

# Pattern-based generation to augment EMF editors with business rules, materialized views, statecharts, transaction demarcation, ORM persistence, and GUI generation hints

Miguel Garcia

Software Technology and Systems Institute (STS),  
Hamburg University of Science and Technology (TUHH),  
Harburger Schlosstr. 20, D-21079 Hamburg, Germany  
[miguel.garcia@tuhh.de](mailto:miguel.garcia@tuhh.de)  
<http://www.sts.tu-harburg.de/~mi.garcia/>

## Table of Contents

1. Motivation.....	2
2. Sub-projects .....	2
2.1 The authoring environment.....	2
2.2 OCL Expressions and Materialized views .....	3
2.3 Reactive behavior: General considerations.....	5
2.4 Reactive behavior: Implementation aspects of Business Rules.....	6
2.5 Reactive behavior: Implementation aspects of Statecharts.....	7
2.6 GUI generation.....	8
2.7 ORM Persistence .....	9
3. Team members and work division.....	11
3.1 Supervision .....	11
References .....	12
Appendix I: Depends-on Analysis of OCL Expressions .....	13
I.a Problem statement.....	13
I.b A more detailed example .....	15

## **1. Motivation**

EMF has raised the bar for modeling infrastructures. It is no accident that developers tap into the capabilities of EMF progressively, adopting patterns along a learning process. We have developed prototypes that shield the developer from coding by hand higher-level abstractions in EMF-based implementations. Our strategy revolves around providing domain-specific language constructs for such abstractions, encoding the knowledge about their realization in translation algorithms. The resulting DSL embodies concepts familiar to the developer: business rules, materialized views, statecharts, transaction demarcation, ORM persistence, and GUI generation hints. The interaction between these capabilities is taken care of by the translation algorithms.

In order to perform the detail-level engineering of these prototypes we have scheduled a number of master theses, for which funding is requested. Project management and dissemination activities have also been scheduled as part of the work package, in order to turn our prototypes into turn-key solutions. In case the grant is not received progress will still be made (for we and our students are convinced of the benefits of the approach) yet at a much slower pace.

During this project technical documentation and wikis will be prepared, describing the generated patterns as well as the generation mechanisms. So far we've found that most of the technical documentation available on EMF comes from Eclipse teams only, and that those technical reports prepared in universities during projects using EMF sometimes do not fully convey best practices on EMF programming. We plan to make a positive contribution in this regard also.

All prototypes and technical reports described in this proposal are available upon request, for use as part of the evaluation of this submission.

## **2. Sub-projects**

### **2.1 The authoring environment**

The translation techniques integral to this submission take as input Abstract Syntax Trees for UML class models, OCL, statecharts, and business rules. We rely on existing authoring environments for UML class models and OCL, while strive for in-house developments of editors for statecharts and business rules.

As for statecharts, a prototype built with Eclipse GMF has proved successful, allowing us to conclude that GMF will be used to jumpstart the final statechart editor, with hand-coded customizations to increase its usability. The increasing reliance of GMF on OCL for defining the well-formedness of visual syntax eased the adoption of GMF in our team.

As to the editor for business rules, we plan to develop a custom text editor, following best practices known for Eclipse-based editors. We are following developments around support for textual notation (e.g. generation of custom text editors) by Xactium.com, the Modeling Project in Eclipse, Chris Laffra's GUIDE and Gymnast, and Structured Source Editors in the Web Tools project, among other sources. The facilities for autocompletion advocated in EMFT OCL are likely to be adopted in our language metamodels (business rules, statecharts).

Regarding the editing of UML class models and OCL, when the development of our prototype started we searched for a tool inspired in compiler technology, that took care of the details of parsing, type binding, variables-to-usages resolving, and cross-checking the consistency of OCL expressions against the underlying UML class model. A decision was made to temporarily architect our main prototype as an extension to the BSD-licensed Octopus OCL tool, <http://octopus.sourceforge.net>. Although Eclipse-based, it constitutes a piece of non-EMF technologies. We see several alternatives for migrating during this project to an EMF-only tooling (and to EPL-only license).

