. The pattern will match a pair of lifelines in a sequence diagram that exchange a message, and yet the classes represented by them are not connected by an association. The structural part captures the existence of the classes (depicted as *ClassA* and *ClassB* in a class diagram), one lifeline typed with each of these classes (depicted as *O1* and *O2* in a sequence diagram), and a message defined between the lifelines. The NAC part defines the missing association between the classes.

B. Overview of EMF-IncQuery

The overall workflow of using EMF-IncQuery separates development time tasks from runtime tasks, as summarized in Figure 2.

First, in development time the queries need to be defined as graph patterns. Our implementation relies on the VIATRA2 model transformation framework [6] to support this development activity. EMF-specific query evaluator components are generated from the query definitions. These components represent the direct interface to Java client applications and rely on the EMF-IncQuery Runtime, which is responsible for evaluating the queries over EMF ResourceSets.

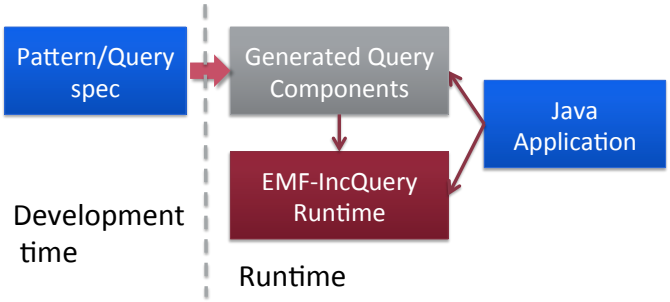

Figure 2. Overview on the EMF-IncQuery Development workflow

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 compared to ad-hoc model traversal. While EMF-IncQuery is not the only technology for declaratively defining queries over EMF (see EMF Query or MDT-OCL), it is distinguished by its incremental evaluation feature.

Incremental query evaluation means that the query results (the match sets of graph patterns) are cached in memory, and can be instantaneously retrieved when queries are issued. These caches are automatically and incrementally maintained upon model updates, using automatic notifications provided by EMF. There is a slight performance overhead on model manipulation, and a memory cost proportional to the cache size (approx. the size of match sets). These special performance characteristics make incremental techniques suitable for application scenarios such as on-the-fly well-formedness checking, live model transformation and other complex use cases. Specifically in well-formedness checking, as the number of constraint violations is usually relatively small, the additional memory footprint is affordable. In [3] we carried out an extensive performance benchmark in an industrial scenario of well-formedness checking, and found EMF-IncQuery to outperform alternatives by a large margin due to its incrementality.

III. Extending EMF Tools with Incremental Model Validation

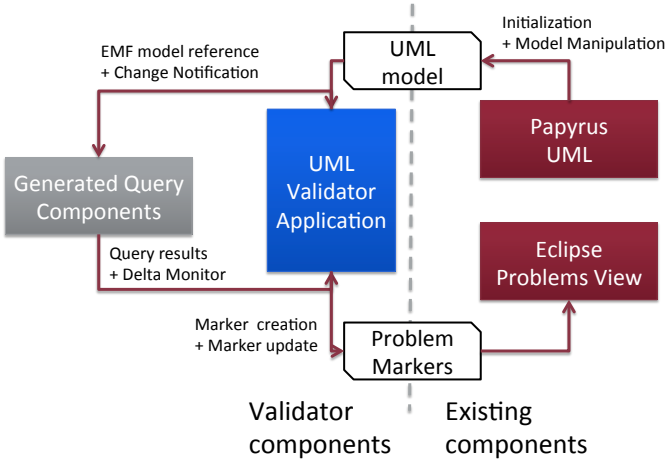

Figure 3. The Papyrus integration of EMF-IncQuery

Components using EMF-IncQuery for incremental query evaluation, including on-the-fly model validators, can be easily integrated into any EMF-based tool. We illustrate this by a case study where the Papyrus UML tool is non-intrusively extended with efficient model validation capabilities. The architecture of the integration is depicted in Figure 3. The following paragraphs will discuss each of the following validator components.

The incremental evaluation feature of EMF-IncQuery requires notifications of changes to the model, so that the cached query results can be incrementally maintained. Fortunately, EMF can automatically provide such change notifications.

After specifying the EMF model that the queries should evaluate against, EMF-IncQuery can automatically subscribe for notifications along with producing the initial result set. In this context, any EMF resource (e.g. XMI file) or resource set (group of interreferencing resources) can play the role of an EMF model (in rare cases an arbitrary EObject containment subtree is also possible).

