# Behavioural Model for a Business Rules Based Approach to Model Services

Ella Roubtsova, Stef Joosten, Lex Wedemeijer
Open University of the Netherlands
Postbox 2960, 6401DL Heerlen. The Netherlands
ella.roubtsova@ou.nl;stef.joosten@ou.nl; lex.wedemeijer@ou.nl

## ABSTRACT

Service-oriented systems are seen as an IT trend impacting businesses. Successful implementation and maintenance of such systems demands their modelling. However most modern modelling approaches do not have the necessary service abstractions and suitable composition techniques for integration of behaviour of services.

The contribution of this paper is the analysis of the behavioural semantics of a declarative, business rules based approach called Ampersand and the identification of the semantic extensions needed for modelling of services and their composition.

## 1. INTRODUCTION

Service-oriented systems are seen as an IT trend impacting businesses. Successful implementation and maintenance of such systems demands their modelling. The danger of the moment is that most modern modelling approaches do not have the necessary service abstractions and suitable composition techniques for integration of behaviour of services.

A service is characterized by two main dynamic features:

- Firstly, a service is a module that is able to appear and disappear at the run time without changing the behaviour of other system modules implemented locally with it. This feature is sometimes called volatility of services [25]. It is also related to service plurality and ability to replace each other.
- Secondly, a service communicates by message passing with other distributed services [18].

Modelling of these features means modelling of service instances and their composition during their execution and interaction.

It has been shown in [2] that many of operational approaches (Statecharts [6], Coloured Petri Nets [14; 15], UML Behavioural State Machines (BSM) [21], Protocol State Machines (PSM) [21] and activity diagrams [21]) need semantic extensions in order to support modelling of service-oriented systems.

The question is whether the declarative approaches are able to model the mentioned features of services in service-oriented systems. The declarative approaches: Z [12], the UML's Object Constraint Language (OCL)[20], Alloy [7; 8], declare behaviour as sets of observable or internal operations. Sets can be easily composed and decomposed. Comparing with the operational models that may contain artificial connections and have a flavor of bureaucracy, the declarative models do not have artificial connections and may be more suitable for modelling of service-oriented businesses with their tendency to adhocracy [11].

In order to give an answer to the question whether the semantics of declarative approaches allows for the adequate modelling of service-oriented systems we describe a behavioural semantics of a declarative approach called Ampersand [26]. As many other declarative approaches [7; 16; 8; 27] Ampersand uses relational logic. The specifics of this approach is the interpretation of expressions in relational logic as business rules. We analyse the behavioural semantics of Ampersand and identify the semantic extensions needed for modelling of services and their integration. The contribution of this paper is the analysis of the behavioural semantics of the declarative approach Ampersand and the identification of the semantic extensions needed for specification of service-oriented systems.

The paper is organized as follows.
Section 2 defines the static model of the Ampersand approach illustrated by an example.
Section 3 describes the behavioural model of Ampersand and estimates its ability to model services.
Section 4 proposes the semantic changes of the behavioral model in Ampersand towards modelling of interactive services and their composition.
Section 5 concludes the paper.

## 2. THE AMPERSAND APPROACH

Among different approaches to formalization of business rules the Ampersand approach uses the relational logic [19] and the formalization of business rules as expressions defined on relations of objects [26][1].

[1]The Business Rules Group defines "a business rule as a statement that defines or constrains some aspect of the business" [1; 4]. Business rules are classified as definitions of business terms, facts relating terms to each other, constraints and derivations [13; 22]


*BM-FA Workshop* 15 June 2010 Paris, France.

## 2.1 Terminology

The vocabulary of relational logic consists of "two classes of words - variables and constants.... Constants (variables) are further subdivided into object, relation and function constants (variables)"[19]. These constants and variables are used to form complex expressions.

There are three types of sentences in Relational Logic: relational, logical and quantified. A relational sentence forms a relation constant, for example, $r(a, b)$, where $a$ and $b$ are parameters. A logical sentence is an expression on relational sentences that uses negations, conjunctions, disjunctions, implications, reductions. subset operations and equivalences [19]. For example, $r \subseteq r_1$. Quantified sentences are formed from a logical sentence and a variable-quantifier that restricts the set where the logical sentence is true. There are two types of quantified sentences. A universally quantified sentence expresses that all objects of have certain property. An existentially quantified sentence expresses that some object has a certain properly.

## 2.2 Syntax of the Language in Ampersand

The language of the Ampersand approach uses object variables called *Concepts*, variables called *Relations* defined on *Concepts*, and expressions called *Business Rules* defined on *Relations*.

