

DYNAMIC PROCESS MODELLING THROUGH MULTI-LEVEL RBNs

*Fourth International Working Conference on Dynamic Modelling and Information Systems,
Noordwijkerhout, Netherlands, September 1994*

Aphrodite TSALGATIDOU, Dimitris GOUSCOS and Constantin HALATSIS

Department of Informatics

University of Athens

TYP A Buildings, Panepistimiopolis, Ilisia 157 71 Athens, GREECE

Phone: ++30-1-7230172,7291885,7217941

Fax: ++30-1-7219561

E-mail: {afrodite,gouscos,halatsis}@uranus.di.uoa.ariadne-t.gr

Keywords: dynamic modelling, Petri nets, control flow, requirements engineering.

Abstract

This paper presents an approach for specifying and validating the dynamic aspects of IS at multiple levels of abstraction, during the requirements specification phase of system development, using a Petri net based model, called Rule-Based Net (RBN). The produced specifications explicitly model the dynamic system behaviour, they are object-oriented, rule-based, executable and can be validated using graphical animation. In this way, we come up with concise, compact, modular, validated and easily modifiable requirements specifications at the very early stages of system development. Furthermore, these specifications can be easily mapped to design and implementation structures, resulting in a system which satisfies user's needs and is flexible enough to incorporate future changes of behaviour.

1. Introduction

Whereas information systems are inherently dynamic, traditional approaches towards Information Systems (IS) development focus on the modelling of static aspects and are not capable of specifying the dynamic properties of these systems. The lack of methodologies and associated tools for modelling the dynamic aspects of information systems has been realised by a growing number of researchers and a few approaches to dynamic modelling of

information systems have recently emerged (DYNMOD-I 1990; DYNMOD-II 1991; DYNMOD-III 1992). Along these lines, this paper presents an approach for specifying the *dynamic aspects* of information systems at *multiple levels of abstraction* using a formal Petri-net (Murata 1989) based model, called Rule-Based Net (RBN). As the last abstraction level of the RBN model has been described elsewhere in detail (Tsalgaidou et al. 1993; Tsalgaidou et al. 1994), a more simplified view of the model is presented and the main focus lies on the various abstraction levels of the RBN model and on the mapping from one level to another.

RBN graphical specifications are developed in an object-oriented rule-based framework. These specifications are executable and they model explicitly *control flow* within an organisation; as far as validation is concerned, the graphical nature of the RBN formalism allows for graphical *animation* techniques, whereas the theoretical foundation of the model provides a basis for formal validation.

