Query-Driven Soft Interconnection of EMF Models*

Ábel Hegedüs, Ákos Horváth, István Ráth, and Dániel Varró

Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
1117 Budapest, Magyar tudósok krt. 2
{hegedusa,ahorvath,rath,varro}@mit.bme.hu

Abstract. Model repositories based on the Eclipse Modeling Framework (EMF) play a central role in the model-driven development of complex software-intensive systems by offering means to persist and manipulate models obtained from heterogeneous languages and tools. Complex EMF models can be assembled by interconnecting model fragments by hard links, i.e. regular references, where the target end points to external resources using storage-specific URIs. This approach, in certain application scenarios, may prove to be a too rigid and error prone way of interlinking models. As a flexible alternative, we propose to combine derived features of EMF models with advanced incremental model queries as means for soft interlinking of model elements residing in different model resources. These soft links can be calculated on-demand with graceful handling for temporarily unresolved references. In the background, the interlinks are maintained efficiently and flexibly by using incremental model queries as provided by the EMF-IncQuery framework.

1 Introduction

The Eclipse Modeling Framework (EMF) [1] serves as the underlying model management infrastructure for various industrial development tools, especially in the avionics and automotive domain. These domains necessitate the handling of large models with potentially millions of model elements. For maintainability and scalability reasons, such EMF models are not persisted in a single XMI document, but stored as an interconnected network of model fragments where each fragment stores a certain part of the entire system model. In other application scenarios, complete EMF models are used which are complemented with external traceability models to explicitly persist traceability links between requirements models, design models, analysis models or source code, for instance. In both scenarios, EMF models are frequently manipulated by several development or verification tools in complex toolchains operated by different design teams.

* This work was partially supported by the CERTIMOT (ERC_HU-09-01-2010-0003) project, the TÁMOP (4.2.2.B-10/1–2010-0009) grant and the János Bolyai Scholarship.

R.B. France et al. (Eds.): MODELS 2012, LNCS 7590, pp. 134–150, 2012.

Unfortunately, the interconnection of complex EMF-based system models imposes several technical problems due to the identification strategies of model elements in the EMF infrastructure. When serializing a model, a model element is either identified by a unique identifier generated by EMF, or by a relative path of containment hierarchy in the given EMF resource. These techniques are used when interconnecting models using associations (EReferences) e.g. for internal traceability purposes: the target end of the association points to an object resided in a different model resource. Such interconnections are also used in external traceability scenarios where inter-model links are introduced from traceability metamodel elements to existing metamodels which cannot be altered.

These scenarios demonstrate various shortcomings of the core EMF technology. First, (1) interconnected EMF model fragments with circular dependencies including only regular references cannot be serialized. Furthermore, without truly intelligent multi-resource transaction management, (2) local changes in a model fragment may introduce broken links unless all dependent model fragments are manipulated together in working memory. Such broken links require tool-specific resolutions — with a worst case scenario of fixing the links manually by the designer using text editors (and not the modeling tool). Finally, (3) all traceability links captured by associations are explicitly persisted every time even if traceability links could be derived from existing unique identifiers.

In the paper, we provide an approach[1] for the soft interconnection of EMF models based on *derived features* and *incremental model queries*. Derived features are attributes and relations of the model calculated at runtime, and their values are often not stored explicitly. When using derived relations, the corresponding links only exist after the models are loaded. Therefore, model fragments can be (de)serialized in arbitrary order issuing warnings about broken links when certain resources are unavailable or not loaded. In order to provide an efficient and flexible handling of such soft links, we use the incremental model query framework EMF-IncQuery as a technical foundation. As a result, it is sufficient to identify a model element by a query instead of local or global identifiers, and less amount of information needs to be persisted for traceability purposes. Furthermore, the underlying model query technique provides excellent performance with little memory overhead [2] for managing inter-model links[2].

The rest of the paper is structured as follows. First, we illustrate interconnected EMF models in Section 2 on an industrial case study and propose derived features for managing soft interconnections. Then we propose model queries as specification means for derived features, and thus for soft links (Section 3). An incremental maintenance technique for soft links is described in Section 4. In Section 5 the application of soft interconnections is described for traceability modeling. Finally, related approaches and tools are described in Section 6 and Section 7 concludes our paper.

[1] Fully implemented and documented at `http://viatra.inf.mit.bme.hu/incquery/new/examples/query-driven-soft-links`

[2] The paper does not include performance specific contributions to EMF-IncQuery, more details are available at `http://viatra.inf.mit.bme.hu/performance`

2 Soft Interconnection of EMF Models: An Overview

2.1 Case Study: Modeling and Managing Business Processes

Our approach will be demonstrated on an business modeling case study inspired by a project carried out together with an industrial partner. While the actual metamodels (shown in Figure 1) are significantly simplified here due to space restrictions, they still demonstrate many practical industrial problems of interconnecting EMF models. In the case study, semi-automatic workflows (captured as a *process model*) contain both automated and manual tasks. Architectural-level deployment decisions are captured by a separate *system architecture model* comprising of jobs and data resources referring to tasks in the business process model. Finally, the instances of the processes managed by operators are captured in an *operation* model containing a checklist for each process with task entries assigned for each operator.

