

# eAssignment - A Case for EMF

Marcel Bruch   Christoph Bockisch   Thorsten Schäfer   Mira Mezini

Software Technology Group  
Darmstadt University of Technology, Germany

{bruch,bockisch,schaefer,mezini}@st.informatik.tu-darmstadt.de

## ABSTRACT

Developing Eclipse plug-ins often involves the creation of data structures and corresponding data processing code. In developing eAssignment, an Eclipse-based application to support electronic programming exercises, we identified several issues with implicit models of data structures and hand-written code needed to access them. In this paper, we report on our experiences of using the Eclipse Modeling Framework to overcome these shortcomings.

## Keywords

Eclipse, Eclipse Modeling Framework, eAssignment, courseware, code generation

## 1. INTRODUCTION

As nearly every application, Eclipse plug-ins involve data, whose model is either implicitly encoded in the source code or explicitly defined in data modeling languages such as UML [5] or XML Schema [9]. For instance, often a plug-in is used for the management of domain entities, it is customizable and thus has to model and persist user properties, or it communicates with other applications by means of pre-defined data structures. Our experiences from developing Eclipse plug-ins suggest the following requirements for modeling and accessing data structures:

**Explicit model** Data should be modeled explicitly. Explicit models help to describe what an application is supposed to do at a higher level of abstraction than code does. Furthermore, such models can be used to generate (parts of) the implementation code.

**Consistency** Consistency between the model and the implementation classes is crucial. Code that is concerned with data handling, i.e., storing, modifying, retrieving, and deleting data entities, is often scattered across different modules. For instance, the data for the user preferences is used in two places: in the preference

pages where users can edit the preferences as well as in the plug-in's core functionality itself. Especially when following a fast-prototyping approach, developers have to update those locations often, because model changes occur frequently.

**Type-safe data access** Frameworks that support storing and retrieving data such as Eclipse often provide a very general API to support flexibility in defining generic storing and retrieving functionality — unfortunately, at the cost of losing type-safety. For instance, developers can read data from the preference store, but have to make sure that they use the right type. Erroneous behavior can only be detected at runtime.

In this paper, we report on our experiences with using the Eclipse Modeling Framework (EMF) [2] to meet these requirements in developing eAssignment. This is an Eclipse plug-in for supporting teachers and students in managing programming assignments of computer science courses. Setting up, distributing, solving, and submitting programming assignments often imposes some “accidental overhead” on both, teachers and students. Teachers need to administrate students and student groups, provide resources, such as libraries, or test classes, distribute the project artifacts to students, and evaluate the returned projects by running different tests, including those that concern the code quality. Students need to install the needed infrastructure, work on the assignment and send the results back to the teacher. Much of this effort does not directly contribute to pedagogical goals of the assignment, especially for technology heavy projects. However, some of these tasks can be automated — and this is the goal of the eAssignment [6] project. Its concrete features include support for project administration and generation, administering student groups, automatic testing of solutions, etc.

In the context of the eAssignment application, we followed a generative programming approach to cope with the issues presented above: We used the Eclipse Modeling Framework (EMF) [2] to generate various data models. EMF was chosen because it (a) provides a lightweight, pragmatic approach to modeling with very low entry cost and is thus suitable for rapid prototyping, (b) unifies key technologies such as Java and XML, and (c) integrates well into Eclipse.

The EMF framework proved very useful in the development of eAssignment. In some cases, explicit data models have been already available in a kind of schema, e.g., as Eclipse XML Schema Definition files. In the other cases, the modeling effort was very low.

eclipse'05, October 16-17, 2005, San Diego, CA  
Copyright 2005 IBM 1-59593-342-5/05/0010...\$5.00

We used EMF's code generation facilities to generate EMF models based on the aforementioned data models. Moreover, we used EMF's introspection capabilities to write generic components that operate on EMF models. The benefits of this approach are twofold: (a) the implementation is automatically synchronized with the model and (b) using the type information from the model enables us to guarantee type-safe access to the data.

In this paper we discuss on three applications of EMF to facilitate the use of different Eclipse by means of eAssignment as case study. First, we have written a generic component that builds a graphical user interface for editing an EMF model. Second, we used EMF to store and retrieve arbitrary data models using the Eclipse preference store. Third, we applied EMF to Eclipse' extension point mechanism. We generate an EMF model based on an Eclipse XML Schema Definition. This model is used to generate the corresponding descriptor classes as well as an input for a generic parser component that obtains all contributions for an extension point.

