Model Driven Management of Complex Systems: Implementing the Macroscope's vision

Mikaël Barbero
ATLAS Group (INRIA & LINA)
University of Nantes, France
Mikael.Barbero@univ-nantes.fr

Jean Bézivin
ATLAS Group (INRIA & LINA)
University of Nantes, France
Jean.Bezivin@univ-nantes.fr

Abstract

Several years ago, first generation model driven engineering (MDE) tools focused on generating code from high-level platform-independent abstract descriptions. Since then, the target scope of MDE has much broadened and now addresses for example testing, verification, measurement, tool interoperability, software evolution, and many more hard issues in software engineering. In this paper we study the applicability of MDE to another difficult problem: the management of complex systems. We show how the basic properties of MDE may be of significant help in this context and we characterize and extend MDE by the concept of a "megamodel", i.e. a model which elements may themselves be models. We sketch the basic characteristics of a tool for handling megamodels and we apply it to the example of the Eclipse.org ecosystem, chosen here as a representative illustration of a complex system. The paper finally discusses how the proposed original approach and tools may impact the construction and maintenance of computer based complex systems.

1. Introduction

Complex systems are hard to characterize. Nevertheless they are more and more frequently met. Examples are a worldwide airline traffic management system, a global telecommunication or energy infrastructure or even the whole legacy portfolio accumulated for more than thirty years in a large insurance company. There are currently few engineering methods and integrated sets of tools to deal with them in practice. The purpose of this work is to study the applicability of Model Driven Engineering (MDE) to the management of complex systems.

Our general goal is to implement the Macroscope's vision of J. De Rosnay [13]. As the Microscope allows seeing the infinitely small and the Telescope allows seeing the infinitely great or far, the Macroscope is described there as a symbolic instrument to manage the infinitely complex. The Macroscope (Figure 1) can be considered as the symbol of a new way of observing, understanding, controlling and acting on complex systems.

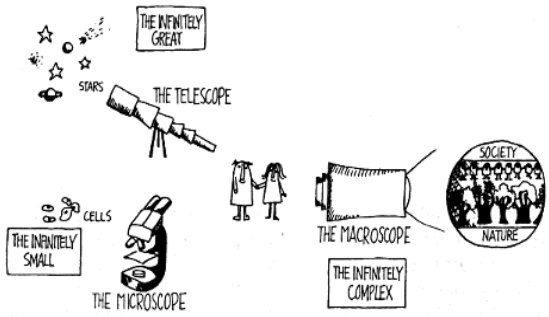

Figure 1. The Macroscope – Illustration from [13]

MDE is a software engineering field based on few simple and sound principles. Its power stems from the assumption of considering everything as a model [7].

Our intuition is that MDE may now provide the right level of abstraction to move the Macroscope from its status of a symbolic instrument to a set of concrete and practical tools, ready to be used by engineers when they collectively work on complex computer based systems.

In order to provide first evidence in support of this intuition, we have been building open source prototypes on the Eclipse platform. This paper presents the current status of the project and discusses achievements and remaining challenges in this field.

Section 2 of this paper introduces some characteristics of complex systems. From these characteristics we show in section 3 how MDE can provide some initial solutions. In section 4 we demonstrate that a generic tool build upon MDE can

Dagstuhl Seminar Proceedings 08331
Perspectives Workshop: Model Engineering of Complex Systems (MECS)
http://drops.dagstuhl.de/opus/volltexte/2008/1600

address some more remaining issues. In section 5 we take the example of the *Eclipse.org* ecosystem itself, considered as a complex system, to illustrate some possible applications. Related work on handling complex information systems is presented in section 6 and section 7 concludes the paper.

2. Complex systems

There are a number of examples of complex biological, ecological or societal complex systems discussed in [13]. In the context of this paper we are interested by Computer Based Complex Systems (CBCS), i.e. complex systems with a significant number of hardware or software components. These parts may be processing elements (processors, programs, processes, etc.) or data elements (memory, disks, repositories, files, etc.) or any kind of composite elements (hardware and software). One of the most important characteristics of a complex system is that it is composed of a very large number of individual parts. But there are also additional properties.

A CBCS is constantly in evolution with a past history, a present, and a future. This evolution is the consequence of the various interactions between the parts of the system. The evolution is permanent, i.e. the CBCS usually never stops, even when some parts are added, removed exchanged or under maintenance or repair.

A CBCS has a structure (or static architecture) and a dynamic behavior. It is composed of elements that may themselves be CBCSs (with structure and behavior) and no limit exists on this deep nesting.

In addition to structure and behavior, a CBCS also has a goal, defining its purpose in the context in which it is operating. As previously stated, this also applies to any component of this system. Important information is also the metadata associated to any component. The categories of metadata are quite diverse.

