FAMIX: Exchange Experiences with CDIF and XMI

Sander Tichelaar
Software Composition Group
University of Berne
Neubrückstrasse 12
CH – 3012 Berne
Switzerland
+41 31 631 3568
tichel@iam.unibe.ch

Stéphane Ducasse
Software Composition Group
University of Berne
Neubrückstrasse 12
CH – 3012 Berne
Switzerland
+41 31 631 4903
ducasse@iam.unibe.ch

Serge Demeyer
Department of Mathematics and
Computer Science
University of Antwerp
Universiteitsplein 1
B-2610 WILRIJK, Belgium
+32 (0)3 820 24 14
Serge.Demeyer@uia.ua.ac.be

February 25, 2000

Abstract

In the FAMOOS project we have developed a set of tools for reengineering object-oriented legacy systems. These tools are based on the FAMIX meta model and exchange information using CDIF, an industry standard exchange format. For several reasons XMI, an emerging standard for information exchange, has appealed to us to be used as our interchange format. In this paper we discuss why XMI is interesting for us and what, to our current experience, are the advantages and disadvantages of XMI over CDIF.

1 Introduction

In the FAMOOS project [DD99], a project on reengineering of object-oriented systems, we had the problem of developing tools on different platforms for analysing different object-oriented languages. To remedy this in a sensible way we have developed FAMIX [DDT99, DTS99], a meta model for representing object-oriented software systems in a language-independent way. First, this allows tools to perform their tasks independently of the underlying programming language. Second, using the standard exchange format CDIF (CASE Date Interchange Format) [Com94a], models can be exchanged and reused between tools in a heterogeneous environment.

Within FAMOOS we have successfully used the exchange format to connect parsers, repositories and analysis tools. We have used different parsers (SNiFF+ [Tak96] for C++ and JAVA, VISUALWORKS [Par98] for SMALLTALK), different repositories (Moose, Nokia Reengineering Environment [DD99]), different metric tools and analysis tools (written in both C++ and SMALLTALK).

However, CDIF has the disadvantage that is not well supported in industry and that it is not being developed any further. Recently XMI (XML Metadata Interchange) [XMI98] has emerged, not only being a standard with considerably more industry backing than CDIF, but having some other interesting properties as well: XMI uses the MOF (Meta Object Facility) [Gro99a] as its meta meta model. UML (Unified Modeling Language) [Gro99b], the de-facto standard for modeling software systems, is also based on the MOF. Moving FAMIX to MOF opens opportunities to investigate a tighter integration with UML as well as different mappings from FAMIX to UML.

In this paper we shortly introduce the FAMIX model. Following, based on our requirements for an interchange format, we discuss our experiences with CDIF and our first experiments with XMI.

2 The FAMIX Model

The FAMIX model provides for a language-independent representation of object-oriented sources and contains the required information for the reengineering tasks performed by our tools (such as CodeCrawler [DDL99] and Gaudi [RD99]). We have chosen not to use UML as our meta model, because we have found it not fit to adequately model source code [DDT99].

First of all, since we are working in the area of reengineering, the model needs to contain *relevant information* for reengineering activities such as visualisation, metrics, heuristics and reorganisation. A second requirement is *language independence*: in the FAMOOS project we had to deal with legacy systems in different implementation languages (C++, JAVA, SMALLTALK and ADA). Therefore, our tools need to be able to work with all of those languages. A third requirement is *extensibility*: we cannot know in advance all information that might be needed in future tools, and for some reengineering problems tools might need to work with language-specific information (e.g. to analyse include hierarchies in C++). Therefore, we allow for language plug-ins that extend the model with language-specific features. Secondly, we allow tool plug-ins to extend the model so tools can, for instance, store analysis results or layout information for graphs. Note that these plug-ins should not break language-independent tools.

Figure 1 shows schematically the use of the FAMIX model: the tools analysing the different languages and exchanging information with each other via FAMIX, possibly extended with language and tool plug-ins.