One alternative consists in waiting for an EPL-licensed UML2 editor, an editor that moreover allows for integrated editing of OCL. Failing that, next-best options consist in transforming the Abstract Syntax Trees from Octopus into their EMF-based counterparts (Eclipse UML2 and EMFT OCL) early in the transformation chain. XMI import of UML + OCL can be offered under all alternatives, yet we resist the idea of offering it as the *only* UML + OCL input mechanism. Building a GMF-based diagram editor for the UML class model with OCL as textual input is also within reach, yet we tend to see that as a distraction from our main goals. The added value of our project lies instead in the transformation algorithms explained next.

## 2.2 OCL Expressions and Materialized views

Consider the case where domain modeling has resulted in a number of global invariants, encoded in OCL. We want to be notified at runtime of broken invariants, in the form of pairs (object, invariant) made available automatically for us at transaction-end time. Moreover, such bookkeeping should be efficient, in the sense that no invariants should be evaluated unless the data locations they depend on have been updated. The design patterns employed to realize these requirements are no longer trivial. In fact, this use case is not supported out-of-the-box by EMFT OCL Validation.

Although EMFT Validation can evaluate an OCL invariant upon update of a structural feature in the invariant's class, it is still the modeler's responsibility to correctly specify (beyond the OCL invariant itself) all data locations that it depends on. Additionally, a deferred evaluation of an invariant (at transaction end time) would be preferable, as opposed to immediately after an individual update operation. As of now, EMF.Editor generates code to check for multiplicity constraints, and only for that.

For the same reasons, operation preconditions benefit from a depends-on analysis to skip evaluations whose result has not been affected by updates. Details of this analysis are provided in *Appendix I: Depends-on Analysis of OCL Expressions*.

Another recurring design pattern is that of in-memory materialized views. Similar to their database counterpart, we want a resultset to be (efficiently) updated whenever its base information changes. This functionality is fundamental and goes by different names according to the chosen implementation mechanism: data-driven computation, Rete algorithm, update percolation.

As to simple OCL queries, optimization measures involve: (a) for interpreted OCL queries, parsing the text representation at class-level, only once for each query, (b) modeling-time, semantics-preserving rewriting of OCL expressions, and (c) compilation instead of interpretation.

Item (b) refers to the simplification of an OCL expression as a result of evaluating constant expressions [1] and applying partial-evaluation techniques [2]. This simplification is performed before the query is evaluated, either by interpretation or by compilation.

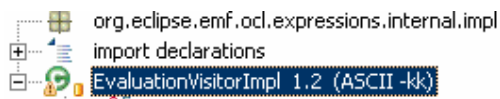
OCL query interpretation follows patterns like: (reproduced from EMFT OCL documentation)

```
Query query = QueryFactory.eINSTANCE.createQuery(
    "Book.allInstances()->select(b : Book | b <> self and
    b.title = self.title)",
    LibraryPackage.eINSTANCE.getBook());

query.setExtentMap(extents);

Collection result = query.evaluate(myBook);
System.out.println(result);
```

Internally, the following visitor is run:



In contrast, a compilation approach generates a method body consisting of statements such as:

```

/**
 * Implements the getter for feature '+ duplicateTitles : Set(Book)'
 *
 * def: duplicateTitles : Set(Book) =
 *   Book.allInstances()->select(b : Book |
 *     b <> self and b.title = self.title)
 */
public Set<Book> getDuplicateTitles() {
    Set<Book> result = new HashSet<Book>();
    Iterator it = Book.allInstances().iterator();
    while (it.hasNext()) {
        Book b = (Book) it.next();
        if (!b.equals(this) && b.getTitle().equals(this.getTitle())) {
            result.add(b);
        }
    }
    return result;
}

```

The code above is shown for illustration only. It is generated by an extension to the Octopus OCL tool, an extension developed in our team. An EMF-based implementation is amenable to optimizations, such as automatic maintenance of *allInstances* (as in-memory materialized view) in a way that does not interfere with garbage collection, among other improvements. The generation implementation in turn would rely on VEP's JEM (Java EMF Model) for the visitor-based generation, and JET for serialization.