Another dimension of a CBCS is engineering heterogeneity. Many components are hardware and software elements produced in the last fifty years, with different types of technologies. For example many different hardware technologies, programming languages, APIs, operating systems, database organizations, network protocols, standards, or normative specifications have been used to build these various components. Furthermore there may be a penalty to the use of any technology. This is often called *accidental* complexity [8] that adds an artificial portion to the *essential* complexity of the base problem. Managing the accidental complexity accumulated by many layers of technological legacy is an important challenge in the management of CBCSs.

A CBCS is also a distributed system, i.e. its elements are located on many widely dispersed physical locations.

By definition a CBCS may not be understood by one unique human operator. On the contrary many stakeholders will have different views on the system. These stakeholders may play different roles (architect, designer, implementer, maintainer, manager, user, etc.). Stakeholders may participate in the global goal of the CBCS.

The interactions between the different parts of a CBCS are not random interactions and they follow specific patterns. Such a system is also characterized by the relationships that hold between its parts. Very often these relationships are informally characterized but in some occasions they may be explicitly represented. In either case they are quite important.

Managing a CBCS means observing it, understanding it and controlling it. However management may imply a lot of additional operations like designing it, constructing it, measuring it, managing it, maintaining it, and many more. We are interested here by the support MDE may bring to all these operations on CBCSs.

3. Model Driven Engineering

In the previous section, we have listed some important characteristics of complex systems. In this section we will take those characteristics and see how MDE may provide the corresponding management solutions.

MDE considers models as first class citizens as illustrated in Figure 2. A model is a representation of a system (relation *repOf*) and the nature of the model (*M*) is defined by its metamodel (*MM*). We say that model *M* conforms to its metamodel *MM* (relation *c2*).

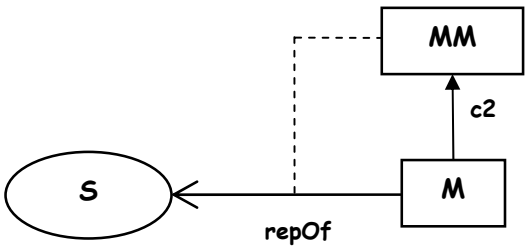

Figure 2. The two basic relationships of MDE

MDE is mainly built on top of these two basic relations of *representation* and *conformance*, like

object technology was mainly based on the relations of *instantiation* and *class inheritance* [9]. MDE may be implemented with the help of object technology (or any other technology like functional). However the basic paradigms of MDE are inherently different.

Any system can be represented by a set of models. Any model is a simplified (but nevertheless precise and faithful) representation of a given system. The relation of representation between system and model is probably one of the most overlooked in the present state of computer science. Until now its study has been mainly limited to ontology engineering, with some exceptions (e.g. [9]). This situation is however changing (see for example [21]) and this concern is more and more integrated in the software engineering perimeter.

Since system may be represented by a set of models, MDE helps to provide a homogeneous representation of a heterogeneous situation or phenomenon.

Metamodels may be used as filters to define matters of interest in a system. Used as a typing system, they provide precise semantics to artifacts and relations between these artifacts [15]. Metamodels and terminal models are abstract models and share many properties. Operations like storage, retrieval, transformation and many more may be applied to any kind of abstract models. A model to model transformation may be considered as an abstract model. Among the consequences of this property we may mention the possibility to use higher order transformations (i.e. transformations that take transformation(s) as input or/and produce transformation(s) as output).

Applying a transformation *Mt* to a model *Ma* may produce model *Mb*. Model *Mt* may be recorded as a relation between *Ma* and *Mb*. Furthermore the execution of *Mt* may produce, as a side effect, traceability model *Mtr* that may also be kept after the transformation.

A transformation is an executable model. There are other abstract models that may represent non executable relations between models and we call them *weaving models*. For example alignment between two metamodels or traceability between a requirement and a design model may be taken into account. Multiple and complex chains of traceability may be established with the support of such weaving models.

A system can be "filtered" by more than one metamodel. As a complex system cannot be understood and managed from one single point of view, being able to have different representations of this system is of great interest. For instance, we can imagine having a model of the static structure of a software application and a model of its execution trace (method calls, etc.) Static and behavioral representations may be models of the same system. Hence some relationships do hold between them. Issues of links between model elements may be addressed by weaving models [16]. Issues of direct links between model themselves are the subject of the present work.

MDE provides some principles and tools to manage complex systems. But this is not sufficient by itself. The distribution and the handling of a high number of artifacts, the representation of complex systems as composition of artifacts that may be complex themselves are not directly addressed. In the next section, we will introduce some enhancements to this purpose.

