DEONTIC PROTOCOL MODELLING
Modelling Business Rules with State Machines

Ashley McNeile, Nicholas Simons
Metamaxim Ltd., 48 Brunswick Gardens, London W8 4AN, United Kngdom
ashley.mcneile@metamaxim.com , nick.simons@metamaxim.com

Keywords: Behavioural Modelling, Protocols, State Machines, Business Rules, Executable Modelling.

Abstract: State machines can be used as a means of specifying the behaviour of objects in a system by describing their event protocols, this being the relationships between the states that the object may adopt and the ability of the object to respond to events of different types presented to it. We describe an extension to this approach whereby different machines in the composition of a single object have different deontic semantics; covering necessary behaviour, encouraged behaviour and discouraged behaviour. This provides a language that has the expressive power to model the way software interacts with the domain in which it is embedded to encourage or discourage behaviours of the domain.

1 INTRODUCTION

Our interest is in building tools that allow behavioural models to be executed and tested early in the development lifecycle, so that the risk that severe behavioural problems are found at late stages of testing, when rectification can be very expensive, is significantly reduced.

Often, subtleties in behavioural requirements specifications concern the nature of the behavioural interaction between the systems and the domain in which it is embedded. For instance, should a system prevent a particular undesirable event from taking place, or only discourage it? If an undesirable event is allowed, how does the system ensure or encourage correction of the resultant state?

This paper describes a technique for modelling event-driven object behaviour that allows different types of behaviour rule to be expressed in a common modelling language. The ideas in this paper build on the concept of "protocol machines" described by the authors (McNeile and Simons, 2006). In that paper, we described how protocol machines are used to describe the essential, domain determined, behaviour of objects. The key observation of this paper is that protocol machines can also be used to express not only what is essential, but also what is allowed and/or desired. This extension to the semantics of protocol machines yields a behaviour modelling language that has the expressive power to address the subtleties in requirements referred to above.

2 DEONTIC MODELLING

2.1 Indicative and Optative Descriptions

When modelling, it is possible to distinguish between two types of description: those that refer to the application domain independently of the existence of the system, and those that pertain to the role of the system in its interaction with the domain. The motivation for this distinction has been made, for instance by Jackson and Zave (Jackson and Zave, 1995) and by Parnas and Madey (Parnas and Madey, 1995). Jackson and Zave use the word *indicative* to refer to descriptions of the domain, and *optative* to refer to descriptions pertaining to the role of the system, and we will follow this convention.

In general, both kinds of description are necessary when developing a system. The reason for making indicative descriptions is that a system tracks the states of an external reality, in the sense that a project administration system tracks projects and people, a stock control system tracks stock levels, and an air traffic control system tracks aircraft. The system is then able to provide its users with information about the reality:

- To which project is Jim assigned?
- How many widgets do we have?
- Where is flight XX123?

When designing a system it is necessary to understand what states are possible in the domain because the system, in order to track the reality, must be able to mirror these states. Indicative models describe these states and the events that cause state change. The behavioural constraints, specifying what events are possible in each state, inherent in indicative models must be obeyed if the state changes of the system are to correspond to meaningful state changes in the domain being modelled. These constraints are properties of the domain and system must ensure that violation of these constraints is prevented.

However a system will also enforce, or help to enforce, user defined rules or policies:

- If the project budget is greater than £x it must should be approved by a director.
- The number of widgets should not fall below the safety stock level.
- Two aircraft should not approach within a minimum distance of each other.

These reflect requirements of the system, as they describe what we want to be true when the domain and the system interact, and are the subject matter of optative descriptions.

Violation of the constraints contained in optative descriptions is both meaningful and possible, the degree of actual compliance depending on the nature of the interaction between the system and the domain.

2.2 Optative Protocol Machines

In the context of indicative descriptions, refusal of an event by a protocol machine denotes that the machine is unable to ascribe a meaning to the event and is therefore unable to adopt a new state. In this paper we extend the use of protocol machines to optative descriptions. Here the semantics of "acceptance" and "refusal" have to be different, as it both possible and meaningful for events to take place that violate the rules of optative descriptions.

Instead of causing the event to be rejected as unprocessable, acceptance or refusal by machines with optative semantics causes feedback to the source of the event (a user, or possibly another system) on the event's appropriateness, but does not prevent the event from being processed. The form of such feedback is discussed later, in Section 4.

