

# Using Non-Determinism for the Separate Specification of Structure and Behaviour

*Pieter Bekaert*

*Eric Steegmans*

*Report CW 337, April 2002*



Katholieke Universiteit Leuven  
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Using Non-Determinism for the Separate Specification of Structure and Behaviour

*Pieter Bekaert*

*Eric Steegmans*

*Report CW 337, April 2002*

Department of Computer Science, K.U.Leuven

## **Abstract**

During the analysis phase of the development of a software system, conceptual models are created to establish a description of the problem domain. In this paper we introduce concepts that allow us to introduce non-determinism in the specification of behaviour in conceptual models and we provide formal and declarative semantics for these constructs. Non-determinism enables us to specify in a very elegant way the requirements that have to be fulfilled, without specifying deterministically how the effect (or solution) will be realised. We integrate non-determinism with the concept of constraintment and show how this can lead to a better separation of structure from behaviour.

**Keywords** : Object-Oriented Analysis, Non-Determinism, Formal Methods, Specification of Behaviour, Formal and Declarative Semantics, Separation of Concerns.

# Using Non-Determinism for the Separate Specification of Structure and Behaviour

Pieter Bekaert\*

Eric Steegmans

Katholieke Universiteit Leuven  
Department of Computer Science

Celestijnenlaan 200A

3001 Leuven, Belgium

{pieter.bekaert, eric.steegmans}@cs.kuleuven.ac.be

<http://www.cs.kuleuven.ac.be/>

April 18, 2002

## Abstract

During the analysis phase of the development of a software system, conceptual models are created to establish a description of the problem domain. In this paper we introduce concepts that allow us to introduce non-determinism in the specification of behaviour in conceptual models and we provide formal and declarative semantics for these constructs. Non-determinism enables us to specify in a very elegant way the requirements that have to be fulfilled, without specifying deterministically how the effect (or solution) will be realised. We integrate non-determinism with the concept of constraintment and show how this can lead to a better separation of structure from behaviour.

**Keywords:** Object-Oriented Analysis, Non-Determinism, Formal Methods, Specification of Behaviour, Formal and Declarative Semantics, Separation of Concerns.

## 1 Introduction

In each object-oriented software development process attention has to be devoted to the description of the problem context of the system. By performing *object-oriented analysis*, a *conceptual model* is developed of this problem domain. We believe that the method used to build a conceptual

---

\*Research Assistant of the Fund for Scientific Research - Flanders (Belgium) (F.W.O. - Vlaanderen)

model should be based on a *declarative formalism*. This urges the modeller to focus on detecting and specifying properties and constraints in the problem domain, and on describing the effects of events. Opposed to using operational approaches, a declarative description does not focus on *how* a problem will be solved, how effects will be realised and how constraints will be verified. We attach a lot of importance to the absence of every such decision in a conceptual model. These decisions have to be postponed to the design phase, during which design decisions guide the transformation from the conceptual model of the problem domain to a design model of the system. The conceptual model then tends to be stable and maintainable, even when the scope, the requirements and the chosen solution evolve. Besides a declarative nature, the language must also have formal semantics, to obtain models with a clear and unambiguous meaning.

During consecutive stages of the development cycle, these declarative specifications will be transformed, introducing concrete architectures and algorithms to implement the intended effect and to realise non-functional requirements. Even if in certain processes analysis and design are not present as very distinct consecutive phases, building a conceptual model that explicitly states all relevant knowledge about the problem domain in a way that is independent from the technical solution, provides a substantial improvement of the quality of the software development process.

By creating and using an object-oriented analysis method with a well-selected set of declarative concepts and formal semantics and in which design issues and non-functional requirements are postponed, we are on our way towards an ideal situation, in which for each perception of the problem domain, there is only one conceptual model, independent from the eventual software solution. From this conceptual model, multiple realisations can result, depending on design and implementation decisions. If the transformation from analysis to design models [6] can be done in a traceable and tool-supported way, the benefits certainly compensate the additional cost.

In our method for building object-oriented analysis models, we introduce the notion of *non-determinism*. An event or more general, a functionality in a conceptual model is said to be non-deterministically specified, if as a consequence of an occurrence of this event, the specification allows more than one way in which the population of a model can evolve.

Non-determinism opens room for variation in the behaviour, which can be filled in during consecutive design and implementation activity. Moreover, within the conceptual model, it is possible to strengthen the non-determinism in generalisation/specialisation hierarchies, thus having specifications in generalisation classes that allow a lot of non-determinism while the redefinitions in specialisation classes restrict this, ultimately introducing a possibly deterministic redefinition.

The behaviour introduced by non-deterministic specifications of functionalities is restricted by the structural constraints in the model. We will

show that non-determinism enables a separation of the specification of behaviour and the specification of constraints. This separation of concerns is a very important tool to manage the complexity in conceptual models.

On the other hand, writing a deterministic specification for a more or less enhanced functionality using a declarative formalism tends to become complex and often already suggests a certain realisation. Even if this suggestion doesn't impose a final choice - and in some cases is not realistic as an implementation - this implementation-minded reading only adds to the complexity of the model.

In the following section, we provide a mental model defining semantics for conceptual models. In Sect. 3, we introduce the notion of non-determinism in conceptual models and provide a formal semantics for the non-deterministic constructs in the context of the mental model for conceptual models. In Sect. 4, non-determinism is integrated into the framework of constraintment, leading to the separation of structure and behaviour. We finally conclude and set out the directions for further research.

While this report is elaborated using the EROOS method (Entity Relationship Object Oriented Specifications [9, 8]), the ideas can be equally valuable in another method and are described in such a way that no familiarity with the EROOS method is required for the understanding of the paper. This work is an extension and completion of [1].

## 2 Semantics for Conceptual Models

In this section we describe a simplified mental model, based on logic and set theory and inspired by the Z specification language [7]. The mental model serves as a basis for defining semantics for conceptual models, in particular EROOS models. These semantics are given by describing the possible populations of the conceptual model and how these evolve in function of the events that occur. The mental model will be used to define formal semantics for the non-deterministic concepts that we introduce in Sect. 3. For the purpose of this paper, a complete, detailed description of this mental model is not needed.