4. MDE for CBCS management

4.1 Megamodel definition

As discussed above, there are a number of properties of MDE that makes this solution a good candidate for the management of CBCS. But with these basis principles only, we would get a large number of independent models representing system's artifacts. We need a mechanism for knitting those models together into a coherent whole and providing a view of the global intent of the CBCS on which we can build a rationale. We propose to use a special kind of model for this mechanism and we call it a *megamodel* [6]. A megamodel is a model such that some of its elements are themselves models (Figure 3).

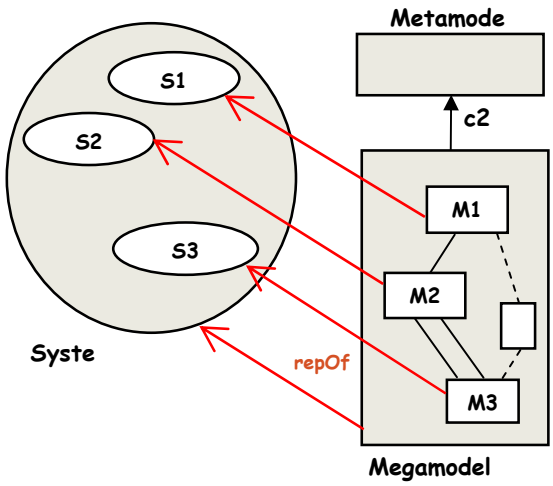

Figure 3. A megamodel: a model which some elements represent models

There is no need to introduce here new specific features to MDE because a megamodel is just a terminal model, conforming to a metamodel. Using

megamodels is just a matter of good modeling practices when dealing with complex systems. However, as we shall see later, some specific tools may be of significant help to support these good practices.

Megamodels are terminal models conforming to different types of metamodels. However some elements are models references, a concept that is defined in the metamodel of a megamodel. Model references may also carry metadata of represented artifacts. But megamodels also address the issue of semantic linking between models. Relationships between models are numerous and diverse: transformations, weaving, provenance, injection, extraction, trace, versioning, refinement, etc. Moreover, each CBCS is different and each stakeholder of a CBCS has different center of interest. From the wide range of relationships between models, we conclude there is no possible standard metamodel for a megamodel. Yet, we reckon it is possible to define a canonical base metamodel of which most megamodels' metamodels may be seen as an extension. A metamodel extension mechanism has been sketched out in [4]. In the remainder of this paper, we call "AM3Core" this base metamodel. A conceptual excerpt of AM3Core is depicted in Figure 4 (in KM3 notation).

```
package AM3Core {
 class Megamodel {
  reference ownedElements[*] ordered container : Element;
 }
 abstract class Element {
  reference metadata[*]  container :  Metadata;
 }
 class Metadata {
  attribute key: String;
  attribute value: String;
 }
 abstract class LocatedElement extends Element {
  reference locator container : Locator;
 }
 abstract class IdentifiedElement extends Element {
  reference identifier container : Identifier;
 }
 abstract class Model extends IdentifiedElement, LocatedElement {
  reference targetOf[*] : Relationship oppositeOf target;
  reference sourceOf[*] : Relationship oppositeOf source;
 }
 abstract class Relationship extends IdentifiedElement {
  reference source[*] : Model oppositeOf sourceOf;
  reference target[*] : Model oppositeOf targetOf;
 }
 abstract class Container extends LocatedElement {
  reference ownedElements[*] container : LocatedElement;
 }
 abstract class Group extends IdentifiedElement {
  reference ownedElements[*] : Element;
 }
 abstract class Chain extends IdentifiedElement {
  reference chainedRelationships[*] ordered : Relationship;
 }
 abstract class Identifier extends Element {
  attribute value : String;
 }
 abstract class Locator extends Element {
  attribute value : String;
 }
}
```

Figure 4. AM3Core: base metamodel of megamodel (in KM3)

The two main concepts are *Model*s and *Relationship*s. A relationship links some models (*source*) to some others (*target*). A *Model* is an *IdentifiedElement* and a *LocatedElement*. An identified element has an *Identifier*. It can be referenced even if it is not locally available. A *LocatedElement* has a *Locator*. This locator can be dereferenced to access to the underlying element. We can then access sub levels of representation of a system by dereferencing a model. We defined separately *Identifier* and *Locator* to make a clear distinction between what some resource management framework call logical identifier and physical identifier. *Identifier*s will be used to reference models that are distributed across platform, resources, networks, etc… *Identifier*s and *Locator*s are abstract concepts to be specialized in an extension of AM3Core. The specialization may be system- or implementation-specific. For instance, an extension may use URI generic syntax specification (RFC 2396) or the Digital Object Identifier (DOI) of the Association for Computing Machinery.

AM3Core also defines grouping concepts. A *Container* is a physical set of located elements while a *Group* only is a logical set of elements of the megamodel. A *Chain* is a special group of relationships being ordered.

