Designing an Adaptive Weaving Workbench using Eclipse Contributions[1]

Marcos Didonet Del Fabro, Jean Bézivin, Patrick Valduriez

ATLAS Group, INRIA & LINA, University of Nantes
{marcos.didonet-del-fabro, jean.bezivin}@univ-nantes.fr, patrick.valduriez@inria.fr

1 Introduction

Establishing links between models are a fundamental issue in model driven engineering (MDE) practices. There are many situations where it is necessary to create links between models, for instance traceability, transformation specification, composition operations, model evolution, etc. Every application scenario may require the creation of different types of links.

The existence of many application scenarios and link types motivates the development of sets of DSLs (Domain Specific Languages) to capture links between model elements. We propose using weaving models [1] to do this. A weaving model conforms to a weaving metamodel. The creation of weaving models requires adapted user interfaces, for example with link management primitives, or matching algorithms.

Eclipse EMF (Eclipse Modeling Framework) [3] is a plugin for Eclipse [2] that provides basic model management primitives, such as reading, updating, inserting, serialization, as well as and a standard user interface. However there is no specific support for creating weavings models.

In this paper we present an adaptive workbench for developing weaving architectures, calling AMW (ATLAS Model Weaver) [1]. It is based on a generic weaving metamodel that supports basic link management (a generic weaving DSL). The workbench uses the dynamic model management API of EMF. It is based in the contribution mechanism [5] of Eclipse: we develop an extensible workbench where other developers can *contribute* to it with other plugins. We define extension points to handle with specific requirements for creating adaptive weaving plugins.

This paper is organized as follows. Section 2 presents a set of requirements to implement a weaving workbench. Section 3 describes a weaving metamodel as a core DSL. Section 4 presents the adaptive weaving workbench based on the Eclipse contribution mechanism. Section 5 concludes.

2 Weaving Requirements

In order to define an adaptive weaving workbench, it is necessary to handle with basic requirements of a weaving platform. Weavings may have different purposes, for example a merge weaving indicates how to merge elements of two models. A traceability weaving indicates that one model is a new version of another. A weaving workbench should have at least the following features:

- The manipulated weavings should express the notion of *links* between model elements;
- Different links semantics (and link types) should be supported by the means of an extensible metamodel (different weaving DSLs);
- The weaving should have an identification mechanism to uniquely identify the model elements;
- Different matching algorithms or heuristics may be executed to semi-automatically create weavings;
- The workbench must support different interfaces (e.g., woven models represented as trees or graphical views);
- It must be possible to weave from 1 to N models.

3 Weaving Metamodel

We propose a core weaving metamodel based on the notion of links between model elements. The weaving workbench is implemented based on this core metamodel. We illustrate it in Figure 1.

[1] Work partially supported by ModelWare IST European project 511731 and a grant from Microsoft Research, Cambridge, UK.

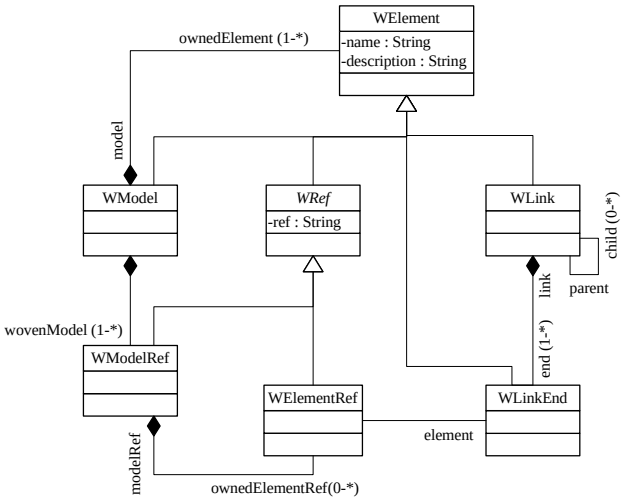

Fig. 1. The weaving metamodel

WElement is the base element from which all other elements inherit. *WModel* represents the root element that contains model elements. *WLink* express a link between model elements, i.e., it has a simple linking semantics. To be able to express different link types, this element is extended by different metamodels. *WLinkEnd* represents a linked model element. This way it is possible to create N-ary links. We associate the *WElementRef* element with an identification mechanism over the related elements. The function takes as parameter the model element and returns a unique identifier for this element. This core is extended with different types of links, for example with equality links, concatenation links, merging links, depending on the application scenario.

4 Weaving Workbench

In this section, first we briefly introduce EMF and its adapter mechanism. Then we describe how we provide our own extension points to implement a generic weaving prototype.

4.1 Eclipse Modeling Framework (EMF)

EMF is a model driven plugin for Eclipse that manipulates models and metamodels conforming to the Ecore [3] metametamodel. EMF relies on the notion of *adapters*. Every model element (or metamodel element) is associated with one or more adapters. An adapter provides a set of interfaces that must be implemented to support different functionalities. There are different types of adapters, for example label adapters, content adapter, etc. The implementation of an adapter is called *item provider*. Consider for instance the standard three-based EMF interface: one label adapter is responsible to show the correct label for a given node (e.g., the element name); the content adapter contains the model element itself.

