

A Reflective Framework for Configurable Workflow Processes and Tools

Michel Tilman, Senior System Architect Unisys
Email: mtilman@acm.org
<http://users.pandora.be/michel.tilman>

1 Abstract

This paper describes some key issues in implementing a reflective object-oriented framework. We use the framework to build applications in administrative environments. These applications share a common business model, and require a mix of database, document management and workflow functionality.

So far, we have focused our development efforts on the central administration of schools in the Flemish Community of Belgium, with increasing emphasis on access to the central applications from within the schools and local boards through the Internet. In addition we implemented a few test cases to verify the validity of our approach. We rewrote a rather large application of the Belgian Police Department in a few weeks; this includes training the developer.

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

2 Configurable end-user tools

One of the main objectives of the framework is a high-degree of end-user configurability. Configuration by end-users is often just skin-deep. In our case it involves all aspects of end-user application development: knowledgeable users adapt the business rules and workflow processes, 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.

Users are becoming increasingly aware that change is a constant factor and that applications are never truly finished. Thus we give them the necessary tools to build and configure their applications.

3 Configurable development tools

End-user application developers are also users of the system, with their own requirements. These may change over time, or depend on the kind of applications they are building. Given the generic functionality of the end-user tools and the configuration options, another major goal of our framework consists of replacing as many dedicated development tools as possible by applications configured in the system itself. Thus any enhancement of the end-user tools automatically becomes available for those development tools as well.

4 Configurable workflow process model

We take a similar view for workflow processes. If we look at current standardization efforts, we discover rather vague one-size-fits-all proposals covering most current solutions. This is not surprising, given different perspectives on workflow by vendors and customers alike.

Even within one organization, different departments often have different views on how workflow systems should support their business. Divergent thinking in teams and organization cultures is not something to be avoided. It is a fact of life, and must be supported.

In the context of our project we aimed for a system that is highly configurable, not only with regards to user interface aspects, but also with regards to the meta-structures, including the types of business rules and process models we can express. In the second part of the paper we describe a particular process model configured in our framework. But we view it as one possible instance, just as our organizational model and types of constraints are not pre-defined.

Even so, the current model is fairly generic, although we do not claim it is universally applicable. For instance, heavy-duty transaction processing was never a goal in our project. What we do want to achieve, however, is the means to adapt our meta-structures whenever the need arises, without having to (completely) rewrite our applications.

Two elements are crucial here: the use of a meta-data driven approach and of reflection.

5 Meta-data architecture

What the user perceives as applications in our framework are in fact manifestations of a small set of tools acting on the 'data' in the repository and driven by the meta-data. Thus instead of building applications, we define specifications of applications and store these in the repository. The tools interpret the specifications at run-time. This way it becomes easier to change the nature of the applications, such as the options we provide. Whenever the nature of application specifications change, we write conversion scripts to transform the applications. This is no harder than maintaining and converting regular 'data' in traditional database applications.

6 Reflection

We aim to develop (most of) the development tools and meta-structures 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, exporting and business rules. For instance, we need event-condition-action rules acting on event-condition-action rules to configure the development tools. To bootstrap this process we use a very limited set of primitive rules.

We make all this happen by modeling many aspects that are often hard-wired in other approaches explicitly in the system. This includes structure of event-condition-action rules, constraints, workflow process templates and even the meta-model.

Thus we can change the basic structure of, say, tasks and processes to a large degree without having to rewrite our workflow applications. Most management tools can be (re-)configured in the system, without having to develop new classes in the framework. In our case we essentially added only one dedicated component to the framework to view and edit processes graphically. The rest is configured in the system itself, using the available tools.

7 Workflow process model

For our current workflow process model we use the following components:

- the meta-model
- the regular user and development tools
- the generic constraint and event-condition action rules, including a generic background process manager.

In addition we developed a dedicated editor / viewer to handle graphical representations of processes. We provide appropriate hotspots in our framework to deal with these needs. In order to keep interactions to a minimum, the graphical editor assumes only a minimal task / process structure consisting of a few abstract super-types for tasks and processes. The 'user' is free to extend this model with appropriate attributes and associations in concrete sub-types.

To keep track of incoming and outgoing task assignments we added specific interfaces (in / out baskets), but these additional framework components are essentially just cosmetic face-lifts of the generic functionality for customizing query and overview list windows.

7.1 Concepts

Workflow processes represent cause-connected activities to be performed to achieve a specific goal, e.g. the central board decision process, ad-hoc processes to register and handle incoming documents, or the process to create and distribute new norms and regulations towards the school.

Activities are modeled by means of task types. Tasks may involve several participants, e.g. the person who issued the task assignment and the recipient of the task.

Process templates provide options to guide the user through default process scenarios. Processes can be triggered automatically by specific events, for instance at the start of each day.

