A Framework for Information Systems Design based on Object-Oriented Concepts and Petri Nets

Peter A.C. Verkoulen

University of Twente[1]
Department of Computer Science
P.O. Box 217
NL-7500 AE Enschede
the Netherlands

1 Introduction

During the past decade, (information) system designers have been confronted with an enormous increase of the complexity of the systems they have to develop. To start with, the systems have become larger. Novel applications have many purposes, contrary to systems that are relatively simple and have limited functionality. Moreover, as the systems to be developed increase, the projects of developing such systems also increase. This requires a good way of communication, in order to avoid misunderstandings between the designers. All this also implies an increase of routine work, which sometimes causes the designers to lose sight of the really important, creative work. When the developer is given the possibility of using a specification method that takes care of (part of) this routine work, he may be able to concentrate on the most important aspects. E.g., a specification method may support reuse of frequently needed components. Finally, it can be observed that in modern information systems, both static and dynamic properties are important and complex. Contemporary IS are neither systems dealing merely with large amounts of information upon which simple operations have to be applied (e.g. a single-user database with primitive query- and update facilities), nor are they systems with elementary data and more complex dynamics (like a communication protocol).

New systems are developed by integrating data- and behaviour specifications or by applying integrated system concepts, i.e. concepts that intrinsically cover both static and dynamic system aspects. In each development phase, both perspectives (statics and dynamics) should be balanced. Such an integrated development method makes tuning of static and dynamic aspects of a system under design more feasible. Moreover, the modelling process becomes more flexible, as one does not have the rigid and sometimes artificial separation between data and process modelling.

In this paper, we describe an *integrated* way of developing (information) systems. This integration has two dimensions. First and foremost, it covers what has been called *horizontal integration* [Ramackers & Verkoulen, 1992]. This means that modelling static and dynamic aspects of a system has been integrated, at all levels of detail. Although this horizontal integration is the most important one, we have also accomplished *vertical integration* to a certain extent: we have designed a formal framework for IS analysis and design, we have developed tool support for this framework and we have dealt with practical application of our approach. Of course, we can only outline these aspects in this short paper.

Our framework has a formal semantics. In our opinion, this is necessary for a sound specification technique. The most important benefit from choosing a framework with a formal semantics is the possibility to obtain *executable specifications*. These can be used for simulation purposes, which gives the designer the opportunity to test his design. Of course, simulation does not provide a means for formal verification. However, in practise, simulation is very useful because it enables the designer to discover and repair errors in his design. Moreover, simulation can be used in order to communicate the design to non-experts. A formal specification technique also helps a designer

[1] The research described in this paper was performed while the author was working at Eindhoven University of Technology, supported by a grant from the Netherlands Organisation for Scientific Research.

to find omissions, contradictions and ambiguities, because he is forced to define exactly and in full detail what he means. The fact that formal specifications are not ambiguous also makes code generation feasible. A formal technique often leads to a hierarchical development method. Furthermore, a formal semantics makes it possible to develop techniques for formal verification. In fact, we propose a technique for *automatic* verification of a design. A final benefit that we want to mention is the possibility to work in a modular way: it is possible to build a specification from smaller models that have already been proven to be correct. Such a correctness proof can only be accomplished when the modules have a formal semantics.

Of course, there are other specification methods that have an underlying formal semantics. However, a considerable number of these approaches suffer from one or more problems [Lyytinen, 1987]. The most important problem is the fact that they are often suited for modelling either static or dynamic aspects, instead of both. And even when they are suited for both data and process modelling, they are often still focussed on one of these perspectives. Moreover, many of these formal modelling techniques do not provide the modeller with sufficient tool support or they are intrinsically difficult. On the other hand, modelling techniques that are comprehensible by the designer are often informal, causing the aforementioned problems. It is also important for an IS development technique to be able to describe the *environment* of the information system for requirements engineering. This is necessary to obtain a better insight in the function of the IS and the way the IS is or will be used.

Our framework is based upon a high-level Petri net model, comparable to Coloured Petri Nets [Jensen, 1992], and an object-oriented data model with an associated data manipulation language. The framework has been called *SimCon*[2]. This paper can only give you an impression of the SimCon approach. Full details can be found in [Verkoulen, 1993].