The common process of EMF is to automatically generate Java code (with a set of adapters) based on the Ecore metamodel. The generated code accesses model elements conforming to Ecore. This way each adapter may be modified as needed to change the application behavior.

4.2 Extensible workbench

We develop a weaving workbench with basic functionalities for creating elements conforming to the core weaving metamodel. However, we do not generate Java code to handle with this metamodel. We use the dynamic API of EMF. The implementation acts over the Ecore metametamodel elements, so it is generic regardless the weaving extension. The interface auto-generates menus for each different metamodel element. This simple core provides standard behavior for a weaving architecture.

To customize it with different interfaces, matching algorithms, metamodel extensions, we provide a set of extension points definitions. An extension point definition describes how different plugins may be plugged to our weaving workbench as well [5].

4.2.1 Metamodel extension point

We define an extension point to be able to add new links. This means we may add a metamodel extension plugin defined in KM3 [4]. We must set a *filename* and a *path* property to indicate where the KM3 file is located. The workbench searches for the metamodel extension to load it automatically.

4.2.2 Adapters (Item providers)

There is an extension point that contributes with new adapters to each metamodel element. This way each metamodel element may be customized. The adapter class with the implementation must extend *IWeaverItemProvider* interface. The adapter may be a provider of all the children classes of the given class. This is specified in the *isChildrenProvider* property.

The adapter may support different identification mechanisms. This means for example that we may identify a model element by its XMI-ID or using XPointer. It is necessary to implement *IIdentifierAdapter* interface, with two methods: *public Object getID();* and *public void setID(Object obj);*.

In the model weaver prototype, the default extension implements a standard adapter that sets an XMI-ID for every created object.

4.2.3 Initialization extensions

This extension point definition enables executing pre-defined operations when loading a weaving model for the first time, for example a matching plugin that contributes to this extension point to partially produce a weaving model. The matching plug-in must implement a predefined interface as well.

4.2.4 Panel extensions

The weaving workbench supports two types of panels: panels that contain the woven models and one panel with the weaving model (they are also contributions to the workbench). We create two extension point definitions: one for the main weaving panel and one for the woven models. This way it is possible to weave several models.

We use the standard EMF interface for implementing both panels. However, it is possible to change the panels, let us say by graphical user interfaces, for example using Eclipse GMF.

Weaving model panel

This extension defines the main component of the Model Weaver workbench. The panel must contain all the logic that handles a weaving section (selecting, creating, updating new links, showing them in a graphical way, etc.). It provides specific interfaces to change the menu behaviors as well.

Woven model panels

This extension definition enables defining different panels for the woven models. The panel is a user interface for a metamodel or a model that will be woven by the Model Weaver. Several panels may be defined. In the first time one creates a weaving model the user is asked which is the desired user interface that is used.

4.3 The prototype

We create a prototype called ATLAS Model Weaver (AMW) that contributes to the extension point definitions described above to create a complete weaver workbench. The prototype is part of the AMMA (ATLAS Model Management Architecture) platform [7]. The AMMA platform is also formed by different plugins, for instance AM3 and ATL plugins. These plugins are all available as Eclipse GMT subprojects [6]. AMW is dependent of ATL plugins to produce a single weaving metamodel by taking a set of weaving metamodel extensions.

The screen-shot of Figure 2 is formed by 3 panels. The panels from left and right (*mantisModel* and *bugzillaModel*) contributes to the *woven panel* extension point definition. These panels use the standard tree interface provided by EMF. The panel in the middle (*WeavingModel*) contributes to the *weaving panel* definition. The weaving model conforms to a weaving metamodel. The weaving metamodel is the result of extending the core weaving metamodel from Section 3. The extension is a KM3 file that is itself a contribution to the *metamodel* extension point. We show an excerpt of the metamodel extension below:

The model element "*Attribute Equal Summary*" from the weaving panel links *summary* in the left panel with *short_desc* in the right panel. This element conforms to the *AttributeEqual* element.

The weaving model is manually created by the help of the user interface. Only the *Bug Model* element is automatically created by a contribution to the extension definition to initialize the model.

The last contributions concern the adapters (item providers) and the identification mechanism. In this example we contribute with an adapter that changes the label of the model elements. The name of the references are explicitly shown (e.g., *child*, *source*, *target*) and the names have different separators (<> and <<>>). The adapters also modify the property panel of each element: it shows not only the main attributes of the model elements but also the value of the child elements. The adapters automatically generate an XMI-ID for every model element. In the example the ID is the type of the model element plus an incremental number (e.g., *EAttribute27*).