### 2.1 The Population

The mental model corresponding with a conceptual model introduces a number of object sets (one for each class in the conceptual model) and value sets (one for each domain in the conceptual model), together with a number of relations between the elements of these sets (according to the attributes and relations defined on top of the classes). At any point in time, each of

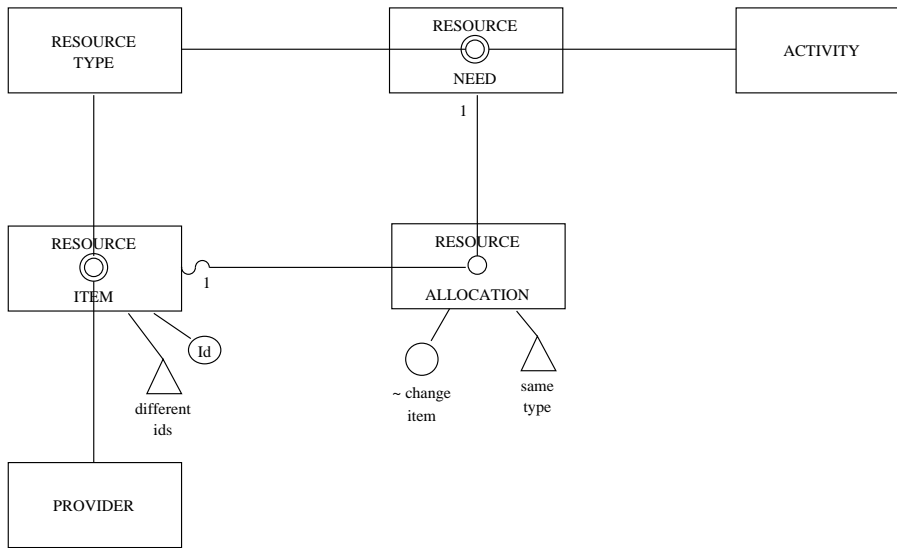


Figure 1: A simple conceptual model for a resource allocation system

the object sets consists of all currently living<sup>1</sup> objects of the corresponding class. The object sets, together with the relations form the *population* of the model. The population of the model evolves over time<sup>2</sup>, as *events* occur.

Conceptual modelling involves the creation of a model for which the semantic mapping on a population and its evolution can in turn be mapped to the world as it is perceived/conceived and deemed relevant by the conceptual modeller.

In this article we will use a simple example conceptual model for resource allocation. The conceptual model is shown in Fig. 1. In short, the problem consists in allocating resource items to activities, in function of need and availability. The population of this conceptual model, at any instance of time, consists of 6 object sets: **PROVIDER**<sup>o</sup>, **RESOURCE ITEM**<sup>o</sup>, **RESOURCE TYPE**<sup>o</sup>, **ACTIVITY**<sup>o</sup>, **RESOURCE NEED**<sup>o</sup> and **RESOURCE ALLOCATION**<sup>o</sup>.

Besides these object sets, the population also consists of the relationships between the objects of related classes. The class **RESOURCE ALLOCATION**, for example, is said to be *refined with a binary relation* involving the classes **RESOURCE ITEM** and **RESOURCE NEED**. This means that every resource allocation object is refined with a relationship involving a (single) resource item and a (single) resource need, introducing for resource allocation objects an

<sup>1</sup>In fact, EROOS also introduces object sets for the dead objects, allowing a better modelling of object life cycles.

<sup>2</sup>Time and evolution can be explicitly incorporated into the mental model by introducing a function that displays the TIME domain on a population, and a function that displays the TIME domain on a set of concurrent event occurrences.

*existential dependency* of both a resource item object and of a resource need object. The refinement of a class with a binary relation<sup>3</sup> introduces in the mental model, for each participating class and at each instance of time, a total function from the object set of the refined class to the object set of the participating class. In the example, the refinement of RESOURCE ALLOCATION introduces the following functions in the mental model, used in post-fix notation:

$\downarrow$ RESOURCE ITEM : RESOURCE ALLOCATION<sup>o</sup> → RESOURCE ITEM<sup>o</sup>

$\downarrow$ RESOURCE NEED : RESOURCE ALLOCATION<sup>o</sup> → RESOURCE NEED<sup>o</sup>

The relations corresponding with the inverse of these functions are indicated as  $\uparrow$ RESOURCE ALLOCATION. When applied to a resource item respectively resource need, these relations will yield the set of resource allocations in which this item respectively need participates.

A class can also be *decorated with attributes*. In the example, the RESOURCE ITEM class is decorated with an *Id*. This is an attribute of the IDENTIFICATION domain. In the mental model, a value set IDENTIFICATION is present, with all possible identifications as elements. On this domain, some relations are assumed to be defined, such as a total order relation  $<$ . The decoration introduces a total function  $\rightarrow$ Id in the mental model, from the object set RESOURCE ITEM<sup>o</sup> to the domain IDENTIFICATION.

The object set corresponding with a class, together with these functions (if the class is refined and/or decorated), is called the population of that class.

## 2.2 Structural and Behavioural Propositions

Both the population at any point in time, as well as each possible evolution of the population over time, are subject to certain rules. These rules originate both from the semantics of the constructs used in the conceptual model and from the explicit constraint and event specifications present in the conceptual model. Note that these rules together do not completely determine the population and its evolution: the population as a collection of sets and functions is considered as central in the mental model, subject to but not defined by the set of all rules.

In our mental model, these rules are translated to a set of (temporal<sup>4</sup>) logical propositions on top of the sets and functions introduced before. The propositions can be characterised as either structural or behavioural, depending on whether the rule concerns a static property or restricts the dynamic evolution of the population. In the context of this paper, there is no need for a complete characterisation of all possible kinds of propositions

---

<sup>3</sup>The ERCOS method introduces unary and binary relations. Unary relations (involving only one participating class) can be considered in an analogous way as binary relations.

<sup>4</sup>In this article, temporal operators will not be used.

for every construct in the method. We will only provide an overview and some basic illustrations, which allow the reader to understand how non-determinism is introduced in the mental model, and that are used in the derivations.

### 2.2.1 Structural Propositions.

Structural propositions have to be fulfilled at any time and follow from structural items in the conceptual model. We will refer to these structural propositions and the rules they represent, as *constraints*. The introduction of a class in itself already implies a number of *implicit constraints*, such as propositions stating that the population of two distinct classes are disjoint object sets<sup>5</sup>.