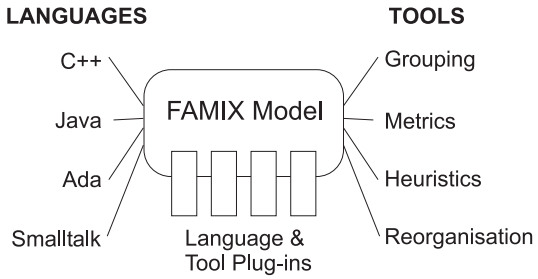

Figure 1: Concept of the FAMIX model

The Core Model

The core model (see figure 2) specifies the entities and relations that are extracted immediately from source code. It consists of the main object-oriented entities, namely Class, Method and Attribute. In addition there

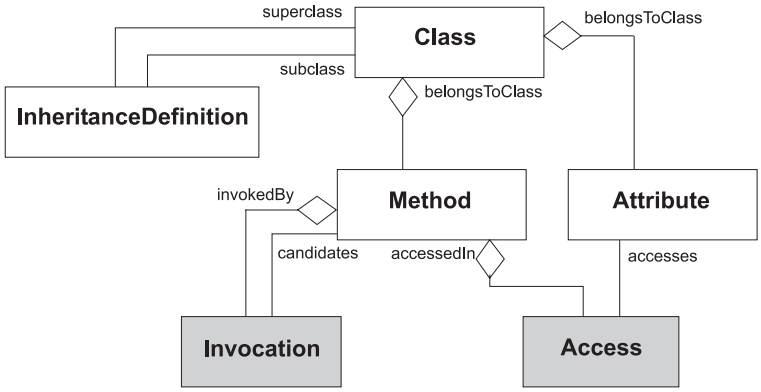

Figure 2: Core of the FAMIX model

are the associations InheritanceDefinition, Invocation and Access. An Invocation represents a Method calling another Method and an Access represents a Method accessing an Attribute. These abstractions

are needed for reengineering tasks such as dependency analysis, metrics computation and reorganisation operations. Typical questions we need answers for are: ”are entities strongly coupled?”, ”which methods are never invoked?”, ”I change the name of this method. Where do I need to change the invocations of this method?”. The complete model consists of much more information, i.e. more entities such as functions and formal parameters, and additional relevant information for every entity. The complete specification of the model can be found in [DTS99].

3 Requirements for an Exchange Format

Next to modeling the relevant information we need to exchange our information between tools in a heterogeneous environment. Apart from supporting the extensibility of the model, our exchange format needs to support the following requirements:

Easy to generate The format should be easy to generate, allowing for quick, ad-hoc integration of tools.

Simple to process. As the exchange format will be fed into a wide variety of tools, the format itself should be easy to convert into the internal data structures of those tools. It should also be easily transformable into input-streams for external tools (e.g., spreadsheets and databases).

Human readable. The exchange format is used by (sometimes buggy) prototypes. To ease debugging, the format itself should be readable by humans. Especially, references between entities should be by meaningful names rather than by identifiers that bear no semantics.

Convenient for querying. A large portion of reengineering is devoted to the search for information. Therefore it should be easy to query the exchange format. Especially, processing by ”standard” file utilities (e.g, grep, sed) and scripting languages (such as Perl, Python) should be easy. Note that this requirement is closely related to both the “simple to process” and “human readable” requirements.

Allows combination with information from different sources. Information about the same software system can come from different sources. Therefore, the representation should support incremental loading and merging of information. Note that just like with the ”human readable” requirement this implies that references between entities should be by meaningful names rather than by meaningless identifiers.

Supports industry standards. Since the tool prototypes must be used within an industrial context, they must integrate with whatever tools already in use. Ad-hoc exchange formats (even when they can be translated with scripts) hinder such integration. When available, the representation should favour an industry standard.