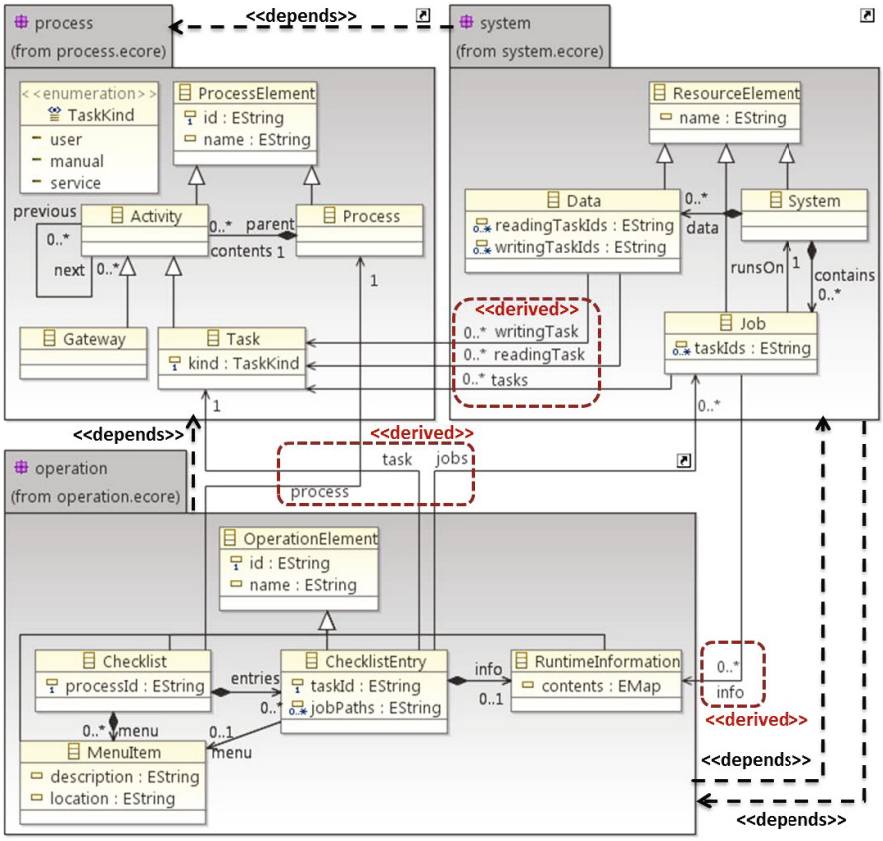


Fig. 1. The metamodels of the case study

Business process metamodel Business processes (process package) are defined by a fragment of the standard XPDL [3] metamodel. The ProcessElement top-level type defines id (unique identifiers) and name attributes for each element. A Process includes Activities that are either Tasks (atomic workflow steps) or Gateways (e.g. fork-join, decision, loops), while the control flow of the process is represented by the next and previous relations between activities. Based upon their kind attribute, tasks can be service (for automated execution through API calls), manual (where the operator initiates some job) and user (when the task itself is performed by an operator or other assigned personnel).

System architecture metamodel The system architecture metamodel (system package) defines a top-level ResourceElement that defines a name for each element. This simplified architecture includes Systems (representing larger components), Data elements that represent application data (e.g. configuration, input or output files) that can be read or written during the execution of tasks in the processes and Jobs (e.g. scripts or one-shot programs) that run on Systems. We assume that each system must have a unique name and each job contained in the same system must have different names. Otherwise, names are not globally unique in this domain.

Operation metamodel The operation metamodel (operation package) is used for representing Checklists followed by operators when performing the manual tasks in processes. The top-level OperationElement adds a name and a unique identifier for each model element. Each Checklist is related to a Process, and includes a number of entries and a menu. The menu contains MenuItems that have textual descriptions and a location, where the operator can access it. The entries are ChecklistEntry elements, each corresponding to one task, an arbitrary number of jobs, and optionally to a MenuItem. Finally, each entry can contain further information (e.g. historical statistics or requirements) stored by a RuntimeInformation element using a content map.

Inter-model connections These metamodels and thus the corresponding model instances heavily depend upon each other (see Figure 1). The following logical interconnections are present in our example: (1) a Job (from system) can be linked to a Task (in process); (2) a Process is a referenced from Checklist; (3) a ChecklistEntry links to both to a process Task and a system Job; (4) a RuntimeInformation (from operation) can be attached to a Job.

Many industrial tools (including the TIBCO Business Studio [4] used in our industrial project for capturing XPDL models and the AUTOSAR standard [5]) store identifiers of external (inter-model) elements using (a list of) simple string (or integer) attributes. In contrast, EMF uses EReferences (corresponding to lazily initialized inter-object pointers) to interconnect different models (or model fragments), which are resolved during the first traversal. For the current paper, they are referred to as *hard links*, as all such cross-model references are explicitly stored in a serialized model. In order to implement such standards over the EMF infrastructure, the main challenge is to provide a transparent reference maintenance mechanism that maps the textual identifiers to in-memory pointers and also allows their modification.

2.2 Soft Links for EMF Models by Query-Based Derived References

In the paper, we propose a soft linking technique for interconnecting EMF models by combining derived features and incremental model queries. The term "query-based soft links" refer to the fact that (1) certain model interconnections only exist at runtime but they are not maintained explicitly in instance models, but (2) the interconnected model elements can be accessed and navigated in a type-safe way along derived features. Furthermore, (3) our query based technique allows to define complex, n-ary interconnections of several model elements, and (4) to identify model elements dynamically based upon query results (instead of static unique IDs).

