A First Experimentation on High-Level Tooling Support upon Fractal

Frédéric Loiret, David Servat, Lionel Seinturier

A first experimentation on high-level tooling support upon Fractal

Frédéric Loiret[1,2], David Servat[1], Lionel Seinturier[2]

[1]CEA-LIST Team L-LSP/Accord, F-91191 Gif sur Yvette Cedex
[2]INRIA Futurs, USTL-LIFL, Team GOAL/Jacquard, F-59655 Villeneuve d'Ascq

{Frederic.Loiret, David.Servat}@cea.fr, Lionel.Seinturier@inria.fr

In this abstract, we present a first experimentation on tooling support based on Fractal component model. The motivations which encouraged us were to provide an homogeneous design environment for the user to specify Fractal applications.

This experimentation is based on Think implementation (and associated tools) of Fractal. However, it deals more on generic structural aspects of the component model rather than its configuration and control ones, it can be regarded as being independent of Fractal implementations.

In the first section, we briefly present functionalities of EMF framework which was used to design Fractal ADL as a meta-model described in the second section. The third part is dedicated to a first attempt about a reverse engineering tool dealing with behavioural aspect of components.

Eclipse Modeling Framework overview

EMF[BSE+04] is a modeling and code generation framework for developing meta-model based applications and design tools. Meta-models can be specified with class and association constructs to describe their structure, hence EMF can be thought of as a Java implementation of (a subset of) the MOF[Gro06a] (OMG's meta-object facility which give grounds to the UML), it is called Ecore. From this core meta-model, the EMF framework provides a code generator to automatically derive a Java API to allow model instance manipulations. A built-in XMI[Gro06b] serialization provides for model persistence support. Moreover a tree-like model editor plugin can be generated, which makes use of the API and may serve as the basis for more elaborated user tooling environments.

Beyong those functionalities support, we have fixed one's choice on this framework because it provides the glue between programming and modelisation worlds. Concretely, it makes it possible to easily implement the link between tools provided by Fractal compilation chain (parsers, generators...) and those of an higher level we implemented.

An EMF meta-model for Fractal architectural concepts

We have used EMF as a means to define a meta-model regrouping all Fractal architectural constructs. For lack of space we restrict our presentation here to the most important ones.

The meta classes introduced can be thought of as the regrouping of both the abstract syntax tree of Fractal's ADL and its IDL ; some minor elements have also been added for data types and package management.

We mainly used a hierarchy of interrelated concrete and abstract Ecore classes to describe this meta model : only the concrete classes will be instantiated and manipulated as model elements. Abstract classes enable the factorization of common features in a set of classes via inheritance relationships.

Classes feature two types of relationships : *containment* which specify an ownership and composition relation from one object to a set of others (e.g. a component with its required and provided interfaces) and reference relationships which specify a visiblity or accessibility from one object to another (e.g. in a `client-server binding` there exists such a reference link from a `required interface` of a client component to a corresponding `provided interface` of a server component).

Together with this structural description it is already possible to add semantical constraints, i.e. right from design time, as all concepts are typed ; in our previous client-server example, a `required` interface can only be associated to a `provided` interface, given the meta model construction.

As said previously, thanks to the EMF code generation, we can derive an API and tree-like model editor in the form of a dedicated plugin associated to the meta model we just defined. This plugin can be connected to the already existing Fractal/Think tools. It is a matter of synchronizing the provided Fractal textual specifications with our EMF model instances. For this we used various existing tools which enable us to browse the AST (Abstract-syntax tree) from Fractal ADL to generate its EMF representation. The converse is done thanks to code generation. We did this for the Kortex Think[1] library.

From a user point of view, the graphical user interface provided with the EMF model editor eases the specification of the application architecture. For *containment* relationships for instance, no wrong assignement can be done, models are correct by construction, with respect to the underlying meta model. The same is true for reference associations, a contextual menu provides with a restricted list of allowed objects that can be referenced : mistakes are avoided. Navigation among model elements is more intuitive for the user who benefits from the overall Eclipse development environment facilities and capitalize on his Eclipse developing experience. In addition, persistent storage of components libraries models (like Kortex one) allows a simple and efficient manner for the user to reuse what exists.

[1] Think V2 tools

Towards behaviour representation

We have already started to investigate further in the development of a higher-level tooling environment for Fractal, which would go beyond the structural issues presented above. We have developed an early prototype version of a code analyzer which parses the source code of primitive components - so far we worked on Think C-code built-in components - to tackle behavioural issues.

The tool analyses the message traces exchanged by components with their environment. Such traces are specified with I/O automatons whose transitions are labelled with respect to the calls invoked on the operations listed on the interfaces of the component. This is mainly inspired from the work by [Bar05] in the context of *SafeArchie*.
From the primitive component source code and its interface description, an abstract syntax tree is produced, then analysed to extract the corresponding control flow graph, finally reduced to the sole invokations of interface operations.

We are actually working on an first extension of our EMF meta-model to include corresponding constructs for this behavioural description. Its interest is double : it could provides to user a representation of execution flows for a given components configuration. It could also offers behavioural analysis capabilities on components assemblies (deadlock, safety, liveliness...). Such functionalities could be integrated in an homogeneous way (above Fractal ADL) to the editor environment described.

Conclusion and perspectives

We think that this homogeneous environment (around Eclipse platform) could allow to extend Fractal users field when it will be fully implemented.
In extension, and as a possible perspective, we can make a projection of those architectural concepts as a UML2 profil (in an automatic way by a tool implemented in our lab) to permit interoperability with UML2 Eclipse plugin or other specification/modelisation tools.

Our field of interest also concern using Fractal component model (and its Think implementation) to design real time applications. For the specification level, we will extend the EMF meta-model to provide annotation mechanism for non-functional properties, like temporal and QoS ones.

Références

[Bar05] Olivier Barais. *Construire et Maitriser l'Evolution d'une Architecture Logicielle à base de Composants*. Thèse de doctorat, 2005.

[BSE+04] F. Budinsky, D. Steinberg, R. Ellersick, E. Merks, S.A. Brodsky, et T.J. Grose. *Eclipse Modeling Framework*. Addison-Wesley, 2004.

[Gro06a] Object Management Group. Meta object facility (mof) core specification, 2006.

[Gro06b] Object Management Group. Xml meta interchange (xmi), 2006.