

ECOOP '98 Workshop on Tools and Environments for Business Rules

Michel Tilman, System Architect Unisys, (mtilman@acm.org)

Context

This paper describes some experiences in implementing business rules within the context of an object-oriented framework. We use the framework to build applications in administrative environments. These applications often share a common business model, and typically require a mix of database, document management and workflow functionality.

The framework [Ti198] uses a repository to store meta-information about end-user applications. This includes object model, object behavior, constraints, specifications of application environments (object types, attributes and associations that can be accessed), query screens, layout definitions of overview lists and forms, authorization rules, workflow process templates and event-condition-action rules. Fully operational end-user tools consult this meta-information at run-time, and adapt themselves dynamically to the application specifications in the database. Thus we effectively separate specifications of a particular organization's business model from the generic functionality offered by the end-user tools. Rather than coding or generating code, we develop end-user applications by building increasingly complete specifications of the business model and the various business rules. These specifications are immediately available for execution.

End-user configurability

One of the main objectives of the framework is a high-degree of end-user configurability. Often end-user tailorability is just skin-deep. In our case it involves all aspects of end-user application development. Thus (knowledgeable) users adapt the business rules, whereas regular end-users adapt the form and overview list layouts and query screens to their own needs. The business rules ensure consistency in all cases, because their specifications are de-coupled from the application functionality.

Giving the users access to the development tools is not sufficient. Users are becoming increasingly aware that change is a constant factor and that applications are never truly finished. We take a similar view with regards to the development tools. For this reason we aim to develop (most of) the development tools in the system itself. Since this requires one or more bootstrapping steps, we originally started with hard-wired tools, such as an object model editor and a tool to define event-condition-action rules. In a later phase we replaced these editors by applications configured in the system, and discarded the original tools. This way we also re-use all the existing functionality of the end-user tools, such as reporting, printing, importing and exporting and support by business rules.

Another advantage arises from the fact that we no longer need to change the framework in order to customize many aspects of the development tools, such as views, additional management tools or consistency checkers.

The need for a reflective architecture

The key to this approach is a highly reflective architecture. We not only model the meta-information explicitly, i.e. structure and relationships of object types, association types and business rules, we also define the meta-model in terms of itself. We store this meta-meta-information in the repository too.

Some examples

While we provide 'off-the-self' constraints, such as totality and uniqueness constraints, the constraint mechanism enables the user to design and re-use new constraints. For instance, we can easily define generic exclusivity constraints that are parameterized by the associations (actually the roles) that should be exclusive. In part this is possible because the constraint definitions have access to their own structure.

Our authorization mechanism consists of a rule-base and a (simple) inference engine. Although the semantics of the rules allow very dynamic and concise definitions, the authorization mechanism is less well suited for the regular end-user. The reflective architecture enables us to develop simpler tools (in the system itself) for more casual usage.

The former types of business rules are rather passive, i.e. they guard over the consistency of data and business policies. Event-condition-action rules on the other hand are much more active: they initialize objects, enforce explicit cascaded deletes and support the user in workflow processes. But we can put them to other uses as well. For instance, since these rules (in fact all types of business rules) are decoupled from the repository interface (object store) and since the meta-model is expressed in itself, we have the means to enhance the structure of the meta-model and implement additional semantics by means of (amongst others) event-condition-action rules. Thus we keep the kernel meta-model and object store small, and delegate implementation of additional features to configuration in the system itself. For instance, the default behavior in case of totality constraint violations is to raise an exception. In several cases however we would like to enforce cascaded deletes automatically. Thus we extended the meta-model with a 'auto-cascaded-delete' flag. We implemented the semantics of the flag by means of event-condition-action rules.

Using appropriate tools and languages

Rather than chasing a 'one-size-fits-all' goal, we prefer to use appropriate tools and languages for the right job. For one, as we explained before, most of the tools can be reconfigured and enhanced to a large degree.

We use Smalltalk as our scripting language in constraints and event-condition-action rules because it is a simple language and it suits our purposes well. For instance, we avoid an impedance mismatch when accessing the framework class library. The scripting language gives the user implicit access to the query language (by means of Smalltalk select-style blocks) and to the authorization mechanism. In our experience, a typical, say, logic-based language is not sufficiently 'expressive' to cover all our needs. Using a 'general purpose' language makes it less suitable for more extensive or formal analysis, however.

The query language provides high-level access to the repository, hiding details of the underlying database and type of database. All the database operations are defined in terms of the object model rather than relational tables and columns. We also define authorization rules in terms of query expressions.

To support design of workflow processes we provide a graphical editor. Internally, this editor manipulates high-level event-condition-action rules, rather than generating, say, pieces of code.

It is also worth stressing that the various business rules are always triggered, whether we access the repository through interactive end-user and development tools, or through the scripting language.

Performance issues

When confronted with dynamic approaches such as ours, people often fear that performance may be severely compromised. We feel this need not be the case.

For one, although it is a dynamic and very reflective architecture, it is still targeted to a particular use, and thus can be optimized with that goal in mind. For instance, as in any application, paying attention to typical patterns of usage and optimizing for these patterns often yields the most significant benefits. In contrast to more hard-wired applications, however, we need only do these optimizations once at the framework level. If done appropriately, most end-user applications and development tools ultimately benefit.

Even more importantly, the very reflective nature of the architecture lends itself well to optimization, as the tools can reflect on the business rules to a large degree. The authorization rules, for instance, when executed single-mindedly, tend to be rather computation intensive in some of our applications. By analyzing the rules, the tools automatically deduce if any rules need to be checked at all.

References

[Til98] Michel Tilman and Martine Devos, A Repository-based Framework for Evolutionary Software Development, Application Frameworks (Mohamed Fayad and Ralph E. Johnson, ed.), Wiley Computer Publishing, 1998