If time permits, advanced features such as a natural language rephrasing of a broken invariant will also be explored, taking as starting point the OCL processing algorithms described in the Master thesis <http://www.cs.chalmers.se/~krijo/gfspec/burke/> and implemented in <http://www.cs.chalmers.se/~krijo/gfspec/ocl2nl.html>

## 2.3 Reactive behavior: General considerations

The query-only capabilities generated so far simplify the implementation of behavior, behavior which was specified in a declarative manner at the DSL level.

Our experience shows that developers use business rules provided that an IDE extension, such as ILOG Business Rules, shields them from coding the interaction between the rule engine and the domain objects. Exactly this principle we are in a position to follow, for we control the language for expressing the actions to be triggered. This action language is to follow the mature design of action languages for Executable UML [3], standardized in UML 1.5. Tools like Kennedy Carter's iUML, BridgePoint, and PathFinder, prove that behavior can be specified at the model level and translated to efficient code. Prototypes in the "UML Virtual Machine" category attest to the growing importance of this technology. Unlike UML Virtual Machines, where a runtime system works with POJOs, we aim at generating implementations where EMF objects are managed.

Allowing the inclusion of arbitrary Java or scripting statements for actions turns out to be a disservice to the programmer, for modeling-time analyses which can be performed on Action Language's Abstract Syntax Trees are not feasible for arbitrary Java or Java-like scripting languages (for example, determining if an action is side-

effects-free with respect to a component, i.e. if the action only external entities are notified as a result of the action's execution.)

The query capabilities and the action language are a natural building block for business rules and for statecharts. The triggering condition of a business rule is expressed in OCL, as well as the guards of statecharts. The triggered action (of a business rule) and the transition's actions (in a statechart) are expressed using the same Action Language. This way, we are able to intercept all relevant updates of structural features and operation invocations because of our strict adherence to Abstract Syntax Trees for all offered high-level abstractions.

A balancing act is to be performed during the detail-level engineering of the action language. Overly restrictive constructs allow for additional analyses, yet drive developers away. For example, a while-loop construct can be so defined as to require that, at runtime, the set of items to be iterated over should be known before entering the loop body, conceptually held in a non-mutable collection (thus remaining constant throughout the loop.) This restriction allows Hoare-logic-style analyses on whether the actions guarantee that the operation's postconditions will be established, for all inputs that satisfy the preconditions. While valuable in some circumstances, a trade-off will be performed in view of the expected use of our tooling analyses.

## **2.4 Reactive behavior: Implementation aspects of Business Rules**

As a note on terminology, we've settled in this document for the term "business rules" as a synonym for "forward-chaining rules" and "production rules".

In the context of the implementation of business rules, in order to detect the activation of a triggering condition we build upon the notification mechanism of EMF, as it provides the low-level Observer pattern on structural features (but not on operations). However, additional programming is required to detect the aggregation of notifications that constitute a composite activation condition. Moreover, without a Rete-algorithm implementation, unnecessary evaluations will degrade performance.

In summary, for business rules specified alongside the input models to be translated, we aim at generating an implementation that realizes nothing short of the Rete algorithm on EMF models. Previous work towards the same goal was carried out in our team in the context of a master thesis [<sup>4</sup>], where generated AspectJ advices that gathered updates to data locations involved in triggering conditions. This implementation still relied on a separate rule engine to perform Rete-processing, yet the activation conditions were already expressed in OCL instead of the proprietary syntax of a rule engine. The insights gained in that project are directly applicable to an EMF implementation.

We aim at letting the modeler choose whether the generated EMF-based implementation allows for updatable rulesets at runtime, a feature that makes business rules attractive in comparison to other behavior-specification alternatives.

## 2.5 Reactive behavior: Implementation aspects of Statecharts

In the context of the implementation of statecharts, detection of an event occurrence does not have to rely on EMF notifications (but see the discussion near the end of this section). The specific detection mechanism depends actually on the implementation strategy chosen for statecharts, for which there's an abundance of recipes in the literature. We settle for letting the modeler choose between two implementation alternatives: compilation or interpretation.