It is even possible to apply EMF-IncQuery on the run-time EMF model used by an application, without any modifications required to said application. Many Eclipse-based applications offer extension points where a reference to their run-time EMF instance model can be obtained. Even if the extension point only provides access to a single EMF model element, it can be used to identify its containing resource or resource set, and thus it is enough to initialize EMF-IncQuery. This is also how the Papyrus integration was performed: the UML validator can be initialized by selecting a context menu entry that was contributed through an Eclipse extension point; no direct modification of Papyrus code was reqiured. There can be, of course, benefits of more cooperation between the program that owns the EMF model and EMF-IncQuery, such as tighter control of when and how the query engine should be initialized, and more direct feedback of query results.

Although model queries and incrementally evaluated queries in particular have many usage scenarios, our case study in particular focuses on finding violations of structural well-formedness constraints. For this purpose, in addition to simply executing the queries and retrieving the results, it is also neccessary to inform the user about the current violations in the model. Fortunately, this task can also be carried out without modifications to Papyrus, as problems can be indicated in a separate view. We have opted to use the standard problem markers provided by Eclipse, where problems are bound to files and can be listed in the Problems View. This provides tight cooperation with the UI facilities of Papyrus, e.g. the ability to indicate the actual location of violations within the model, directly in the graphical editor of the tool.

The final task is handling model evolution, i.e. reacting to changes in the result set of the query. When a new violation is detected, a new problem marker should be created; when a previously detected violation ceases to occur, the corresponding problem marker should be removed. The challenge is in detecting when these changes occur. The incremental query evaluation engine of EMF-IncQuery includes a feature called *delta monitor* that serves the purposes of monitoring the difference of the result set of a query from a given point in time onward. In our particular case, delta monitors can be used to report which violations have appeared or disappeared since the last time they were checked. There still remains one issue: *when* to check. EMF-IncQuery provides a callback hook that is invoked whenever the result set has been updated. This can be used entirely automatically and without cooperation from the modeling tool, so our case study relies on this callback mechanism to update the problem markers.

Unfortunately, each step of a complex model manipulation operation will do some partial updates in the model, resulting in the invocation of the callback mechanism mentioned above. In this case, the routine that updates problem markers may observe the model in an inconsistent intermediate state. Therefore the problem markers may show erroneous results for a very short amount of time, while the modeling tool is in the middle of a model manipulation sequence. In case of tighter integration between the modeling tool and the model validator, it can be ensured that the problem marker update routine is called only when the model is in a semantically consistent state. For example, it would be a good idea to call this routine at the end of each EMF Transaction (if the transaction framework is used), or any application-specific unit of model manipulation. With the latter approach, there are two easy mistakes to consider. The first is neglecting to call this update routine after every kind of model manipulation (e.g. performed by a background transformation as opposed to a GUI operation), so that the results become stale until a subsequent operation is performed that does properly refresh them. The second is updating the markers regularly but too infrequently (e.g. only when saving the model). In both cases, the outdated results can rob the on-the-fly validation approach of its greatest advantage: immediate feedback to the modeler.

We consolidated the common components described above as a light-weight model validation framework built on the EMF-IncQuery runtime. This library handles the registration and initialization of the EMF-IncQuery pattern matchers, and the creation of problem markers from query results and delta markers. To extend this validator with a new well-formedness constraint, only the high-level specification of the corresponding graph pattern needs to be annotated accordingly, as the tooling supports full code generation.

IV. Related Work

Model queries over EMF: There are numerous technologies for providing declarative model queries over EMF. Here we give a brief summary of the mainstream techniques, none of which support incremental behavior.

EMF Model Query [8] provides query primitives for selecting model elements that satisfy a set of conditions; these conditions range from type and attribute checks to enforcing similar condition checks on model elements reachable through references. The query formalism has several important restrictions: (i) it can only describe tree-like patterns (as opposed to graph patterns); (ii) nodes cannot be captured in variables to be referenced elsewhere in the query; and (iii) the query can only traverse unidirectional relations in their natural direction. Indeed, the expressive power of Model Query is weaker than first order logic. Therefore, more complex patterns involving circles of references or attribute comparisons between nodes cannot be detected by EMF Model Query without additional coding.