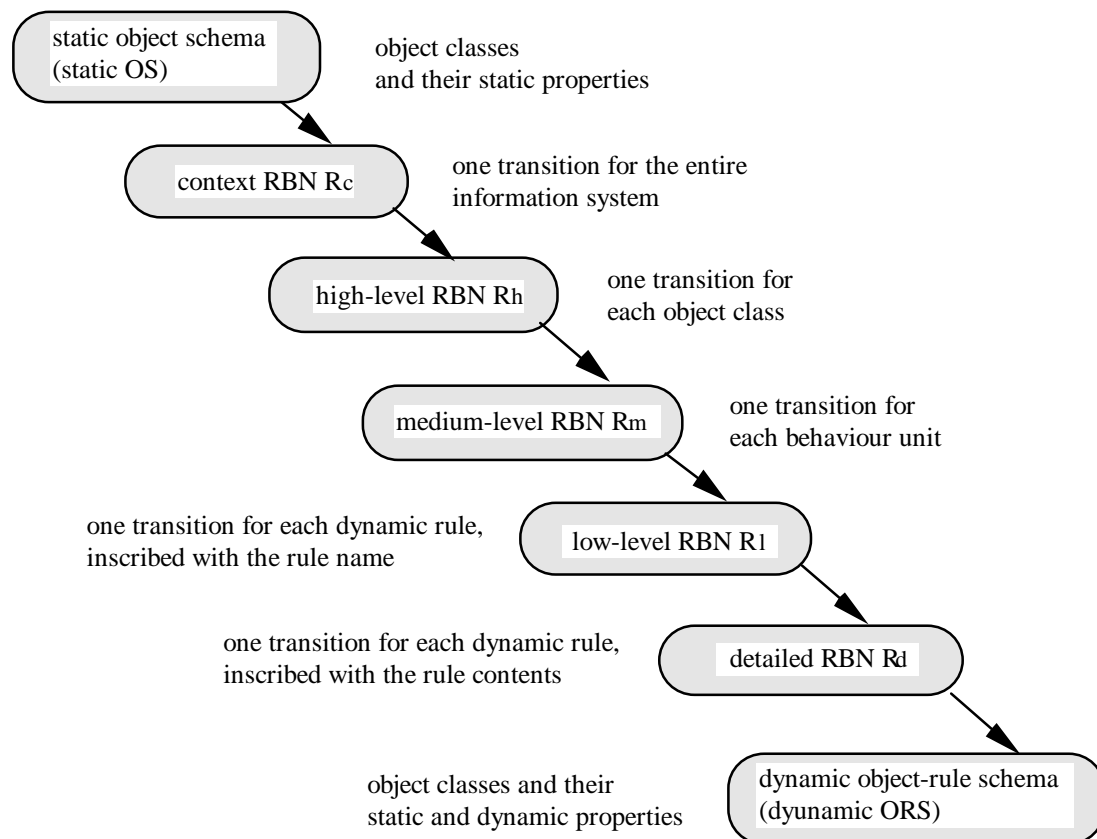


Figure 1. The sequence of models developed during building dynamic object-oriented rule-based requirements specifications in a top-down fashion

More specifically, the objective of our approach is to explicitly model and validate the dynamic behaviour of an information system at the early stages of system development. This is accomplished by first constructing an object-rule schema, which contains only the static aspects of the application at hand, and subsequently enhancing the schema with dynamic behaviour. The latter enhancement takes place according to a stepwise, top-down

approach, and an intermediate RBN model is produced in each enhancement step. Fig. 1 depicts the sequence of models employed during this process.

The proposed approach is described in the next section, and the various schemas employed (static Object Schema, dynamic Object-Rule Schema, context, high-level, medium-level, low-level and detailed RBNs) are briefly defined in the following. Section 5 discusses RBN validation and the last section concludes the paper.

2. Modelling the Dynamic Behaviour of an Information System

The dynamic behaviour of an information system is captured in a dynamic Object-oriented Rule-based Schema (dynamic *ORS*) in terms of behaviour units (*BUs*) which are defined per object class. Each *BU* has a certain behaviour described in terms of dynamic *rules*. Each *rule* has a *THEN*-part which contains various actions and, optionally, an *IF*-part containing some preconditions.

The behaviour of each *BU* is triggered by a signal of a specific *signal type*. Signals are produced either by the external environment or by the execution of the actions of a dynamic rule. When a *signal* of a specific type is created (either by the external environment or by the execution of the actions of a rule) it is directed to the appropriate *BU* and triggers its behaviour (which is described in terms of dynamic rules). This means that the rules of that *BU* which have their *IF*-part satisfied (or rules with no *IF*-part) will execute the actions described in their *THEN*-part. The execution of these actions may generate other signals which are sent either to the external environment or to another *BU* of the same or of another object class of the dynamic *ORS*. In this way, signals constitute the sole means of communication between the various parts of the information system under study and between the information system and its external environment. An example of an *ORS* may be seen in fig. 2.

The procedure that has to be followed in order to construct a dynamic *ORS* is the following: First a static object schema (static *OS*) is constructed. The static *OS* is a set of object classes of the application at hand, and contains only their static class and instance properties. Details about the construction of the static *OS* may be found in (Tsalgatiidou et al. 1994). Having constructed a static *OS*, the next step is to develop a context RBN R_C . R_C is a Petri net that contains one transition (called \bullet INFORMATION SYSTEM Σ) standing for the whole information system under study and other transitions which model the external environment. Places of R_C model signals exchanged between the information system and its external environment, see fig. 3(a).

At the following step, the \bullet INFORMATION SYSTEM Σ transition of R_C is decomposed into a number of transitions corresponding to the application object classes (that have already been defined in the static *OS*) and into a number of places which model the signals exchanged between object classes and are inscribed with the signal names. Transitions and places of R_C which model the interaction of the system with the external environment remain the same, unless it is realised, during this decomposition, that the signals exchanged

between the system and its external environment need to be modified. The RBN produced at this stage is a high-level RBN R_h ; fig. 3(b) provides an example of such an RBN.

