MODULAR BEHAVIOUR MODELLING OF SERVICE PROVIDING BUSINESS PROCESSES

Ella Roubtsova, Lex Wedemeijer, Karel Lemmen,
Open University of the Netherlands
ella.roubtsova@ou.nl; lex.wedemeijer@ou.nl; karel.lemmen@ou.nl

Ashley McNeile
Metamaxim Ltd, UK
ashley.mcneile@metamaxim.com

Keywords: service providing business process; requirements modelling; standard requirements; flexibility; modular model; executable model; composition

Abstract: We examine possibilities for modularizing the executable models of *Service Providing Business Processes* in a way that allows reuse of common patterns across different applications. We argue that this requires that we can create independent models for different aspects of the process, and then compose these requirements related partial behavioral models to form a complete solution. We identify two areas of modeling that should be separable from the main, application specific, process model: the underlying subject matter with which the process is concerned, and standard reusable process-level behavior that is common across many processes. Using an example of a Service Providing Business Process concerned with *Accreditation of Prior Learning* we show that the *Protocol Modeling* approach has the capability to support modularization of functional and non-functional requirements, when other paradigms cannot completely support it.

1 INTRODUCTION

A *Service Providing Business Process (SPBP)* is an interactive process that transforms the requests of users and the information presented in rules, law regulations or databases of official organizations into a physical product or a document. A Service Providing Business Processes (SPBP) may be either an independent process or an elementary process of a service-oriented business process in the context of SOA (Erl, 2004; Reisig et al., 2007).

SPBPs are subject to constant change caused by social and technological factors. The changes have to be reflected both at the modelling and the implementation level. The need to support continual change motivates the use of modular and compositional approaches in development of SPBPs, both at the modeling and implementation level.

To provide its service, any SPBP utilizes some underlying *subject matter* implemented as databases, resources of the semantic web or official information sites. These provide the raw material on which the service acts. The underlying subject matter may be represented by an *underlying model*. The underlying model has its own dynamics, because the business objects of the underlying model have own life cycle.

Conceptually separate from this, an SPBP itself has a *process model* that describes how it:
(1) Uses and acts on the state and data content of the underlying subject matter to perform tasks (such as data transformation and analysis) specific to the application;
(2) Provides such common (or standard) process-level functions as registration, archiving, security, payment and acknowledgement (of business steps and decisions) to the service-consumer.

In this paper we show that the *Protocol Modeling* approach (McNeile and Simons, 2006) has the capability to support modularization of the underlying model and crosscutting functional requirements, when other paradigms cannot completely support it. Section 2 of this paper describes an example in the SPBP domain. Section 3 presents the modeling semantics that allows separating standard and specific requirements for an SPBP and relate them to models to make models manageable and flexible. Section 4 draws conclusions about flexibility and reuse of Protocol Models.

2 EXAMPLE OF AN SPBP

An example of an SPBP is the process of Accreditation of Prior Learning (APL)(LIFELONG LEARNING, 2008).

The APL service is triggered by a student who follows an education provided by a university. The student requests the university to grant exemptions from the officially listed courses of the current education. The request form contains the student number, the official name of the current education, the official name of the former education of the student and the certificate of the former education. If the former education is not on the list of educations recognized for exemption, the request is rejected and the student is informed. If the former education is on the list of recognized educations, the request is registered, the student receives an acknowledgement and a request for payment. When payment has been received, the assessment team receives the request of the student.

The assessment of the certificate of former education is fulfilled by a trained assessor. If the certificate is complete and valid, the student gets exemption from a set of courses of his current education. The student is informed by receiving an official document about the exemptions. If the certificate is not valid, the assessment is negative. If the certification is incomplete, the student gets one chance to send a complete certificate and have the assessment repeated.

The APL process transforms a request of a student and the information in the official sources into a product: a document of assessment. The official sources about *Education*, *Exemption*, *Student* and *Assessor* form the underlying subject matter of the service. The APL process also implements the process-level functions of request registration, payment, acknowledgement and security.

3 MODULARIZATION OF SPBPS

In order to achieve modularization of requirements in behavioural models we have investigated many popular modeling techniques: Coloured Petri Nets (Jensen, 1997), statecharts (Harel and Politi, 1998) and several UML profiles. We have built executable models of our case study using all those techniques and trying to localize requirements in the models. Although sequential functional requirements defining the APL process (registration, payment, assessment) are localisable, such requirements as security, acknowledgement and archiving demand copies of all other models and cannot be localized in conventional modelling notations. We have found that there are three reasons for this. The conventional techniques do not have

- event-refusal semantics needed to model interaction and so do not allow behavior synchronization within a SPBP. Synchronization is usually used to present collaboration of SPBPs (Su. et al., 2008).
- a concept of a derived state. Without this concept, handling changes in a state value that is not represented as a single data attribute requires the generation of artificial internal events.
- event and state abstractions needed for specification of cross-cutting functional requirements.

We show that the Protocol Modelling technique (McNeile and Simons, 2006) possesses the listed concepts and allows localization and composition of functional and non-functional requirements in models.