Irrespective of the realization approach, the chosen statechart language has to support a minimum of proven constructs in order to be useful: hierarchical states, orthogonal regions, history, entry/exit actions, and communication with other statecharts over event queues. All along we leave concurrency and transaction support to the underlying infrastructure, assuming either (a) that statecharts are invoked from a single thread; or (b) that declarative transaction demarcation is enough to ensure consistency. In order to avoid race conditions a simple method-level locking strategy can be included in the generated actions for transitions, but other than that concurrency handling lies outside the scope of this proposal. We do provide however for the well-formedness analysis of statechart Abstract Syntax Trees, as for example to disallow at modeling time a transition (ending at state S1) whose action raises an event to transition to a different state S2. The statechart semantics to follow is that of Rhapsody [5].

In the compilation approach, an event is consumed when its associated method is invoked. The generated method body is responsible for:

- Throwing an invalid state exception if there's no outgoing transition labeled with that event in the current state configuration
- Finding the innermost state  $S_{OUT}$  with an outgoing transition labeled with the event being consumed.
- Evaluating its guard (which is side-effects free by virtue of OCL). If false, remain at current state, the event is considered processed.
- Execute the (nested, from innermost to outermost) exit actions up to  $S_{OUT}$ , recording for history purposes the last state at each level
- Execute the transition's actions, passing event parameters if any
- Execute the on entry actions from the outermost target state to its innermost one, choosing history states if present, updating the state configuration along the way.

As can be seen, under normal development conditions, an EMF developer cannot be expected to implement anew the above design pattern whenever a statechart is needed. Notice also that we've elided handling orthogonal regions and inter-statechart communication over event queues.

We plan to field-test the statechart compilation capability by specifying GUI-level behavior, following the case study of the book *Constructing the User Interface with Statecharts* [6]. We have tried the Unimod tool [7] for this purpose, but we were left unconvinced by its non-standard notation and by the fact that it manages POJOs and not EMF Objects.

As for the interpretation approach, it has the unique advantage that the runtime representation of a statechart can be updated or replaced altogether at runtime (while in a quiescent state), provided that the external event interface remains stable. This capability is a pre-requisite for *adaptive processes*, functionality exhibited by prototypes such as the following:

- a) Stefano Ceri, Peter Dolog, Maristella Matera, and Wolfgang Nejdl:  
Adding Client-Side Adaptation to the Conceptual Design of e-Learning Web Applications  
Journal of Web Engineering, 4(1):21-37, March 2005  
<http://www.l3s.de/~dolog/pub/jwe2005.pdf>
- b) Reichert, M.; Rinderle, S.; Kreher, U.; Dadam, P.:  
Adaptive Process Management with ADEPT2.  
Proc. Int'l Conf. on Data Engineering, ICDE 2005, Tokyo, April 2005, Demo Session  
<http://www.informatik.uni-ulm.de/t3-aristaflow/fileadmin/publications/ulm/RRKD05.pdf>

Not everyone is asked to develop adaptive applications, where the workflow needs to be modified at runtime. We believe the cause-effect relationship to be the opposite: lack of infrastructural support for adaptive processes prevents their widespread use.

The design patterns to generate for a statechart interpreted are similar to those found in a traditional interpreter. This time the EMF notification mechanism can be put to use, given that events are reified and assigned to structural features. Moreover, Commands in EMF.Edit are an excellent realization vehicle for events. Custom values in genmodel are necessary to prevent generation of EMF.Edit commands that do not make sense for a statechart implementation, as for example deleting an event.

## 2.6 GUI generation

The generated EMF-based implementations can be used as the Model (in MVC sense) in sophisticated modeling or information management tools (supporting use cases such as *live document editing*, *constraint-based visual editing*, *warnings as-you-type*). However, the software artifacts necessary to provide a user experience exhibiting such functionality involve GUI functionality whose generation has not been discussed so far.

The phrase “GUI generation” is commonly equated to mean CRUD (Create-Retrieve-Update-Delete). The problem with generators of CRUD GUIs is that the only model-level information they have at their disposal is the class model without OCL. On another extreme, some research prototypes demand an explicit User Task Model to generate GUIs that are aware of the workflows to enforce at the GUI level. While the generated GUI is aware of workflows, the price to pay is the cumbersome maintenance of these detailed task models. Somewhere in between approaches such as WebML ([www.webml.org](http://www.webml.org)) promote a concise specification of workflows, a specification that is amenable to refactoring, that takes into account different user roles, and one that moreover is platform independent.