4 The use and evaluation of CDIF

We have adopted CDIF [Com94a] as the basis for the actual exchange of information using the FAMIX model. CDIF is an industrial standard for transferring models created with different tools. We have chosen CDIF as it matches well our requirements for the representation of FAMIX-based information. CDIF satisfies the ”supports industry standards” and the ”extensible” requirements. Moreover, CDIF is open with respect to the specific format for a transfer, or to state it in CDIF terminology, it allows for different syntaxes and encodings. By adopting the CDIF syntax SYNTAX.1 with the plain text encoding ENCODING.1 (see [Com94c] and [Com94b]), we also satisfy the ”human readable” and ”simple to process” requirements. Figure 3 shows a simple example. Note that we don’t show the meta model definition and headers, which are normally required in a transfer. And for reasons of simplicity, we only show a few attributes per entity.

However, to get to this point we had to make some design and interpretation decisions, which we discuss in the following paragraphs.

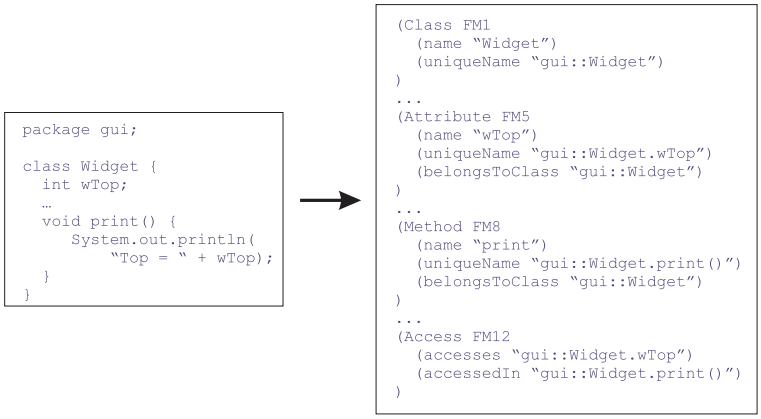

Figure 3: Java code and its CDIF representation

Avoid explicit relationships

The FAMIX model is designed as a flat repository-friendly model. Relations are encoded via unique names rather than explicit relations between entities. We avoid explicit relations because they have the following consequences. They lead to verbose transfers where meaningless identifiers — unique within a transfer only — are used to connect the different entities. Information gets scattered around a document, severely compromising the ”human readability” and ”convenient for querying” requirements. And the fact that the identifiers are unique within a transfer only, makes it difficult to incrementally load or delete entities and to merge information from different sources.

Below is an example of how we use explicit naming: a ”belongsToClass” attribute in the Method entity links the method to the class, instead of creating a link by defining an explicit ”Class.HasMethod.Method” relationship.

```
(Class FM17
   (name "Widget")
)

(Method FM35
    (name "print")
    (belongsToClass "Widget")
)
```

instead of

```
(Class FM17
   (name "Widget")
)

(Method FM35
    (name "print")
)

(Class.HasMethod.Entity FM56 FM17 FM35)
```

Lack of multi-valued strings

Another practical problem — especially when named relations are used — is that CDIF lacks a multi-valued string attribute (although it supports multiple values for, for instance, integers). To deal with one-to-many relationships we need multi-valued string attributes. Representing these using relations poses the same problem as for the explicit associations in general: verbose, hard to read and hard to merge with other information. Therefore, we have encoded multi-valued strings in CDIF text values (see [DTS99] for details).

Evaluating the context

CDIF has proven to be a proper solution for our purposes. It matches all our requirements, although this is partly because we made the design decision to avoid explicit relationships. However, apart from the technical properties of CDIF, there are some issues that make CDIF a less interesting solution. First, there is hardly any support in industry for CDIF. Direct consequence is that not many tools are available, for instance for parsing or verifying CDIF files. Furthermore, the OOPSLA'99 workshop on CDIF indicated that CDIF stopped evolving (although the people involved will try to influence other standards such as XMI with CDIF ideas and experience). So CDIF seems to be dying and is therefore not a preferred solution anymore.