$\langle Relation\ Description\rangle$ ::=
$\langle Relation\rangle$ " :: " $\langle Concept_A\rangle$ " $*$ " $\langle Concept_B\rangle$
$[UNI|TOT|INJ|SUR]$
$= [(\text{"}Element_{Concept_A}\text{"},\text{"}Element_{Concept_B}\text{"}), ....,].$

The properties of relations can be presented as quantified expressions, however in Ampersand they are declared as $[UNI|TOT|INJ|SUR]$ or expressed as business rules:
$UNI$- univalent. Each element of domain of $\langle Concept_A\rangle$ corresponds to at most one element of domain of $\langle Concept_B\rangle$.
$TOT$ - total. Each element of domain of $\langle Concept_A\rangle$ corresponds to at least one element of domain of $\langle Concept_B\rangle$.
$INJ$ -injective. Each element of element of domain of $\langle Concept_B\rangle$ corresponds at most one element of domain of $\langle Concept_A\rangle$.
$SUR$ - surjective. Each element of domain of $\langle Concept_B\rangle$ corresponds to at least one element of domain of $\langle Concept_A\rangle$.

Business Rules in Ampersand are logical expressions of a predefined structure:

$\langle Business\ Rule\rangle$ ::=
$\langle Expression\rangle\langle SetOperator\rangle\langle Expression\rangle$
$\langle SetOperator\rangle$ :: " $=$ "|"! $-$ "|" $-$ |""
$\langle Expression\rangle$ :: $\langle Expression\rangle$""|
$\langle Expression\rangle$ :: " $-$ "$\langle Expression\rangle$|
$\langle Expression\rangle$" $\wedge$ "$\langle Expression\rangle$|
$\langle Expression\rangle$" $\vee$ "$\langle Expression\rangle$|
$\langle Expression\rangle$";"$\langle Expression\rangle$|
$\langle Relation\rangle$|
$\langle Concept\rangle * \langle Concept\rangle$|
$IdentityRelation[\langle Concept\rangle]$
where if
$\langle Relation_1\rangle ::= \neg\langle Relation_2\rangle \mid \langle Relation_2\rangle \wedge \langle Relation_3\rangle|$
$\langle Relation_2\rangle \vee \langle Relation_3\rangle|\ \langle Relation_4\rangle; \langle Relation_5\rangle$
then
$\langle Relation_1\rangle$ :: $\langle Concept_A\rangle$" $*$ "$\langle Concept_B\rangle$
$\langle Relation_2\rangle$ :: $\langle Concept_A\rangle$" $*$ "$\langle Concept_B\rangle$
$\langle Relation\rangle_3$ :: $\langle Concept_A\rangle$" $*$ "$\langle Concept_B\rangle$
$\langle Relation_4\rangle$ :: $\langle Concept_A\rangle$" $*$ "$\langle Concept_C\rangle$
$\langle Relation_5\rangle$ :: $\langle Concept_C\rangle$" $*$ "$\langle Concept_B\rangle$.

Symbol "|−" means $\subseteq$ Symbol "−| " means $\supseteq$.

The properties of a relation

$$\langle Relation\rangle :: \langle Concept_A\rangle\text{" }*\text{ "}\langle Concept_B\rangle$$

can be expressed as following business rules
$[UNI]$ ::=
$\langle Relation\rangle$"˜" ; $\langle Relation\rangle$"| − "$IdentityRelation[\langle Concept_B\rangle]$
$[TOT]$ ::=
$\langle IdentityRelation[\langle Concept_A]\rangle$"| − "$Relation\rangle; \langle Relation\rangle$"˜"
$[UNI, TOT]$ ::=
$\langle Relation\rangle$"˜" ; $\langle Relation\rangle$" = "$IdentityRelation[\langle Concept_B\rangle]$
$[UNJ]$ ::=
$\langle Relation\rangle; \langle Relation\rangle$"˜" "| − "$IdentityRelation[\langle Concept_A\rangle]$
$[SUR]$ ::=
$IdentityRelation[\langle Concept_B\rangle]$"| − "$\langle Relation\rangle$"˜" "; $\langle Relation\rangle$

## 2.3 Example of a static model in Ampersand

Let us model an order management system. The requirements for this system are the following:

An order containing an item and addressed to a provider is issued by a client. The order is accepted by the provider if the account of the client is not overdrawn and if the item are in the stock of the provider. A client has only one known bank account. The provider of an accepted order sends a bill to the client. When the client pais the bill, the order is delivered to the client.