This paper proposes to use megamodels as the main tool to implement the Macroscope's vision. The complexity of the systems is harnessed: some elements of the megamodels may be considered as gangways of more concrete details of it. Although we have a structure for managing complexity, we also need tools to handle it efficiently and in a user-friendly way. We propose two related kinds of generic tools for this purpose grouped into what we call the "megamodel manager":

- A megamodel editor: tool for viewing and editing entities and relationships of megamodels.
- A megamodel browser: tool for navigating trough pre- or user-defined views on the megamodel.

They can handle any megamodel conforming to AM3core as stated before.

Finally, handling efficiently a large number of models is a big requirement for MDE being the best candidate for CBCS management. We propose to use models repositories. Models repository would provide

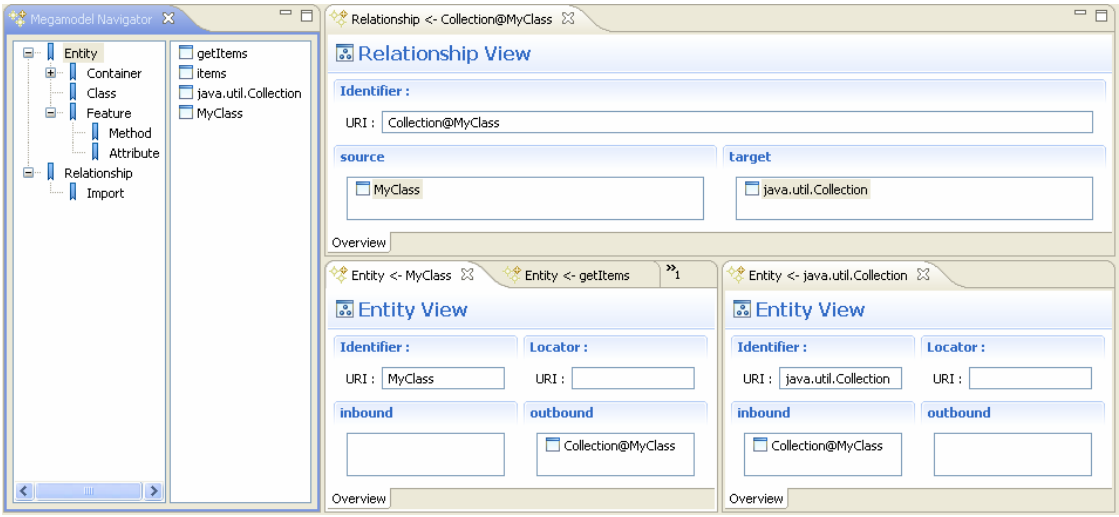

Figure 5. The megamodel browser prototype

interface for storing models in a centralized or distributed fashion. Compared to current solutions (see section 6), we support explicitly the basic relationship of MDE (i.e. the c2 conformance relation) leading to a clean distinction between modeling levels. The separation of modeling levels helps to deal with well delimited models without any dangling edges between models. No dangling edges lead to avoid considering the repository as one big graph and falling into a trap of complexity. The issue of linking elements within different models with symbolic link end is addressed by models weaving techniques [16].

4.2 Megamodel implementation

Megamodels of complex systems are not usable without tooling to help browsing, editing, and building custom views. We are currently implementing some tools (a megamodel browser and a megamodel editor) in the Eclipse AM3 project. A screenshot of the current megamodel editor prototype, currently in development, is depicted in Figure 5. This shows a (very) small megamodel of a Java program with dependencies between classes. We can see two panes: on the left side a split view (named Megamodel Navigator) and on the right side some editors. The megamodel navigator contains the concepts of the metamodel of the megamodel (left) and their instances (right).

Concepts are organized with respect to their inheritance tree. The right pane shows some editors of those instances. We see a relationship between two classes. This relationship is an “Import” of the Java class *java.util.Collection* (target) in the class *MyClass* (source). Below, there are two entities editors. We see that *java.util.Collection* is the target of an import relation (inbound) and that *MyClass* is the source of an import relation (outbound).

The editor is entirely generic and can handle any kind of megamodel conforming to an extension of AM3Core. Thus, relationships between entities are specified in the extended metamodel of AM3Core. It gives them their semantic.

The choice of a Java program as a complex system in the example of Figure 5 is a big simplification made in the purpose of illustration. In the next section we take another example, still illustrative and simplified, but a bit more realistic.

5. Illustrative example

This section takes the example of the whole *Eclipse.org* ecosystem to illustrate the notion of a CBCS. It shows how the tool presented in the previous section may be applied to handle the corresponding situation.