5 Evaluating XMI

Currently we are considering moving to XMI (XML Metadata Interchange) [XMI98]. XMI is a set of rules to produce XML content from object-oriented data based on the MOF meta meta model. The MOF standard is an extensible meta meta model for defining meta models such as UML and FAMIX.

The main purpose of XMI is to enable easy interchange of meta data between tools in heterogeneous environments. It allows meta data to be interchanged as streams or files with a standard format based on XML [BPSM98].

The XMI proposal supports the interchange of any kind of meta data that can be expressed using the MOF specification, including both model and meta model information. It defines two sets of production rules, one for producing XML DTD files from meta models and one for producing XML documents from actual data.

The following example shows the same piece of Java code as in the CDIF example, but now represented in XMI. Again, as in the CDIF example, we don't show the meta model definition (the DTD) and the headers that are normally required.

```
<Famix.Package xmi.id="_1">
  <Famix.Entity.name>gui</Famix.Entity.name>
  <Famix.Entity.uniqueName>gui</Famix.Entity.uniqueName>
</Famix.Package>
<Famix.Class xmi.id="_2">
  <Famix.Entity.name>Widget</Famix.Entity.name>
  <Famix.Entity.uniqueName>gui::Widget</Famix.Entity.uniqueName>
  <Famix.Class.belongsToPackage>gui</Famix.Class.belongsToPackage>
</Famix.Class>
<Famix.Method xmi.id="_3">
  <Famix.Entity.name>print</Famix.Entity.name>
  <Famix.Entity.uniqueName>gui::Widget.print()</Famix.Entity.uniqueName>
  <Famix.BehaviouralEntity.signature>print()</Famix.BehaviouralEntity.signature>
  <Famix.Method.belongsToClass>gui::Widget</Famix.Method.belongsToClass>
</Famix.Method>
<Famix.Attribute xmi.id="_4">
  <Famix.Entity.name>wTop</Famix.Entity.name>
  <Famix.Entity.uniqueName>gui::Widget.wTop</Famix.Entity.uniqueName>
  <Famix.Attribute.belongsToClass>gui::Widget</Famix.Attribute.belongsToClass>
</Famix.Attribute>
<Famix.Access xmi.id="_5">
  <Famix.Access.accesses>gui::Widget.wTop</Famix.Access.accesses>
  <Famix.Access.accessedIn>gui::Widget.print()</Famix.Access.accessedIn>
</Famix.Access>
```

Evaluating the representation

XMI is based on the MOF meta meta model. This means that you *have* to use the MOF to represent your own meta model. Once you have expressed your meta model in terms of the MOF, and your data in terms

of your MOF-based meta model, the XML DTD files (the meta model definition in XML) and XML files representing your data can be generated using the XMI production rules. We can hardly call this “easy to generate”. To use XMI in a convenient way, tools will have to be based on (or migrated to) the MOF and then the exchange documents are easily generated.

The format itself is more verbose than CDIF. While normally verbosity implies human readability, in the case of XMI we feel it is “over-verbose”. Consequently, it is less “simple to process”, less “convenient for querying” and less “human readable” than CDIF.

XMI supports our requirement for incremental loading and merging information from different sources. Due to fact that we use our own scheme of unique naming in FAMIX two documents are always disjoint from the XMI point of view. XMI supports extension and linking between documents, but we do not have any experience with this functionality yet.

Evaluating the context

The XMI standard is a young standard in progress. Many vendors are backing it, but it is still evolving. However, it is based on two standards with quite some support already (MOF and XML) and it is currently the only remaining candidate for the OMG SMIF (Stream-based Model Interchange) proposal.

For XML there are already many free tools such as parsers available. For XMI there is some support (such as the XMI toolkit from IBM alphaworks). We feel that tool support is crucial for the easy adaptation of XMI, especially since its structure is more complex than, for instance, CDIF. On the other hand, when more vendors start shipping products using this standard, integration with these tools should be rather easy.

