From Platform-Independent to Platform-Specific Models using Democles

Christian Glodt
University of Luxembourg
christian.glodt@uni.lu

Pierre Kelsen
University of Luxembourg
pierre.kelsen@uni.lu

Nuno Amálio
University of Luxembourg
nuno.amalio@uni.lu

Qin Ma
University of Luxembourg
qin.ma@uni.lu

Abstract

Democles is an executable modeling tool. It is based on a formally defined language named EP that allows both the structure and behavior of a system to be represented. Earlier versions of the tool allowed platform-independent models to be described using EP-models, which have the same level of granularity as classes. The present demonstration will focus on two new features of Democles: a high-level grouping of EP-models into domains which more faithfully represent the different subject matters that make up a complex software system, and a mechanism for mapping the platform-independent model to a concrete platform, namely Java, using platform bindings.

Categories and Subject Descriptors D.1.5 [*Programming Techniques*]: Object-oriented Programming; D.1.7 [*Programming Techniques*]: Visual Programming; D.2.6 [*Programming Environments*]: Graphical Environments; D.2.6 [*Programming Environments*]: Integrated Environments

General Terms Design, Languages

Keywords executable models, platform-independent model, platform-specific model, domains, visual programming, code generation

1. Introduction

The Democles tool is a research tool whose main purpose is to advance the state of the art in executable modeling.It targets a model-centric approach to software development. At its core is a declarative executable modeling language named EP that allows both structure and behavior of a system to be specified.

The high-level modules of a software system modeled with Democles are domains and bridges. Domains represent distinct, self-contained subject matters while bridges provide mechanisms for propagating behavior across domains. Democles provides an executable modeling environment. While several other tools exist for executable modeling (e.g., executable UML [7]), our tool distinguishes itself from these other tools by having a formally defined modeling language [3, 5] and providing a declarative description of behavior.

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.
ACM 978-1-60558-768-4/09/10.

2. The Underlying Framework

In this section we give a brief overview of the theoretical framework in [2, 4] that underlies the Democles tool. At the heart of the framework is the EP-language that expresses both the structure and behavior of a system. The key concepts are events and properties. These are grouped in EP-models (similar to UML classes). A property expresses a structural feature of an EP-model while an event represents a behavorial feature. Events can impact properties and thereby modify the state of a system. Event propagation is represented by edges connecting events.

EP-models constituted of events and properties can be used to describe a platform-independent model of a system. This was demonstrated by an earlier version of the Democles tool [1]. In the present version of Democles two more important features have been included: a higher-level structuring concept based on domains and bridges was introduced, and a mechanism for mapping to a concrete platform has been provided with platform bindings. In the remainder of this section we describe domains and bridges in more detail. Platform bindings will be discussed in the next two sections.

Complex software applications deal with many different subject matters. As an example consider a Web Application for electronic banking. Such an application has a business domain dealing with the business objects relevant for a banking application, a graphical user interface domain representing concepts related to the graphical user interfaces of web applications, a security domain that expresses the security policies, to name just a few.

To express these different subject matters we introduce the notion of domain as a first-class concept: a domain is a self-contained subject matter. It is expressed as a collection of EP-models that constitute the domain and that do not have references external to this domain. Domains are not sufficient to express a working system since the domains have to interact to realize a software system. As an example consider the *GUI* and *Banking* domains of the aforementioned banking system: events in the GUI domain (triggered by the user) may affect the accounts managed in the Banking domain (e.g., by adding a new account). This implies that we have to be able to propagate behavior from one domain into another domain.

Bridges constitute the main mechanism for effecting this propagation. A bridge is defined over a number of domains. In con-

trast to domains, bridges have external links into the domains over which they are defined. Through these external links events can be propagated from one domain to another domain. Bridges are also responsible for converting data types from one domain to another domain.

We can view the bridge and its underlying domains as a new domain since the union of these structures does not have external links. We can thus view a system model as a hierarchy of domains, with a single top-level domain representing the application.

3. Tool Functionalities

3.1 Modeling Domain Hierarchies

Democles provides a fully featured environment for modeling using Domain Hierarchies and the EP language. In addition to the graphical modeling tools, it offers, among other features, immediate syntax checking, model refactorings and model debugging[6].

The modeling process is supported by a number of views: (1) *Domain View*: displays the structure of a domain hierarchy as a collection of domains and bridges; (2) *Event Tree View*: displays a graphical representation of the event propagation across domains and bridges; (3) *Code View*: allows quick viewing and editing of code snippets associated with certain model elements;(4) *Event Navigator View*: lists events within a system and allows quick access to the containing models.