5.1 Presentation

Eclipse.org may be considered as a relevant example of CBCS. It is composed of an important number of artifacts of different natures. There is code, documentation, committers, projects, plug-ins, update sites, legal process, committer’s election process, Wiki, web pages, bugs report facilities, newsgroups, etc. *Eclipse.org* also is a large cluster of projects: there are 75 projects distributed into 11 top level projects with different maturity levels: from incubation to mature phase. Regarding the “Eclipse Platform” project only, 186 contributors have submitted code and 75 have done so in the last year. The code base

represents 4,779,778 lines of code being an estimated effort of 1,460 person years (source: Ohloh [24]). The evolution is very quick (there is one commit to the platform every 22 minutes – [11]).

Moreover, projects and entities are tightly related to each others. For instance, latest Test & Performance Tool Platform (TPTP) release (4.4.0.3 as the time of writing) relies on EMF 2.3.1, WTP 2.0.1, BIRT 2.2.1, GEF 3.3.1 and DTP 1.5.1. Those dependencies are expressed in different locations: releases notes, builds manifests, etc. Dependencies are even hardest to manage due to projects sovereignty regarding their timelines and release schedules.

Eclipse.org has a lot of different stakeholders. They consider *Eclipse.org* from different point of views. Members of the management board of the Foundation take care about download stats, good health conditions of the projects (conformity to the intellectual property policy, state of development: active or not, etc.) and information about members and sponsors. Committers for their part check more technical concerns such as bugs reports, unit tests results, etc. Simple users only care about projects versions and dependencies, compatibility between builds.

5.2 MDE Support

From the list of informal notions enumerated above, we may select concepts that will appear in the metamodel of *Eclipse.org* megamodel. In the remainder of this section, we show some of the benefits that could be reaped from a MDE representation of this CBCS.

From a project lead point of view, we need to offer some facilities. For instance, producing a regular summary of all reported bugs classified by some criteria (severity for instance) is a typical need. It is possible to do it "by hand", i.e. to create a database request to get all bugs on a project, to give the result to a program to compute the desired metrics (typically Excel) and to call some export functions to have a nice display. With reverse engineering tools, Eclipse's Bugzilla system can be represented as a model [22] and the metric as model-to-model transformation targeting a metric model. With canonical metric models, the "metric to visualization format" (table to HTML, table to SVG bar chart, etc.) transformations are easily reusable (Figure 6).

Every artifact is handled in a seamless way thanks to the homogeneity provided by modeling principles. The degree of reusability of independent model transformations increases accordingly. Delivering web feeds of the bugs' metrics is only a matter of doing a transformation between the model of metrics to an RSS one. Atom format is provided by reusing Web Syndication interoperability transformations [3].

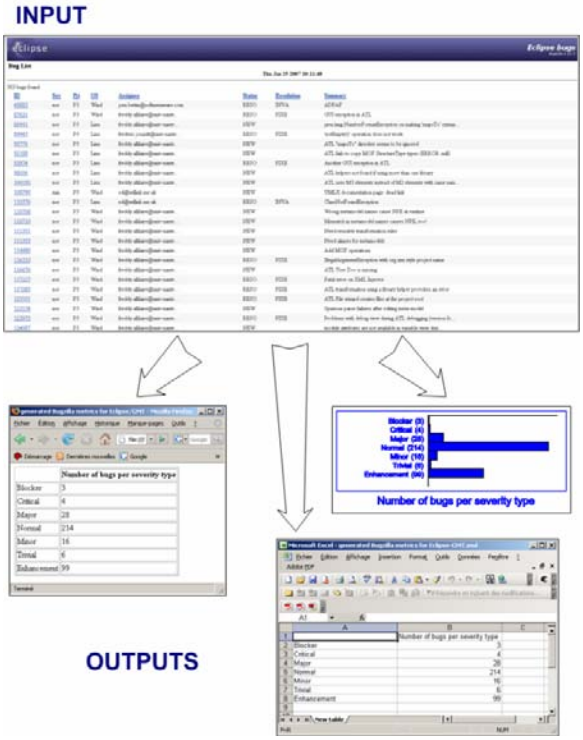

Figure 6. Visualization format for metrics on a Bugzilla model

Interoperability of the Eclipse Bugzilla system with other software quality control tools like Mantis can also be achieved thanks to the homogeneity [5]. It is possible to do it either by using a pivot metamodel [3] or directly by capturing differences between the two tools in a weaving model of the two metamodels [17].

Computing metrics on the project's code is another concern of an *Eclipse.org* project lead. Here again, MDE allow to factorize common pattern and reuse them. For instance, metrics on class diagram of Unified Modeling Language (UML) models already are available in [3]. Models of the Java code can be created with some Java discovery tools like [14] and [22]. The tool [22] gives the abstract syntax tree of Java classes. It can be transformed to an UML class diagram and only hold back the design level. The tool [14] gives directly an UML model of the Java code. Once the UML model is created, transformations computing the measures can then be fired. Finally graphical representation can then be chose to display computed metrics (Figure 7 and Figure 8).