We plan to leverage the information specified not only in the class model but also in the OCL and statechart specifications when generating an editor. Consider what happens once the mapping between GUI gestures and statechart events or class operations has been specified. Those events which are not allowed for the current

statechart state, as well as those operations whose precondition is not fulfilled, should be grayed out in the GUI. Letting generators weave this functionality increases the ROI that modelers get from the time invested in adding statecharts, OCL, and business rules at the model level.

Consider for example the following code snippet about enabling or not an action depending on the current selection (reproduced from the Eclipse RCP book):

```
org.eclipse.rcp.hyperbola/AddContactAction
public void selectionChanged(
    IWorkbenchPart part, ISelection incoming) {
    // Selection containing elements
    if (incoming instanceof IStructuredSelection) {
        selection = (IStructuredSelection) incoming;
        setEnabled(selection.size() == 1 &&
            selection.getFirstElement() instanceof ContactsGroup);
    } else {
        // Other selections, for example containing text or of other kinds.
        setEnabled(false);
    }
}
```

A question to ask is: which part of this code *cannot* be generated from information already available in the set of models that our tools process? In fact, all of it can be generated, and in this case, no further customization is necessary.

As another example, the EMF book details how to constrain the targets of a reference offered in a combo box. Without demanding programming, the same can be achieved by an OCL invariant specifying the set of valid items for a structural feature. The generator, either targeting a forms-and-menus interface or a GMF-based one, can follow this contract.

We aim at generating EMF.Edit commands and an RCP-based GUI with support for:

- Invocation of operations
- A temporary storage location to instantiate parameters
- Choices for binding GUI gestures to statechart events
- Reliance on OCL to specify valid values at different phases of interaction
- Light-weight navigation modeling

The proposed architecture opens the door in the future to generation of non-SWT based editors, given the platform-independent nature of the GUI hints and navigation model. This task however lies outside the scope of this proposal.

The design of this generator will take into account the ideas on hints for GUI generation, and their runtime support, described by Michael Scharf in “Annotations to Customize EMF Editors” at <http://www.eclipsecon.org/2006/Sub.do?id=423>

## 2.7 ORM Persistence

While direct interpretation of an OCL query over an EMF model persisted in an RDBMS does (conceptually) get the job done of retrieving from the RDBMS the

objects of interest, in practice this approach does not scale. Performance succumbs because of the big-inhale problem and due to in-memory nested object navigation (by “big-inhale” is meant bringing to main memory large sets of objects instead of letting the RDBMS process them in place.)

We’ve designed and implemented non-trivial compile-time reformulations of OCL queries for RDBMS-side processing. As part of this proposal, we plan to migrate them to our EMF tooling. In a nutshell, the translation from OCL first prepares a relational query in terms of table projections, Cartesian products, natural joins, and SQL built-in functions. This syntax-independent query can be translated to EJB3QL as well as to SQL’92, provided the DB schema to which the EMF model is mapped is known. The correctness of the translation can be assured (i.e. both formulations return the same resultset). Additionally, readable textual EJB3QL or SQL’92 is obtained, after eliminating double negations and other pretty-printing rules. A simple, one-pass translation (as for example with a template-engine), does not exhibit these advantages.

We see the rewriting of OCL into efficient EJB3QL or SQL queries as critical for the real-world usage of ORM mapping for EMF, and we don’t see this issue being addressed by the existing ORM prototypes available from EMF Corner.

With these techniques, the model-conformant ORM generator for JSR-220 Persistence that has been developed and field-tested in our team is able to scale to large data sets of POJOs.

EMF supports at runtime both compile-time generated and runtime defined eclasses. Usually, an ORM solution assumes a static schema, with a corresponding static DB schema. For supporting EMF-based persistence, besides DB tables whose structure was determined at compile-time, additional tables are needed to store dynamically defined EMF objects. This affects our proposed rewriting of OCL queries into EJB3QL, as well as some DB schema management utilities (for example, to materialize in the DB schema tables for the dynamically introduced EMF classes, migration of the eobjects to those tables, etc)

An evaluation of the impact on performance for different design alternatives will be carried out, in order to document the chosen trade-off between flexibility and performance.