In order to simplify the evolution of models in a system a number of refactorings are available. These allow to rename bridges, domains, models, events and properties, resulting in changes to all entities that refer to them by name, thus maintaining the integrity of the system.

3.2 Platform Bindings

An important feature of Democles is that it generates compilable and executable Java code from the domains and bridges in a system. Recently, "platform bindings" for domains have been added, allowing them to be realized and to interact and integrate with the underlying platform.

Several such bindings can be present for each domain. Thus, a generic domain, eg. a "GUI" domain containing models of graphical user interface elements, can have realizations using different GUI libraries (for example Swing and SWT). Only one binding per domain can however be marked as active at runtime. By not activating any binding, a system can still be executed as a platform-independent model, and interacted with using Democles' debugging component.

In addition to its previously-described modeling functionalities [6], Democles now provides support for creating, exporting, importing and managing these platform bindings.

4. Tool Architecture

The Democles tool is an Eclipse plugin that makes use of the graphical editing framework GEF and the OCL component of Eclipse's model development tools project. It was developed using current best practices such as revision control, issue tracking, unit testing and continous integration.

Democles implements platform bindings for domains using adapter classes. These adapter classes effectively map EP semantics to platform semantics. In general they instantiate a platform specific counterpart for a model instance, and forward events and property changes both ways, from platform to EP runtime and vice-versa.

These adapters are instantiated at runtime by a factory, using a naming convention to locate the adapter class for a model instance. Switching between different platform binding implementations for a domain is currently implemented by adjusting the Classpath such that the desired implementation is found. When needed, more elaborate adapter lookup schemes can easily be implemented.

For the EP runtime to be able to use the adapters, they must implement the *IBinding* interface which specifies methods that the runtime will call on certain EP-level occurrences, namely changes of property values and event invocations. The adapters must also provide a constructor taking an *IInstance* object, representing the EP model instance that they bind to the platform, as parameter.

This protocol for implementing adapters leaves freedom to the binding developer to cope with different platform idioms. For example, *Components* in Swing are added to their parents by calling the parent's *add(...)* method, while *Widgets* in SWT take their parent as a constructor parameter.

5. What the audience will see

In order to introduce the audience to Democles, we will first give a quick overview of the structure and semantics of the EP language.

Using an example application we will show how Domains are used in Democles to model self-contained subject matters, and how Bridges are used to combine Domains into functional applications. This part of the demonstration will make use of Democles' domain view, as well as other features that simplify the modeling process.

To illustrate the runtime behaviour of EP systems, we will run the example application under Democles' debugger, without using any platform bindings. The application's object graph will be clearly inspectable, and while the application will not be able to perform input or output operations, the debugger will allow interacting with it in order to simulate user interaction.

Next we will demonstrate the platform bindings of the GUI domain. The application will be run with the Swing and the SWT implementation of the bindings, resulting in the corresponding user experiences. We will show the structure of platform bindings, and the idioms employed in the development of these bindings, by elaborating on the implementation of a binding of a common GUI widget.

References

[1] Democles Website. `http://democles.lassy.uni.lu/`

[2] Pierre Kelsen and Qin Ma. Domain Hierarchies: a Basic Theoretical Framework for Integrating Software Domains, Extended Abstract, TASE 2009, July 29 - 31, 2009, Tianjin, China.

[3] P. Kelsen and Q. Ma. A formal definition of the EP language. Technical Report, University of Luxembourg, 2008. `http://democles.lassy.uni.lu/documentation/TR_LASSY_08_03.pdf`

[4] P. Kelsen and Q. Ma. A language for domain hierarchies with applications to the domain integration problem. TR-LASSY-08-05, University of Luxembourg, 2008. `http://democles.lassy.uni.lu/documentation/TR_LASSY_08_05.pdf`

[5] P. Kelsen and Q. Ma. A lightweight approach for defining the formal semantics of a modeling language. MODELS 2008, LNCS 5301, pages 690–704, 2008.

[6] C. Glodt, P. Kelsen, E. Pulvermueller: DEMOCLES: a tool for executable modeling of platform-independent systems. OOPSLA Companion 2007: 870-871.

[7] C. Raistrick, P. Francis, and J. Wright. *Model Driven Architecture with Executable UML(TM)*. Cambridge University Press, 2004.