The listing of a static model of an order management system in Ampersand is presented below[2] .

```
-1-CONTEXT Trading
-2-
-3-PATTERN OrderManagement
-4-
-5-isIssued ::Order*Client [UNI,TOT,SUR]
=[("1","Company A")].
-6-
-7-
-8-isAccepted::Order*Provider [UNI,TOT].
-9-
-10-hasAccount ::Client*Account [UNI,TOT,INJ]
=[("Company A","12345")].
-11-notOverdrawn ::Account*NotOverdrawn[UNI,INJ,SUR].
-12-isAddressed::  Order*Provider [UNI, TOT].
-13-isChecked::NotOverdrawn*Provider [UNI, TOT,INJ].
-14-contains ::Order*Item [UNI,TOT]=[("1","Software X")].
-15-hasInStock ::  Item*Provider [TOT]
=[("Software X","Company P")].
-16-isAccepted -| isAddressed
-17-isAccepted-| contains; hasInStock
-18-isAccepted-|
isIssued;hasAccount;notOverdrawn;isChecked
-19-
-20-
-21-makeBill::  Provider*Bill [INJ,SUR].
```

[2]Dashed numbers represent lines of the listing.

```
-22-billFor::  Bill*Order [UNI,TOT, INJ].
-23-billTo::Bill*Client [UNI].
-24-makeBill;billTo -| isAccepted˜;isIssued
-25-
-26-isPaid::  Bill*Client [UNI,TOT].
-27-isPaid-| billFor; isIssued ⋀ billTo
-28-
-29-
-30-isDelivered::  Order*Client [UNI].
-31-isDelivered -| isAccepted;makeBill;isPaid
-32-
-33-ENDPATTERN
-34-ENDCONTEXT
```

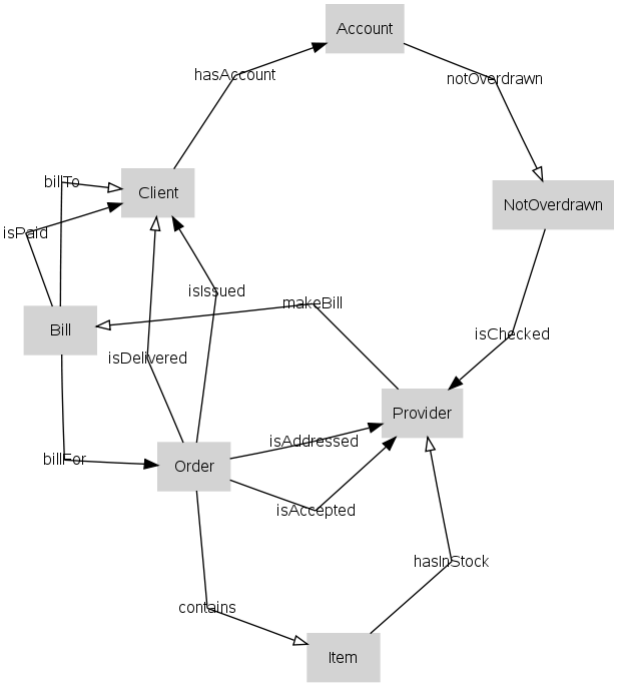


**Figure 1: Static Model of a order-management system**

- The names of concepts start with a capital letter: *Client, Order*, *Provider* etc..
- The identifiers of relations start with a small letter: *isIssued*, *hasAccount*,*isAccepted* etc.
- The business rules are expressed as properties of relations $INI, TOT.etc$ or as expressions on relations.

Figure 1 shows the model graphically.

## 2.4 Semantics of Ampersand Models

The semantics of the Relational Logic is based on conceptualization of the world. The choice of the right conceptualization depend on the goal of modelling.

A semantics of a language based on Relational Logic is an interpretation of constants of the language as elements of the conceptualization. Each object constant is mapped into an element in the universe of discourse. Each relation constant is mapped into a relation population.

### *Business objects and Concepts.*

The universe of discourse of an Ampersand model includes the business objects that exist in the modelled business. The business objects are classified using *concepts*. A concept presents a set of objects of the same type. Concepts can be infinite or finite sets. The interpretations or populations of concepts in models are finite sets of elements:

$$Client = \{"Company\ Bits", "Person\ M.Bos"\}.$$

$$Order = \{"1", "2", "3\}.$$

### *Relational basis set.*

"An interrelationship among the objects in the universe of discourse is called a relation... The set of relations emphasized in a conceptualization is called the relational basis set"[19].

The interpretation of a binary relation

$$\langle Relation\rangle :: \langle Concept_A\rangle * \langle Concept_B\rangle$$

in Ampersand is a finite set of pairs defined on the populations of two concepts $p_{C_A} \times p_{C_B}$. The finite set of pairs can be empty.