Moreover, we expect the persistence concern not to result in a tangled programming model (where domain-related statements end up intermingled with JSR-220-specific code). Instead, we aim at relying on annotations (for transaction demarcation, as in EJB) to cater for future use cases such as swapping an ORM runtime with an another targeting instead an XML database, for example.

Research on persistence solutions for object systems with dynamic classes (CLOS) [8] was carried out at Hamburg University and those research results are known to members of our team.

### 3. Team members and work division

#### 3.1 Supervision

	Prof. Dr. Ralf Möller <a href="http://www.sts.tu-harburg.de/~r.f.moeller/">http://www.sts.tu-harburg.de/~r.f.moeller/</a>
Background	Description Logics, Conceptual Data Models, Distributed Systems, Semantic Web, Schema Integration, Data Integration, Stream-Oriented Processing, Semantic Middleware, Software-Engineering, Software Technology
Responsibilities in the project	Overall project supervision.

	Miguel Garcia, M.Sc. <a href="http://www.sts.tu-harburg.de/~mi.garcia/">http://www.sts.tu-harburg.de/~mi.garcia/</a> <a href="http://groups.yahoo.com/group/octopus-is-ocl-for-mda/">http://groups.yahoo.com/group/octopus-is-ocl-for-mda/</a>
Background	Ph.D. candidate. His main interests are the application of model compilers in the field of enterprise software architecture. Mr. Garcia designed and implemented the depends-on analysis for OCL expressions, and has co-supervised a number of master theses on tooling for model-driven development processes.
Responsibilities in the project	Language definition, ensuring that the translation algorithms correctly weave the capabilities expressed as high-level abstractions, implementation of the Rete-algorithm on EMF models for business rules.

## References

- 
- <sup>1</sup> Constant folding: Information from Wikipedia.com, <[http://en.wikipedia.org/wiki/Constant\\_propagation](http://en.wikipedia.org/wiki/Constant_propagation)>, date accessed: 30 March 2006.
- <sup>2</sup> M. Giese, D. Larsson. Simplifying Transformations of OCL Constraints. Proceedings of Models 2005, Montego Bay, <http://www.ricam.oeaw.ac.at/people/page/giese/pub/models05.pdf>
- <sup>3</sup> C. Raistrick, P. Francis, J. Wright, C. Carter and I. Wilkie, Model Driven Architecture with Executable UML, Cambridge University Press, March, 2004.
- <sup>4</sup> Meng Xue, Prototypes for Solving Consistency Problems in Model Engineering, Diplomarbeit, TU Hamburg-Harburg, January 2006.
- <sup>5</sup> D. Harel and H. Kugler, "The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML)", Integration of Software Specification Techniques for Applications in Engineering, (H. Ehrig et al., eds.), Lecture Notes in Computer Science, Vol. 3147, Springer-Verlag, 2004, pp. 325-354 <http://www.wisdom.weizmann.ac.il/~dharel/papers/RhapsodySemantics.pdf>
- <sup>6</sup> Constructing the User Interface with Statecharts, by Ian Horrocks, Addison-Wesley, 1998. ISBN 0-201-34278-2
- <sup>7</sup> <http://unimod.sourceforge.net/>. An Eclipse IDE extension and runtime framework for the visual editing, execution, and debugging of statecharts.
- <sup>8</sup> Heiko Kirschke. Technical Report FBI-HH-B-179/95: Persistence in the object-oriented programming language CLOS shown with the PLOB! system (in German). <http://lki-www.informatik.uni-hamburg.de/~kirschke/diplom/bericht.ps>

## Appendix I: Depends-on Analysis of OCL Expressions

### I.a Problem statement

Although EMFT Validation can evaluate an OCL invariant upon occurrence of some event, it is still the modeler's responsibility to correctly specify (beyond the OCL invariant itself) all data locations that it depends on. Additionally, a deferred evaluation of an invariant (at transaction end time) would be preferable, as opposed to immediately after an individual update operation. As of now, EMF.Editor generates code to check for multiplicity constraints, and only for that.

