Behavioural Model
for a Business Rules Based Approach

Ella Roubtsova, Stef Joosten, Lex Wedemeijer

Open University of the Netherlands
ella.roubtsova@ou.nl, stef.joosten@ou.nl, Lex.Wedemeijer@ou.nl

Abstract. In this paper we define a behavioural model for a declarative approach called Ampersand, which specifies a system using declarations of business rules that should be maintained by the system. Defining events corresponding to the declarations, Ampersand has a derived semantics for simulation of specified business models. Violation of business rules is considered as a driving force of behaviour. Declarations of business rules are used to generate the feedback during model execution and to trace requirements. We assess the level of the current behavioural model in Ampersand and propose changes in modelling semantics allowing specification of service-oriented applications. These changes include partitioning of the underlying static model and definition of high level events and states. Moreover, we define the refuse and failure semantics being the necessary conditions for composition techniques in models and modelling of interactions.

1 Introduction

There is a long tradition of specifying system behaviour in operational approaches. Such approaches as Statecharts [3], Coloured Petri Nets [10, 11], UML Behavioural State Machines (BSM) [14] and Protocol State Machines (PSM) [14] and activity diagrams [14] are operational. They show states and transitions within operating units as well as the relationships between these operating units. The behaviour presented in operational approaches is usually a workflow or a computation tree built from observable as well as internal events in the system.

On the one hand, operational approaches have many advantages. They support reachability analysis techniques and may be used to prove a desired set of properties of designed models. On the other hand, most of operational behaviour modelling approaches lose traceability of requirements in models, i.e. the ability to describe and follow the life cycle of requirements, in both a forward and backward direction at any phase of specification development [6, 12]. This makes the models less reusable and less flexible when requirements are changed. Moreover, operational behaviour models may contain artificial connections and sequences and may have a flavor of models for strong bureaucracy. They often do not fit to the behaviour of modern businesses with their tendency to adhocracy [7].

That is why behaviour modelling has a trend towards declarative approaches. Declarative approaches to system modelling provide flexibility in creating domain-
specific languages. Such approaches as Z [9], the UML’s Object Constraint Language (OCL) [13], Alloy [4, 5] hide the workflow and declare behaviour as sets of observable or internal operations. In order to be sure that the declarations define the desired behaviour, the hidden operational semantics is added to declarative approaches like it is added to Alloy [18] and this enables simulation of declarative models. In declarative approaches “the analysis is not "complete”, it examines only a finite space of cases. But the space of cases is usually huge ... and it therefore offers a degree of coverage unattainable in testing” [5].

At the moment many approaches are experimenting with declarations and their interpretation for business system modelling. A large group of approaches uses business rules [8, 15] as declarations. In this paper we define the behavioral model behind one of declarative approaches called Ampersand [17]. This approach specifies a business system using declarations of business rules that should be maintained by the system. Ampersand has a derived semantics for simulation of specified business models. Violation of business rules is considered as a driving force of system behaviour. The declarations of business rules are used to generate the feedback during model execution and to trace requirements. We discuss the level of the current behavioural model in Ampersand and its possible development for specification of service-oriented applications.

The paper is organized as follows.
Section 2 defines the static model and the hidden behavioural model of the Ampersand approach illustrated by an example.
Section 3 assesses the level of the behaviour model in Ampersand for modelling of modern service-oriented businesses and proposes semantic changes of the behavioral model in Ampersand towards separation of concerns and modelling of interactive behaviour.
Section 4 concludes the paper.

2 The Ampersand Approach

Among different definitions of business rules [8, 15] the Ampersand approach uses the formalization of business rules as relations of concepts [17]. This form of model presentation is accompanied with a hidden behavioral model and a simulator. The aim of the simulation is to test the constrains specified by business rules by execution of events available in the model.