Derived features in EMF models represent computed information which can be calculated from other model elements. Essentially, we distinguish between *derived attributes*, which provide a data store for a(n instance of a) class and *derived references*, which represent "virtual" interconnections between model element instances (represented graphically by the derived stereotype in Figure 1). Derived features for soft links will be defined by using a declarative, high-level graph-based query language (Section 3) and evaluated truly incrementally (Section 4) as offered by the advanced model query framework EMF-IncQuery [6]. Our soft interconnection technique offers the following advantages:

- **Handling circular dependencies:** Circular dependencies between EMF models can be handled easily with soft links. For instance, metamodels system and operation are mutually dependent on each other along references jobs and info, which can materialize in a circular dependency on the model level preventing serialization using auto-generated regular EMF methods. As soft links are not serialized, this problem no longer occurs.
- **Graceful management of broken links.** When EMF models are manipulated by multiple tools, inter-model links can be easily broken, which result in runtime exceptions when the corresponding model element is attempted to be accessed along a broken link. Soft links provide graceful behavior in case of broken links by issuing warnings in case of unresolved elements.
- **Improved persistence.** Whenever a model interconnection can be calculated by a query, this does not necessarily have to be explicitly persisted into traceability models. As result, the load time of complex interconnected models can be reduced.
- **High performance.** Due to the incremental caching mechanism of EMF-IncQuery [2], derived features can be reevaluated very efficiently even in case of complex definitions (e.g. transitive closures [7]). As a result, the maintenance of soft links will be efficient with low memory overhead even for large models with complex traceability structures.

3 Definition of Soft Links as Model Queries

In order to support the runtime management of soft interconnections between models using derived features of EMF models, the graph pattern based model

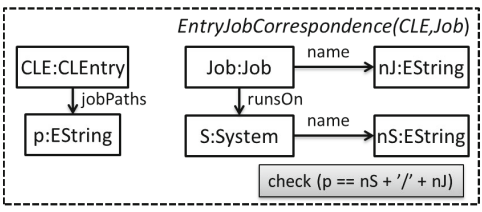
query language of EMF-IncQuery is used as the specification language for derived features. Therefore a brief introduction to this query language is provided first, followed by a detailed description on how this general purpose query language is adapted to specify the derived features for soft interconnections.

3.1 Model Queries by Graph Patterns: An Overview

Graph patterns [8] are 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, as well as *expressions* to define *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 not be satisfied). The complete query language of the EMF-IncQuery framework is described in [9], while several examples will be given below.

3.2 Soft Links as Model Queries

Sample Soft Link First, we demonstrate on an example how the graph pattern EntryJobCorrespondence(CLE,Job) (Figure 2) can be used to express the soft links captured by the derived EReference jobs (connecting *ChecklistEntry* and *Job* in Figure 1), that is, to identify those jobs that correspond to a task execution as part of the checklist entry.




```
1  // ChecklistEntry.jobs link
2  pattern EntryJobCorrespondence
3   (CLE, Job) = {
4    Job.name(Job,JobName);
5    System.name(System,SysName);
6    Job.runsOn(Job,System);
7    ChecklistEntry.jobPaths
8     (CLE,JobPath);
9    check(JobPath ==
10    SysName+'/'+JobName);}
```

Fig. 2. Model query to define EntryJobCorrespondence in graphical and textual syntax

This model query formulated as a graph pattern has two parameters: CLE and Job, denoting the source and the target end of the soft link. The query defines the designated set of jobs by checking the names of the given job element Job and the system S it *runs on* (nJ and nS, respectively) and the path p stored

in the entry. Model queries for the other soft links captured by derived features defined in the metamodel are defined similarly in Listing 1.1 and Listing 1.2.

```
// Job.tasks link
pattern JobTaskCorrespondence
 (Job,Task) =
{
  Task.id(Task,TaskId);
  Job.taskIds
   (Job,TaskId);
}
// Data.readingTasks link
pattern DataTaskReadCorrespondence
 (Data,Task) = {
  Task.id(Task,TaskId);
  Data.readingTaskIds
   (Data,TaskId);}
// Data.writingTasks link
pattern DataTaskWriteCorrespondence
 (Data,Task) = {
  Task.id(Task,TaskId);
  Data.writingTaskIds
   (Data,TaskId);}
```

Listing 1.1. Resource-Process mapping

```
// Job.info link
pattern JobInfoCorrespondence
 (Job,Info) = {
  ChecklistEntry.info(CLE,Info);
  RuntimeInformation.id
   (Info,InfoId);
  find EntryJobCorrespondence
   (CLE, Job);}
// ChecklistEntry.task link
pattern EntryTaskCorrespondence
 (CLE, Task) = {
  Task.id(Task, TaskId);
  ChecklistEntry.taskId
   (CLE,TaskId);}
// Checklist.process link
pattern ListProcessCorrespondence
 (Checklist, Process) = {
  Process.id(Process,ProcessId);
  Checklist.processId
   (Checklist,ProcessId);}
```

Listing 1.2. Checklist entry mapping

The query language also supports the following language constructs:

- check(JobPath == SysName + '/' + JobName) checks that the model element bound to variable *JobPath* is equal to the concatenated value of *SysName* and *JobName* (note that the evaluation will use String.equals to compare the value of EStrings).
- Using the find keyword, graph patterns are allowed to reuse other graph patterns. Therefore, if a soft link is defined as a model query by a corresponding graph pattern, this definition can be reused in other queries, and thus, in other soft links (along derived features).