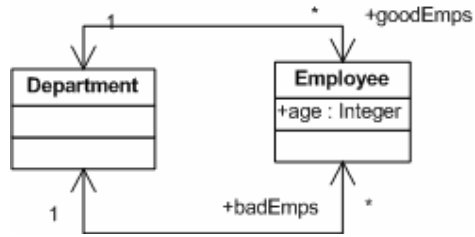
In order to relate this discussion to current invariant support, a Live OCL invariant specified as per EMFT Validation is shown next, as an extension to `org.eclipse.emf.validation.constraintProviders`:

```
<constraint
  lang="OCL"
  severity="ERROR"
  mode="Live"
  name="%example2.name"
  id="example2"
  statusCode="102">
  <description>%example2.desc</description>
  <message>%example2.msg</message>
  <!-- This constraint applies to books, triggered when
        the author reference is set. -->
  <target class="Book">
    <event name="Set">
      <feature name="author"/>
    </event>
  </target>

  <!-- Books must have authors. -->
  not author.oclIsUndefined()
</constraint>
```

If localizable messages are used (such as “`%example2.name`” in the example) additionally a `plugin.properties` file with at least an English version has to be generated.

The complete set of events that may affect the value of an OCL invariant (or in general, an OCL expression) can be determined after computing the depends-on tree of the OCL expression. The following example illustrates this task:



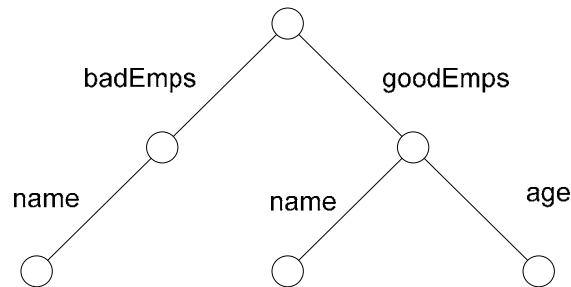
**Fig. 2. A department with good and bad employees.**

The invariant noJohn (for a Department) requires the union of two sets (all bad employees and those good employees over forty) to contain no one called John:

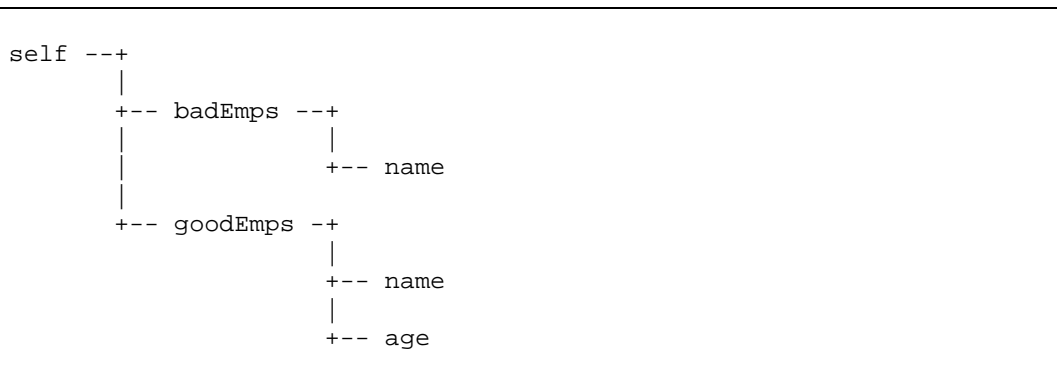
```

context Department
inv noJohn :
  badEmps->union(goodEmps->select(age > 40))
  ->select(name = 'John')->isEmpty()
  
```

Renaming any employee (good or bad) may affect the value of the invariant (among other events). However, changing the age of only a good employee may affect this invariant. The resulting depends-on tree is depicted on Figure 3.