2.1 Terminology

Concept. The basic building block of a business rule in Ampersand is a Concept. We denote concepts as $A_c, B_c, C_c$. In business modelling example of concepts are Order, Provider, Client. A Concept presents a domain or a type.

Population. A population $A$ of a concept $A_c$ in Ampersand is a set of values of concept $A_c$. Concepts may be infinite, but their populations are finite. A population of a concept is a finite set of elements:

$$A = \{a_i\}, i = 1,...,n_A, a_i \text{ is of type } A_c, n_A \in \mathbb{N}.$$
Relation. The Cartesian product of populations of two concepts is called the complete relation

\[ V_r = A \times B. \]

A Relation is a set of pairs defined on two populations of Concepts. A relation is a subset of their Cartesian product:

\[ r \subseteq (A \times B) = \{(a_i, b_j), a_i \in A, b_j \in B, i = 1, \ldots, n_A, j = 1, \ldots, n_B\}. \]

Business Rule. A business rule \( br \) is a boolean expression defined on relations. All possible operations on sets and relations can be used to express business rules. Any business rule can be normalized and presented as a conjunction of compositions of relations [17]:

For example,

\[ br : r \subseteq (r_1 \cap r_3; r_4, r_5); \]

is a business rule where ";" is the operator of composition of relations and

\[ r_1 \subseteq A \times B; r_2 \subseteq B \times C; r_3 \subseteq A \times B; r_4 \subseteq B \times D; r_5 \subseteq D \times C; r \subseteq A \times C \]

Another example of a business rules structure is

\[ br : r = (r_1 \cap r_3; r_4, r_5). \]

2.2 Static Model of a Business System.

The static model of a business system in Ampersand is a tuple of system’s concepts, relations and business rules:

\[ SP = (C, R, BR) : \]

- \( C \) is a finite set of concepts and \( c \) is used to range over \( C \).
- The underlying model of a specified system is presented in Ampersand with relations \( R \) defined on concepts \( C \). We use \( r \) to range over \( R \).
- \( BR \) is a finite set of business rules and \( br \) is used to range over \( BR \).

In the static model all business rules of the system are satisfied.

2.3 Example of modelling in Ampersand

Table 1 represents requirements for an order management system and its static model in Ampersand.

- The Concepts: Client, Order, Provider etc. start with a capital letter.
- Relations \( r_1, \ldots, r_{10} \) are named issues (considered as equal to isIssued’), hasAccount, status, checks, contains, areSold, isAccepted, bills, pays, delivers.
Business rules are named \( br_1, \ldots, br_5 \).

For example, \( b_2 :: \text{isAccepted}=\text{isIssues};\text{hasAccount};\text{status};\text{isChecked} \) means 
"A Provider accepts an order if the bank Account of the Client is not Overdrawn".

