Decision modules in models and implementations

Citation for published version (APA):

Document status and date:
Published: 01/01/2014

Publisher Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Modularization plays an important role in system evolution. The chosen granularity and the content of the modules are able either improve or complicate system modification, requirements traceability and code analysis. Commonly used Object Life Cycle Modules (OLCM) hide the process control points, i.e. the decisions on which path to follow, inside objects. To modify a program or generate a test, the control points inside OLCM have to be analysed. In this paper we propose decision modules (DM), show their advantages for system evolution and investigate the possibility of their implementation in Java programs.

The necessity to modularise the control points had been recognised by the Business Rules community. Modules called enablers were suggested [1] there. An enabler has varying interpretations depending upon the nature of the correspondent object: it may permit (i.e., enable) the creation of a new instance; permit another action assertion; permit an action execution [1]. An enabler is often called an integrity constraint, a condition or a test.

We extend the idea of enablers and define a decision module as an abstract description of the system actions and the states before and after these actions allowing or forbidding these actions. A decision module contains a description allowing to make a choice of an action from a predefined set or to make no choice. Our decision module may be seen as a module because it can be associated with different objects as a separate entity. We name this module a decision module because it forms the condition for the acceptance or refusen of an action. The condition is derived from the pre- and post-states of the action in the life cycle modules of objects.

In a rather advanced form, such an approach to modularization can be seen in protocol models [3]. Protocol modeling [4] uses the CSP parallel composition [2] defined at the level of event accepting and refusing and extended for modules with internal data. The decision modules, localised in protocol models, possess unidirectional dependency. Unidirectional dependency (also known as obliviousness) means that the decision modules can read the information about the state of OLCMs but the OLCMs "do not know" about existence of decision modules. The composition of decision modules with OLCMs does not change the execution sequences specified by OLCMs. This property is called observational consistency. Unidirectional dependency and observational consistency of
decision modules make the system evolution less laborious: adding, modifying
and deleting decision modules do not require changes in the related OLCMs.

It is desirable to implement such properties in executable programs. We have
carried experiments with different implementation techniques (Table 1). Our
experiments have shown that within the common Java paradigm with object
composition, the desired unidirectional dependency cannot be implemented as
the OLCMs have to explicitly invoke decision modules. The event-driven mecha-
nism is also absent.

Using Publisher-Subscriber design pattern, the event-driven mechanism can
be implemented. In this case, OLCM becomes a listener of an event. Java Re-
flection allows the decision module to read the state of OLCMs. However, the
OLCMs still need to invoke decision modules and, thus, “know about” them.

EJB 3.0 specification completely supports implementation of decision mod-
ules with the interceptor mechanism. In order to be composed with decision
module, the OLCM should contain an @Interceptors annotation of the busi-
ness method corresponding to an event. This annotation informs the application
server that before invocation of this business method the corresponding decision
module has to be invoked. The disadvantage of implementation of the decision
modules using EJB3 is obvious: it’s too heavy as it needs an application server
to implement the interceptor mechanism. However, if the system is already im-
plemented as an enterprise application, this may be a viable solution. For the
classes that have the same external behaviour (implement a certain interface)
the decision modules can be implemented with the Decorator design pattern.

These results form a promising start for implementation of decision modules
for real Java projects.

References
3. A. McNeile and E. Roubtsova. Aspect-Oriented Development Using Protocol Mod-
4. A. McNeile and N. Simons. Protocol Modelling. A Modelling Approach that Sup-
107, 2006.