For example, relation "A Client issues an Order"

$$issues :: Client * Order\ [TOT]$$

may be interpreted as population
$\{("Company\ Bits", "2"),$
$("Company\ Bits", "3"),$
$("Person\ M.Bos", "1")\}.$

Any relation can be expressed as an active or a passive sentence of requirements, the interpretations of such presentations are equal (the syntax rule is define in section 2.2): $isAccepted = isAccepted˜ = accepts$.

### *Properties of Relations and Business Rules.*

The properties of relations are the simplest business rules. The properties $UNI, TOT, INJ, SUR$ of relations are used to enforce multiplicity information about data structures.

If a relation $\langle Relation\rangle :: \langle Concept_A\rangle" * "\langle Concept_B\rangle$ is total, then each element of domain of $\langle Concept_A\rangle$ corresponds to at least one element of domain of $\langle Concept_B\rangle$. If the property is not satisfied, then it is violated.

For example, the model presented in subsection 2.3 contains the instance `("1","Company A")` of relation `isIssued` (line -5-) and the instance `("Company A","12345")` of relation `hasAccount` (line -10-) and this results in violation of totality of relations `isAddressed` (line -12-), `contains` (line-14-) and `isAccepted` (line -8-) that have no pairs with client `"Company A"`.

### *Expressions and Business Rules.*

An interpretation of a business rule compares the populations of composed relations presented by the the left hand side and the right hand side expressions.

A business rule may express that the left hand side composed relation is equal to the right hand side composed relation. In this case a business rule is maintained if the population of the left hand side composed relation contains the same pairs as the population of the right hand side composed relations. Otherwise, this business rule is violated.

A business rule may express that the left hand side composed relation is a subset of the right hand side composed relation (or the other way around). In this case this business rule is maintained if the population of the the left hand side composed relation is a subset of the population of the right hand side composed relation.

For example, expression

$$isDelivered \ -| \ isAccepted; makeBill; isPaid$$

means that each pair of the population of the composed relations

$$isAccepted; makeBill; isPaid$$

is a pair of the population of relation *isDelivered*. If a business rule is not maintained, then it is violated.

## 2.5 Ampersand and UML

The static Ampersand model can be presented as a UML domain model with the OCL expressions presenting business rules. However, as it can be seen in [16] both the UML and the Object Constraint Language (OCL) contain more semantic elements than it is necessary for a business rules based specification. So, Ampersand can be seen as a language with the minimal semantics for business rules based specification.

# 3. BEHAVIOURAL MODEL OF AMPERSAND AND SIMULATION

In a behavioral model of a living business system new elements of relations are created and other elements are deleted, so that the populations of relations are changed. New relations often contain concepts of other relations that do not have the corresponding element in their populations. So, with inserting or deleting of relations the properties of relations and business rules become violated.

The Ampersand approach is supported with a simulator for insert- and delete- actions that can potentially violate or restore the completeness of relations and business rules. A model in the simulator implements the following pair of the static model ($SM$) and behavioural model ($BM$):

$$SM = (C, R, BR),$$

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

- $C$ is a finite set of concepts, $C_1, ...., C_N$.
- $R$ is finite set of relations, $R_1, ...., R_M$.
- $BR$ is a finite set of business rules, $BR_1, ..., BR_K$.
- $S$ is a finite set of instances of all concepts, or a final set of populations of concepts $S = \{p_{C_1}, ..., p_{C_N}\}$. A state is an element of this set where each population $p_{C_J}$ is a final set of instances $c_J$ of concepts: $c_J \in p_{C_J}$.

  The values of business rules are evaluated on instances. For each state $s \neq s_f$ there is subsets (that may be empty) of violated business rules. Any state can be initial.
- $E$ is a finite set of events $e$.
  There are only two types of events in the current version of Ampersand defined for each relation in the static model:

  (1) Insert an element of a relation $(a_J, b_K)$,

  $$\{pre : s_i = p_{C_1}, ..., p_{C_J}, ....p_{C_K}, ....p_{C_N},$$
  $$a_J \text{ is of type } C_J, \ b_K \text{ is of type } C_K\}$$

  $$INSERT\langle Relations\rangle((a_J, b_K))$$
  $$\{post : s_j = p_{C_1}, ..., (p_{C_J} \cup a_J), ..., (p_{C_K} \cup b_K), ...., p_{C_N}\}$$