The soft links defined as model queries using the graph pattern based language of EMF-IncQuery in the case study have two parameters, the first parameter denotes the source (i.e. the container EClass) while the second parameter denotes the target of the soft link. However, in the actual query language, this rule can also be satisfied by using *pattern annotations* for multi-parameter queries that explicitly specify which of the parameters is the context and which one will correspond to the target (or value). Furthermore, the adherence to this rule is checked at editing time by a built-in query language validator in the EMF-IncQuery tooling [6].

4 From Incremental Query Evaluation to Soft Links

In this section, we outline how the soft links can be managed using the efficient querying features of the EMF-IncQuery framework. Our approach can

be integrated to notification based applications (like EMF) in a deep and transparent way by mapping model changes to the values of derived features using incremental evaluation.

4.1 Incremental Evaluation of Queries: an Overview

The key to efficient evaluation and change notification for derived features is the incremental graph pattern matching infrastructure of the EMF-IncQuery framework (first introduced in [10]), see the internal architecture in Figure 3.

The input for the incremental graph pattern matching process is the EMF instance model and its Notification API where callback functions can be registered to instance model elements that receive notification objects (e.g. ADD, REMOVE, SET etc.) when an elementary manipulation operation is carried out.

Based on a query specification, EMF-IncQuery constructs a RETE rule network [10] that processes the contents of the instance model to produce the query result at its output node. Query results are then post-processed by *auto-generated query components* to provide a type-safe access layer for easy integration into applications. This RETE network remains in operation as long as the query is needed: it continues to receive elementary change notifications and propagates them to produce *query result deltas* through its *delta monitor* facility, which are used to incrementally update the query result. These deltas can also be processed externally, which is a key feature for the integration of derived features (Section 4.2).

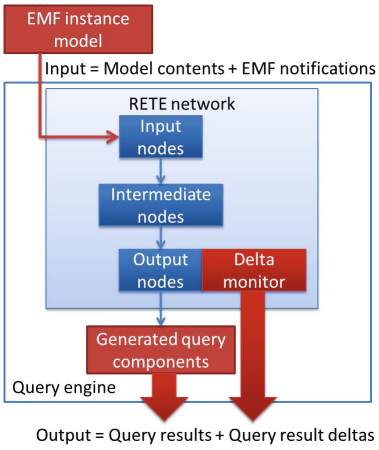


Fig. 3. The EMF-IncQuery architecture

By this approach, the query results (i.e. the match sets of graph patterns) are continuously maintained as an in-memory cache, and can be instantaneously retrieved. Even though this imposes a slight performance overhead on model manipulation, and a memory cost proportional to the cache size (approx. the size of match sets), EMF-IncQuery can evaluate very complex queries over large instance models very efficiently. These special performance characteristics, reported in [2], allow EMF-IncQuery-based derived features to be evaluated instantly in most cases, regardless of the complexity of the query or the size of the instance model.

4.2 Integration Architecture

To support soft links captured as derived features, the outputs of the EMF-IncQuery engine need to be integrated into the EMF model access layer at two points: (1) *query results* are provided in the getter functions of derived features, and (2) *query result deltas* are processed to generate EMF Notification objects that are passed through the standard EMF API so that application code can process them transparently. The overall architecture of our approach is shown in Figure 4.

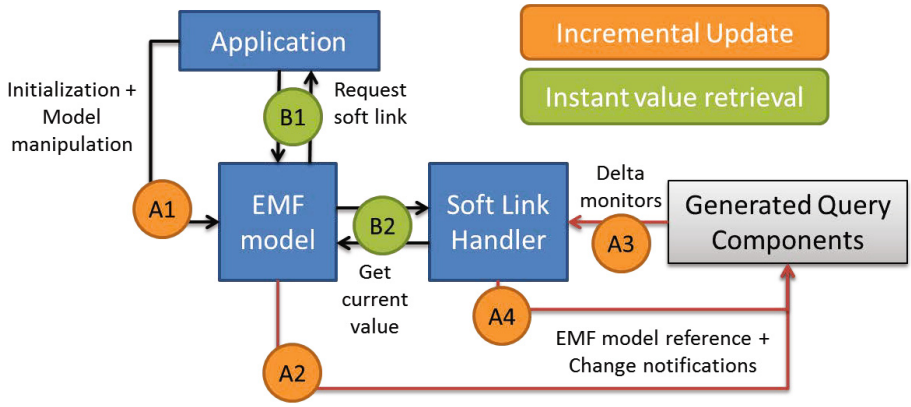


Fig. 4. Overview of the integration architecture, adopted from [11]

The application accesses both the model and the query results through the standard EMF model access layer – hence, no modification of application source code is necessary. In the background, as a novel feature, *soft link handlers* are attached to the EMF model objects that integrate the generated query components (pattern matchers). This approach follows the official EMF guidelines of implementing derived features and does not require more effort to integrate than ad-hoc Java code, or OCL expression evaluators. Note that these handlers can be used for managing regular derived features as well.

When an EMF application intends to read a soft link (B1), the current value is provided by the corresponding handler (B2) by simply retrieving the value from the cache of the related query. When the application modifies the EMF model (A1), this change is propagated to the generated query components of EMF-IncQuery along notifications (A2), which may update the delta monitors of the handlers (A3). Changes of soft links and derived features may in turn trigger further changes in the results sets of other derived features (A4).