7.2 Tasks

A task specifies the interaction between participants in order to perform a specific activity. Participants can be any objects, since object types are independent of the actual workflow engine, process editor or in / out basket model.

As previously noted, some workflow tools, such as the graphical process editor, assume the existence of a minimal set of common properties for all task objects:

- the status property defines the various relevant states for a task with regards to commitment of the participants of the task;
- the previous task / next task association maintains task dependencies;
- timing information includes date sent, reminder date and deadline.

Other properties can be added as needed in the various subtypes of the Task object type. These properties can be viewed or edited using regular list views and forms, or listed in the graphical process history overview.

7.3 Processes

Processes represent the running history of a workflow process, i.e. the tasks performed so far. Task activations are always associated with exactly one process. A process will automatically be created when the user activates a process template. Sub-processes may be activated within the context of a process.

7.4 Process templates

A process template defines default scenarios for a workflow process, by listing the default tasks to be performed, and, at each step, the options to move from one task to another. These default scenarios are essentially modeled as a collection of event-condition-action rules.

Whether users may deviate from this default process is ultimately determined by the authorization mechanism. Instead of relying on programmed exceptions, the process template definition gives users the means to let non-default processes snap back into the default flow if some rule condition is satisfied. Alternatively, users can synchronize explicitly with one or more key steps in the template by activating a rule specifically configured for these purposes.

The mixture of ad-hoc / pre-defined processes and tasks / sub-processes, allows us to define process templates incrementally. For instance, a process template may be initially defined at the level of individual departments, and refined later within each individual department. These sub-processes may be more or less strict, depending on each department's culture.

7.5 Implementing other process models

Other workflow process models can be implemented too. We illustrate two existing approaches here:

- Workflow processes based on intelligent work objects can be modeled using script rules. We model work objects so that they contain the necessary data and state information. Script rules describe the routing algorithms and may be activated whenever the user opens a form on the work objects.
- Process models based on Speech Acts can be modeled using high-level rules. Conversation objects contain the necessary data and state information. The condition expression refers to a matrix describing permissible state transitions. Default acts on conversations, such as promise, decline or counter-offer can be configured by means of action rules.

7.5.1 Event-condition-action rules

Objects obey global constraints defined in the model and exhibit behavior that is used across all applications. End-user applications sharing these objects usually require additional, often context- and contents-sensitive functionality and semantics (business rules). Action rules capture these requirements. While action rules allow semantics to be different across applications or workflow processes, they do not violate the global constraints defined in the model.

The main elements of action rules are events, conditions, actions and rule context. The type of event determines whether the rules implement a business rule or provide additional functionality to the user. If an event happens, one or more rules may be triggered. For each rule we check the condition. If it is satisfied, we execute the action part. A rule may be active within the context of a particular application or workflow process only.

Examples:

- When creating documents we set the creation date and creator. When committing a modified document, we update modification date and remember the last author.
- When removing objects, we delete attached electronic documents by default.
- We add extra functionality to perform operations not directly provided by the generic tools, or to support repetitive work. In the context of workflow processes we often need to synchronize the state of different related objects and guarantee the overall consistency; we configure the functionality to perform the necessary operations in a controlled way. All these rules are typically triggered by events corresponding to menus that only appear in the relevant context (application environment and object type). Forms, for instance, adapt themselves to the type of object displayed (which may be task, a work object, ...), and provide the options that can be performed on that particular object at that time in a particular application or process context. We present these options as context-specific menu items that trigger the corresponding rules.
- Several tasks must be performed automatically, often at regular times. Rules can be triggered by time events.

We provide essentially two types of rules: script rules and high-level rules

7.5.2 Script rules

Script rules require our scripting language (basically Smalltalk), hence they are not targeted towards the average end-user.

7.5.3 High-level rules

To allow end-users to define their own rules, we provide high-level rule types. High-level rules explicitly model common script practices in the repository, such as:

- initializing objects to default values (constants or results of query expressions)
- updating an object at commit time or when a property has been modified
- creating a follow-up task in a workflow process, based on certain conditions, and updating the state of the current task.

7.6 *Background processing rules*

Background processing rules specify actions to be performed automatically, often on elements in the repository. Typical uses are process managers to convert notification requests generated by the system into E-mail messages, and automated tasks within the context of a particular application or workflow process.

Background processing rules used to be dedicated objects and framework components. Now they are implemented by means of action rules triggered by time events. Time events enable us to set up elaborate scheduling strategies, including recurrent events.

8 References

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

Michel Tilman, A Repository-based Framework for Evolutionary Software Development, Meta-Data Pattern Mining Workshop, University of Illinois, 1998

Michel Tilman, Tools and Environments for Business Rules, ECOOP'98 Workshop, 1998