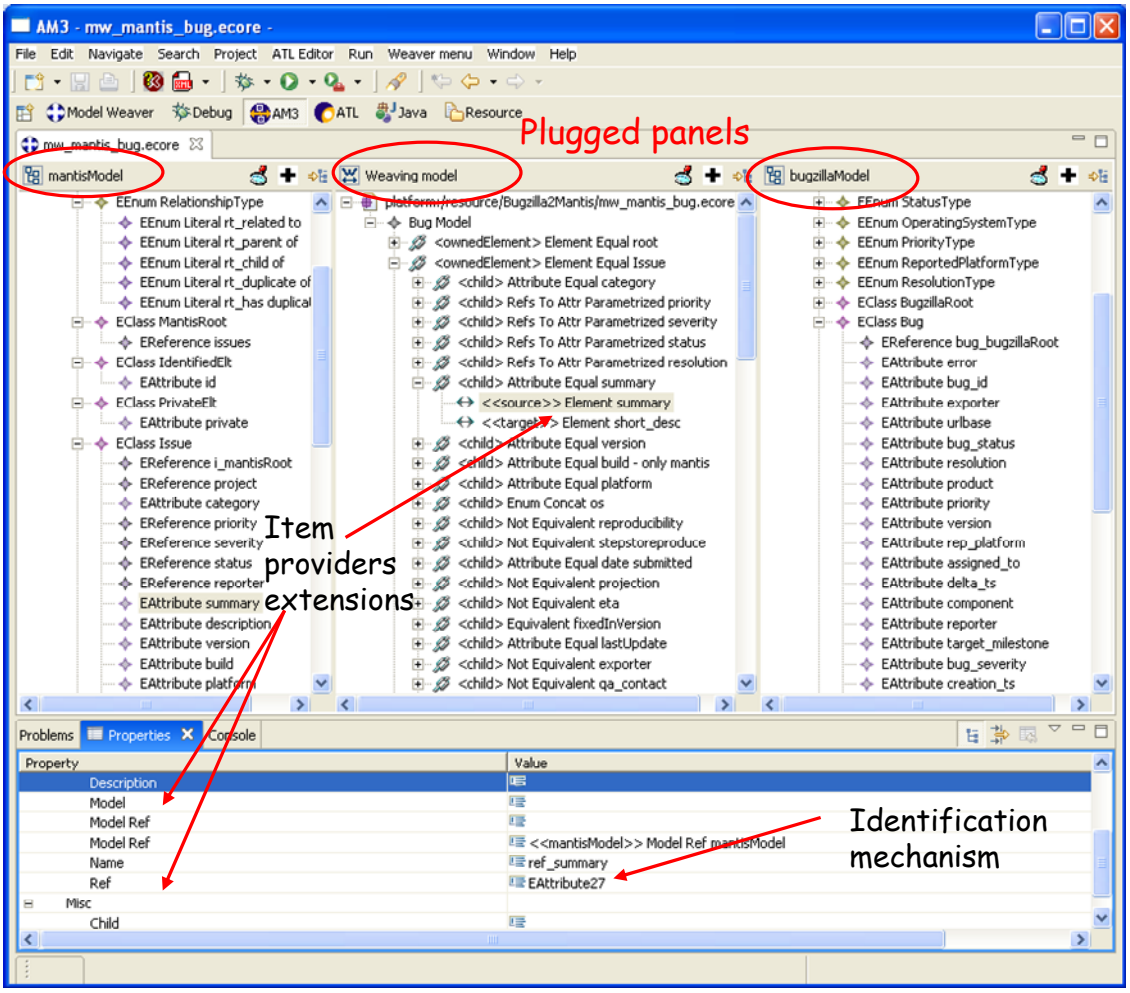

Fig. 2. The AMW prototype

5 Conclusions

In this paper we have shown how we used the Eclipse contribution mechanism to create an adaptive weaving workbench. The weaving workbench is part of the AMMA model management architecture. The prototype uses different functionalities provided by other AMMA plugins (for instance the ATL engine plugin).

We have presented a set of extension point definitions to contribute to the core weaving workbench. It is possible to add different panels (weaving models and woven models); to create adapter extensions; to add weaving metamodel extensions; to execute different algorithms in the moment models are loaded and to associate different identification mechanisms with the metamodel elements.

The different extension point definitions enabled a quick customization of the weaving workbench for different applications. They should be well document to be able to use them correctly. We contributed to this extension points by implementing the current version of the AMW prototype.

References

1. Didonet Del Fabro M, Bézivin J, Jouault F, Valduriez P. Applying Generic Model Management to Data Mapping. (BDA05), 2005, Saint-Malo, France
2. Eclipse Foundation. http://www.eclipse.org
3. EMF Eclipse Modeling Framework. http://www.eclipse.org/emf
4. Jouault F, Bézivin J. KM3: a DSL for Metamodel Specification. In proc. of 8th FMOODS, LNCS 4037, Bologna, Italy, pages 171-185
5. Gamma E, Beck K. Contributing to Eclipse: Principles, Patterns, and Plug-Ins Addison-Wesley, Boston, 2004. 395
6. GMT Home Page. http://www.eclipse.org/gmt/
7. The AMMA platform. http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/