Because EMF is heavily used by an increasing number of Eclipse projects (e.g., VE [11], UML2 [10] and the IBM WebSphere Studio product family [12]) and the APIs mentioned above are used by most plug-ins, we assume that other plug-in developers can benefit from our experiences. Moreover, we have released our EMF utilities for Eclipse via our project website [6], so that the Eclipse community can use them.

The remainder of this paper is organized as follows. In the next section we provide an overview of the eAssignment plug-in. Then, we give a short introduction to the Eclipse Modeling Framework in Section 3. Section 4 presents a case study covering the usage of EMF in our eAssignment application. In Section 5 we summarize.

## 2. EASSIGNMENT

In this section we present eAssignment which led us to developing the EMF utilities. Later we will use examples from the implementation of eAssignment to illustrate our requirements and how they are met by our implementation with EMF.

Programming courses are an essential part of a computer science curriculum. For instance, in a first programming course students learn basic concepts of computer science, while graduate students can take courses covering e.g., software component technologies. In both cases, programming exercises typically accompany the lectures to augment them. An exercise is made up of several single assignments. Unfortunately, an assignment requires a large effort from teachers. First, it is often necessary to provide some resources, such as libraries or test classes. The assignment projects are then distributed to and modified by the students. After accomplishing the assignment, they send the results back and the teacher has to test the submission, give the students a feedback and keep hold of the results. To reduce the effort and improve the handling of assignments we are developing eAssignment [6], a set of plug-ins for the Eclipse IDE. In the following, we list the features of our product, present the typical workflow for an assignment project, and report on our experiences of the first application of eAssignment.

### 2.1 Features

From our experiences with programming exercises, we de-

rived the following features to simplify and standardize administration tasks for both, students and teachers.

**Project generation** Teachers can generate an assignment project by creating a regular Eclipse project and set up the roles the resources play. For instance, a resource can set to be read-only, only visible to teachers, or as modifiable by students. This information is then used to create the assignment projects for the students.

**Bidirectional distribution** Teachers can upload assignment projects to a server. The students can download them directly into the Eclipse IDE, process them and submit their solution by uploading their projects to the server. Teachers can then import all student submissions into their IDE.

**Automatic processing** It is possible to execute tests or checks for e.g., implementation restrictions automatically. This execution can take place inside the students IDE (for instance to avoid submitting solutions with compile errors), as well as in the teachers IDE (e.g., to see if the solution functions properly).

**Administration** The administration module enables teachers to keep track of students, exercise groups, and the results for each assignment and student.

The first two features are already implemented, and we are currently working on the implementation of the latter two. For the future, we have planned two additional features. First, we want to provide cross-review capabilities. Based on the distribution facility, this feature enables students to review solutions of their fellows. Second, we want to use the automatic tests and checks to generate reports sent out to the students to improve the awareness of their mistakes.

The different users of eAssignment, i.e., teachers and students, may use different kinds of functionality. For example, only students can submit solutions, while publishing assignment projects is restricted to teachers. But, there are also some functions used by both user groups, e.g., the setup of the server to be used. To allow flexible construction of versions for the different users, the project is divided into five plug-ins. The *Core* plug-in provides common functionality to be used by all user groups. The *Student* plug-in provides the students' user interface, while the *Teacher* plug-in provides the teachers' user interface. The *Administration* plug-in provides means to maintain collections of courses, students that are attending a course and their grades. Finally, the plug-in *EMFUtilities* contains utilities to ease the usage of Eclipse APIs with EMF, as described in detail in Section 4.

### 2.2 Using eAssignment

In this subsection, we describe a typical workflow of using eAssignment, assuming that the features described above are fully implemented.

To use eAssignment, the teacher starts with creating a regular Eclipse project, adds the necessary resources and customizes the project, e.g., by setting the class path. Next, he/she activates eAssignment and tags all resources which influences the way they are published. Currently, possible tags are *student read/write* and *student read-only*, *teacher sample solution* and *teacher private test*. After the assignment project is complete, the teacher publishes it. This

involves the creation of a student assignment project that does not contain any teacher resources and uploading the project to a server.

Then, students can download the assignment project into their IDE. They may view and use read-only resources, but only resources that are tagged as *student read/write* can be modified. After a student finishes the assignment, the submission feature is used. eAssignment checks whether all necessary preconditions are met (e.g., the project has no compile errors), and sends the project to the server.