EMF Search [9] is a framework for searching over EMF resources, with controllable scope, several extension facilities, and GUI integration. Unfortunately, only simple textual

search (for model element name/label) is available by default; advanced search engines can be provided manually in a metamodel-specific way.

EMF-INCQUERY is not the first tool to apply graph pattern based techniques to EMF [10], [11], but its incremental pattern matching feature is unique.

OCL evaluation approaches: OCL [12] is a standardized navigation-based query language, applicable over a range of modeling formalisms. Taking advantage of the expressive features and wide-spread adoption of OCL, the project MDT OCL provides a powerful query interface that evaluates OCL expressions over EMF models. However, backwards navigation along references can still have low performance, and there is no support for incrementality.

Cabot et al. [13] present an advanced three-step optimization algorithm for incremental runtime validation of OCL constraints that ensures that constraints are reevaluated only if changes may induce their violation and only on elements that caused this violation. The approach uses promising optimizations, however, it works only on boolean constraints, and as such it is less expressive than our technique.

An interesting model validator over UML models [14] incrementally re-evaluates constraint instances (defined in OCL or by an arbitrary validator program) whenever they are affected by changes. During evaluation of the constraint instance, each model access is recorded, triggering a re-evaluation when the recorded parts are changed. This is also an important weakness: the approach is only applicable in environments where read-only access to the model can be easily recorded, unlike EMF. Additionally, the approach is tailored for model validation, and only permits constraints that have a single free variable; therefore, general-purpose model querying is not viable.

V. CONCLUSION AND FUTURE WORK

Previously we presented EMF-INCQUERY as the next evolutionary step in efficiently executing complex queries over EMF models, by adapting incremental technologies [4] for graph pattern matching. In the current paper, we examined how a simple and efficient model validator based on EMF-INCQUERY can be integrated non-intrusively into an existing EMF-based modeling environment.

In the typical workloads associated with well-formedness checking and other structural model validation, incremental techniques can provide excellent performance. Our previous measurement results [3] have confirmed this, but also the fact that the designer needs to keep the memory impact in mind. Practical applications of this technology include on-the-fly model validation, interactive execution of domain-specific languages, live incremental model synchronization and incremental maintenance of (aggregated) model views for development tool environments.

Further work is expected in performance tuning and general usability improvements of EMF-INCQUERY. Finally, we plan to work on integrating OCL as a query specification language. As it has been shown [15], a significant subset of OCL can be mapped to the graph pattern formalism, especially with extensions such as those in [7].

ACKNOWLEDGEMENTS

This work was partially supported by the SecureChange (ICT-FET-231101) European Research Project, the CERTIMOT (ERC_HU-09-01-2010-0003) Project and the János Bolyai Scholarship.

REFERENCES

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

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

[3] 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, MODELS'10*, ser. LNCS, vol. 6395. Springer, 2010.

[4] 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.

[5] *MDT Papyrus*, The Eclipse Project, http://www.eclipse.org/modeling/mdt/papyrus/.

[6] D. Varró and A. Balogh, "The Model Transformation Language of the VIATRA2 Framework," *Science of Computer Programming*, vol. 68, no. 3, pp. 214–234, October 2007.

[7] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró, "A Graph Query Language for EMF models," in *Proc. of ICMT'11, 3rd Intl. Conference on Model Transformation*. Springer, 2011.

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

[9] *EMFT Search*, The Eclipse Project, http://www.eclipse.org/modeling/emft/?project=search.

[10] E. Biermann, C. Ermel, and G. Taentzer, "Precise Semantics of EMF Model Transformations by Graph Transformation," in *MoDELS '08*. Springer, 2008.

[11] H. Giese, S. Hildebrandt, and A. Seibel, "Improved flexibility and scalability by interpreting story diagrams," in *Proceedings of GT-VMT 2009*, vol. 18. ECEASST, 2009.

[12] *Object Constraint Language, v2.0*, The Object Management Group, May 2006, http://www.omg.org/spec/OCL/2.0/.

[13] J. Cabot and E. Teniente, "Incremental integrity checking of UML/OCL conceptual schemas," *J. Syst. Softw.*, vol. 82, no. 9, pp. 1459–1478, 2009.

[14] I. Groher, A. Reder, and A. Egyed, "Incremental consistency checking of dynamic constraints," in *FASE 2009*, ser. LNCS, vol. 6013. Springer, 2010.

[15] J. Winkelmann, G. Taentzer, K. Ehrig, and J. M. Küster, "Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars," *ENTCS*, vol. 211, 2008.