Protocol Model. A unit of a Protocol Model is a Protocol Machine $PM = (S, D, E, T)$, where

- $S = \{s_1, s_2, ...s_n\}$, $n \in N$, is a non-empty finite set of stored states. A stored state has a corresponding set of attribute values, including the state name.
- $D = \{d_1, d_2, ..., d_k\}$, $k \in N$, is a finite set of derived states calculated using the states of the machine itself and other protocol machines. D can be empty.
- $E = \{e_1, e_2, ..., e_m\}$, $m \in N$ is an alphabet of events, i.e. a non-empty finite set of recognized events coming from environment.
- $T = \{t_1, ..., t_p\}$, $p \in N$ is a set of transitions. A transition t can be of types $t = (s_1, e, s_2)$, $t = (d_1, e, \text{any state})$ or $t = (\text{any state}, e, d_2)$.

A protocol machine (Figures 1,2) may be represented as a set protocol state diagrams, where

- a stored state is depicted as an ellipse;
- the values of attributes in a stored state are presented near the state in a bubble;
- a derived state is presented as a double line ellipse;
- the rules of derivation of a derived state are presented as an expression of the protocol machine;
- a transition is depicted
  - as an arc, i.e. a pair states labeled by the events that cause this transition or
  - as an arrow coming from or to a derived state and labeled by the events that cause this transition.

The semantics of a protocol state diagram is different from the UML state diagram (OMG, 2003) and statecharts (Harel and Politi, 1998).

Behavior of a Protocol Machine. The state diagram of a protocol machine has the following behavioral semantics.

- If an event does not belong to the alphabet of recognized events, the machine ignores it.
- If the machine is in the state that has an output arc labeled by the alphabet-event, it accepts the event and transitions to another state. This is the only possible cause of a transition.

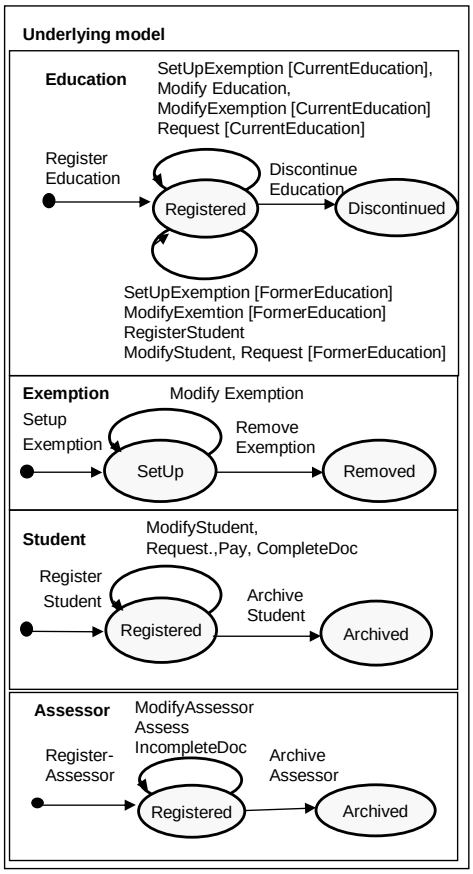

Figure 1: Underlying model of the APL system

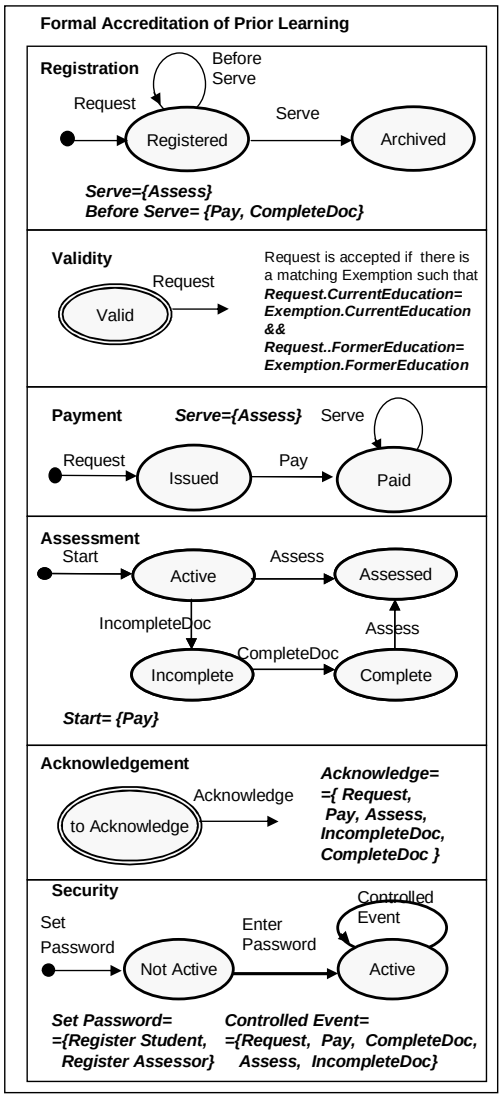

Figure 2: Process Model of the APL System

- If the machine is in the state that has no output arc labeled by the alphabet-event, it refuses the event.
  - If the arrow labeled by an event has no start state, it presents the transition from *any* state to the given derived state.
  - If the arrow labeled by an event has no end state, it presents a transition from the given derived state to *any* state.