<table>
<thead>
<tr>
<th>( n )</th>
<th>Requirement</th>
<th>Ampersand Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \in R_P )</td>
<td>A Client issues an Order.</td>
<td>issues :: Client * Order.</td>
</tr>
<tr>
<td>( r_1^- )</td>
<td>isIssued :: Order * Client.</td>
<td></td>
</tr>
<tr>
<td>( r_2 \in R_O )</td>
<td>A Client has an Account</td>
<td>hasAccount :: Client * Account.</td>
</tr>
<tr>
<td>( r_3 \in R_O )</td>
<td>An Account is not Overdrawn</td>
<td>status :: Account * notOverdrawn.</td>
</tr>
<tr>
<td>( r_4 \in R_P )</td>
<td>A Provider checks if an Account is not Overdrawn</td>
<td>checks :: Provider * notOverdrawn.</td>
</tr>
<tr>
<td>( r_4^- )</td>
<td>isChecked :: notOverdrawn * Provider</td>
<td></td>
</tr>
<tr>
<td>( r_5 \in R_P )</td>
<td>An Order contains Goods</td>
<td>contains :: Goods * Order.</td>
</tr>
<tr>
<td>( r_6 \in R_O )</td>
<td>Goods are sold by a Provider</td>
<td>areSold :: Provider * Goods.</td>
</tr>
<tr>
<td>( r_7 \in R_P )</td>
<td>An Order is accepted by a Provider.</td>
<td>isAccepted :: Order * Provider.</td>
</tr>
<tr>
<td>( r_7^- )</td>
<td>accepts :: Provider * Order.</td>
<td></td>
</tr>
<tr>
<td>( r_8 )</td>
<td>A Provider bills a Client</td>
<td>bills :: Provider * Client.</td>
</tr>
<tr>
<td>( r_8^- )</td>
<td>isBilled :: Client * Provider.</td>
<td></td>
</tr>
<tr>
<td>( r_9 \in R_O )</td>
<td>A Client pays to a Provider</td>
<td>pays :: Client * Provider</td>
</tr>
<tr>
<td>( r_9^- )</td>
<td>isPaid :: Provider * Client</td>
<td></td>
</tr>
<tr>
<td>( r_{10} \in R_P )</td>
<td>An Order is delivered by a Provider</td>
<td>delivers :: Provider * Client.</td>
</tr>
<tr>
<td>( r_{10}^- )</td>
<td>isDelivered :: Provider * Client.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Specification of an order management system.

2.4 Behavioural Model

Any business rule should be maintained by the system. When a business rule \( br \) is not satisfied, the expression presenting this rule is false. For example,

\[
 r \neq (r_1; r_2 \cap r_3; r_4; r_5).
\]
Since relations $r, r_1, r_2, r_3, r_4, r_5$ are defined on populations of concepts, it is defined that the false expression presenting a business rule means that at least one of the relations are incomplete. Incomplete relations may be made complete by inserting and deleting elementary pairs to it. In such a way the false business rule can be made true.

The Ampersand approach is supported with a behaviour simulator for execution of the transition caused by events that can potentially both violate and restore invariance of business rules. The simulator implement the following pair of the static model and the state transition system

$$P = (S, E, T) :$$

- $S$ is a finite set of states $S = \{s_i, i = 1..N_s\}$ such that $s_i$ is all populations of all relations $R$.
- Any state can be initial.
- A final state $s_f \in S$ is a state where all relations are complete and all business rules are satisfied: $BR = br_1 \cap ... \cap br_n = true, n \in N_{BR}$.
- For each intermediate state $s_i \neq s_f$ there are business rules that are not satisfied $BR = br_1 \cap ... \cap br_n \neq true, n \in N_{BR}$ and there is a set of incomplete relations $R_N \subseteq R$.
- $E$ is a finite set of events

$$E = \{e_j, j = 1,...,N_E\}$$

There are only two types of events in the current version of Ampersand:
- $INSERT(a_r)$ insert an element of a relation $r$, $r \in R$, 
- $DELETE(a_r)$ delete an element of a relation $r$, $r \in R$.

Both event types change the population of a relation $r \in R$.

All events of these types for relations $r \in R$ are possible, however the violated business rules give the indication which relations $r_N \in R_N$ should be completed to guarantee the progress of the business process.

$T$ is a finite set of possible transitions:

$$T = \{t_k, k = 1,...,N_T\} t_k = (s_i, s_j, e), s_i, s_j \in S, e \in E\}.$$ 

Any event is accepted in any state.

The behaviour of this state-transition system is a set of accepted events that never refused. In the simulator, every accepted event is accompanied with a message describing violated business rules.