Illustrative Example. Figure 5 illustrates a detailed elaboration EMF-IncQuery handlers, which process elementary model manipulation notifications to update, and generate notifications for derived features. The figure corresponds to a case

where the user assigns a new Job to a ChecklistEntry through the Editor which is essentially a cle.getJobPaths().add(jobPath) method call on the Model. During the add method, the ChecklistEntry EObject sends an ADD notification to the Notification Manager, which will notify the EMF-IncQuery Query Engine about the model modification. The Query Engine updates the match sets of each query and registers the match events in the Deltamonitor. Once its finished with updating the RETE network, it invokes the callback method of each IncqueryFeatureHandler. Each handler has a Deltamonitor from which it retrieves the new and lost match events since the last callback to processes them. During the processing, the handler may send notifications of its own (e.g. the value set of the info soft link of job is updated) that is propagated to listeners. Anytime the soft link value is retrieved from the model (e.g. job.getInfo()), it accesses the handler for the current value of the derived feature, which is returned instantly.

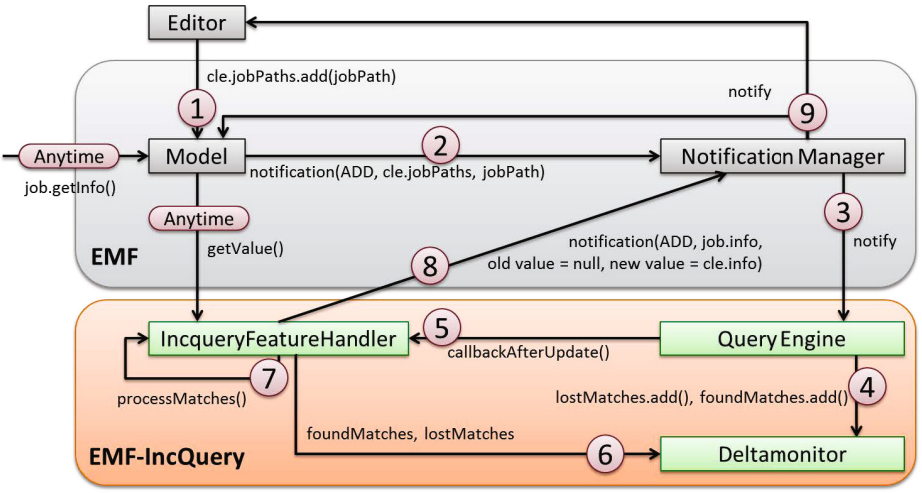


Fig. 5. Elaboration of the execution

Summary. In summary, the combined pattern matching and notification processing ensures that EMF-IncQuery-based soft links (and derived features) behave exactly as reeegular features of EMF instance models. This behavior ensures that user interfaces, model validators etc. can safely depend on soft interconnections built on soft links, without on-demand querying.

5 Applications in Traceability Modeling

The approach proposed in this paper can be interpreted in an external traceability modeling context. Figure 6 illustrates a typical architecture applied to

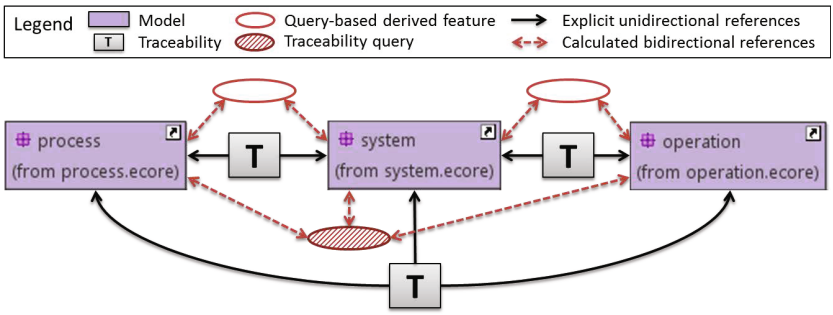


Fig. 6. External traceability modeling scenario

the examples of Section 2.1, where interconnections between three distinct models (belonging to the *process*, *system* and *operation* domains, respectively) are augmented with *explicit (external) traceability models* T.

In such a scenario, trace models T in EMF typically conform to a custom traceability metamodel that may describe simple binary (source-target) relationships with the help of association classes that use explicit unidirectional references to point to elements of the host models. In more complex cases, T may also include ternary (or hyper-) edges that interrelate multiple elements (e.g. three element types from all three domains, as in Figure 6).

5.1 Traceability-Specific Challenges

While this commonly used approach has an obvious advantage over *internal traceability/correspondence links* (as used in our previous examples), namely that the external models do not require the modification of the host metamodels, it also involves a number of frequently encountered problems as mentioned in Section 2.2:

- *Fragility*: Cross-resource hard EReferences are fragile, they may break when a host model is manipulated without the traceability model being loaded simultaneously. Additionally, in some scenarios, such as when using file-based EMF resources, traceability links may even break during external operations (e.g. when the files are moved within the workspace [12]).
- *Identification of target elements*: to work around the fragility issue, traceability modeling solutions may use IDs or fully qualified naming schemas (as presented in our previous examples) to store cross-references, even for external traceability models. However, such identifying attributes need to be present in the host models, and also necessitates an auxiliary mechanism that ensures consistency rules (such as uniqueness) within the host domains. If these prerequisites are not met, then additional, auxiliary techniques have to be used (such as ECore annotations, or genmodel modifications to add ID

maintenance capabilities to EMF domains, as used e.g. by EMFStore [13] and CDO [14]).
- *Persistence scalability issues*: in complex system modeling scenarios, the amount of EReferences can grow to be the dominant factor in storing the entire model space, in terms of both in-memory and serialized persistence overhead [2]. Hence, the performance of all model management-related operations (e.g. serialization) may be severely negatively affected as the size of the model resources grow, especially when taking the fragility issue into consideration (i.e. that traceability models with hard EReferences need to be loaded and manipulated together with host model fragments).