At the next abstraction level, a medium-level RBN R_m is produced by decomposing each object class-transition of R_h into a number of transitions modelling the behaviour units of the corresponding object class and a number of places modelling the signals exchanged between these behaviour units. Fig. 3(c) presents the medium-level RBN which resulted from the decomposition of the high-level net of fig. 3(b).

From a medium-level RBN R_m , a low-level RBN R_l may be produced which depicts the dynamic rules that describe the behaviour of each behaviour unit. Thus, each BU-transition of R_m is decomposed into a number of transitions corresponding to the dynamic rules of this BU. At this stage, transitions are inscribed only with the rule names (see fig 3(d)).

A detailed RBN R_d is produced at the lowest abstraction level. Each transition of R_d is inscribed with the contents, i.e. the *IF*- and *THEN*- parts of the represented rule, the BU and the object class where the rule is defined, as well as the object classes that inherit this rule (more details may be found in (Tsalgatidou et al. 1993)).

```
object class CUSTOMER_ORDER subclass of ORDER
instance properties
  name ArrivalTime domain TIME
  name Priority domain PRIORITY
  name Customer domain CUSTOMER
  name Invoice domain 0:1 INVOICE
end instance properties
class behaviour
  class BU CUSTOMER_ORDER.ProcessOrdersOnHold
    triggered by ProcessOrdersOnHold(Product)
  rule R1
    execute NotifyOrders(signal.Product,OnHold) ;
  end BU CUSTOMER_ORDER.ProcessOrdersOnHold
end class behaviour
instance behaviour
  instance BU CUSTOMER_ORDER.ProcessNow
    triggered by ProcessNow( )
  rule R1
    if self.Quantity > self.Product.StockQuantity
    then execute SetPriority(OnHold) ;
         NoShipment(self.OrderNo) -> EXT ENV ;
  rule R2
    if self.Quantity <= self.Product.StockQuantity
    then execute SetPriority(Processed) ;
         IssueInvoice(self.ObjectId) -> INVOICE ;
         DecreaseStock(self.Quantity) -> self.Product ;
  end BU CUSTOMER_ORDER.ProcessNow
end instance behaviour
end object class CUSTOMER_ORDER
```

Figure 2. Part of a dynamic ORS

In summary, the specification of the dynamic behaviour of an information system starts with development of a static *OS* from which a context RBN R_C is constructed. Subsequent refinements of R_C in lower abstraction levels result in a detailed RBN R_d which contains all information necessary for modelling in a graphical manner, and within an object-oriented rule-based framework, the dynamic behaviour of the system under study. During RBN refinements, the static *OS* is also gradually enhanced with the dynamic behaviour of each object class, thus resulting at a complete dynamic *ORS* when the RBN model has been refined down to the lowest abstraction level.

All the aforementioned models are formally defined in the following section, while fig. 1 depicts the sequence in which these models are created. It is worth noting that, although the refinement process outlined above proceeds in a top-down fashion, a bottom-up approach is also possible, since the mappings from one model to another are bidirectional.

3. The static Object Schema and the dynamic Object-Rule Schema

3.1. The static Object Schema (static OS)

Modelling of an information system starts by construction of a static Object Schema (static *OS*) which consists of a set of object classes ($classes(OS) = \{C_1, C_2, \dots\}$) and their static class and instance properties. In a static *OS* there is a unique class OBJECT, and class C_i has a set of subclasses $\{C_{i1}, C_{i2}, \dots\} = sub(C_i)$. A static *OS* supports single inheritance, so that each class C_i other than OBJECT has a unique superclass and it inherits all the properties defined in its unique superclass as well as in all the upper superclasses up to class OBJECT. More details about *OS* and its construction may be found in (Tsalgaidou et al. 1994).

3.2. The dynamic Object-Rule Schema (dynamic ORS)

A dynamic Object-Rule Schema (dynamic ORS) is an object-oriented schema which consists of a set $classes(ORS) = \{C_1, C_2, \dots\}$ of object classes and a set $sig(ORS) = \{S_1, S_2, \dots\}$ of signal types. Each object class has a set of class and instance properties and a set of class and instance behaviour units; the latter describe the behaviour of the class and its instances in terms of rules. Each behaviour unit B consists of a set of rules $rules(B) = \{R_1, R_2, \dots\}$ with preconditions and actions and is triggered by a signal of a specific type S ($when(B) = S$); each rule R_i has certain preconditions and actions and, at the time of triggering, the rules of B which have their preconditions satisfied are fired and execute their actions.