**Behaviour Simulator.** The described behaviour model is implemented in the current version of the behaviour simulator in Ampersand. The behaviour defined by the model is a set of traces of accepted events. Any trace starts from an initial state and is intended to finish in a new state where all business rules are satisfied. In principle, the $INSERTs$ and $DELETEs$ can be infinitely repeated and the final state will be never achieved. It is, however, assumed that the driving force of business rules always directs the traces to the final state.
In our order management system (Table 1) an example of a trace in the simulator may be:

TRACE-EXAMPLE:
INSERT-isIssued(1, Company A),
INSERT-contains(1, Software),
INSERT-hasAccount(Company A, 67589),
INSERT-notOverdrawn(67589, not Overdrawn),
INSERT-isSold(Software, Company P),
INSERT-isAccepted(1, Company P),
INSERT-bills(Company P, Company A),
INSERT-pays(Company A, Company P),
INSERT-delivers(Company P, Company A).

The initial state of this process is a state where all business rules are satisfied, although any state could be initial.

When a Company A issues an order, event INSERT-isIssued(1, Company A) takes place and all business rules \( br_1 \) – \( br_5 \) become violated. Business rules are presented to the modeler in a semi-natural language as an indicator of relations that should be completed. The relations are shown as tables on the screen.

Elements of the tables can be deleted or new elements can be added. In our trace, relations contains, hasAccount, notOverdrawn, isSold are completed by events of type INSERT. After event INSERT-isAccepted(1, Company P) only business rules \( br_3, br_4, br_5 \) remain violated. This gives an indication of the progress in the business process. Each of the next events in the trace decreases the set of violated business rules.

**Traceability of Requirements.** The attractive feature of the Ampersand approach is the traceability of requirements. Business rules are always traceable in the model at run time. Business systems supported by Ampersand models can estimate legibility of any step in the process. That is why the simulator of the Ampersand is used for requirements engineering. Simulation in Ampersand is finding executions that satisfy relations of the underlying model and lead to satisfaction of business rules.

2.5 Restrictions of the current static and behavioural model

The current static and behavioural models restrict the class of the systems that can be modelled in Ampersand.

The current behavioural model is suitable only for simulation of a business system on a local machine that possesses the complete underlying model. This means that

- the models of large local systems with large underlying models become not observable;

the service-oriented systems and systems with distributed components that do not possess the complete underlying model cannot be modelled in Ampersand.

In other words the Ampersand models do not separate concerns and do not support modelling of distributed services.

Our TRACE-EXAMPLE for an order-management system illustrates this statement. The presented trace contains events that add an Account for Client and check if it is overdrawn:

\[
\text{INSERT-hasAccount(Company A, 67589)),}\n\text{INSERT-notOverdrawn(67589, not overdrawn)).}
\]

These events should belong to the traces of another concern, namely, to an account management system.

Event

\[
\text{INSERT-isSold(Software, Company P)}
\]

should belong to the traces of another concern, namely, to a product management system. The order management may only have the read-only access to relations changed by other concerns. However, the separation of concerns is not implemented in the current version of the simulator and the \text{INSERT}(a_r) or \text{DELETE}(a_r) events for relations changed by other concerns can be changed in the order management system.

The work on the simulator is a work in progress. The necessity to separate concerns and support the modelling of distributed services that do not have access to the underlying model of each other drives changes of the behavioural model of Ampersand. In section 3 we propose the semantic changes of the behavioral model in Ampersand.

3 Development of Ampersand

We propose a modification of the static and behavioural models in Ampersand. New models build the basis for separation of concerns and modelling of systems from services. We propose

1. to partition the set of relations in the static model,
2. to define the refuse and failure semantics in the behavioural model,
3. to construct high level events and states in the behaviour model.

3.1 Static model with partitioning the underlying model.

We propose to partition the set of relations \( R \) in the static model into two subsets:
- the subset of relations that can be changed by the modelled system $R_P$.
- the subset of relations that are derived from concepts of other systems $R_D$.

The modified static model of a business process is a tuple

$$SP = (C, R_P, R_D, BR).$$

### 3.2 Refuse and Failure semantics in the behaviour model

The partitioning of the static model changes the set of events used by the behavioral model $P = (S, E, T)$:

- The finite set of events $E$ includes events that change relations $R_P$:
  - $\text{INSERT}(a_r)$ insert an atom of a relation $r$, $r \in R_P$ or
  - $\text{DELETE}(a_r)$ delete an atom of a relation $r$, $r \in R_P$.
  