Although MDE presents some benefits for source code measurement, the size and the number of modeling artifacts involved in this process is very

high. This problem is only addressed by megamodeling.

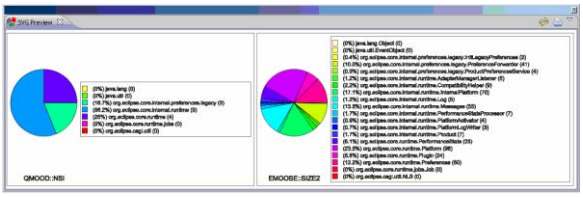

Figure 7. Example of SVG Pie Chart presentation of measures on a UML model

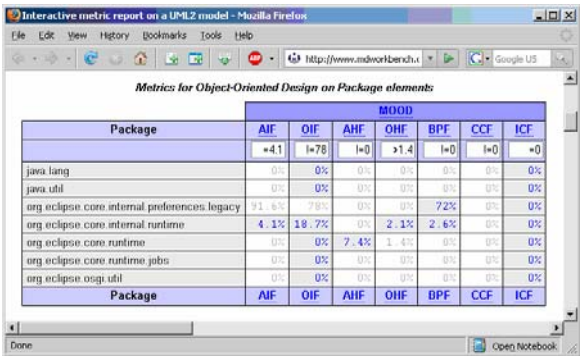

Figure 8. Example of interactive XHTML with CSS presentation of measures on a UML model

For instance, the Java 5 SDK (J2SE 1.5) is composed of 2701 classes and 912 interfaces within 187 packages. The number of links between classes (import statements) and methods (call statements) blows up. Handling such an amount of data in a single model would be unfeasible. Not technically speaking but from human's eye point of view. One solution is to provide different levels of representation. For instance we should have a megamodel of packages (187 elements) with their dependencies (sub-packages, imports). At a sub level, we should have a megamodel of classes by package (approximately 14 classes and 5 interfaces by package) with class-specific dependencies such as import, inner classes, etc. It is possible to manage the models of the code with this magnitude of artifacts number.

But MDE can be used by a project lead even at a higher level. For instance, the Eclipse Architecture Council would like to begin population of a catalog of functional elements, i.e. contributed extensions points and other facilities for each project. In this goal, the lead would build a model of the plug-ins of the project. Building the catalog is then to select information that are required and extract it in static and/or browsable format. A discoverer of models of plug-ins does not exist yet but a metamodel already is available [1]. It defines dependencies between plug-ins, extensions points and their attributes, packages it contains and also contributions to other extension points. Several applications can be defined from such a model. For instance, a dependency graph can be created. It helps at better displaying inter-plug-ins dependencies and detecting circular ones. The target model could be conforming to DOT (an automatic graph layout program from Graphviz) like the KM32DOT transformation [3]. The catalog of components can be generated as a Wiki or HTML page and can be used as a documentation in other Eclipse projects or, more generally, by anyone extending the Eclipse platform.

A model of a plug-in can also be represented as a megamodel. Java packages are sure enough references to models previously described for code measurement.

Finally, we will illustrate how Model Driven Management of CBCS may be well suited for systems' evolution management. The metamodel of plug-ins we were talking about just this minute will be accurate until the next evolution in the Eclipse plug-ins architecture. The metamodel will have to be upgraded but models conforming to may not as easily. They have to migrate to the new one. Semi-automatic migration is possible thanks (once again) to the homogeneous representation of system's artifact. The former and the new metamodel are matched by reusing techniques and tools like the one used to capture differences between Bugzilla and Mantis (see above). It is possible to define a metamodel independent approach [3] to the difference representation. This approach is used to compute the migration transformation [2].

All models we presented in this section are representation of some artifacts of the *Eclipse.org* system. It helps understanding each artifact but not the system as a whole. Knitting those models (and the underlying artifacts) is done within a megamodel (Figure 9). We take the example of a Java project and a bug reports tool. At the first representation level, we see links between Java projects and components declared in a Bugzilla product. Bugs of "Java App 1" are declared in "Bugzilla Comp21" as do bugs of "Java App 3".

As long as a Java project is represented by a model, details of "Java App 3" can be observed at the sub representation level. It is composed of Java files with import dependencies. A Bugzilla component may also be represented as a model. It is composed of Bugs and attachments. The relationship between a bug and its Java file is described in a weaving model. On the picture, there is only one link: "bug1" is related to the "B.java" file. The model of a Java project is a megamodel which Java file elements are models. From the "B.java" model element, it is possible to retrieve

the model of the abstract syntax tree of the Java B class.