There are two types of optative machine semantics, corresponding to whether feedback is triggered by acceptance or refusal of the event. This is shown in Table 1. The second column of the table indicates which disposition (accepted or refused) of an event by a machine causes feedback to be returned to the source of the event. The third column maps the two types of machine to natural meanings, which we refer to as deontic semantics as they correspond roughly to the ideas of "obligatory" and "forbidden" (or "encouraged" and "discouraged") in deontic logic systems, as discussed for instance in (Hilpinen and Føllesdal, 1971).

Table 1: Types of Optative Machine

Type	Significant Event Disposition	Semantics
D	Acceptance	An *accepted* event is Desired. Example: Replenishing stock that has fallen below the safety stock level.
A	Refusal	A *refused* event is not Allowed. Example: Borrowing a reference book.

Together with machines that describe indicative behaviour (which we refer to as Type E, for "Essential") we now have a scheme of three *deontic types* of machine (E, D and A) which can be used in combination to describe an object's behaviour.

2.3 Syntax for Optative Machines

Our general goal is to use the same syntax, described in (McNeile and Simons, 2006), for protocol machines of all deontic types. However the different nature of optative machines (Types D and A) means that they are subject to special rules of form and syntax, which we now describe.

Suppose that an object o is described by a heterogeneous set of machines of all three deontic types. Without loss of generality, we can take it that o is described by exactly three machines (m_E, m_D and m_A), one of each type. This is because:

- We allow multiple machines of a given deontic type to be composed, using the composition rules described in (McNeile and Simons, 2006), to yield a single machine of the same type.
- If o has no machine of a particular type, we can add a machine with an empty repertoire, which will ignore all events presented to it, to establish the complete complement of three types.

The two rules that must be observed if the model of o is to be well formed are:

$$\lambda(m_D) \cup \lambda(m_A) \subseteq \lambda(m_E)$$

$$\sigma(m_D) \cup \sigma(m_A) = \{\}$$

where λ(m) denotes the repertoire of a machine m and σ(m) denotes its local state.

The first of these rules states that the repertoire of the optative machines is a subset of the repertoire of the indicative machine. In other words, the optative machines m_D and m_A do not introduce any new repertoire (event-types) for the object o, but only qualify the event behaviour already defined in m_E through the feedback they provide.

The second rule says that the optative machines have no local state. This is because indicative machines, including their local state, are "analogic models" of the domain, in the described by Jackson in his work on "Problem Frames" (Jackson, 2001); however, optative machines are only advisory, and have no analogic relationship to the domain.

3 EXAMPLE

3.1 Base Example

To illustrate the ideas described in this paper we will use a simple example Project Administration System which tracks projects, their budgets, and who is assigned to work on them. A base protocol machine model is shown in Figure 1.

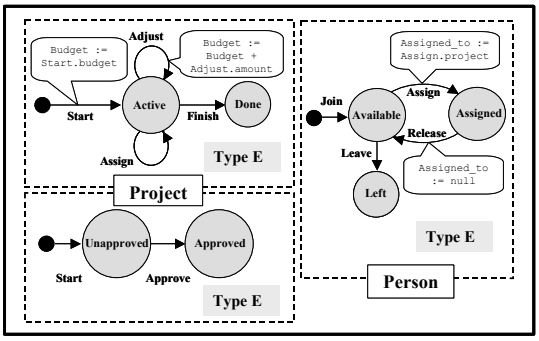

Figure 1: Model for a Project Administration System

This model uses three indicative (Type E) machines: two to define the behaviour of the Project object and one to define the behaviour of the Person object. Note the following:

- The fact that the Assign event appears in both Project and Person machines means that an Assign event cannot take place unless the Project involved in the event is in the state "Active" and the Person involved is in the state "Available".
- The second (lower) machine for Project specifies that, once started, a project budget may be approved. Approval may or may not happen, but can only happen once.

As it stands, this model is purely indicative. We now proceed to extend this base model to illustrate the use of optative machines.

3.2 Addition of a Type D Machine

Figure 2 shows a further, optative, machine with deontic type D for the Project object.

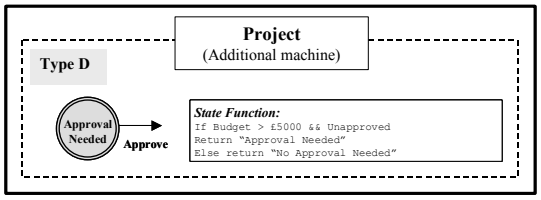

Figure 2: Type D machine for Project

This new machine specifies that approval of a project is *desired* if the Project is currently unapproved and the budget exceeds £5000. This machine has deontic type D. The new machine will not (and cannot) force approval, but will provide feedback if the budget of a Project exceeds £5000 and it is currently unapproved, to encourage approval to take place.