The refinement of a class with a binary relation can give rise to additional *integrated constraints*, corresponding with the specified connectivity and multiplicity. Consider once more the refinement of RESOURCE ALLOCATION as depicted in Fig. 1: the single circle in the refined class denotes a binary relation with single multiplicity, i.e. prohibiting duplicates. This gives rise to the following proposition in the mental model:

$$\forall ra_1, ra_2 \in \text{RESOURCE ALLOCATION}^\circ : ra_1 \neq ra_2 \Rightarrow \\ \neg(ra_1 \downarrow \text{RESOURCE ITEM} = ra_2 \downarrow \text{RESOURCE ITEM} \wedge \\ ra_1 \downarrow \text{RESOURCE NEED} = ra_2 \downarrow \text{RESOURCE NEED})$$

The "1" at the end of both lines towards the participating classes establishes a one-to-one connectivity, introducing the following propositions in the mental model:

$$\forall ra_1, ra_2 \in \text{RESOURCE ALLOCATION}^\circ : \\ ra_1 \downarrow \text{RESOURCE ITEM} = ra_2 \downarrow \text{RESOURCE ITEM} \Rightarrow \\ ra_1 \downarrow \text{RESOURCE NEED} = ra_2 \downarrow \text{RESOURCE NEED}$$

$$\forall ra_1, ra_2 \in \text{RESOURCE ALLOCATION}^\circ : \\ ra_1 \downarrow \text{RESOURCE NEED} = ra_2 \downarrow \text{RESOURCE NEED} \Rightarrow \\ ra_1 \downarrow \text{RESOURCE ITEM} = ra_2 \downarrow \text{RESOURCE ITEM}$$

The wave at the end of RESOURCE ITEM indicates that the participating resource item can change over time. The participation at that side is called mutable. A straight line indicates an immutable participation, which gives rise to a temporal logic proposition in the mental model.

Besides these implicit and integrated constraints, *explicit constraints* further restrict the possible populations of a conceptual model. In the example, the constraint named *same type* imposes the rule that no resource allocation

---

<sup>5</sup>In the context of generalisation/specialisation this has to be relaxed. Of course non-leaf classes in the inheritance hierarchy can share objects. In ERCOS these classes are by definition abstract. All objects belong to one concrete leaf-class. The final assertion then states that the populations of concrete classes must be disjoint.

can exist for which the allocated item is of a different resource type than the type that is needed. The specification of this constraint adds the following proposition to the mental model:

$$\forall ra \in \text{RESOURCE ALLOCATION}^\circ : ra \downarrow \text{RESOURCE ITEM} \downarrow \text{RESOURCE TYPE} = \\ ra \downarrow \text{RESOURCE NEED} \downarrow \text{RESOURCE TYPE}$$

The conjunction of all structural propositions introduced by the conceptual model is denoted as *SP*.

### 2.2.2 Behavioural Propositions.

Behavioural propositions state how the population evolves over time. The only way in which the population can change is when events occur. Such an event can correspond with the construction of a new object, with the destruction of an object, with the mutation of a characteristic or it can be a shell event, which is an event that provokes the occurrence of other events, and thus has as effects, the combined effects of these events. The occurrence of an event introduces (corresponding with its specification) propositions in the mental model, describing the evolution of the population from the state before the event occurred to the state after the event has occurred.

Consider, for example, the occurrence on a certain resource allocation object *ra*, of the mutator *change item* with a certain resource item object *item* as an argument. This event introduces the following behavioural proposition, which must be satisfied by the new population at the time the event has been successfully completed, given the original population:

$$(\text{new } ra) \downarrow \text{RESOURCE ITEM} = \text{item}$$

This proposition will be denoted by  $BP_{ra.change\ item}(item)$ . In this particular case, the proposition doesn't express the new state in function of the original state. In general, however, the state change will depend on the original state. So we must be able in the specification of an event to distinguish between the state before the event has occurred and the state after the event has occurred. We follow the convention that an unqualified object expression denotes the object in its original state, while qualifying an object expression with the **new** keyword denotes the object in its new state. This keyword is not allowed in shell events: the specification of elementary behaviour must be done in constructors, destructors and mutators.

The evolution of the population from one point in time to the next<sup>6</sup> point in time is then described by the original population and the conjunction of all structural propositions (both in the original and in the new state), all behavioural propositions introduced by the events that occur at the current

---

<sup>6</sup>In fact, time is considered to be continuous, and the next point in time has to be interpreted as the limit of the time going to the current point in time, from the future.

point in time and finally the *inertia propositions*<sup>7</sup> stating that the characteristics of which nothing is said in the behavioural propositions do not change. The inertia propositions corresponding with the simultaneous occurrence of a set of events  $A$ , will be denoted in what follows as  $IP_A$ . These inertia propositions can't be written as such for each event, but depend on the complete set of simultaneous events describing the transition to the next point in time. The inertia propositions can be written if this set is known, since we assume that the world is closed, i.e. that everything that is relevant for us is present in the model. These inertia propositions will show to be very important in defining the semantics for non-deterministic specifications.

The effects of an event in our approach have *transaction semantics*. It is considered to be atomic for an outside observer as well as internally: no intermediate state can be observed, but also internally in the mental model no intermediate states are considered. Either the total effect is obtained or the event is rejected as a whole. If the behavioural propositions introduced by an event (in conjunction with the structural propositions and the inertia propositions) give rise to a contradiction, and thus cannot be satisfied, the event as a whole is rejected and the population will evolve as if the event had not occurred. If the behavioural propositions do not lead to a contradiction, the effects will be realised. In the example of the mutator *change item*, the mutation will be rejected when the behavioural proposition conflicts with the structural propositions introduced by the connectivity and multiplicity, stating that an item can participate in an allocation only once, and by the constraint *same type*. In this situation, the new state will be as if the mutation had never occurred.

These semantics of failure and success can be explicitly stated in the formalisation. We don't do this in the context of this paper and focus on the semantics of events in the case that they succeed. Note that, at the level of analysis, the decision of how these contradictions will be detected and how the failure will be reported and processed is not relevant. What is important is that the system can never reach an inconsistent state. During design, this can be translated into invariants and preconditions and/or exceptions or using other techniques [10].

### 3 Non-Determinism in Conceptual Models

In object-oriented analysis, modellers tend to create deterministic specifications. The behavioural propositions in the mental model, introduced by the occurrence of an event, typically have a very deterministic nature: the specification of events gives a complete description of what the effect will

---

<sup>7</sup>The question of how to deal with those characteristics of which nothing is said, is known in artificial intelligence as the Frame Problem. We solve this Frame Problem with the Inertia Axiom, stating that everything of which nothing is said will remain unchanged.

be after the event has occurred, allowing only one possibility for the new population. This is often a kind of over-specification that severely reduces the elegance of a conceptual model. In fact, many problem domains are to a certain extent non-deterministic, even if modellers tend to make deterministic models of them. In this section we will elaborate a way to use non-determinism in specifications.

The formalism described in the previous paragraph, offers a lot of room for multiple populations: in the context of our mental model, different logical and mathematical operators lead to propositions that can be satisfied in more than one way. The usage of this kind of operators to form structural propositions in conceptual modelling is commonly accepted and applied. It is however not this kind of alternatives that is meant when we talk about non-determinism. Nevertheless the fact that the structural propositions of conceptual models offer these alternatives – and thus have multiple models in a model theoretic interpretation, corresponding with multiple populations in our mental model – is a crucial observation, allowing behaviour to be specified in a non-deterministic way.