Next, the teacher can download all submissions into his or her IDE. During this import step, the read-only resources are added from the original assignment project to ensure that they have not been modified. Furthermore, automatic tests are executed. The submissions can be reviewed and executed directly by the teacher. The results of an assignment are then persisted using the administration feature.

### 2.3 Experience report

We used eAssignment in our course Software Component Technologies during summer term 2005. At this time, the first two features described in Section 2.1 have been available. Apart from some minor usability issues we observed and fixed during the first assignments, the application was very successful.

From a student's point of view, there is no more need to set up a project, download necessary libraries and classes, build the class path and so on. Although a lot of libraries such as those required by the frameworks Spring [7] and Hibernate [3] were used in the assignments, students were freed from the installation and had to deal with the assignment's subject only.

The benefits for the teachers were even bigger. First, we provided one project setup that was used by all students. This enabled us to support this single setup. In previous exercises, some problems raised due to a wrong project setup and it took some time to get familiarized with a student's settings. The distribution framework enabled us to easily import all solutions into our workspace. We could then run the application as well as tests without any further setup like setting the correct class path.

In short, eAssignment helped to minimize those tasks that are of no educational use, e.g., downloading libraries and setting up a project. After downloading the assignment project, students could start instantly solving the problem. Teachers, however, could concentrate on making code reviews and give feedback to the students instead of figuring out how to run the applications.

## 3. ECLIPSE MODELING FRAMEWORK

The Eclipse Modeling Framework (EMF) Project [2] is intended for building tools and other applications based on a structured data model. This section shortly describes the three building blocks of the framework; for a detailed discussion of EMF's features we refer to [1].

First, there is the core framework which includes a meta model for describing models called ECore. ECore is a small and simplified subset of UML and is based on XMI as its canonical form. Existing models specified in XMI, Java interfaces, or XML Schema can be imported. With ECore it is possible to model (a) objects together with their attributes and operations, (b) relationships between objects, and (c) simple constraints on objects and relations. More-

over, runtime support for the models is provided in the core framework. This includes change notification, persistence support with a default serialization to XMI, and a very efficient introspective API for EMF objects.

Second, the EMF.Edit framework includes a generic set of reusable classes for building editors. These editors consist of a single widget, e.g., a tree viewer, which represents the whole ECore model. The framework provides content and label provider classes, property source support, and other convenience classes that allow ECore models to be displayed using standard desktop viewers and property sheets. Moreover, a command framework is part of EMF.Edit, including a set of generic command implementation classes for building editors that support undo and redo.

Third, the EMF code generation facility is capable of generating arbitrary data-processing code. Files produced by the EMF generator are intended to be a combination of generated pieces and hand-written pieces. Thus, generated classes can be enhanced by adding methods and instance variables. EMF is capable to regenerate the model as needed while preserving those additions.

## 4. EMF IN EASSIGNMENT

During the development of the eAssignment application, we came across some data models for which we used EMF to comply with the requirements claimed in Section 1. In the following subsections we discuss the uses of EMF by means of three different kinds of data models. Using EMF already entails that an explicit data model is provided as input. Hence, our first requirement is accomplished per se.

The example in Section 4.1 is concerned with code for handling data entities in our domain, e.g., students and courses. The user should be able to view and edit these entities in a form. From a programmer's point of view, the code for creating the form and the data model must be kept consistent, making this an example for our second requirement.

We discuss our third requirement, type-safety, using the example of properties. Properties are used in the eAssignment application, e.g., to let a user specify the host name and port number to be used for down- and uploading projects. The Eclipse platform provides a generic framework for editing, accessing and persisting preferences. The genericity, though, comes at the cost of type-safety. Section 4.2 shows how we use EMF to regain type-safety while exploiting Eclipse' infrastructure.

Extension points are used in eAssignment to enable third-party plug-ins to augment its functionality. The Core plugin includes a repository view that shows all projects – either assignments from the teacher or solutions from the students – on a server. We provide, e.g., an extension point to plug in filters for projects in the repository view. Extension points are, as the preferences, handled very generically by Eclipse. Similar to the form data, the structure of an extension point influences different parts of a program. Consequently, this is an example of improving consistency and type-safety at the same time by using EMF.

The EMF-based tools that we developed for eAssignment can also be used in other Eclipse-based projects. A distribution of the tools can be downloaded from our project website [6].