Our work is part of the *ExSpect*[3] project. This project started in the late 80's. It aims at providing a formal framework for (information) systems design. ExSpect combines a functional language with a high-level Petri net model. Also, an advanced CASE tool is under development. This tool is commercially available since January 1993. We have used ExSpect successfully in a number of practical situations. Examples of such practical applications are cases in the context of the TASTE project [Van der Aalst, Voorhoeve & Waltmans, 1989], a study at Schiphol Airport and a project with Dutch Railways. In this paper, we build upon the ExSpect approach.

More details about ExSpect can be found among others in [Van Hee & Verkoulen, 1991; Van Hee, Rambags & Verkoulen, 1993; Houben & Verkoulen, 1991; Verkoulen, 1993].

2 Outline of the SimCon Approach

In this paper, we introduce the SimCon approach, which we claim to be a framework that meets the aforementioned demands. In our approach, data and process modelling have been integrated in a consistent and comprehensible way. For describing the static aspects of a system, we have designed an object-oriented data model. This object model is used to model the passive entities in a system. For describing the properties of the active entities, a high-level Petri net model has been incorporated. Both models have been integrated at a high conceptual level. Our approach forms the fundament of a specification tool, which will be introduced too. Finally, we provide a means of automatic verification of constraints from the object model in the net model.

In this section, we introduce the components of the SimCon framework. In [Verkoulen, 1993], these components are dealt with in full detail. Section 3 provides a small example that illustrates the application of SimCon in practise.

[2] This stands for *Simple Integrated Model for Complex Object Networks*, but it also contains the prefixes of the two major concepts in the SimCon data model, viz. simple objects and container objects.

[3] ExSpect stands for *Executable Specification Tool*.

2.1 The SimCon Object Model (SCOM)

We have developed an *object model* that is close to existing and well-known models that are being used in practise, like the Entity-Relationship model. The SimCon Object Model (SCOM) also incorporates the most important object-oriented concepts from [Atkinson, Bancilhon, DeWitt, Dittrich, Maier & Zdonik, 1989], such as object identity and inheritance. SCOM has some resemblance with GOOD [Gyssens, Paredaens & Van Gucht, 1990]. We have kept in mind the purpose of this object model: we did not want to develop yet another object model, but the object model to be used in our framework must be suited for integration with a process model. Therefore, we have incorporated the most important concepts that have proved to be useful in practise, with some extensions for making the integration possible. We have combined these aspects, together with some new features, in a new object model: this way, we have full control over the concepts in our model, which makes integration and verification more feasible. We have defined the formal semantics of SCOM in terms of set theory.
There are two levels in SCOM. The first one is the level of the so-called *simple objects*. This level is meant for global state space modelling, without paying attention to things like *distribution*, *location* and *timing*. A simple object has a unique *identity*, *relationships* with (other) simple objects and some *attributes*. Simple objects are classified into *simple types*; when the state space of a system has to be described, a *simple schema* is introduced to define the existing simple types and their properties. Moreover, *inheritance* between simple object types is supported.
The second level is that of the *container objects*. This level extends the simple level by describing location, distribution and timing aspects. Moreover, it serves as "glue" for making the integration possible. A container (object) has a unique *identity*, it contains a *set of simple objects* (not necessarily of the same type) and it has a *location* and a *time-stamp*. Again, container objects are classified into *container types*; the notion of a simple schema is extended to that of an *object schema*: besides defining the existing simple types, an object schema also defines which container types exist. On both levels, *constraints* can be expressed.

2.2 The SimCon Algebra (SCA)

It is obvious that it is not sufficient to develop an object model without some means to *manipulate* objects: we need some language for this purpose. It is preferable that the concepts in this manipulation language are on the same level of abstraction as the object model. The object model describes the *structure* of the objects, whereas the manipulation language serves for expressing *creation, retrieval, update* and *deletion* of objects. Both formalisms are equally important for modelling static and dynamic aspects of a system. It would also have been possible to define the manipulation language at the mathematical level that represents the formal semantics of the object model. However, in that case, it would only be understandable for experts with full knowledge of this formalisation. Although our approach is not intended to be used by laymen who have no background in (formal) computer science, this is highly undesirable: if someone who wants to apply our approach, would have to study and comprehend the *complete* formalisation *in advance* and he would have to express some system properties at this level of abstraction, it would not be very clear what is gained by applying the SimCon approach.