Non-determinism occurs when the specification of an event allows the population to evolve (in the general case) in more than one way for a given occurrence of the event with given arguments. When we consider the evolution of the population in the mental model, there is only one population of the model at any point in time. But for every step forwards in time, different new states can be possible: the conjunction of all structural, behavioural and inertia propositions can have multiple models, from which one will be taken as the new population. The choice happens in a non-deterministic way. Notice that in particular cases, events can occur for which the non-deterministic specification turns out to have a deterministic effect. This occurs when there happens to be only one alternative to choose from.

Non-determinism can only be introduced in the evolution of a population via the non-deterministic specification of events: the inertia propositions fix every element of the population of which nothing is said in the behavioural propositions. This also implies that non-determinism only offers alternatives for the next state: later on in the evolution of the population, a non-deterministic choice of the past cannot be revised.

More concrete, non-deterministic behaviour can be introduced in constructors (e.g. concerning the initialisation of a property of the new object), in mutators (i.e. concerning the new value of the property that is changed), and most general, in shell events. The result of a query can also be defined in a non-deterministic way. This is the case when the specification allows the result to be one of multiple possible values. Typically, the result of such a query is used in the specification of an event, turning this event in a non-deterministic event.

The usage of non-determinism in the specification of behaviour is not common-practice in object-oriented conceptual modelling, as opposed to

the usage of the operators mentioned above in (the formalisation of) the structure of a conceptual model.

In the rest of this section, we will propose two non-deterministic operators. First we will introduce the example of the *allocate item* event and present a deterministic solution. Then the *one of operator* will be introduced in the context of the example. We will then define the semantics of this operator. Next, we will introduce the *non-deterministic choice operator*, which is more general, and provide semantics for this operator. Finally we will show how specifications using the *one of operator* can be rewritten in terms of a *generalised non-deterministic choice*. In this way, we can in the future reason about non-determinism in a uniform way, i.e. in terms of the non-deterministic choice operator.

We consider it a good methodological rule to specify non-determinism in an explicit way using these non-deterministic operators, in contrast with a specification where non-determinism would be present as an implicit consequence of the presence of logical and relational operators allowing multiple alternative populations and evolutions, as mentioned in the beginning of this section. The use of these latter operators in the specification of functionality is therefore limited to logical expressions that are evaluated in the original state and not asserted.

### 3.1 The *allocate item* Event: a Deterministic Solution

We will introduce non-determinism through an example. Suppose, in the context of the conceptual model in Fig. 1, that we want to specify an event *allocate item*, which – when applied to a resource need object – creates a resource allocation object with the involved resource need object as one participant and an available resource item object of the required resource type as the other participant. If no appropriate item is present, the event should fail and as a consequence be rejected.

Each specification of this event, deterministic or non-deterministic, will involve the construction of a new resource allocation object with a resource item object and the given resource need object as arguments. In a deterministic specification for the *allocate item* event, a selection of an appropriate item among all available items must be made, and this item must be passed to the constructor of RESOURCE ALLOCATION. A possible specification constructs the set of all available items conforming to the connectivity constraint stating that an item can only be allocated once, and the *same type* constraint stating that the type of the allocated item must be the same as the required type. A possible choice would then be the item with the lowest *Id*. The deterministic specification of the event in ERCOS is as follows:

**Spec. 1** *Deterministic specification of the allocate item event*

```

class RESOURCE_NEED
  event
    shell
      allocate item
    effect
      let appropriate items = {item in RESOURCE_ITEM°:
        item↓RESOURCE_TYPE = self↓RESOURCE_TYPE
        and
        item↑RESOURCE_ALLOCATION = empty set}
      in
        if appropriate items ≠ empty set then
          let item = [item in appropriate items:
            (for all item2 in appropriate items:
              item2→Id ≥ item→Id)]
          in
            RESOURCE_ALLOCATION.allocate(self, item)
          else
            false
        end RESOURCE_NEED

```

In this specification, we use a deterministic selection operator [ ], which delivers the unique element from the given set that satisfies the given condition. For this operator to be used in a syntactically correct way, it must be certain that there exists one and only one such element (here guaranteed by the constraint *different ids*, and the explicit test that the set is not empty). The specification also explicitly states that the event will fail if no such element is found.

The choice of the item in Spec. 1 is made using a quite arbitrary criterion. However, which item is chosen should not be relevant, as long as the item is an appropriate item. In fact, if this selection of the item with the lowest *Id* is not part of the reality, the specification above cannot be part of a correct model for this reality. If the problem domain contains an inherently non-deterministic functionality, a deterministic conceptual model cannot model this in a correct way. The absence of non-deterministic constructs often leads to this kind of inappropriate models.

### 3.2 A Non-Deterministic Solution Using the one of Operator

Instead of selecting an item from the set of appropriate items according to a deterministic criterion, one could specify that it doesn't matter which item is chosen. As a first instrument to specify this, we introduce the **one of** operator. This operator can be applied to an expression yielding a set of alternatives, and forms an expression that will yield exactly one element of

this set. In correspondence with the deterministic selection operator, the use of the `one of` operator is only correct, when it can be proven that the set to which the operator is applied cannot be empty. This delivers the following non-deterministic specification for the *allocate item* event:

**Spec. 2** *Non-deterministic specification of the allocate item event, using one of*

```

class RESOURCE NEED
  event
    shell
      allocate item
    effect
      let appropriate items = {item in RESOURCE ITEM°:
        item↓RESOURCE TYPE = self↓RESOURCE TYPE
        and
        item↑RESOURCE ALLOCATION = empty set}
      in
        if appropriate items ≠ empty set then
          let item = one of (appropriate items)
          in
            RESOURCE ALLOCATION.allocate(self, item)
        else
          false
    end RESOURCE NEED

```

In this specification the deterministic selection operator (including the selection criterion) is replaced with the non-deterministic `one of` operator.