  (2) Delete an element of a relation $(c_J, c_K)$,

  $$\{pre : s_i = p_{C_1}, ..., p_{C_J}, ....p_{C_K}, ....p_{C_N},$$
  $$c_J \in p_{C_J}, c_K \in p_{C_K}\}$$

  $$DELETE\langle Relations\rangle((c_J, c_K))$$
  $$\{post : s_j = p_{C_1}, ..., (p_{C_J} \cap c_J), ..., (p_{C_K} \cap c_K), ....p_{C_N}\}$$

  Both event types change the populations of concepts and the state of the model. All events of these types for relations $r \in R$ are possible, however the violated business rules give the indication that the population of elementary relations business rules can to changed to make progress in the business process.
- $T$ is a finite set of possible transitions.

  $$T = \{t_k, k = 1, .., N_T|$$

  $$t_k = (s_i, s_j, e), \ s_i, s_j \in S, \ e \in E\}.$$

  The events possible in a particular state $s_i$ are defined by the preconditions of events. The result state $s_j$ of a transition $(s_i, s_j, e)$ is defined by the postcondition of the event.
  If the precondition of an event is not satisfied in a state, then the event is ignored by the simulator and the state is not changed.

The behaviour of this state-transition system is a set of traces of accepted events.

The behavioural model and the simulator are used to validate the model against use cases. In the simulator, every accepted event is accompanied with a message describing the violated business rules. Ideally each trace has to be finished in a new state where all business rules are complete. In practice, the *INSERTs* and *DELETEs* can be infinitely repeated. It is, however, assumed that the business logic and business rules always give the user, who simulates traces, a direction to the final state.

The following trace is a use case for our order-management system tested in the Ampersand simulator:

*TRACE-EXAMPLE:*
*INSERT-isIssued(("1","Company A")),*
*Properties of isAddressed, contains, isAccepted and the business rule (line 18)*
$isAccepted -| isIssued; hasAccount; notOverdrawn; isChecked$ *are violated*
*INSERT-isAddressed)("1","Company P")),*
*Properties of contains and the business rule for isAccepted are violated*
*INSERT-contains(("1","Software X") ),*

*The business rule for isAccepted is violated*
*INSERT-hasAccount(("Company A","67589")),*
*The business rule for isAccepted is violated*
*INSERT-notOverdrawn(("67589","67589")),*
*The business rule for IsAccepted is violated*
*INSERT-isInStock(("Software X","Company P")),*
*The business rule for IsAccepted is violated*
*INSERT-isChecked(("67589", "Company P")),*
*The business rule forIsAccepted is violated*
*INSERT-isAccepted(("1","Company P")),*
*Business rule makeBill;billTo = isAccepted ;isIssued is violated*
*INSERT-makeBill(("Company P","B-500=2010")),*
*Properties of billFor and business rule makeBill;*
*billTo = isAccepted ;isIssued are violated*
*INSERT-billFor(("B-500-2010","1")),*
*Business rule makeBill;billTo = isAccepted ;isIssued is violated.*
*INSERT-billTo(("B-500-2010","Company A")),*
*Business rule billTo |- isPaid is violated.*
*INSERT-isPaid(("B-500-2010","Company A")),*
*Business rule isAccepted;makeBill;isPaid |- isDelivered is violated.*
*INSERT-isDelivered("1", "Company A"))*

## 3.1 Analysis of the current behavioural model in Ampersand

The simulator of the Ampersand is used to estimate legibility of any step in a trace with respect to the business rules. However, in the current model nothing prevents the simulation from infinite traces. Moreover, the resolution of violations of business rules can be always achieved by emptying the data store (by the *DELETE* operations) which would break the semantic link between model and its real-world interpretation. A potentially promising way to fix this fundamental flaw of invariant business rules is by introducing the semantics of event refusal into the model.

The current behavioural model of Ampersand demands specification of all relations and business rules in one model. It does not have semantics for separation of functionality of services. This means that the Ampersand can be used only for modelling of a monolith business system that possesses the complete underling model. The models of large systems become unobservable.

The current Ampersand model does not recognizes business process instances. A trace in the current behavioural model not necessarily corresponds to the process. For example, a trace in the current behavioural model not necessarily corresponds to the process of management of one order, it may correspond to the management of a set of orders.

The events in the current model always come from the user of the simulator. The message based communication of services cannot be modeled.

In order to support modelling of service-oriented systems the semantics of the static and the behavioural models of Ampersand has to be changed.

Services are relatively independent parts of business functionality that have to be modeled and tested separately and then composed during the model execution in such a way that the behaviour of services is preserved in the behaviour of the complete system. The CSP parallel composition [5; 9] of locally implemented services possess such a property, also known as local reasoning or observational consistency. As the distributed services communicate by message passing, the CCS [23] composition technique can be used for modelling of behaviour of distributed services.