With the addition of this machine the Project object, which already had 2 machines in Figure 1, now has 3 machines: two Type E and one Type D.

3.3 Addition of a Type A Machine

Now we suppose that the Project object has a method:

```
self.estimated_cost()
```

that calculates the estimated cost of the project based on its duration and the people assigned to work on it. (This requires some extra attributes, e.g., duration for Project and daily rate for Person, which we have not shown but which are simple to add to the model.)

Figure 3 shows a further machine for Project, determining when it is allowed to add resources to a Project, with deontic type A. The rule described by this machine is that assignment of a Person to a Project is *not allowed* unless the estimated cost of the project both before and after the assignment is less than the budget. This machine has deontic type A. This machine will not prevent an assignment, but will provide feedback if a Project is in a state in which assignment would violate this rule.

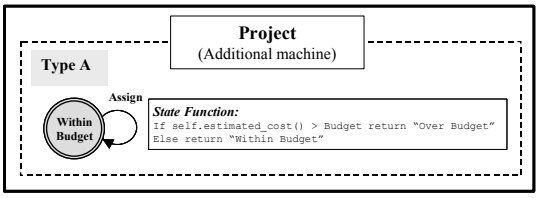

Figure 3. Type A machine for Project

The definition of the Project object now comprises 5 machines, 3 of Type E and one each of Type D and Type A.

4 TOOL SUPPORT

4.1 Purpose of Tool Support

Our interest is in using behavioural models to explore requirements early in the systems development process, using tools that allow behavioural models to be directly executed. This helps reduce the risk that severe behavioural problems are found at late stages of testing, when rectification can be very expensive.

The executable models can be viewed as a form of prototype, and the testing and exploration of such prototypes provides a vehicle for users and other stakeholders to engage in the modelling process, even if they have no understanding of the notations and concepts used to build the model.

4.2 Illustration

Figure 4 shows the appearance that the user interface might take when executing the Project Administration model.

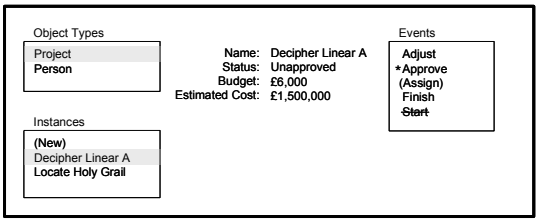

Figure 4: User interface during model execution

The boxes on the left hand side allow the user to browse the contents of the model. The user selects an Object Type (Project in this case) whereupon a list of instances is displayed below.

On selecting an instance in list on the lower left side, the attributes of that instance are shown to the right along with a list of the events that are available on that instance. The list of available events is generated from the model, and a simple coding scheme is used to indicate the constraints imposed on the event by the current state of the model, as follows:

- If the event is not possible according to the Type E machines of the object, it is struck-through (= disabled). Example: ~~Start~~
- If it is possible but not allowed by the Type A machines of the object it is in parentheses. Example: (Assign)
- If it is possible and desired by the Type D machines of the object has an asterisk against it. Example: *Approve
- Otherwise it is shown without any adornment. Examples: Finish, Adjust

Events that are shown as struck-through are disabled, so selecting one of these has no effect. Selecting an event that is not struck-through causes controls to be displayed that allow the attributes of the event to be entered and the event submitted for processing.

In Figure 4, the project "Decipher Linear A" cannot be started as it has already started. Approval is desired as it is unapproved and has a budget of over £5000. Assigning additional people is not allowed as the estimated cost currently exceeds the budget.

REFERENCES

Hilpinen, R., Føllesdal D., (1971) *Deontic Logic: An Introduction*. Deontic Logic: Introductory and Systematic Readings, D. Reidel, Dordrecht 1971, pages 1-35.

Jackson, M., and Zave, P., (1995) *Deriving Specifications from Requirements: An Example*. ICSE17, vol. 1995, pages 15-24.

Jackson, M., (2001) *Problem Frames.* Addison-Wesley, 2001.

McNeile, A., and Simons, N., (2006) *Protocol Modelling*. The Journal on Software and System Modeling. To appear February 2006. Available on-line at http://springerlink.metapress.com.

Parnas, D., and Madey, J., (1995) *Functional Documentation for Computer Systems Engineering.* Science of Computer Programming (Elsevier) 25(1), Oct 1995, pages 41-61.