The preconditions $if(R_1)$ of a rule R_1 of a behaviour unit BU_{ij} may refer to the properties of the class C_i where BU_{ij} is defined ($def(BU_{ij}) = C_i$) and to the parameters of the signal type S which triggers that behaviour ($when(BU_{ij}) = S$). The actions $then(R_1)$ of R_1 may also refer to the parameters of the triggering signal type S and the properties of class C_i and, furthermore, they may also modify the values of the properties of class C_i and produce signals of specific types which trigger other behaviour units or which are sent to the external environment. In case BU_{ij} is a class behaviour unit, actions $then(R_1)$ may create or delete instances of C_i . An example of an object-rule schema may be found in fig. 2.

There are two possible cases for the flow of signals of a signal type S : either (a) they are produced by the external environment X and directed to an object class or (b) they are produced by (the execution of the actions of a dynamic rule of) an object class and directed either to the external environment or to a behaviour unit of the same or of another object class. Therefore, the producers $prod(S)$ as well as the consumers $cons(S)$ of a signal type S constitute two sets made up of the external environment X and/or of object classes of the dynamic ORS. For any signal type S , $prod(S)$ and $cons(S)$ are not necessarily disjoint, but there is no signal type S such that $prod(S) = cons(S) = \{X\}$, i.e. there is no signal which is both produced and consumed solely by the external environment.

As already mentioned, each class C_i of a dynamic ORS has a set of subclasses which inherit all the static features of C_i and of its superclasses, and may also define some new ones; the dynamic features of a class, i.e. its class and instance behaviour units and rules, may also be inherited as they are or redefined accordingly (Tsalgatidou et al. 1993), and new dynamic features can be defined as well.

4. The Rule-Based Net (RBN) models

The RBN models are based on Petri nets (Murata 1989) and more specifically on Predicate-Transition nets (PrT-nets) (Genrich and Lautenbach 1981). Petri nets have been used by many researchers for modelling the dynamics of information systems; see for example the work by Conrath et al (Conrath et al. 1991), where Petri nets are used for specifying the work flow of an office automation system or the work by Ang et al. (Ang et al. 1991) where Petri nets are used as an office process specification formalism. Guha et al. (Guha et al. 1991) employ generalised stochastic Petri nets (GSPNs) to model system performance.

In the proposed approach, Petri nets are used within an object-oriented framework as the underlying formalism for modelling the dynamic aspects of an information system in terms of rules. The various RBN models which were mentioned in section 2 and lie at different levels of abstraction are defined in the rest of this section. All these models can be formally mapped to each other (Gouscos 1994), and the mapping process can proceed either top-down towards more detailed models, or bottom-up towards more abstract models. At all levels of abstraction defined below places represent signal types, tokens present in a place model signals of the corresponding signal type which are exchanged within the various parts of the information system and between the information system and its external environment, and each specific marking corresponds to a state of the modelled system.

4.1. Context RBNs

A context RBN R_C is defined as a tuple

$$R_C = \langle P_C, T_C, F_C, O_C, sig_C \rangle$$

where P_C is a set of places, T_C is a set of transitions, F_C is a set of arcs connecting places to transitions and transitions to places, O_C is the underlying structure, (which has resulted from the static OS with the addition of some signals from and to the external environment) and sig_C is a function that maps places to signal types. T_C contains one transition which corresponds to the information system under study and is conventionally named • INFORMATION SYSTEMΣ and other transitions X_i which model the external environment that produces (consumes) various signals sent to (coming from) the information system. Each place p of P_C is inscribed with a signal type $sig(p)$ and the parameters of this signal type. Thus, places in a context RBN are inscribed with the signals consumed or produced by the external environment (see fig. 3(a)).

4.2. High-level RBNs

A high-level RBN R_h is defined as a tuple

$$R_h = \langle P_h, T_h, F_h, O_h, sig_h, class_h \rangle.$$

where $class_h$ is a function mapping transitions to object classes. O_h is the underlying structure of R_h and it has resulted from the corresponding O_C of the context RBN R_C with the addition of the signals exchanged between the various object classes. P_h , T_h , F_h and sig_h are similar to the corresponding elements of the context RBN,

but have different values. Here, the unique • INFORMATION SYSTEMΣ transition of R_C has been decomposed into a number of transitions (corresponding to and inscribed with the object classes of the underlying structure) and into a number of places representing and inscribed with the signals exchanged between these object classes (see fig. 3(b)).