5.2 Traceability Management with Soft Links and Queries

The traceability architecture (components with black outline in Figure 6) can be augmented or even replaced with model-integrated *soft links* (symbolized by red outlined empty ovals) and *traceability queries* that can be accessed through the EMF-IncQuery API (oval with dashed fill). Both techniques share incremental, on-the-fly evaluation as their background.

Soft Links in a Traceability Context. From the traceability perspective, the most important advantages of soft links are that they are (logically) *bidirectional* references that are *maintained on-the-fly*. Thus, given that host metamodels are allowed to be augmented, such traceability links can be added without regard for circular serialization dependencies, that is, it is entirely up to the language designer to specify where such EReferences are going to reside, making trace link navigation also starting from host model elements feasible.

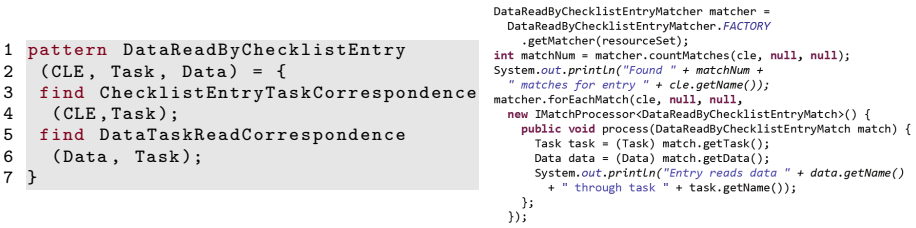
Additionally, as soft links provide graceful behavior for broken traceability references, erroneous trace records may be marked with warning markers, instead of throwing exceptions or runtime errors. These markers can then be corrected by e.g. a user-aided, on-demand resolution process, which may be further supported by helper queries that locate the most likely target host model element (esp. in the case when non-ID keys are used to identify model elements, such as EntryJobCorrespondence in Figure 2 – in this case, a helper query may enumerate those elements whose local names are similar).

As EMF-IncQuery query results can be represented by derived features as well as generic collections of EObjects, this feature may be used in a straightforward way to fine-tune which EReferences are going to be explicitly persisted and which ones are going to be calculated on-demand, when the models are loaded into memory. This gives the tool developer precise control over performance vs. compliance considerations (i.e. when certain traceability information is required to be stored persistently).

Finally, soft links behave exactly like normal EReferences (send notifications), easing the integration with user interface components or on-the-fly validators.

Using Traceability Queries for N-ary Links. If host metamodels cannot be modified, or hyperedges (multilinks, connecting three or more element types)

are desired for traceability modeling, the architecture of Figure 6 can be augmented with generic queries. Such a case is illustrated by Figure 7. In this case, a ChecklistEntry is connected to Tasks and, consecutively, to Data elements to represent the traceability information between data elements that are read by a given check list element. Such a ternary relationship (with * multiplicities) may be implemented by the DataReadByChecklistEntry pattern (shown on the left in Figure 7).



```
pattern DataReadByChecklistEntry
 (CLE, Task, Data) = {
 find ChecklistEntryTaskCorrespondence
  (CLE,Task);
 find DataTaskReadCorrespondence
  (Data, Task);
}
```

```
DataReadByChecklistEntryMatcher matcher =
  DataReadByChecklistEntryMatcher.FACTORY
    .getMatcher(resourceSet);
int matchNum = matcher.countMatches(cle, null, null);
System.out.println("Found " + matchNum +
  " matches for entry " + cle.getName());
matcher.forEachMatch(cle, null, null,
  new IMatchProcessor<DataReadByChecklistEntryMatch>() {
    public void process(DataReadByChecklistEntryMatch match) {
      Task task = (Task) match.getTask();
      Data data = (Data) match.getData();
      System.out.println("Entry reads data " + data.getName()
        + " through task " + task.getName());
    };
  });
```

Fig. 7. Ternary links with traceability queries

This approach shares the functional benefits of soft links, with the one exception that it is not integrated into the EMF model layer and as such, it is not API-transparent to EMF-based tools. Instead, the query results can be accessed through an additional API provided by EMF-IncQuery (illustrated on the right in Figure 7). Here, the results of the DataReadByChecklistEntry pattern are processed using a generated DataReadByChecklistEntryMatch data transfer class and the IMatchProcessor<> visitor interface. Though not shown in Figure 7, the EMF-IncQuery API also exposes the *delta monitor* facility (Section 4) that allows to track the changes in the result of such a query.

Summary. Soft links and traceability queries can be used to overcome the challenges presented by traceability-specific applications by complementing external traceability models and supporting incrementally maintained bidirectional links between interconnected model elements.