# 4. SEMANTIC EXTENSIONS TO MODEL SERVICES

We propose to apply the Protocol Modelling semantics [3] for the Ampersand approach. As the Protocol Modelling semantics is able to model services [10] and uses the CSP-parallel composition and the CCS-composition for composition of behaviour of services, this ensures us that our new behavioural model of the Ampersand will be suitable for modelling of service-oriented systems. Our model extensions are aimed to separate the state space of the model into state spaces of services, enable modelling service instances and define the protocol rules for of communication of service instances.

## 4.1 Static Model of a Service Instance

We propose to define for each service the following static model

$$SM_{Service} = (C, CD, R, RD, BR)\ where$$

- a set $C = \{C_1, ..., C_N\}$ of own concepts;
$c_i$ is an element of $C_n$, $n = 1, ..., N$
- a set of concepts $C_D = \{C_1, ..., C_K\}$ of other services;
$c_k$ is an element of $C_k$, $k = 1, ..., K$.
- a set of relations $R = \{R_1, ..., R_M\}$ that can be changed by the service;
$r_m$ is an element of $r_m$ $m = 1, ..., M$.
- a set of relations $RD = \{R_1, ..., R_H\}$ that are derived from concepts of other services;
$r_h$ is an element of $r_h$ $m = 1, ..., H$.
- a set of own business rules $BR = \{BR_1, ..., BR_L\}$ ;
$br_l$ is an element of $br_h$ $m = 1, ..., h$.

The concepts and relations form the state space of a service. If the state spaces of services are intersected then these services are not distributed and can read but cannot alter the state space of each other.

The elements of business rules $br_1, ..., br_L$ are business rules evaluated on the elements of relations and concepts $c_1, ...., c_N, cd_1, ...., cd_K, r_1, ..., r_M, rd_1, ..., rd_H$. An element of a business rule is maintained if the the pair corresponding to the element on the left hand side relation is equal to the pair of the element of the right hand side relation.

### *A static model of a service instance.*

includes one element of each concept and one element of each relation. A statics model of a service instance is a tuple of elements of all service concepts and relations and elements of instances of business rules evaluated on the elements of concepts and relations.

$$SM_{Service\ Instance}$$

$$= (c_1, ...., c_N, cd_1, ...., cd_K, r_1, ..., r_M, rd_1, ..., rd_H, br_1, ..., br_L).$$

## 4.2 Events

In the new model an event is defined in such a way that it is able to change a subset of own relations $R$ of a service and to change the value of a subset of business rules of the service.

An event is modeled with its metadata or its parameter schema[3]. The parameters of events are concepts of the relations, populations of which have to be changed or read during the handling of the event.

The construction a parameter schema of an event is a part of requirements engineering. The events are of the high level and their names reflect their role in the business process.

For, example, in our order management system we recognize events of type

$$\bullet Issue\ Order(Order, Client, Item, Provider)$$

This event changes the populations of relations
$isIssued :: Order * Client$,
$isAddressed :: Order * Provider$ and
$contains :: Order * Item$
All concepts of these relations become types of parameters of this event.

$$\bullet Accept\ Order(Order, Client, Item, Provider)$$

This event changes the population of relation $isAccepted :: Order * Provider$
and should evaluate the business rules
$isAccepted = isAddressed$
$isAccepted = contains; hasInStock$
$isAccepted = isIssued; hasAccount; notOverdrawn; isChecked$
that can be done only when the populations relations of other services (an account management and a stock management ) can be accessed:
$hasAccount :: Client * Account$
$notOverdrawn :: Account * NotOverdrawn$
$isChecked :: NotOverdrawn * Provider$
$hasInStock :: Item * Provider$
The parameters of the event contain concepts needed to find element of all those relations.

$$\bullet Send\ Bill(Bill, Provider, Client, Order)$$

The parameters of the event contain concepts needed to find element of all relations of the business rule
$makeBill; billTo = isAccepted\ ; isIssued$

$$\bullet Pay\ Bill(Bill, Provider, Client, Order)$$

The parameters of this event needed to evaluate business rule: $isPaid - |billFor; isAccepted; makeBill; billTo$

$$\bullet Deliver\ Order(Order, Provider, Bill, Client)$$

The parameters of the event contain concepts for evaluation of $isDelivered - |isAccepted; makeBill; isPaid$

## 4.3 Service Instances. Event Handling

A behavioural model $BM_{Service\ Instance}$ of a service is defined for a service instance.

$$SM_{Service\ Instance} =$$

$$(c_1, ...., c_N, cd_1, ...., cd_K, r_1, ..., r_M, rd_1, ..., rd_H, br_1, ..., br_L).$$

$$BM_{Service\ Instance} = (S, S_0, S_f, E, T)$$

- A service has a set or an alphabet of high level events $E$ that it recognizes. Events that do not belong to the alphabet are ignored by the service.