**Fig. 3. Depends-on analysis for the noJohn invariant.**



To correctly determine that the age attribute accessed over goodEmps but not over badEmps has an effect on the value of the invariant, a new notion is needed, for the recursive depends-on analysis of the AST of an OCL expression: an OCL expression is reduced to two sets of navigation paths, those further navigable (f-n paths) and those not (n-f-n paths). These terms will be used in Sect. 3 to describe the reduction of OCL loop operations (which apply a property to the items in a source collection).

## I.b A more detailed example

As a second example, for the following OCL invariant

```

context LoyaltyProgram
  inv noAccounts: partners.deliveredServices->
    forAll(pointsEarned = 0 and pointsBurned = 0 )
      implies Membership.account->isEmpty()
  
```

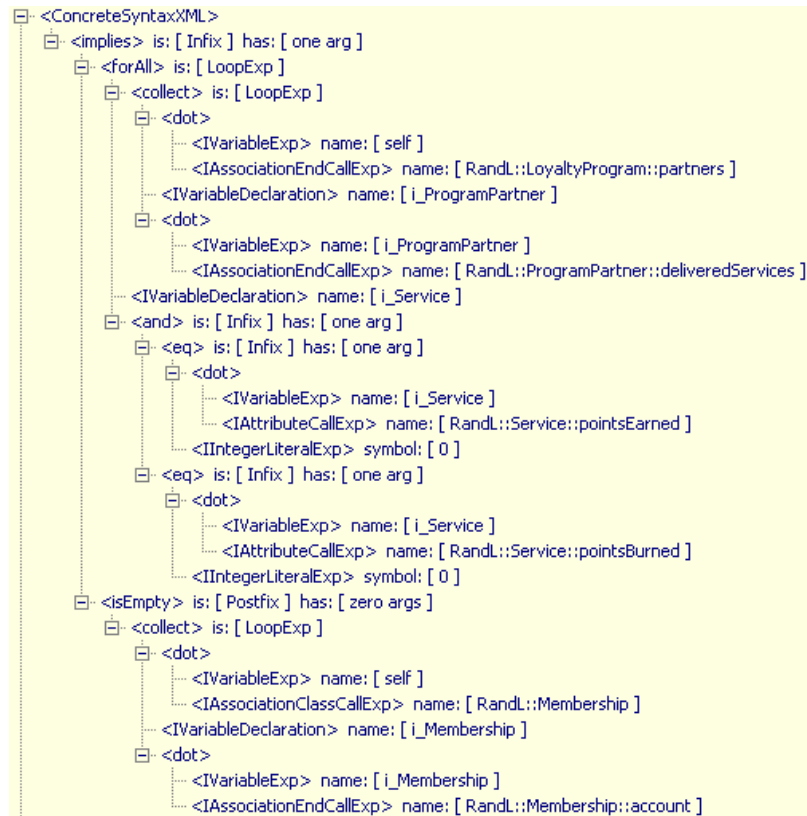


Fig. 4. The AST of the noAccounts invariant.

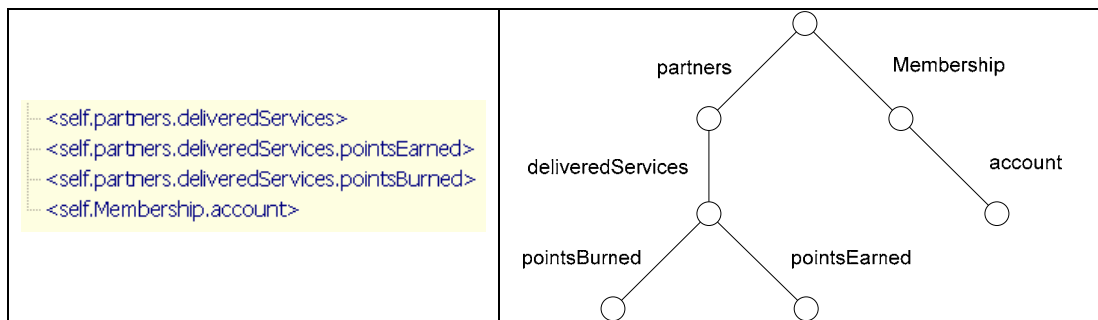


Fig. 5. The depends-on tree of the noAccounts invariant.