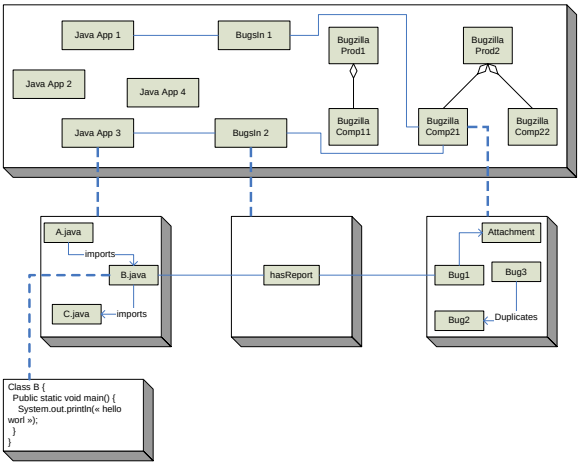

Figure 9. Megamodel of Java project and bug reports

In this section, we have presented how MDE can help to understand the *Eclipse.org* CBCS. The number of models and links between them that are generated from this study comforts us in the need for the megamodel mechanism and tooling for handling them in a manageable and consistent way.

6. Related work

Managing complex systems is not a recent concern and some tools have already been developed. Most of them provide a way to associate metadata to the artifacts of the system. Those metadata are more or less structured. For instance, we can cite the Resource Description Framework [27] (RDF) being used to described online resources by providing a lightweight ontology system or Dublin Core [12] being a set of standard metadata that support a broad range of purposes.

Description of complex systems and especially legacy portfolios may also be achieved by reverse-engineering tools and Application Portfolio Management (APM) systems. Those tools, like the CAST workbench [10], provide an understanding of complex systems through different configurable views, measures, and monitors. Main issue with those kinds of application is that they do not provide homogeneous views of the systems. Data can be only retrieved in heterogeneous formats such as Excel, XML, HTML, etc. As far as we know, they provide only limited ad-hoc facilities to navigate through different representation levels.

There is some well known work around models repositories. For instance, the Eclipse Modeling Framework (EMF) [18] and the NetBeans MetaData Repository (MDR) [26] implements MOF-like repositories. EMF is based on Ecore as a metametamodel and provides an in-memory repository of models with lazy loading facilities. MDR is based on Java Metadata Interface (JMI) and provide a full database based repository of models. These repositories may be viewed as low level megamodels without much extensibility potential.

The notion of megamodel has been previously used in the work of J.M. Favre with a similar meaning. Ongoing work includes [19]. The notion of megamodel is also central to other projects like the SITRA Norwegian project [23].

One of the projects that are the closest to the work reported here is the MMTF (Model Management Tool Framework) at the University of Toronto [25]. In this project the CBCS is limited to a model based complex system, i.e. all the considered artifacts are models and relations between models. This is also a goal pursued by our toolset, but here we have presented a much broader view. Of course, by restricting the domain of CBCS to model-based systems, we may obtain more advanced results. For example the work reported in [25] mentions the objectives of: supporting arbitrary model and model mapping types and operators over them, supporting easy integration of existing independently developed model-related components including editors and operators, providing the capability to import/create/modify/view particular collections of models and mappings, and providing the capability to interactively apply relevant operators to sets of models and mappings to derive new (resultant) models and mappings and to define new operators. One interesting emphasis in this work is on the inference calculus of explicit relationships between models from implicit knowledge.

7. Conclusion

This paper has described a vision and the initial development of an open source model-based framework for the management of CBCS. We have implemented an initial set of tools in the GMT/AM3 Eclipse project and we are presently starting the implementation of a first set of use cases. From there we plan to validate the conceptual framework and to improve the practical tools.

At the center of the proposed toolset, we place what we call the "megamodel manager". As we have seen, there is very little specificity to this tool, and it may be considered more as good practice guidance than as a heavyweight and constraining set of tools. One lesson

learnt in this work is that, even without any extension, MDE provides an excellent support for the management of CBCS. In spite of the fact we provided clues for handling issues of CBCS, we did not address the management of big sized artifacts. This is a subject of future work.

However by using this idea of a megamodel, we have an interesting opportunity to leverage model engineering to enhanced practices. The current achievements are illustrated by a set of experiments visible on the Eclipse site (see [1], [2], [3] and [22]). Some of these have been summarized in section 5. The initial conclusions are rather positive and clearly demonstrate the feasibility of the approach. Among the contribution of this work, we may mention in particular:

- The idea of a megamodel with variable metamodel allowing to consider a number of different situations including nested systems.
- The consideration of a model transformation as a model, allowing to keep track of certain explicit relations between models.
- The use of incomplete and non executable relations between models by so-called weaving models. A weaving model summarizes a lot of fine-grained relations between model elements.
- The possibility to use weaving models or transformation models to link chains of versions of different artifacts like metamodel.
- The possible specialization of model transformations to produce measurement or verifications of parts of the modeled system.
- The possible use of model transformations to provide different kinds of views on a given model.
- The integration of a lot of facilities like the ones mentioned above in a tool for megamodel handling, giving a first idea of a possible engineering workbench for complex system management.