6 Related Work

In this section we first give an overview of existing approaches and tools that deal with interconnection between models, then we briefly describe other model query techniques for EMF. Finally, we list approaches that rely on derived features and therefore may take advantage of our incremental evaluation techniques.

Interconnecting EMF Models. In [15] correspondences between models are handled by matching rules defined in the Epsilon Comparison Language, where the application conditions (called guards) use queries similarly to our approach. Additionally, Epsilon also manages model integrity between EMF models using the

novel Concordance framework [12]. It is able to handle intermodel links when models are moved/renamed and helps in correcting invalid models caused by metamodel changes. Anwar [16] introduces a rule-driven approach for creating merged views of multiple separate UML models and relies on a correspondence metamodel and OCL expressions to support model merging and composition. VirtualEMF [17] allows the composition of multiple EMF models into a virtual model based on a composition metamodel, and provides both a model virtualization API and a linking API to manage these models. The approach is also able to add virtual links based on composition rules. In [18] an ATL-based method is presented for automatically synchronizing source and target models of a given transformation, based on the definition of the transformation.

Compared to them, the main distinctive features of our approach is (1) the fully incremental evaluation of queries for model interconnections, and (2) flexible support for query-based, computed soft links. It is a nice task for future research to combine the benefits of our current approach with the benefits of these existing solutions.

Model Query Approaches. OCL [19] 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 Eclipse OCL through its *Essential OCL* language provides a powerful query interface that evaluates OCL expressions over EMF models. Additionally, it also supports the definition of invariants and operations to enrich the Ecore metamodel using either the *Complete OCL* [20] or the *OCLinEcore* [21] languages. Balsters [22] presents an approach for defining database views in UML models as derived classes using OCL. The derived classes in this case are the result set of queries, which is similar to the match sets provided by EMF-IncQuery.

There are several technologies for providing declarative model queries over EMF, e.g. EMF Model Query 2 [23] and EMF Search [24]. Other graph pattern based techniques like [25,26] have been successfully applied in an EMF context.

Cabot et al. [27] present an algorithm for incremental runtime validation of OCL constraints and uses promising optimizations, however, it works only on boolean constraints. An interesting model validator over UML models [28] incrementally re-evaluates constraint instances whenever they are affected, but relies on environments that support the recording of read-only access to the model, unlike EMF. Additionally, general-purpose model querying is not viable.

These approaches provide possible alternatives to implement model queries, thus, they can potentially be used for providing soft links. However, many of them lack incremental evaluation support or require significantly more integration effort to enable their use for soft links.

Application of Derived Features. The PROGRES language [29] allows the rule-based programming of graph rewriting systems and uses derived attributes for encoding dynamic semantics. ConceptBase.cc [30] is a database (DB) system for metamodeling and method engineering and defines active rules that react to events and can update the DB or call external routines, the latter could be

applied in models as derived features representing data stored in the ConceptBase.cc DB. Neither tool has adopted EMF up to our best knowledge.

In [31] Diskin describes a formal framework for model synchronization that uses derived references for propagating changes between corresponding models. A recent work by Diskin et al. [32] proposes a theoretical background for model composition based on queries using Kleisli Categories, in their approach derived features are used for representing features merged from different metamodels. The conceptual basis is similar to our approach in using query-based derived features, however, it offers algebraic specification, while our approach might serve as an implementation for this generic theoretical framework.

The MOF 2.0 tool in [33] allows the definition of derived features using OCL. It handles derived attributes and operations as custom code provided by the user and redirects calls using reflection. The FUJABA [34] tool suite also supports derived edges by path expressions. Both tools work in a non-incremental way.

JastEMF [35] is a semantics-integrated metamodeling approach for EMF. It uses derived features as side-effect free operations (i.e. queries) and refers to them as the static semantics of the model. Therefore, our query-based approach could be integrated with JastEMF without any problems.

In a previous paper [11], we offer an algorithm for incremental evaluation of derived features and present technical details on the integration of existing native implementations. The current paper provides details on applying incremental queries for soft interconnections by using derived features in EMF.

7 Conclusion

Interconnections between model fragments of complex EMF models are usually represented as regular associations and persisted using storage-specific URIs. This approach proves to be rigid and error-prone in some application scenarios.

We proposed to use derived features as a flexible alternative to provide soft interlinking between model fragments, and demonstrated an approach for incremental evaluation of soft links with the use of model queries on an industrial case study. Our approach supports circular dependency between models, graceful handling for unresolved links and is implemented using EMF-IncQuery, which provides efficient evaluation capabilities for incremental model queries.

As a primary direction for future work, we plan to integrate traceability queries into the EMF model layer by constructing *derived classes* whose instances behave like EObjects but their lifecycles are managed by an underlying incremental query. Such constructs could be used to create n-ary traceability models that are automatically kept in-sync, retaining the graceful handling of soft links.