The static type<sup>8</sup> of the expression as a whole is the static base type of the operand set, i.e. the most specific common super-type of the static types of all elements in that set. In other words, if the set is of static type  $P(A)$ , then the `one of` will yield a result of static type  $A$ . This common super-type  $A$  must of course exist and be a correct type corresponding to the context in which the `one of` expression is used. In the context of the specification above, it is obvious that this means that all objects of the set `appropriate items` must be syntactically acceptable as the indicated argument for the constructor of `RESOURCE ALLOCATION`, i.e. a resource item object. We stress the importance of these strict static rules. An alternative would be to accept models that do not offer these guarantees. This could then be defined to lead to failure when the event occurs and the set would

---

<sup>8</sup>We use the term *static type* to indicate the type that can be derived for the expression from its specification, regardless of any actual evaluations. The type in the context of an actual evaluation, is called the *dynamic type*, and is guaranteed to be a subtype of the static type.

be empty or contain objects of a wrong type. We reject these models as incorrect. We are convinced that this leads to higher quality modelling, since it helps to eliminate errors in models. This, however, comes with the extra costs of validating the obligations.

### 3.3 Semantics for the one of Operator

From an operational viewpoint, the **one of** operator introduces a choice that will be made when evaluated. For *allocate item* as specified above, every choice will lead to a successful realisation of the effect of the event, since all constraints will be satisfied as a consequence of the definition of **appropriate items**. After a successful occurrence of the event, one of the appropriate resource items will be allocated to the resource need through a newly created resource allocation object.

A first approach to provide declarative semantics for the **one of** operator is using the set-theoretic  $\in$  operator. The assertion **item = one of (appropriate items)** in the specification gives rise to  $item \in appropriate\ items$  in the behavioural propositions in the semantic domain. Specification 2 is then mapped (after some derivations conserving the semantics) on the introduction of following behavioural proposition  $BP_{rn.allocate\ item}$  in the semantic domain, stating the effects, when the *allocate item* event occurs on a resource need object *rn*:

**Spec. 3** *Semantics for Spec. 2 using  $\in$  for one of*

$$\begin{aligned}
BP_{rn.allocate\ item} = & \\
& \exists appropriate\ items \in P(\text{RESOURCE ITEM}^\circ) | \\
& \quad appropriate\ items = \{item \in \text{RESOURCE ITEM}^\circ | \\
& \quad \quad item \downarrow \text{RESOURCE TYPE} = rn \downarrow \text{RESOURCE TYPE} \wedge \\
& \quad \quad item \uparrow \text{RESOURCE ALLOCATION} = \emptyset\} \wedge \\
& \exists item \in appropriate\ items | \\
& \quad new\ \text{RESOURCE ALLOCATION}^\circ = \\
& \quad \quad \text{RESOURCE ALLOCATION}^\circ \cup \{new\ object\} \wedge \\
& \quad \quad new\ object \downarrow \text{RESOURCE NEED} = rn \wedge \\
& \quad \quad new\ object \downarrow \text{RESOURCE ITEM} = item
\end{aligned}$$

This specification uses the existential quantifier  $\exists$  for mapping the **let** construct and *new object* is used to denote a newly created object. If a **one of** expression is used in-line, e.g. as an argument, first a substitution is needed, introducing a **let** construct and a new variable. Note that here it is no longer needed to differentiate on whether or not *appropriate items* is empty. The syntactical rule stating that for a model to be correct, the set from which is chosen must be guaranteed to be not empty, is not needed here after the mapping in the semantic domain. The corresponding test in

the specification has also disappeared in the semantics, since the semantics of the `else` case are equivalent with the semantics of the existential quantification.

The inertia proposition  $IP_{rn.allocate\ item}$  corresponding with the isolated occurrence of the *allocate item* event on *rn*, states that besides the creation of a new object of the `RESOURCE ALLOCATION` class, and the initialisations for the respective participants, nothing has changed in the population of the model.

### 3.4 Non-Determinism Using the Non-Deterministic Choice Operator

The `one of` operator can be used when the non-determinism is parameterised by an element to be chosen from a set. This is clearly the case for the *allocate item* event: the structure of the effect does not depend on which item is chosen. In general however, non-determinism can introduce a choice between two or more alternatives that do not have that much structural commonalities. The alternatives can ultimately be completely different. To make such specifications possible, we introduce a non-deterministic choice operator  $\nabla$ .

The non-deterministic choice operator  $\nabla$  can be used to combine two arbitrary specifications for alternative effects into one. Note that these two constituent parts must both be syntactically correct. The combination denotes an effect that results from the realisation of one of both effects.

As an illustration of the usage of this operator, we introduce a simple example of a model, shown in Fig. 2, in which there is a class `RECTANGLE`, with two attributes *Width* and *Height*. Let's now introduce an event *double width or height*, which should either double the width or double the height of the rectangle involved in the event<sup>9</sup>. The specification of the event is:

**Spec. 4** *Non-deterministic specification of the double width or height event using the non-deterministic choice operator  $\nabla$*

```
class RECTANGLE
  event
    shell
      double width or height
    effect
      set height(self→Height * 2)
   $\nabla$ 
```

---

<sup>9</sup>We don't consider other alternatives to double the surface as part of the intended semantics of this event: we intend to introduce an operator that introduces a choice between two alternatives.

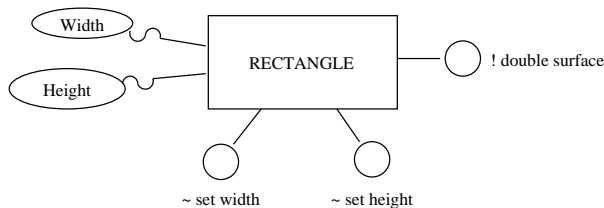


Figure 2: A RECTANGLE class with width and height

```

set width(self→Width * 2)
end RECTANGLE

```

Of course more alternatives can be combined by using more than one non-deterministic choice operator. The non-deterministic choice operator is among others commutative, associative, and distributive with respect to the logical conjunction. The operator cannot be used in the specification of queries, since that specification consists of an expression denoting result of the query. Non-determinism for queries is modelled using the `one of` operator.

### 3.5 Semantics for the Non-Deterministic Choice Operator

In this section, we try to formalise the informal semantics of the  $\nabla$  operator, as introduced above. From an operational viewpoint, one of the alternative effects is chosen and this effect will be the effect of the whole construction. Providing declarative semantics in terms of propositions in the mental model is more complicated. Suppose that we want to provide semantics for an event with a specified effect  $e_1 \nabla e_2$ , where  $e_1$  and  $e_2$  denote effect specifications.

Initially, we searched for something of the form:  $BP_{e_1 \nabla e_2} = BP_{e_1} \otimes BP_{e_2}$ , i.e. we tried to map the construct by mapping both constituent effects, and then combining them using an appropriate logical operator. We considered the disjunctive or operator as well as the exclusive or operator, but they both did not fit in. The problem is that, as a consequence of the intended semantics (i.e. one and only one of the alternatives is chosen), both alternatives might de facto be realised. On one hand, this excludes the usage of the disjunctive or operator, since this would allow both effects to take place. On the other hand, if the effect of one alternative is realised by the non-deterministic choice, other events occurring at the same time can still imply the effect of the unchosen alternative. This would not be possible when we define the semantics in terms of the exclusive or.