To meet these demands, we have introduced the SimCon Algebra (SCA). The SimCon Algebra consists of some basic operations resembling graph manipulations. As has been demonstrated [Verkoulen, 1993], these operations give sufficient expressive power. In particular, we have proven that SCA is at least as expressive as the Nested Relational Algebra (NRA) (and thus also as the flat Relational Algebra) and GOOD [Gyssens, Paredaens & Van Gucht, 1990]. We even described an operation for which it has been proven that it cannot be expressed in GOOD, but which is expressible in SCA.
In order to increase the expressive comfort of the algebra, we have proposed a graphical version, called SCA_G. We also have defined some high-level operations in terms of the aforementioned basic operations. As these high-level constructs have been defined in terms of primitive ones,

they do not have to be considered in our theoretical contemplations. We may develop *libraries* of dedicated high-level operations that can be used when modelling special kinds of systems. It has been recognised before that this may support the designer of a system well.

2.3 The SimCon Net Model (SCNET)

Actually, the SCNET model is the core of our integrated model. An SCNET is a special kind of Petri net. Below, we describe why we have chosen Petri nets as the basis for modelling dynamic aspects of a system. First, the most important concepts from Petri net theory are mentioned.

Basically, a Petri net is a bipartite graph consisting of *transitions* and *places*. Transitions are active components, that communicate with each other by exchanging so-called *tokens*. The places connect the transitions. They contain the tokens. When a transition *occurs* or *fires*, it consumes tokens from each of its input places and produces tokens for its output places. In classical Petri net models these tokens do not have a value: the only information they represent is their presence. In high-level Petri net models like Coloured Petri Nets (CPN) [Jensen, 1992] and ExSpect, these tokens have a value. In the tools that support these high-level net models, these values are specified using the type system of a functional language, while the functionality of the transitions is specified by functions from that language. However, when developing large and data-intensive applications, this way of defining the state space of the net is not always satisfactory. The main reason for this is that it may be difficult to acquire an overview of the data aspects of the specification. In the SCNET model, the state space of the net is defined in terms of the SimCon Object Model: tokens become SCOM (container) objects. To each place, an SCOM container type is assigned, defining which kind of objects can be contained by that place. Moreover, the functionality of a transition is defined by an SCA expression. This has proven to be a natural and dexterous way of working.
We use high-level Petri nets as a basis for describing the dynamic aspects of a system for a number of reasons. We want to mention the most important ones. High-level Petri nets are inherently simple, yet they give sufficient modelling power and comfort, especially because of their graphical nature. Moreover, they have a firm mathematical foundation which is necessary to obtain a formal semantics and which is a prerequisite for the development of new theories for proving properties of such nets. At the moment, many theoretical results have been accomplished. Last but not least, advanced computer tools have been designed for high-level Petri nets. Two important ones are Design/CPN [Jensen, 1992] and the ExSpect tool [Van den Broek & Verkoulen, 1992].

2.4 Automatic Verification

In SCOM, it is possible to define constraints. When an SCOM schema is used to model the state space of an SCNET, we want to know whether this net satisfies the constraints that have been formulated upon the schema. Again, it would be possible to let the designer perform this verification process at the level of the formal semantics of our models. However, this is cumbersome. We want to support the designer in this verification process. We would prefer a means of *automatic verification*: in that case, the designer only has to *formulate* the constraints (which may be difficult enough). The automatic verifier then takes care of the verification. Such a method for automatic verification has been described for a certain class of constraints, the so-called *compositional* constraints [Verkoulen, 1993]. We mention the main idea here. In the verification process, the state space of the SCNET under consideration is divided into a *finite* number of equivalence classes: in order to check whether the net will violate a constraint, it is checked whether each of the transitions in the net leaves the constraint invariant. In this way, the number of states that have to be checked is reduced drastically because we do not consider all possible interleavings of transition firings. This is a method that has been recognised in literature before. However, the number of states that have to be checked this way is still very large (often even infinite). Therefore, the number of relevant states is reduced further, by looking only at a set of characteristic states, called *prime states*. We have proven that checking the prime states is necessary and sufficient to prove (or disprove) the invariance of a compositional constraint. This set of prime states is

relatively small. In particular, it is small enough to do the checking of the constraints by means of a tool, in a reasonable amount of time.