The final two rules apply only to machines that have a derived state, where the state is given by a derivation function rather than stored by the machine.

Composition of Protocol Machines. The rules of the composition of protocol machines are the *CSP parallel* composition rule (Hoare, 1985): If there are several protocol machines that recognize an event then this event will be only accepted if all those machines can accept it. A Protocol Model of the APL system is a composition of protocol machines shown in Figures 1,2.

The event synchronization semantics allows easy separation and composition the functionalities of the APL service providing business process. The synchronization works as follows.

- If a student initiates a request, he/she should correctly enter his password, i.e. the *Security* machine should be in state *Active.*
- If the password corresponds to the saved password, then student may initiate the *Request* event. If the model is presented with event

Request: (StudentNumber, Password, FormerEducation, CurrectEducation, Certificate),

the protocol machines *Registration*, *Validity*, *Student*, *CurrentEducation*, *FormerEducation*, *Payment*, *Acknowledgement* and *Security* recognize this event.

- The protocol machine *Validity* checks if there is an

exemption that corresponds to the current and the former education mentioned in the request. If there is such an exemption, the request is registered. This means that the *Registration* machine goes to the state *Registered*, the *Acknowledgement* machine generates an acknowledgement for registration, and the *Payment* machine goes to the state *Issued*, so that the event *Pay* becomes possible for the student. This way all the machines combine to produce the system behavior.

Event Abstractions. Protocol machines may use *generic* events, i.e. the aliases of events from a set. For example, event *Acknowledge*={*Request, Pay, Assess, IncompleteDoc, CompleteDoc* }.
Each event that belongs to an alias causes the same behavior of the protocol machine that reacts to the alias. For example, each event of the last set has to be acknowledged. A generic event is a means to separate and reuse the models of requirements.

Local Reasoning. It is proven in (McNeile and Roubtsova, 2008) that the composition of protocol machines results in behavior that preserves the individual behavior of its component machines. This property of models allows local reasoning on behavior of parts about behavior of the complete system. For example, analyzing only the *Assessment* module we can say that the sequence of events *"Pay, IncompleteDoc, CompleteDoc, IncompleteDoc, CompleteDoc"* does not belong to the behavior of the system, i.e. it is imposable to update documents more than once.

4 FLEXIBILITY AND REUSE OF PROTOCOL MODELS

Separate models of requirements make Protocol Models manageable and allow modification of the model without remodeling. A new requirement may
(1) reduce the set of models. For example, if the *Assessment* were free of charge, then the generic event $Start = \{Register\}$ and *Payment* protocol machine should not be used;
(2) modify a condition for acceptance of an existing event in terms of the pre-state or the post-state. For this case, modification is needed of the machine that embodies the state-based condition for event acceptance (as, for example, the *Validity* machine does for the *Request* event);
(3) modify the results of event acceptance in terms of states. This means modification of the topology and/or state calculation of the corresponding machine.
(4) define a set of sequences of related reactions on new and old events. This extension usually means modeling additional functionality as a separate protocol machine, and synchronizing it with the rest of the protocol model using shared events.
Each the model modifications outlined above is local, i.e. has to be done in one protocol machine and does not demand modification to the states or transitions of other protocol machines.

Protocol Models corresponding to requirements are easy to reuse. For example, the *Registration*, *Payment*, *Acknowledgement* and *Security* machines can be reused in a different SPBP. The same set of machines may be reused for an *Examination* service. The *Assessment* protocol machine will be replaced with the *Examination* protocol machine and the generic events will be changed in correspondence with service events, i.e. $Serve = \{Examine\}$. With some extension the same model can be used for a hotel reservation or a conference registration services.

This means that Protocol Modelling supports reuse of common requirements-oriented patterns and simplifies building of executable models among different applications in the SPBP domain.

REFERENCES

Erl, T. (2004). *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall.

Harel, D. and Politi, M. (1998). *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill.

Hoare, C. (1985). *Communicating Sequential Processes*. Prentice-Hall International.

Jensen, K. (1997). *Coloured Petri Nets*. Springer.

LIFELONG LEARNING (2008). *http://ec.europa.eu/education/llp/*.

McNeile, A. and Roubtsova, E. (2008). CSP parallel composition of aspect models. In *AOM '08: Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*, pages 13–18, New York, NY, USA. ACM.

McNeile, A. and Simons, N. (2006). Protocol Modelling. A modelling approach that supports reusable behavioural abstractions. *Software and System Modeling*, 5(1):91–107.

OMG (2003). *Unified Modeling Language: Superstructure version 2.1.1 (with change bars) formal/2007-02-03*.

Reisig, W., Bretschneider, J., Fahland, D., Lohmann, N., assuthe, P., and Stahl, C. (2007). Services as a Paradigm of Computation. *LNCS 4700*, pages 521–538.

Su., J., Bultan, T., Fu, X., and Zhao, X. (2008). Towards a Theory of Web Service Choreographies. *LNCS 4937*, pages 1–16.