We finally found an appropriate way to provide formal semantics for the non-deterministic choice operator, by taking also the inertia propositions into consideration. As we already indicated, stating the behavioural proposition is not straightforward, and the same is true for the inertia proposition,

since this of course must correspond with the option that is taken for the non-deterministic choice. However if we consider the logical conjunction of both, corresponding with the semantics of the advancement of time in Sect. 2.2, we can define the semantics of the non-deterministic choice operator. Suppose that the event with specified effect  $e_1 \nabla e_2$  occurs in isolation, then we define the semantics as follows:

**Spec. 5** *Semantics for the non-deterministic choice operator in an isolated occurrence of  $e_1 \nabla e_2$*

$$BP_{e_1 \nabla e_2} \wedge IP_{e_1 \nabla e_2} \equiv (BP_{e_1} \wedge IP_{e_1}) \vee (BP_{e_2} \wedge IP_{e_2})$$

This corresponds with the informal semantics we described when we introduced the operator. It gives us a mapping in the semantic domain, for the isolated occurrence of an event involving the non-deterministic choice operator at the top level of the effect specification. The reasoning can be generalised to a specification in which more than two alternatives are offered. Due to the selection of constructs available to specify the effect of an event and the properties of the non-deterministic choice operator, every effect specification can be transformed into a form in which zero or more top-level non-deterministic choice operators combine one or more alternatives that are each in their selves deterministic. Combining this transformation with the semantics given in this paragraph, semantics can be given for any given event involving non-deterministic choice operators. In a similar way, concurrent occurrences of events involving the non-deterministic choice operator can be dealt with.

In Spec. 6 we derive the semantics of the isolated occurrence of the event *double width or height* in Spec. 4 on a rectangle object  $r$ , including the inertia proposition, limited to the width and the height of this rectangle:

**Spec. 6** *Semantics for an isolated occurrence of the double width or height event*

$$\begin{aligned} & BP_{r.double\ width\ or\ height} \wedge IP_{r.double\ width\ or\ height} = \\ & BP_{(r.set\ height(r \rightarrow Height*2) \nabla r.set\ width(r \rightarrow Width*2))} \wedge \\ & IP_{(r.set\ height(r \rightarrow Height*2) \nabla r.set\ width(r \rightarrow Width*2))} = \\ & (BP_{r.set\ height(r \rightarrow Height*2)} \wedge IP_{r.set\ height(r \rightarrow Height*2)}) \vee \\ & (BP_{r.set\ width(r \rightarrow Width*2)} \wedge IP_{r.set\ width(r \rightarrow Width*2)}) = \\ & ((new\ r) \rightarrow Height = r \rightarrow Height*2 \wedge (new\ r) \rightarrow Width = r \rightarrow Width) \vee \\ & ((new\ r) \rightarrow Width = r \rightarrow Width*2 \wedge (new\ r) \rightarrow Height = r \rightarrow Height) \end{aligned}$$

### 3.6 The Generalised Non-Deterministic Choice Operator

The non-deterministic choice operator can be used in a generalised way as follows:  $\nabla_{x \text{ in } X} e(x)$  denotes the combination using the non-deterministic choice operator of the effect<sup>10</sup> for each element of the set  $X$ . It must be guaranteed that this set is not empty. Semantics for the generalised non-deterministic choice operator follow easily from the semantics given in Spec. 5:

**Spec. 7** *Semantics for the generalised non-deterministic choice operator*

$$BP_{\nabla_{x \text{ in } X} e(x)} \wedge IP_{\nabla_{x \text{ in } X} e(x)} \equiv \bigvee_{x \in X} (BP_{e(x)} \wedge IP_{e(x)})$$

Every usage of the `one of` operator in an event can be rewritten in terms of the generalised non-deterministic choice operator, allowing us to reason in the future about non-deterministic events in terms of the non-deterministic choice operator: a fragment  $e(\text{one of } X)$  of the effect specification of an event, can be replaced by  $\nabla_{x \text{ in } X} e(x)$ . This transformation is possible because we demanded that each element of the set to which the `one of` operator is applied, is of an appropriate type for the context in which the result of the operator is used. As an illustration and as a basis for the following section, consider the specification of the *allocate item* event. After applying the substitution, we obtain the following:

**Spec. 8** *Non-deterministic specification of the allocate item event, using the generalised non-deterministic choice operator*

```
class RESOURCE NEED
  event
    shell
      allocate item
  effect
    let appropriate items = {item in RESOURCE ITEM°:
      item↓RESOURCE TYPE = self↓RESOURCE TYPE
      and
      item↑RESOURCE ALLOCATION = empty set}
    in
      if appropriate items ≠ empty set then
        ∇item in appropriate items RESOURCE ALLOCATION.allocate(
          self, item)
      else
        false
end RESOURCE NEED
```

---

<sup>10</sup> $e$  denotes a specification fragment parameterised by the variable of the generalised non-deterministic choice operator.

## 4 Separating Structure and Behaviour

As we already indicated in Sect. 2.2, and as is stressed in [10], all structural rules that are deemed relevant in the problem domain, must be present in the conceptual model. This can be realised via the implicit or integrated constraints of the concepts introduced in the model, as well as via the explicit specification of constraints further restricting the possible populations. In this section we will look at the interaction between these constraints and the non-deterministic specification of events.

We will show that non-determinism significantly increases the potential for the separation of concerns in conceptual modelling. A deeper look at the semantics of the occurrence of a non-deterministic event and the advancement of time, will learn us that the effect of an event can be specified separately from the constraints that must be fulfilled by (the evolution of) the population. We will see that there is no need to duplicate knowledge from structural constraints in the non-deterministic specification of behaviour, and even more, that this duplication gives rise to bad models. We will illustrate this for the *allocate item* event from Spec. 8. We will finally present a non-deterministic specification for the event which is completely separated from the structural constraints in the model.