[3]Such approach to modelling events with attribute schemas is used by M.Jackson [17], S.Cook and J.Daniels [24] A.McNeile and N. Simons [3].

- A behaviour of a services instance is a process initiated with an external event. There is always an initiating event that creates an instance of a service. This event inserts an initial combination of instances of some relations owned by the service.
- A state of a service that does not contain this initial combination of instances of concepts and relations is the initial state $S_0$ of the service instance. When a servies is instantiated, the static model (the structure of instances of concepts and relations) of a new instance is created.

  After creating a service instance, the service goes to another state, the initiating event is not allowed in this new state. If an initiating event happens again with the same data, it is refused.
- In any state $s \in S$ of a service instance some concept instances are filled in with data and same of them do not. So, states have different sets of violated instances of business rules. The events of the alphabet that add data to instances of concepts and restore the violated business rules are possible in the state.

  If an event, possible in the state, happens, then it is accepted by the service instance. The alphabet events that are not possible in the state are refused in the state.
- The final state $S_f$ of a service instance is a state where all instances of concepts and relations of the static model of the service instance are filled in with the data and all the business rule instances are not violated.
- The transitions $T$ are defined as triples each of which contains
  - two states with different data of instances of concepts and different sets of violated instances of business rules and
  - one high level event that changes the data of instances of concepts.

## 4.4 Service Composition

The static model of a service-oriented system is a union of the static models of service instances.

The behavioural model of a service-oriented system is a union of the behavioural models of service instances and the behavioural models of necessary composition techniques.

### *Composition of local services.*

The alphabets of events of non-distributed services with intercepted sets of concepts are intersected. If an event belongs to the alphabets of several services, then these services use relations of each other to evaluate their business rules. These services should be synchronized. For services with intersected alphabets of events the synchronization can be implemented as a CSP parallel composition algorithm:

- If all servies are in the state where the shared event can be accepted, then the event is accepted by the composition of services.
- If at least one of the services is in the state where the shared event is refused, the event is refused by the composition.

The semantics of event refusal makes it possible to use the CSP-parallel composition [5] of local services that have access to the state of each other.

In the current behavioural model in Ampersand any element of any relation can be inserted or deleted in any order, then the new behavioural model restricts such a freedom. A static model of a service instance defines the set of relations that have to be filled in with data by a service instance. Thanks to the complex events, adding and deleting of elements of some relations is synchronized leading a service instance from one state to another. Thanks to the sematic of event refusal, events are ordered (partially ordered) so that a service-process can be constructed and simulated.

### *Composition of distributed services.*

The alphabets of events of distributed services are not intersected. However, the distributed services may have complementary events with the same metadata except the element type that has two values: send-event (!$e$ ) or receive-event (?$e$).

The semantics of event refusal is also the necessary condition to model communication and the CCS composition of distributed services. 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 [23]. If the service-receiver refuses to receive massage ?$e$, no behaviour composition takes place and then the sender knows about this refuse.

The CSP composition technique guarantees the propertied of local reasoning on services about the behaviour of their composition. The CCS composition is supported by the model checking techniques for analysis of the behaviour of the result of service composition [9]. The combination of these composition techniques enables modelling and analysis of service-oriented systems.

The composition techniques need to be implemented in the simulator. The implementation of the new semantics and the composition techniques in Ampersand is our future work.

# 5. CONCLUSION

The Ampersand approach belongs to a growing group of declarative approaches adding "small amount of syntax to the logic for structural descriptions"[8]. We have described the static and behavioural model of Ampersand and analyzed its ability to model services. We have proposed the semantic changes necessary to enable modelling of services and their composition. The behavioural model of a service is defined at the level of service instances. Its semantics is enriched with the business process level events and the refusal semantics of event handling. The proposed semantic extensions enable behaviour composition techniques for composition of service models.

The behavioural model of services proposed in this paper may be used by other declarative approaches [16] generating behavioural models from static models.


### *Acknowledgement..*

We thank the anonymous reviewers of the Second International workshop on Behavioural Modelling-Foundation and Applications (BM-FA 2010) for very useful comments and questions.


# References


[1] A. Walker, M. McCord, J.F. Sowa, W.G. Wilson. *Knowledge Systems and Prolog: Developing Expert, Database and Natural Language Systems.* Addison Wesley, 1990.