### 4.1 Forms

Graphical environments are the interface between the user

and an underlying data model. The environment reads the model, displays its data to the user, and enables the user to edit the data. One example in `eAssignment` for data models that the user interacts with is administrative information about students, courses and assignments. For instance, a teacher must be able to view the students attending his/her course and edit their attributes like matriculation number or email address.

A developer who creates a form for viewing and editing a data model has to perform some tasks common to all forms. The user can only input `Strings` which must often be converted to some special data type. Additionally, the programmer must check whether the entered `String` has the correct form, i.e., can be converted to the data type, and, if this is not possible, provide an error message to the user. For example, if a data attribute is an email address, the developer will use a text input widget. The entered text must, however, have the form `name@domain`.

Graphical user interfaces (GUIs) in Eclipse are created with the Standard Widget Toolkit (SWT) [8] which basically provides simple widgets, like labels, text fields, buttons and containers. To create a form, first, the developer instantiates the widgets and sets their properties, like an initial value for a text field. The initial value is read from the data model underlying the form. The developer must also implement handlers for events which occur, e.g., when the user changes the value of a text field or clicks a button. In these event handlers the changes must be mapped back to the underlying data model.

JFace is a high-level framework for graphical user interfaces in Eclipse that builds on top of SWT and relieves the developer from the presented tasks with `FieldEditors`. A `FieldEditor` displays a single-valued data attribute and allows the user to edit it. It has a correspondence to a value in an `IPreferenceStore`<sup>1</sup> which is used as the underlying data model.

To use a `FieldEditor`, the developer needs to specify a preference key identifier, the data type to use and the store where the data is read from and written to. However, an `IPreferenceStore` is not always an appropriate representation of data as it is neither structured nor typed. We will discuss this topic in Section 4.2.

However, we were inspired by the concept of `FieldEditors` and present a more general concept for creating forms with an underlying `ECore` model.

### *EFields*

Putting the `FieldEditor` and EMF concepts together lead to the idea of what we call `EField`. An `EField` is basically little more than a JFace `FieldEditor`. But, we employ a more general notion of a data model than a preference store, i.e., an `ECore` model.

An `EField` displays an attribute's value to the user and allows him/her to edit it. Value changes in the graphical representation are committed to the underlying model just as changes in the underlying data model lead to an update in the GUI. The EMF Validation Framework is used to validate the syntax and semantics of user input to ensure the integrity of the data. If the input does not pass the validation, a context sensitive error message is reported to the user.

<sup>1</sup>`IPreferenceStore` is a type in JFace that stores key-value pairs, similar to `java.util.Properties`.

### *EToolkit*

To further automate the creation of forms, we developed the `EToolkit`. An `EField` provides an abstraction for a single data attribute. The `EToolkit` provides functionality to create complete forms for displaying and editing data structures of `ECore` models. To create a form, the `EToolkit` is passed a data model and an SWT container that should display the form. The `EToolkit` creates appropriate controls and adds them to the container.

Using EMF's introspection facility, the structure of `ECore` models can be explored at runtime and, based on the data type of a model attribute, the `EToolkit` automatically instantiates an appropriate `EField` and adds it to the SWT container. In addition, the `EToolkit` automatically creates the labels for each `EField` using information from the underlying model.

## 4.2 Preferences

Eclipse provides the `IPreferenceStore` to save user settings across sessions. A problem with the preference store is that the type of a preference property is not preserved. Preferences are written to and read from the store by using methods such as `void setValue(String name, <type> value)` and `<type> get<type>(String name)`, where `<type>` can be any of Java's primitive types or `String`. Internally all properties are stored as `Strings`. Thus, it is possible to write an `int` property with the call `store.setValue("Name", 0)` and read it with `store.getBoolean("Name")`. This can either cause unexpected behavior of the program or even a crash.

There is also an issue of usability. Each Eclipse plugin has only one preference store which is an unstructured list of key-value pairs. In general, preferences are thought to be single valued and grouping of preferences is not supported. Usually, the developer will define constants for the keys to be used when accessing the preferences in his/her project. Logical grouping of preferences can be achieved by using common prefixes for constants specifying the keys for preferences in the same logical group. Finally, in Eclipse the developer must set the default value of a preference in a programmatic way, if it differs from the type's system default value.