8. Acknowledgements

This work is being supported by the French *FLFS*, *IdM++*, and *Happy* projects as well as the *ModelPlex* European Integrated project (FP6-IP 034081). We thank all the members of the AMMA project and specially Hugo Brunelière, Brahim Khalil Loukil and Frédéric Jouault for their contributions to the ideas presented here.

9. References

[1] AM3 Atlantic Zoo, *Eclipse plug-in: metamodels in KM3*,http://dev.eclipse.org/viewcvs/indextech.cgi/org.eclipse.gmt/ AM3/org.eclipse.am3.zoos.atlantic/EclipsePlugIn.km3

[2] AMW website, use cases section, http:// www.eclipse.org/gmt/amw/usecases/

[3] ATL website, *use cases section and Transformation Zoo*, http://www.eclipse.org/m2m/atl/usecases/

[4] Barbero, M., Jouault, F., Gray, J. and Bézivin, J.: *A Practical Approach to Model Extension*, In: Model Driven Architecture- Foundations and Applications, Third European Conference, ECMDA-FA 2007, Haifa, Israel, June 11-15, 2007, Proccedings, LNCS 4530, edited by David H. Akehurst, Régis Vogel, and Richard F. Paige. Springer, pages 32--42.

[5] Bézivin, J, Brunelière, H, Jouault, J, and Kurtev, I: *Model Engineering Support for Tool Interoperability*. In: Proceedings of the 4th Workshop in Software Model Engineering (WiSME 2005), Montego Bay, Jamaica.

[6] Bézivin, J, Jouault, F, and Valduriez, P: *On the Need for Megamodels.* In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. 2004.

[7] Bézivin, J: *On the Unification Power of Models. In:* Software and System Modeling (SoSym) 4(2):171—188. 2005.

[8] Brooks, F. P: *No Silver Bullet: Essence and Accidents of Software Engineering*. 1987.

[9] Cantwell Smith, B: *On the origin of objects*, The MIT Press, ISBN: 0-262-19363-9, 1996.

[10] CAST: *The CAST Application Intelligence Platform*, http://www.castsoftware.com/

[11] CIA.v, *an open source version control informant*, http://cia.vc/

[12] DCMI*: The Dublin Core Metadata Initiative*, http://dublincore.org/

[13] De Rosnay, J: The macroscope, Harper & Row, New York, 1979.

[14] Dennis Wagelaar, *Jar2UML Eclipse plugin*, http://ssel.vub.ac.be/ssel/research:mdd:jar2uml

[15] Didonet Del Fabro, M, and Bézivin, J: *Generic Model Management: from Theory to Practice*. In: First Intl. Workshop on Towers of Models 2007 Co-located with TOOLS EUROPE. 2007.

[16] Didonet Del Fabro, M, Bézivin, J, Jouault, F, Breton, E, and Gueltas, G: *AMW: a generic model weaver*. In: Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05). 2005.

[17] Didonet Del Fabro, M. Bézivin, J. and Valduriez, P: *Model-Driven Tool Interoperability: An Application in Bug Tracking*, In: The 5th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE'06), LNCS 4275, 2006, pages 863--881.

[18] Eclipse.org: *Eclipse Modeling Framework (EMF)*, Eclipse Foundation, http://www.eclipse.org/modeling/emf/, 2007

[19] Favre, J.M., Nguyen, T.: *Towards a Megamodel to Model Software Evolution Through Transformations*. Electr. Notes Theor. Comput. Sci. 127(3): 59-74 (2005)
[20] Jouault, F, and Kurtev, I: *Transforming Models with ATL*. In: Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica, October 2-7, 2005, pages 128—138. 2006.
[21] Kiczales, G: *Context, Perspective and Programs*, OOPSLA' 07 keynote, Montreal, Canada.
[22] Modisco Website: *Uses cases and toolbox*, http://www.eclipse.org/gmt/modisco/
[23] Nytun, J.P. *A Generic Model for Connecting Models in a Multilevel Modeling Framework*, ICSOFT'2006, First International Conference on Software and Data Technologies, http://www.icsoft.org
[24] Ohloh.net: *Open source network*, http://www.ohloh.net/projects/3855?p=Eclipse+Platform+Project
[25] Salay, R., Chechik, M., Easterbrook, S. Diskin, Z., McCormick, P., Nejati, S., Sabetzadeh, M., Viriyakattiyaporn, P. *An Eclipse-Based Tool Framework for Software Model Management*, Eclipse Technology Exchange Workshop at OOPSLA 2007, Montreal, October 2007.
[26] Sun NetBeans: *Metadata Repository (MDR)*, http://mdr.netbeans.org/ 2007
[27] W3C: *Resource Description Framework (RDF)*, http://www.w3.org/RDF/