A generic solution for weaving business code into executable models

Eric Cariou, Olivier Le Goaer, Léa Brunschwig, Franck Barbier

University of Pau / LIUPPA, France
Introduction

• Interests of model execution
  • Clear separation between behavioral and business parts

• Business
  • What to do: call of a Web service, request on a data-base...

• Behavior
  • When and why doing something
  • Specified by a state machine, a Petri net, a workflow...

• Software implementation
  • Weaving business operations with a behavioral model
    → Technical/scientific problem
Challenges

• Developing an executable DSL and its execution engine
  • Well-known: Ecore, Java EMF, Kermeta, GEMOC ...
• How to weave business operations with the executable model and its elements?
  • Java methods with various number and type of parameters with returned values becoming parameters of other methods
    • Need to manage a data flow
  • The execution engine is agnostic: independent of the content of the model to execute

exec: registration(EXE18) → exec: buy_plane_ticket(): ticket → exec: take_plane(ticket)
Challenges

● One solution
  ● Developing the business parts in parallel of the model
  ● Final application obtained by full code generation mixing executable elements and business methods

● Limits
  ● Require to develop business code in an Eclipse/EMF-based tool
  ● If you want or need to use another IDE or reuse legacy code?
    ● How to develop an Android mobile app without Android Studio?
    ● We must be able to escape the Eclipse/EMF world

● Proposition
  ● Xmodeling Studio: a tool for defining executable DSL and execution engines usable in any Java development
Xmodeling Studio

• EMF plugin for helping in the definition of executable DSL

For the language engineer

• Provide generic meta-classes for defining business operations that can be associated with meta-elements of any Ecore meta-model
• Provide generic EMF Java code for automatically calling the business operations within the execution engine
  • By using the Java reflection mechanisms

For the software engineer

• Implement his/her Java business methods on one side
• Specify his/her executable model on another side
• Embed the execution engine and its executable model in any Java development, independently of Eclipse/EMF
Example: Process Definition Language (PDL)
Extended PDL

- Meta-model transformation
  - Executable elements are annotated
  - Activity can now define operations

- An operation has
  - A name
  - An object name/tag on which the operation is called
  - Parameters through tags of objects
  - A returned value with a tag

- In the Java implementation
  - A map associates concrete objects with their names/tags
  - We provide generic code to execute the business operations and manage the data flow
PDL Execution engine

- Main code of the engine: the `executeProcess()` operation of Process

```java
public void executeProcess() {
    // get the first activity of the process
    Activity act = this.getStart().getReference();
    do {
        // update the current activity
        this.setCurrentActivity(act);
        // execute the operations of the activity if
        // defined by calling our implemented methods of
        // Xmod_Action that Activity is specializing
        act.onEntry();
        act.onDo();
        act.onExit();
        // go to the next activity
        act = act getNext();
        // end the loop if there is no further activity
    } while (act != null);
}
```
An Android-based PDL model

process {
    t1 {
        label "Get all SMS"
        call as entry getAllSMS on sms result allSMSContent
    }
    The business Java method to call
    On which object

    t2 {
        label "Convert Cursor to JSON"
        call as entry cursor2JSON(allSMSContent) on cloud result json
    } next of t1
    The returned value becomes parameter of another operation

    t3 {
        label "Backup in Cloud"
        call as do save(json) on cloud
    } next of t2
    }

Software engineer: app. implementation

```
// create the initial contents of the map with business objects on which methods will be called
HashMap<String, Object> map = new HashMap<>();
SMSManager smsManager = new SMSManager(...);
CloudManager cloudManager = new CloudManager(...);
map.put("sms", smsManager);
map.put("cloud", cloudManager);
// load the contents of the PDL model through our generated utility class
Process proc;
proc = PDLXmodUtil.loadProcess("SMSBackupWorkflow.xmi");
// set the map through our generated utility class
PDLXmodUtil.setMap(map);
// execute the process: the operation of activities will be automatically called by our generic meta-classes
// and the data flow is managed by the tags in the map
proc.executeProcess();
```
Conclusion

- As a proof of concept: an Android mobile app
  - Add 3 .jar files of EMF in the Android Studio project (size of 2 MB)
  - Add the .jar file of the EMF PDL project
  - Add the .xmi model to execute
  - Successfull deployment and execution on an Android smartphone

- Critics
  - Strange way and perhaps not efficient way of programming
    - Not yet tested for developing large applications
  - Intrinsic problem of executable models due to the complete separation of behavioral and business parts?

- To test it: http://www.pauware.com → Technology