We solved these drawbacks by using `ECore` data models for the preferences of the `eAssignment` application; they are type-safe and structured. To benefit from the Eclipse infrastructure for persisting preferences we implemented an adapter between the `ECore` data model and the preference store. EMF provides a modification tracker that we use to propagate changes of the data model to the preference store at the time a value is changed to keep the `ECore` model and the Eclipse preference store consistent.

When using an `ECore` model for preferences, the `EFields` as introduced in Section 4.1 can be used to automatically generate pages for editing preferences. Additionally, default values can be defined declaratively in the data model without the need of writing code.

## 4.3 Extensions

Eclipse's flexible architecture is mainly based on the plugin concept. Plug-ins can (a) extend the Eclipse platform itself or an Eclipse plug-in and (b) provide extension points that can be used by other plug-ins to add functionality. For instance, `eAssignment` is an Eclipse extension, but also pro-

vides an extension point to add filters for the projects viewed in the repository view. This view shows all projects available from a repository's location, i.e., assignments published by a teacher and solutions submitted by students. Students can only download the assignment projects, but not solutions from their fellows. This is guaranteed by access rights on the server. To hide projects that are not downloadable, the repository view applies all filters contributed to the extension point on the projects available from the repository location. Only the projects passing the filters are displayed in the view.

The extension point is declared in the descriptor of eAssignment's Core plug-in. A plug-in that wants to provide an extension defines a respective entry in its own descriptor. The syntax to which this entry must adhere is defined in an Eclipse XML Schema Definition (EXSD) file also provided by the Core plug-in. The module that is affected by extensions, in our case the repository view, retrieves all its extensions via the extension point registry.

Obtaining all contributions to an extension point from the registry is a tedious and error-prone job. To avoid scattering of registry access code, it has become common practice to encapsulate it in a parser that creates an object-oriented model for contributed extensions. An example for such a model are the *Java filters* in the JDT [4]. To access the contributions the extended plug-in uses the object-oriented model rather than the registry.

However, if the extension point definition is changed all involved classes must be reviewed. Usually the parser used to explore the contributions as well as the object-oriented representation of the contributors need to be adjusted or completely rewritten. Again, we use EMF to automate the task of keeping the extension point's data model, the parser and the model classes consistent. Since version 2.1, EMF allows to define custom model importers that can be used to create ECore models from various sources. EXSD files are an appropriate data source for model imports.

Based on a schema file the EXSD model importer generates a complete set of model classes. In the EXSD file XML elements constituting an extension are defined. Further it is specified, which attributes and child elements an element can have. Elements and attributes become the model classes and their single-valued attributes. References between model classes are created from the child element relations, which can either be *choice* or *sequence*.

We implemented a generic extension point parser that builds data structures complying to an ECore model of an extension. Using EMF's introspection, the data model is analyzed and the registry is read out accordingly, while building the data.

## 5. SUMMARY

The downside of the high degree of genericity of some data features in the Eclipse platform is the loss of important features of data models. Data models should be explicit, used consistently and be structured and type-safe. In this paper we have shown how we regained these features by using ECore data models and generic implementations of intermediary layers that marshal between Eclipse' untyped data format and the statically typed ECore data model, using EMF's introspective features.

## Acknowledgements

We thank Matthias Merz who is implementing the automated test feature for the eAssignment plug-in. This work was supported by the eLearning initiative of Darmstadt University of Technology, "TUD-Online 2005", and an IBM Eclipse Innovation Grant.

## 6. REFERENCES

- [1] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2003.
- [2] Eclipse Modeling Framework. <http://www.eclipse.org/emf/>.
- [3] Hibernate. <http://www.hibernate.org/>.
- [4] JDT. <http://www.eclipse.org/jdt/>.
- [5] Object Management Group. Unified modeling language (UML), version 1.5. <http://www.omg.org/docs/formal/03-03-01.pdf>.
- [6] Software Technology Group. eAssignment. <http://www.st.informatik.tu-darmstadt.de/eAssignment>.
- [7] Spring. <http://www.springframework.org/>.
- [8] SWT. <http://www.eclipse.org/swt/>.
- [9] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures second edition – W3C proposed edited recommendation 18 march 2004. <http://www.w3.org/TR/xmlschema-1/>.
- [10] UML2. <http://www.eclipse.org/uml2/>.
- [11] Visual Editor. <http://www.eclipse.org/vep/>.
- [12] IBM WebSphere Studio product family. <http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=products/studio>.