2.5 Tool Support

The tool that supports the SimCon approach is based upon the ExSpect tool. The ExSpect tool has been extended with an editor for the SimCon Object Model [Van den Broek & Verkoulen, 1992].

The tool supports *hierarchical* Petri nets as is it possible to define *subnets*[4]. This subnet concept has also been introduced in the context of Coloured Petri Nets; there, a subnet has been called a *page* [Jensen, 1992]. A subnet consists of transitions, places and subnets. This construct allows for top-down development and hiding of implementation details. Of course, bottom-up development of a specification is also possible.

3 A Small Example

In this section, we present a small example which illustrates our approach. In Figure 1, a graphical representation of a SimCon model that partly specifies a transport company, is given. We suppose that the company owns a number of trucks and employs some drivers. It carries bulked goods. In order to keep the example simple, we assume that trucks may be either filled completely or totally empty[5]. Upon receiving an order from a client, the company schedules a trip, consisting of a truck and a driver who has a licence for that truck. The empty truck is loaded at the storehouse and then the trip is executed. After a while, the truck (that is now empty again) returns at the company: the empty truck is put into the garage and the driver goes into the waiting-room.

Figure 1 gives a graphical representation of a SimCon model that describes this situation. In the data part, we see that there are three simple types `driver`, `truck` and `order`, respectively with attributes `name`, `capacity` and `quantity`. Moreover, a `driver` has a `licence_for` a number of `trucks` and an `order` is `carried_by` a `truck`. Note that we do not make a difference here between the references `licence_for` and `carried_by`. However, there is a difference: in general the licence information of drivers is more or less static (persistent), whereas the data about which order is carried by which truck is liable to continuous fluctuations. In a more extensive example, this may be expressed by modelling the storage and possible changes of the licence information explicitly, instead of simply assuming that each driver knows for which kinds of trucks he has a licence.

In the net part, we see places and transitions. The places may hold container objects, which are clusters of simple objects. Some of them are singletons, like the trucks in the garage. Others contain more than one simple object, like the trips in `manned_truck`: a `trip` consists of a `driver` and a `truck`. The behaviour of the transitions is defined in terms of the SimCon Algebra: the functionality of a transition is described by an SCA expression. Moreover, SCA is used for formulating preconditions; a transition may only fire when its precondition is satisfied. In the example, the transition `TripPlanning` may only fire when it can "consume" a `truck` from the garage with sufficient `capacity` for the `order` and a `driver` from the `waiting-room` that has a `licence for` that `truck`. This is formalised by the following precondition[6]:

$$_{\texttt{order}}[\texttt{quantity}] \leq {}_{\texttt{truck}}[\texttt{capacity}] \wedge {}_{\texttt{truck}}[\,] \subseteq {}_{\texttt{truck}}[\overset{\texttt{licence\_for}}{\leftarrow}{}_{\texttt{driver}}\,[\,]]$$

The expressions that define the transitions are trivial. The interested reader should consult [Verkoulen, 1993]. However, in real practical applications, one will use the graphical version of the algebra and/or a library of high-level operations.

[4] In ExSpect terminology, a subnet is called a *system*.

[5] This assumption is not as artificial as it may look: for certain types of trucks, this constraint really holds, as a half-full truck might topple.

[6] We cannot explain the complete syntax of SCA here, but this expression may give you an impression.