[2] A.T. McNeile and E.E Roubtsova. Composition Semantics for Executable and Evolvable Behavioural Modeling in MDA. *BM-MDA'09: Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture*, pages 1–8, 2009.

[3] A.T. McNeile, N. Simons. Protocol Modelling. A Modelling Approach that Supports Reusable Behavioural Abstractions. *Software and System Modeling* , 5(1):91–107, 2006.

[4] Business Rules Group. Defining Business Rules.What Are They Really? *http://www.businessrulesgroup.org/ first_paper/ br01c0.htm*, 1993.

[5] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[6] D. Harel, M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach.* McGraw-Hill, 1998.

[7] D. Jackson. Alloy: A Lightweight Object Modelling Notation"ACM Transactions on Software Engineering and Methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM) archive Volume 11 , Issue 2, ISSN:1049-331X*, pages 256 – 290, 2002.

[8] D. Jackson. *Software Abstractions: Logic, Language, and Analysis.* MIT Press, Cambridge, MA, 2006.

[9] E.E. Roubtsova, A.T. McNeile. Abstractions, Composition and Reasoning. *Proceedings of the 13th IntŠl Workshop on Aspect-Oriented Modeling, Charlottesville. Virginia, USA, ACM DL,ISBN:978-1-60558-451-5; http://doi.acm.org/10.1145/1509297.1509303*, pages 19–24, 2009.

[10] E.E. Roubtsova, L. Wedemeijer, K. Lemmen, A.T. McNeile. Modular Behaviour Modelling of Service Providing Business Processes. *Internation Conference on Enterpise Information Systems, ICEIS 2009, May 2009. Milan. Italy* , pages 338–341, 2009.

[11] H. Mintzberg and A. McHugh. Strategy Formation in an Adhocracy. *Administrative Science Quarterly, Vol. 30, No. 2 (Jun., 1985), http://www.jstor.org/stable/2393104*, pages 16–197, 1985.

[12] J. Woodcock and J. Davies. *Using Z* . MPrentice-Hall (ISBN number 0-13-948472-8), 1997.

[13] J.L.G. Dietz. On the Nature of Business Rules. *LNBIP 10*, pages 1–15, 2008.

[14] K. Jensen. *Coloured Petri Nets.* Springer, 1997.

[15] L. Kristensen, J. Jørgensen, K. Jensen. Application of Coloured Petri Nets in System Development. *J.Desel, W.Reisig and G.Rosenberg (Eds) ACPN 2003, LNCS 3098*, pages 626–685, 2003.

[16] M. Albert, O.Cabot, C. Gómez, V.Pelechano. Automatic generation of basic behavior schemas from UML class diagrams. *Journal Software and Systems Modeling, Springer*, 9(1):47–67, 2010.

[17] M. Jackson. *Problem Frames.* Addison-Wesley, 2001.

[18] M.P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. page 3, 2003.

[19] M.R. Genesereth. Computational Logic. *Stanford University . USA. http://logic.stanford.edu/classes/cs157/2009/ cs157.html*, 2000.

[20] OMG. Model Driven Architecture: Object Constraint Language, OMG Document Number: formal/2010-02-01, http://www.omg.org/spec/OCL/2.2/PDF/.

[21] OMG. Unified Modeling Language, Superstructure, v2.2. *OMG Document formal/09-02-02 Minor revision to UML, v2.1.2. Supersedes formal 2007-11-02*, 2009.

[22] P. Kardasis, P Loucopoulos. Expressing and organizing business rules. *Information and Software technology*, 46, 2004.

[23] R. Milner. *Communicationg and Mobile Systems - the Pi-Calculus.* Cambridge University Press, 1999.

[24] S. Cook, J. Daniels. *Designing Object Systems -Object-Oriented Modelling with Syntropy.* Prentice Hall, 1994.

[25] S. De Labey, M. van Dooren, E. Steegmans. ServiceJ: A Java Extension for Programming Web Services Interactions . *IEEE International Conference on Web Services (ICWS 2007)*, pages 505–512, 2007.

[26] S. Joosten. Deriving Functional Specification from Business Requirements with Ampersand. *http://icommas.ou.nl/wikiowi/images/ e/e0/ampersand_draft_2007nov.pdf*, 2007.

[27] Th. Giannakopoulos, D.J. Dougherty, K. Fisler,and S. Krishnamurthi. Towards an Operational Semantics for Alloy. *Book FM 2009: Formal Methods DOI 10.1007/978-3-642-05089-3, LNCS 5850, Springer*, pages 483–498, 2009.