4.3. Medium-level RBNs

A medium-level RBN R_m (see fig. 3(c)) lies at one level of abstraction lower than the corresponding high-level RBN and is defined as a tuple

$$R_m = \langle P_m, T_m, F_m, O_m, sig_m, bu_m \rangle$$

where bu_m is a function mapping transitions to behaviour units. O_m is the underlying structure of R_m and it has resulted from the corresponding O_h of R_h with the addition of the behaviour units of each object class. P_m , T_m , F_m , and sig_m are similar to the respective elements of the corresponding high-level RBN, but have different values: here, each object class-transition of the high-level RBN is decomposed into a number of behaviour unit-transitions and into a number of places corresponding to signals exchanged between behaviour units of the same object class.

4.4. Low-level RBNs

A low-level RBN R_l lies at one level of abstraction lower than R_m . More specifically, each behaviour unit-transition of R_m is here decomposed into a number of transitions corresponding to the rules of that behaviour unit. Each transition t in a low-level RBN is inscribed only with the name of the corresponding rule ($rule(t)$). A low-level RBN R_l is formally defined as a tuple

$$R_l = \langle P_l, T_l, F_l, O_l, N_l, sig_l, rule_l \rangle$$

where O_l is the underlying structure of R_l and has resulted from the structure O_m of R_m with the addition of the rules of each behaviour unit. $rule_l$ is a function that maps each transition t of R_l to a rule of the underlying structure O_l . P_l , T_l , F_l , and sig_l are similar to the corresponding elements of the medium-level RBN, but have different values. N_l is a multiplicity function of arcs, and each arc f of F_l is inscribed with its multiplicity $N_l(f)$, where $N_l(f) = 1$ or $N_l(f) = u$, $u \geq 1$. u represents an a multiplicity which is undefined at modelling time but takes a certain value at run-time (see fig 3(d)). It should be noted that this notational convention does not violate standard Petri net models because, for any net with arcs of undefined multiplicity, one can always construct a second net with arcs of multiplicity 1 such that for any given initial marking M_0 , the reachability trees of both nets have the same terminal markings (projected to the places of the first net).