### 4.1 Full Semantics of the Occurrence of a Non-Deterministic Event

When reasoning about the evolution of the population as a consequence of the occurrence of a collection of events  $A$  at a given point in time, we must consider the behavioural propositions and the inertia propositions, *as well as* the structural propositions in the context of two populations: the population before and the population after the occurrence of the events. The overall proposition that describes this step – supposed that the events are successful – becomes (where  $SP_{new}$  indicates the structural propositions in the new state):

**Spec. 9** *Semantics for the progress of time, given the occurrence of a set of events  $A$*

$$SP \wedge SP_{new} \wedge BP_A \wedge IP_A$$

For the isolated occurrence of a non-deterministic event  $\nabla_{x \text{ in } X} e(x)$ , we can derive the following semantics, using the semantics given in Spec. 7:

**Spec. 10** *Semantics for the progress of time, given the occurrence of a non-deterministic event  $\nabla_{x \text{ in } X} e(x)$*

$$\begin{aligned}
SP \wedge SP_{\text{new}} \wedge BP_{\nabla_{x \text{ in } X} e(x)} \wedge IP_{\nabla_{x \text{ in } X} e(x)} &= \\
SP \wedge SP_{\text{new}} \wedge (\forall x \in X (BP_{e_1(x)} \wedge IP_{e_1(x)})) &= \\
\forall x \in X (SP \wedge SP_{\text{new}} \wedge BP_{e_1(x)} \wedge IP_{e_1(x)}) &
\end{aligned}$$

This shows that the non-deterministic event as a whole will be successful as long as there is at least one alternative that allows success, and that a choice will be made between all such alternatives. Only if all alternatives would give rise to a contradiction, the event will fail and as a consequence be rejected. It is not necessary to eliminate explicitly from the set of alternatives, those alternatives that – although syntactically correct – are not guaranteed to succeed. Or equivalently, it is not necessary to explicitly guarantee that all specified alternatives will lead to success and a fortiori are consistent with the structural propositions.<sup>11</sup>

## 4.2 Redundant Specification of *allocate item* Leads to Bad Model

In the specification of the *allocate item* event in Spec. 8, we can observe the presence of redundancy in the overall propositions describing a step in time involving the occurrence of the *allocate item* event. This redundancy exists because we tried to explicitly eliminate contradictory alternatives. We will show that the specification of the event contains two fragments which are superfluous respectively inadequate.

First consider the issue that the item that is allocated in the event must be of the resource type that is needed. In Spec. 8, we observe the following part in the set comprehension for `appropriate items`, evaluated in the original state:

`item`↓RESOURCE TYPE = `self`↓RESOURCE TYPE

Since the participation of a resource type object in both a resource item object as well as in a resource need object is immutable, this preselection explicitly ensures that if *any* of the items in `appropriate items` is used in the construction of a RESOURCE ALLOCATION, the constraint *same type* will hold for the new resource allocation. It is however easy to see that this fragment in the specification of the event is superfluous: the propositions in  $SP_{\text{new}}$ , introduced by the constraint *same type*, imply that for a successfully created resource allocation, the type of the resource item and the type of the resource need must and thus will be the same. So the preselection of items with respect to their resource type, is not needed. Those alternatives that would not realise this constraint would lead to a failure and will thus not be selected.

---

<sup>11</sup>If in an event a non-deterministic query is evaluated, the same reasoning applies. The query evaluation can be seen as an expansion of the result expression.

The second fragment – also part of the set comprehension for **appropriate items** – states that the item must not be allocated:

$$\text{item} \uparrow \text{RESOURCE ALLOCATION} = \text{empty set}$$

This is also evaluated in the original state. The rationale behind this fragment was to ensure that no double allocations could be made for the same item, by selecting those items that were not allocated in the original state. In other words, the rationale was to ensure that the connectivity and multiplicity constraints for the binary refinement of **RESOURCE ALLOCATION** would be conserved. Again, since only the alternatives that do not give rise to contradictions (here in combination with the integrated constraints of the refinement) are taken into consideration, it is not necessary to perform the selection explicitly in the event.

From a separation of concerns perspective, it is not a good idea to ensure the satisfaction of structural constraints explicitly in the event. This gives rise to the duplication and spreading of knowledge. For example, the knowledge stating that the item that is allocated must be of the same type as the type that is needed, is present in the constraint *same type* as well as in Spec. 8 for the *allocate item* event. Not only is this difficult to maintain (e.g. when our understanding of the problem domain evolves), and does this increase the complexity and thus decrease the comprehensibility of the specification, it is also very prone to errors. The example above contains a mistake that illustrates this: we supposed that, to be sure that no double allocations pop up, it was necessary and sufficient to verify that there were no allocations for the selected item in the original state. Although in an isolated occurrence of the event, our preselection is adequate, this is not true when multiple events occur at the same time.

In a first scenario, suppose two *allocate item* events occur at the same time, for two resource needs with the same type of item needed. In this scenario, the **appropriate items** set for both events will be the same. Supposed that the integrated constraints on the refinement were not present, this preselection would not guarantee that there are no double allocations, since both non-deterministic choice operators could choose the same item. The constraints however, prevent the double allocation: if possible, two different items will be selected.

A second scenario is even erroneous: suppose there is no item available (in the original state), but that at the same time a resource allocation withdrawal and the *allocate item* event occur, both related to the same resource type. The set-comprehension for **appropriate items** will produce an empty set in this scenario, and the *allocate item* event will fail and thus be rejected. This is too stringent: it would be preferable that the de-allocated item could be used in the *allocate item* event.

### 4.3 A Better Solution Separating Behaviour and Structure

We propose to eliminate this redundant and inadequate preselection. The *allocate item* event can be specified in a concise and elegant way as follows:

**Spec. 11** *Non-deterministic specification of the allocate item event, without redundant preselection*

```
class RESOURCE NEED
  event
    shell
      allocate item
    effect
      if RESOURCE ITEMSo ≠ empty set then
        ∇item in RESOURCE ITEMo RESOURCE ALLOCATION.allocate(
          self, item)
      else
        false
    end RESOURCE NEED
```

Notice that this specification will have the same effect as Spec. 8, on condition that the latter does not fail as indicated in the second scenario above. In that scenario, the new specification will be successful.

While a completely deterministic specification of a functionality can become extremely complex and will duplicate most if not all constraints in order to fulfil them, a non-deterministic specification can typically be a lot clearer. The benefits of separating the specification of structure from the specification of behaviour are already clear in the context of the simple event *allocate item*. We expect that in more complex conceptual models, the benefits will become important. Non-determinism is the key concept allowing this separation of concerns.

Of course, in a later phase towards the implementation of the software system, an algorithm might be needed: while in certain implementation paradigms, like logic programming, the separation can be conserved, most implementation languages require that a deterministic choice is made, satisfying all constraints. As far as the *specification* of the implementation code is concerned, the separation can be maintained to a certain degree: in the context of an object-oriented implementation language, the structural constraints can be specified as class invariants, while the behaviour can be specified in the post-conditions of the method implementing the functionality, in accordance with the contract paradigm [5]. However, a policy is still needed to deal with exceptional behaviour, i.e. with situations in which the event is not successful. When using pre-conditions, this will in general

require repeating the knowledge about the structure. In a defensive approach, this repetition can be avoided at the cost of under-specifying the circumstances in which an exception could be thrown.