A last point which is important for us, is that both UML and XMI are based on the MOF. When we base FAMIX on the MOF as well we can use XMI for information exchange and at the same time explore the relationship between FAMIX and UML using the MOF meta meta model. Things we are interested in from the reengineering point of view are the integration of FAMIX information with UML information, and experimentation with mappings from FAMIX to UML, i.e. basically how to map source code to UML.

6 Conclusion

In this document we have reported on our experiences with CDIF and XMI in the context of the reengineering tools developed within the FAMOOS project. Within this project we have developed a set of tools based on a language-independent meta model, FAMIX. Exchange between these tools, which are not all in-house, is accomplished by encoding our models using the CDIF industry-standard interchange format. The combination of a model that provides us with the level of information we need in a conveniently accessible way, together with an exchange format that matches our requirements, covers our current needs. However, there are several reasons to evaluate XMI as its replacement. First of all, there is few support for the CDIF standard and it is not being developed anymore. XMI is based on the MOF and XML, two standard that have a lot of backing in the industry, with a lot of support already available. Secondly, one of our reengineering goals is to extract UML from source code, and to experiment with various mappings from FAMIX to UML. Using the same meta meta model will make it easier to experiment. Although XMI is more complex and therefore less trivial to use than CDIF, we will probably move FAMIX to use the MOF and use XMI as our interchange format.

Acknowledgements

This work has been funded by the Swiss Government under Project no. NFS-2000-46947.96 and BBW-96.0015 as well as by the European Union under the ESPRIT programme Project no. 21975 (FAMOOS).

References

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 - w3c recommendation 10-february-1998. Technical Report REC-xml-19980210, World Wide Web Consortium, February 1998.

[Com94a] CDIF Technical Committee. Cdif framework for modeling and extensibility. Technical Report EIA/IS-107, Electronic Industries Association, January 1994. See http://www.cdif.org/.

[Com94b] CDIF Technical Committee. CDIF transfer format encoding ENCODING.1. Technical Report EIA/IS-110, Electronic Industries Association, January 1994. See http://www.cdif.org/.

[Com94c] CDIF Technical Committee. CDIF transfer format syntax SYNTAX.1. Technical Report EIA/IS-109, Electronic Industries Association, January 1994. See http://www.cdif.org/.

[DD99] Stéphane Ducasse and Serge Demeyer, editors. *The FAMOOS Object-Oriented Reengineering Handbook*. University of Berne, October 1999. See http://www.iam.unibe.ch/~famoos/handbook.

[DDL99] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *WCRE'99 Proceedings (6th Working Conference on Reverse Engineering)*. IEEE, October 1999.

[DDT99] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why unified is not universal. UML shortcomings for coping with round-trip engineering. In Bernhard Rumpe, editor, *Proceedings UML'99 (The Second International Conference on The Unified Modeling Language)*, LNCS 1723, Kaiserslautern, Germany, October 1999. Springer-Verlag.

[DTS99] Serge Demeyer, Sander Tichelaar, and Patrick Steyaert. FAMIX 2.0 - the FAMOOS information exchange model. technical report, University of Berne, August 1999.

[Gro99a] Object Management Group. Meta Object Facility (MOF) specification. Technical Report MOF V1.3 RTF, Object Management Group, September 1999.

[Gro99b] Object Management Group. *Unified Modeling Language (version 1.3)*. Object Management Group, June 1999.

[Par98] Parc Place. *VisualWorks 3.0*, 1998. User Guide, Cookbook, Reference Manual.

[RD99] Tamar Richner and Stéphane Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99 (International Conference on Software Maintenance)*, pages 13–22. IEEE, September 1999.

[Tak96] TakeFive Software GmbH. *SNiFF+*, 1996.

[XMI98] XML Metadata Interchange (XMI). Technical Report OMG Document ad/98-10-05, Object Management Group, February 1998.