  Only events changing relations $r \in R_P$ become possible. $\text{INSERT}(a_r)$ and $\text{DELETE}(a_r)$ events for relations $r \in R_D$ are ignored because they do not belong to the business process.

  Instead of $\text{INSERT}$ events,
  - $\text{READ}(a_r)$ events are used for access to relations of set $R_D$.
  - The next possibility is to calculate values of functions using read values as arguments $\text{DERIVE}(\text{READ}(a_{r1}), \ldots, \text{READ}(a_{rn}))$.

- The set of transitions $T$ cannot not be completely defined by the set of the $\text{INSERT}$ and $\text{DELETE}$ events for relations $r \in R_P$.
  
  There are states where $\text{INSERT}(a_r)$ and $\text{DELETE}(a_r)$ events for relation $r \in R_P$ are refused at the simulation time because $r \in R_P$ is in conjunction with the violated relations $r \in R_D$.

  So, in this semantics the violation of some business rules combining relations changed by the specified system and by other systems plays a role of a guard for events.
- The set of states $S$ is the set of all populations.

For example, in our order managements system:

- Event $\text{INSERT}-\text{isAccepted}(1, \text{Company P})$ is refused if the order 1 requests $\text{Goods}$ that are not sold by the $\text{Provider}$. The acceptance is guarded by the business rule $br_1$.

- Event $\text{INSERT}-\text{isAccepted}(1, \text{Company P})$ is refused if the $\text{Client}$ of order 1 does not have a bank $\text{Account}$ or if this account is $\text{Overdrawn}$. The acceptance is guarded by the business rule $br_2$.

A restriction on event handling. If a system contains several separated concerns then only one event of all separated concerns is handled at each moment. During the time of the event handling all other events are refused. This restriction prevents deadlocks, so it is impossible for two events to access the same relation at the same moment.
Separation of concerns and failure semantics. The semantics of the behavioral model is changed: satisfaction of some business rules cannot be achieved because of relations changed by other processes. Instead of a sequence of accepted events of types INSERT and DELETE, the transition system will specify the failure sets defined by Brookes [1], where a failure is a pair of a trace and a refusal set.

For example, in our order management system a possible failure is a pair

\[(sequence, failure set) :\]

where

\[sequence = (\text{INSERT}-isIssued(1,\text{Company A})) \text{ and}\]

\[failure set = \{ \text{INSERT}-isAccepted(1,\text{Company P}), \]

\[\text{INSERT}-bills(\text{Company P}, \text{Company A}), \]

\[\text{INSERT}-pays(\text{Company A}, \text{Company P}), \]

\[\text{INSERT}-delivers(\text{Company P}, \text{Company A}) \}.\]

The trace of this failure consists from one event

\[(\text{INSERT}-isIssued(1,\text{Company A}))\]

and the refusal set contains all other events that change the relations of this system.

3.3 High level events and states

The goal of Ampersand is to model systems that are built from services. Services should interact with each other, users and environment. Interactive events are usually events of the higher level than events of INSERT and DELETE types.

High level events. In our order management system we may think of an event of type

\[\text{IssueOrder}(\text{Order}, \text{Client}, \text{OrderedGoods}, \text{Provider})\]

that may cause a set of lower level events:

\[\text{INSERT}-isIssued(1,\text{Company A}), \]

\[\text{INSERT}-contains(1,\text{Software}), \]

\[\text{READ}-hasAccount(\text{Company A}, 67589)), \]

\[\text{CALCULATE}-notOverdrawn(67589, \text{not Overdrawn}), \]

\[\text{READ}-isSold(\text{Software}, \text{Company P}), \]

\[\text{INSERT}-isAccepted(1,\text{Company P}).\]

After this sequence of low level events, business rules \(br_1\) and \(br_2\) are satisfied and event IssueOrder is accepted. Any of READ and CALCULATE may cause a refuse to accept event IssueOrder.