References

1. The Eclipse Project: Eclipse Modeling Framework, http://www.eclipse.org/emf
2. Bergmann, G., Horváth, Á., Ráth, I., Varró, D., Balogh, A., Balogh, Z., Ökrös, A.: Incremental Evaluation of Model Queries over EMF Models. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part I. LNCS, vol. 6394, pp. 76–90. Springer, Heidelberg (2010)
3. Workflow Management Coalition: XML Process Definition Language, v2.1. (2008), http://www.wfmc.org/xpdl.html
4. TIBCO Developer Network: TIBCO Business Studio (2012), http://developer.tibco.com/business_studio
5. AUTOSAR Consortium: The AUTOSAR Standard, http://www.autosar.org
6. Bergmann, G., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., Varró, D.: Integrating Efficient Model Queries in State-of-the-Art EMF Tools. In: Furia, C.A., Nanz, S. (eds.) TOOLS 2012. LNCS, vol. 7304, pp. 1–8. Springer, Heidelberg (2012), https://www.inf.mit.bme.hu/en/research/publications/integrating-efficient-model-queries-state-art-emf-tools
7. Bergmann, G., Ráth, I., Szabó, T., Torrini, P., Varró, D.: Incremental pattern matching for the efficient computation of transitive closures. In: Sixth International Conference on Graph Transformation, Bremen, Germany (submitted, 2012)
8. Varró, D., Balogh, A.: The Model Transformation Language of the VIATRA2 Framework. Science of Computer Programming 68(3), 214–234 (2007)
9. Bergmann, G., Ujhelyi, Z., Ráth, I., Varró, D.: A Graph Query Language for EMF Models. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 167–182. Springer, Heidelberg (2011)
10. Ráth, I., Bergmann, G., Ökrös, A., Varró, D.: Live Model Transformations Driven by Incremental Pattern Matching. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 107–121. Springer, Heidelberg (2008)
11. Ráth, I., Hegedüs, Á., Varró, D.: Derived Features for EMF by Integrating Advanced Model Queries. In: Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H., Kolovos, D. (eds.) ECMFA 2012. LNCS, vol. 7349, pp. 102–117. Springer, Heidelberg (2012), https://viatra.inf.mit.bme.hu/sites/viatra.inf.mit.bme.hu/files/attachments/ecmfa2012.pdf
12. Rose, L., Kolovos, D., Drivalos, N., Williams, J., Paige, R., Polack, F., Fernandes, K.: Concordance: A Framework for Managing Model Integrity. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 245–260. Springer, Heidelberg (2010)
13. The Eclipse Project: EMFStore (2012), http://www.eclipse.org/emfstore
14. The Eclipse Project: The CDO Model Repository (2012), http://www.eclipse.org/cdo
15. Kolovos, D.S.: Establishing Correspondences between Models with the Epsilon Comparison Language. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 146–157. Springer, Heidelberg (2009)
16. Anwar, A., Ebersold, S., Coulette, B., Nassar, M., Kriouile, A.: A rule-driven approach for composing viewpoint-oriented models. Journal of Object Technology 9(2), 89–114 (2010)
17. Clasen, C., Jouault, F., Cabot, J.: Virtual Composition of EMF Models. In: 7èmes Journées sur l'Ingénierie Dirigée par les Modèles (IDM 2011), Lille, France (2011)

18. Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.: Towards automatic model synchronization from model transformations. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, ASE 2007, pp. 164–173. ACM, New York (2007)
19. The Object Management Group: Object Constraint Language, v2.0 (May 2006), http://www.omg.org/spec/OCL/2.0
20. Willink, E.D.: Aligning ocl with uml. ECEASST 44 (2011)
21. Eclipsepedia: MDT/OCLinEcore (2012), http://wiki.eclipse.org/MDT/OCLinEcorel
22. Balsters, H.: Modelling Database Views with Derived Classes in the UML/OCL-framework. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 295–309. Springer, Heidelberg (2003)
23. The Eclipse Project: EMF Model Query 2, http://wiki.eclipse.org/EMF/Query2
24. The Eclipse Project: EMFT Search, http://www.eclipse.org/modeling/emft/?project=search
25. Biermann, E., Ermel, C., Taentzer, G.: Precise Semantics of EMF Model Transformations by Graph Transformation. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 53–67. Springer, Heidelberg (2008)
26. Giese, H., Hildebrandt, S., Seibel, A.: Improved flexibility and scalability by interpreting story diagrams. In: Proceedings of GT-VMT 2009, vol. 18. ECEASST (2009)
27. Cabot, J., Teniente, E.: Incremental integrity checking of UML/OCL conceptual schemas. J. Syst. Softw. 82(9), 1459–1478 (2009)
28. Groher, I., Reder, A., Egyed, A.: Incremental Consistency Checking of Dynamic Constraints. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 203–217. Springer, Heidelberg (2010)
29. Schürr, A.: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language. In: Nagl, M. (ed.) WG 1989. LNCS, vol. 411, pp. 151–165. Springer, Heidelberg (1990)
30. Jeusfeld, M.A., Jarke, M., Mylopoulos, J.: Metamodeling for Method Engineering. The MIT Press (2009)
31. Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (eds.) GTTSE 2011. LNCS, vol. 6491, pp. 92–165. Springer, Heidelberg (2011)
32. Diskin, Z., Maibaum, T., Czarnecki, K.: Intermodeling, Queries, and Kleisli Categories. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 163–177. Springer, Heidelberg (2012)
33. Scheidgen, M.: On implementing mof 2.0—new features for modelling language abstractions (2005)
34. Nickel, U., Niere, J., Zündorf, A.: The FUJABA environment. In: Proc. ICSE 2000, pp. 742–745 (2000)
35. Bürger, C., Karol, S., Wende, C., Aßmann, U.: Reference Attribute Grammars for Metamodel Semantics. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 22–41. Springer, Heidelberg (2011)