## 5 Conclusion

In this paper we introduced two non-deterministic constructs that can be used in the formal and declarative specification of events and queries. The `one of` operator enables us to write expressions that will non-deterministically evaluate to an element of a set. This operator is appropriate if all alternative effects have the same specification structure. The non-deterministic choice operator offers the possibility to specify the effect of an event as a non-deterministic choice between two or more alternative effects. This operator allows alternatives that have a completely different structure.

We have provided formal semantics for these operators in the context of a mental model for conceptual models. We have also shown how non-determinism can be used to separate the specification of structure from the specification of behaviour, leading to significantly better models.

Of course determinism or non-determinism is a characteristic that can be inherently observed in the problem domain. Non-determinism is not necessary a form of under-specification that – through refinement of specification – should be reduced. If a problem domain is inherently non-deterministic, any valid conceptual model will by definition be non-deterministic and valid conceptual models for deterministic problem domains will be deterministic. And provided that the implementation platform supports non-determinism, there is no inherent reason why the non-determinism could not be conserved.

On the other hand, it will also be possible to use non-deterministic constructs to build deterministic models (e.g. in the context of generalisation/specialisation). Moreover, in many cases, conceptual modelling is not only observing and describing the problem domain, but also creating the problem domain. It is here that modellers typically incline towards deterministic specifications, while a non-deterministic approach could be superior.

## 6 Future Work

We focused on the opportunities for non-determinism in the context of object-oriented conceptual modelling, from a methodological viewpoint. Non-determinism as a concept has been widely studied in computer science, from various viewpoints and in various context. A study hereof is needed, and is expected to bring interesting insights.

In this paper, discussion was limited to simple cases in which only one non-deterministic event takes place at a time. In general, and certainly when using the method to make models of more complicated problem domains,

multiple concurrent occurrences of events can take place. We are investigating the usage of non-determinism in complex planning and resource allocation problems. We expect that non-determinism can be used at its best here. In this context, the semantics of failure also has to be further elaborated.

In the treatment of non-determinism in this paper, we introduced a choice that was almost completely non-deterministic. The only restriction is that the choice must be compatible with the structural propositions. But in certain cases the modeller wants to be able to influence the way in which the choice is made by introducing certain criteria. How this can be done without losing the power of non-determinism is a subject for further study.

We dealt with the concept of inertia propositions and used it to provide semantics for the non-deterministic choice operator. A complete study of the frame problem and the inertia propositions in the context of conceptual modelling and more particular of our mental model, has not yet been carried out. Also as this mental model is concerned, alternative approaches and their impact on non-determinism are under investigation. More in particular, a translation to IDLogic logical theories is being made [2]. This opens the possibility for a multi-paradigm method combining the best from both worlds. The translation of non-deterministic specifications of behaviour in ERCOS to IDLogic tasks is still under investigation [3] and promises a lot of useful insights.

Generalisation/specialisation is a differentiating concept of the object-oriented paradigm. The principle of strengthening aspects of a more general class in a specialisation class will be re-evaluated in the context of non-deterministic specifications of functionality. We expect that the integration of non-determinism in generalisation/specialisation will add to the power of non-determinism as well as of generalisation/specialisation. Non-determinism is needed if we want to provide a relevant specification of functionalities at higher levels of abstraction in the class hierarchy while respecting sub-typing conforming the Substitution Principle of Liskov [4].

The notion of non-determinism has been introduced in this paper in the context of the ERCOS method for conceptual modelling, using the declarative specification of effects of events as a vehicle. Another issue is how non-determinism can be dealt with in methods based on the Unified Modeling Language. This can deliver interesting results, given the wide variety of techniques to model behaviour in UML.

Another direction for further research lays in the field of tool support: rapid prototyping tools under development can be extended with support for non-deterministic specification of behaviour. Moreover static analysis, including an analysis of the states that can be reached and of the presence of failure, can be a very interesting technique to assess the qualities of a non-deterministic specification.

## References

- [1] Bekaert, Pieter and Steegmans, Eric, Non-Determinism in Conceptual Models, Proceedings of the Tenth OOPSLA Workshop on Behavioral Semantics, OOPSLA'01, 2001.
- [2] Bekaert, Pieter, Van Nuffelen, Bert, Bruynooghe, Maurice, Gilis, David and Denecker, Marc, On the Transformation of Object-Oriented Conceptual Models to Logical Theories, Submitted for the 21st International Conference on Conceptual Modeling (ER2002).
- [3] Bekaert, Pieter, Van Nuffelen, Bert, Bruynooghe, Maurice, Gilis, David and Denecker, Marc, Transforming EROOS Conceptual Models to ID-Logic Theories: A Case Study, in preparation, draft at <http://www.cs.kuleuven.ac.be/~dtai/kt/btw/>, Report, Department of Computer Science, K.U.Leuven, Leuven, Belgium.
- [4] Liskov B., Data abstraction and hierarchy, ACM SIGPLAN Notices, **23**(5), 1988.
- [5] Meyer B., Design by Contract, Advances in Object-Oriented Software Engineering, eds. Mandrioli D. and Meyer B., Prentice-Hall, 1991.
- [6] Said, J., and Steegmans, E., Automatic Transformation of Conceptual Models into Design Models, Automating the Object-Oriented Software Development Methods Workshop, European Conference on Object-Oriented Programming (ECOOP) 2001, Budapest, Hungary, 2001.
- [7] Spivey, J.M., The Z notation: a reference manual, Prentice Hall, 1989.
- [8] Steegmans, E., Lewi, J., De Backer, S., Dockx, J., Swennen, B., and Van Baelen, S., EROOS Reference Manual Version 1.1, Department of Computer Science, K.U.Leuven, Leuven, B, 1995, 173 p.
- [9] Van Baelen, S., Lewi, J., Steegmans, E., and Van Riel, H., EROOS: An Entity-Relationship based Object-Oriented Specification Method, Technology of Object-Oriented Languages and Systems TOOLS 7, eds. Heeg, G., Magnusson, B., and Meyer, B., Prentice-Hall, Hertsfordshire, UK, 1992, pp. 103-117.
- [10] Van Baelen, S., Lewi, J., and Steegmans, E., Constraints in Object-Oriented Analysis and Design, Technology of Object-Oriented Languages and Systems TOOLS 13, eds. Magnusson, B., Meyer, B., Nerson, J.-M., and Perrot, J.-F., Prentice-Hall, Hertsfordshire, UK, 1994, pp. 185-199.