If we model interactions then INSERTs, DELETEs and READs become invisible. A high level event can be built from the Ampersand specification. The concepts that are directly related define the content of high level events.
For example, event Issue Order naturally combines the concepts Order, Client, Goods, Provider in correspondence with direct relations $r_1, r_5, r_7 \in R_P$ (Table 1).

**High Level states.** The new behaviour model of Ampersand recognizes only states with different sets of violated business rules. Transitions from one such state to another are defined by the high level events.

In our order management system the high level states are an Initial state, Accepted, Billed, Payed, Delivered.

A pair of the static and the behavioural model is defined as follows:

$$SP = (C, R_P, R_O, R_F, BR)$$

$$P = (S, E, T)$$

- $S$ is a finite set of states $S = \{s_i, i = 1..N_S\}$ such that each state $s_i$ is all the populations of all relations with its unique set of violated business rules.
- Any state can be initial.
- Final states $s_f \in S$ are the states where all business rules are satisfied and all relations are complete or the states where business rules $r \in R_O \cup R_F$ are violated.
- $E$ is a finite set of high level events (interactions) $E = \{e_j, j = 1, ..., N_E\}$ uniquely defined for every business system on the basis of related concepts.
- Any moment only one event can be handled;
- The set of possible transitions is defined by the set of accepted high level events.
  - If business rules guarding event $e$ involve only relations changed by the modelled system, then the set of INSERTs and DELETEs will implement this high level event and this event will cause a transition to another state.
  - If business rules guarding event $e$ involve relations changed by other systems and those relations are complete, then the set of INSERTs and DELETEs will implement this high level event and the event will cause a transition to another state.
  - If business rules being guards of event $e$ involve both relations changed by the modelled system $r \in R_P$ and relations changed by other systems $r \in R_O \cup R_F$, and relations $r = R_O \cup R_F$ are incomplete, then event $e$ is refused.

Such a semantics corresponds to the failure model of the CSP [1] and opens the possibilities for the CSP-parallel composition of separated concerns [2]. The rules of the CSP parallel composition are based on the notions of acceptance and refusal of an event, $e$:
- If all concern models that have $e$ in their alphabet accept $e$, then $e$ is accepted.
- If at least one of those concern models refuses $e$, then $e$ is refused.
The refuse semantics is also the necessary condition to model communication of distributed service-oriented systems using message based communication. If a service model is in the state where it can send message \( e \) and another service is in the state where it can receive message \( ?e \), then communication \( e \) can take place. Services then update their state as defined by their models [16]. If the service-receiver refuses to receive message \( ?e \), no behaviour composition takes place and then the sender knows about this refuse.

The proposed semantic changes and the composition rules in the simulator of Ampersand are the necessary modifications allowing support of modelling of service-oriented systems.

4 Conclusion

Business rules based modelling in Ampersand is a declarative form of model presentation suitable for requirements engineering. We have defined a state-transition semantics of the current simulator of the Ampersand to estimate what should be done for separation of concerns in Ampersand models and modelling of interactions.

We have found that the behavioral model of Ampersand is of a lower level, i.e events are closely coupled to relations of the underlying model. We have proposed the semantic changes in the static and behavioural model of Ampersand allowing separation of concerns and modeling of interactive services. The separation of concerns means partitioning of the underlying model and combining the low level events into high level events suitable for interaction modelling. We have defined the refuse and failure semantics for the behavioural model that is the necessary condition for composition techniques in models.

The Ampersand approach belongs to a growing group of declarative approaches adding "small amount of syntax to the logic for structural descriptions"[5]. Our behavioural model and its semantics may enrich this group of approaches for modelling of service-oriented applications. Moreover, the behavioural model can be used for recognition of business rules in models build in operational approaches.

References