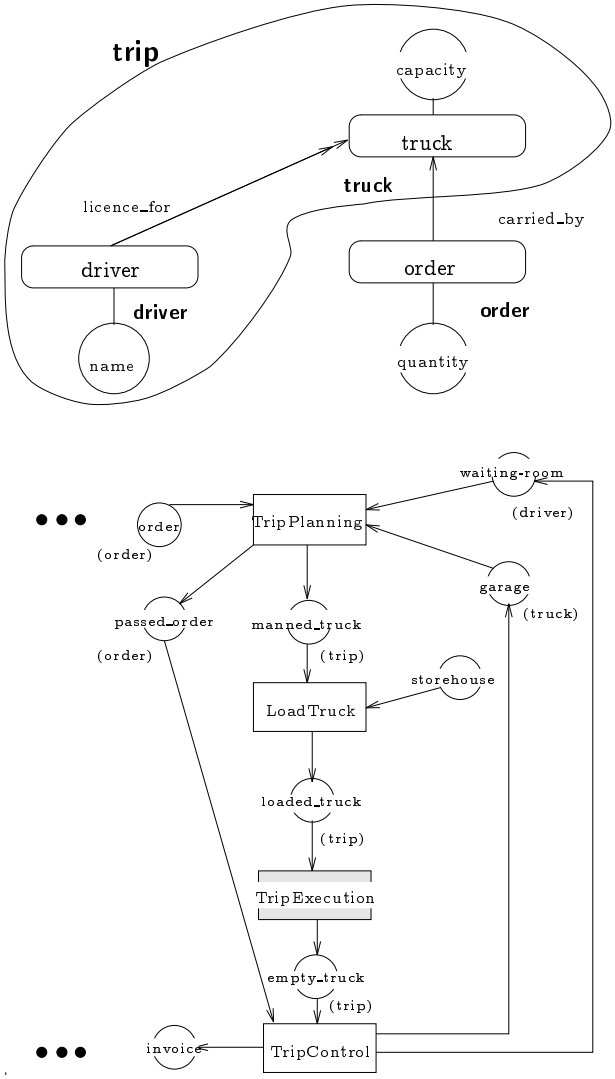

Figure 1: The Trip Example

Bibliography

Aalst, W.M.P. van der, M. Voorhoeve, and A.W. Waltmans [1989], The TASTE project, *Proceedings of the 10th International Conference on Application and Theory of Petri Nets*, Bonn, Germany, 371–372.

Atkinson, M., F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik [1989], The Object-Oriented Database System Manifesto, *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 40–57.

Broek, E.M.M.A. van den, and P.A.C. Verkoulen [1992], A Tool for Integrated Modelling of Static and Dynamic Aspects of Systems, in: K. Lyytinen and V.-P. Tahvanainen (eds.), *Next Generation CASE Tools*, IOS Press, 75–98.

Gyssens, M., J. Paredaens, and D. van Gucht [1990], A Graph-Oriented Object Database Model, *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, 417–424.

Hee, K.M. van, P.M.P. Rambags, and P.A.C. Verkoulen [1993], Specification and Simulation with ExSpect, in: P.E. Lauer (ed.), *Functional Programming, Concurrency, Simulation and Automated Reasoning*, Lecture Notes in Computer Science 693, Springer-Verlag, 296–328.

Hee, K.M. van, and P.A.C. Verkoulen [1991], Integration of a Data Model and High-Level Petri Nets, *Proceedings of the 12th International Conference on Application and Theory of Petri Nets*, Gjern, Denmark, 410–431.

Houben, G.J., and P.A.C. Verkoulen [1991], An Integrated Approach to Modelling Structural and Behavioural Aspects of Complex Objects, in: J. Göers, A. Heuer, and G. Saake (eds.), *Third International Workshop on Foundations of Models and Languages for Data and Objects*, Informatik Bericht 91/3, Aigen, Austria, Technische Universität Clausthal, 47–64.

Jensen, K. [1992], *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, EATCS Monographs on Theoretical Computer Science. Springer-Verlag.

Lyytinen, K. [1987], Different Perspectives on Information Systems: Problems and Solutions, *ACM Computing Surveys* **19**, 5–46.

Ramackers, G.J., and P.A.C. Verkoulen [1992], A Formally Integrated Conceptual Model based on Objects, *Third International Conference on Dynamic Modelling of Information Systems*.

Verkoulen, P.A.C. [1993], *Integrated Information Systems Design – An Approach Based on Object-Oriented Concepts and Petri Nets*, Ph.D. thesis, Eindhoven University of Technology.