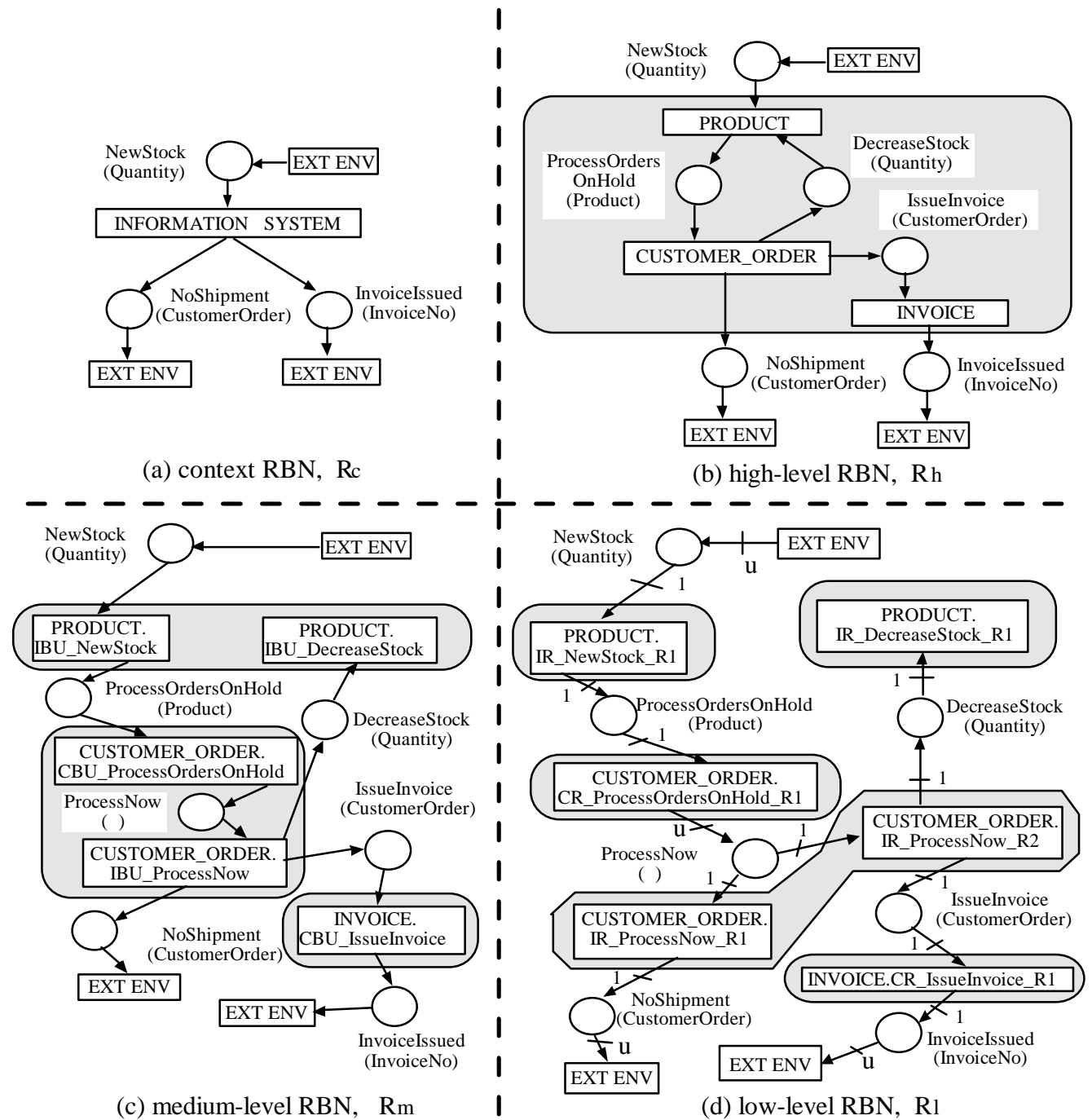


Figure 3. RBNs at various abstraction levels

4.5. Detailed RBNs

A detailed RBN R_d is defined as a tuple

$$R_d = \langle P_d, T_d, F_d, O_d, N_d, sig_d, class_d, bu_d, if_d, then_d \rangle,$$

where P_d is a set of places, T_d is a set of transitions, F_d is a set of arcs connecting places to transitions and transitions to places, N_d is a multiplicity function of arcs, O_d is the underlying structure of R_d (i.e the full-detail dynamic *ORS* which resulted from O_l with the addition of the contents of the various rules), sig_d is a function that maps places (i.e. elements of P_d) to signal types, $class_d$ is a function which maps transitions to object classes, bu_d is a function mapping transitions to behaviour units, if_d is a function mapping transitions to preconditions and $then_d$ is a function which maps transitions to actions.

In R_d every transition t corresponds to a dynamic rule of the underlying structure and is inscribed with the corresponding preconditions, actions, behaviour unit and class, defined respectively as $if_d(t)$, $then_d(t)$, $bu_d(t)$ and $class_d(t)$. Each place p of P_d is inscribed with a corresponding signal type $sig_d(p)$ and the parameters of $sig_d(p)$. Fig. 4 depicts an example of a detailed RBN R_d which resulted from the gradual decomposition of the context RBN R_c of fig. 3(a); a very small part of its underlying structure is depicted in fig. 2.

5. Validation of RBN specifications

Validation of requirements specifications is a hard process which refers to the production of specifications that are complete and consistent, and include no ambiguous or vague points. One way to validate the dynamic requirements specifications constructed by means of the methodology outlined above is graphical animation and execution of the RBNs models, which can be performed at any level of abstraction, from context down to detailed RBNs.

Graphical animation is a valuable tool for reducing errors during requirements specification and for ensuring that the intended system behaviour has been properly captured and modelled. The proposed approach is supported by a set of tools which enable stepwise construction and graphical animation of the RBN models, and the latter can proceed either (a) in a user-driven fashion, where by "user" we mean a systems analyst working together with end-users of the application or (b) automatically. In the first case, the user decides which rules will fire and which will not, and comments on the obtained results, whereas in the second case, and for some given initial marking, the system evaluates all alternative paths (i.e. the preconditions of all triggered rules), executes the corresponding actions and produces various alternative markings. Therefore, it can be checked if and under what conditions certain RBN markings (and, consequently, certain system states) are reachable. Furthermore, missing rules, dead-end rules (i.e. rules which can never fire), wrong rules (i.e. rules that produce wrong results) and circular rules can also be identified. User feedback can be used to correct any errors discovered, thus helping to design the rule base interactively.

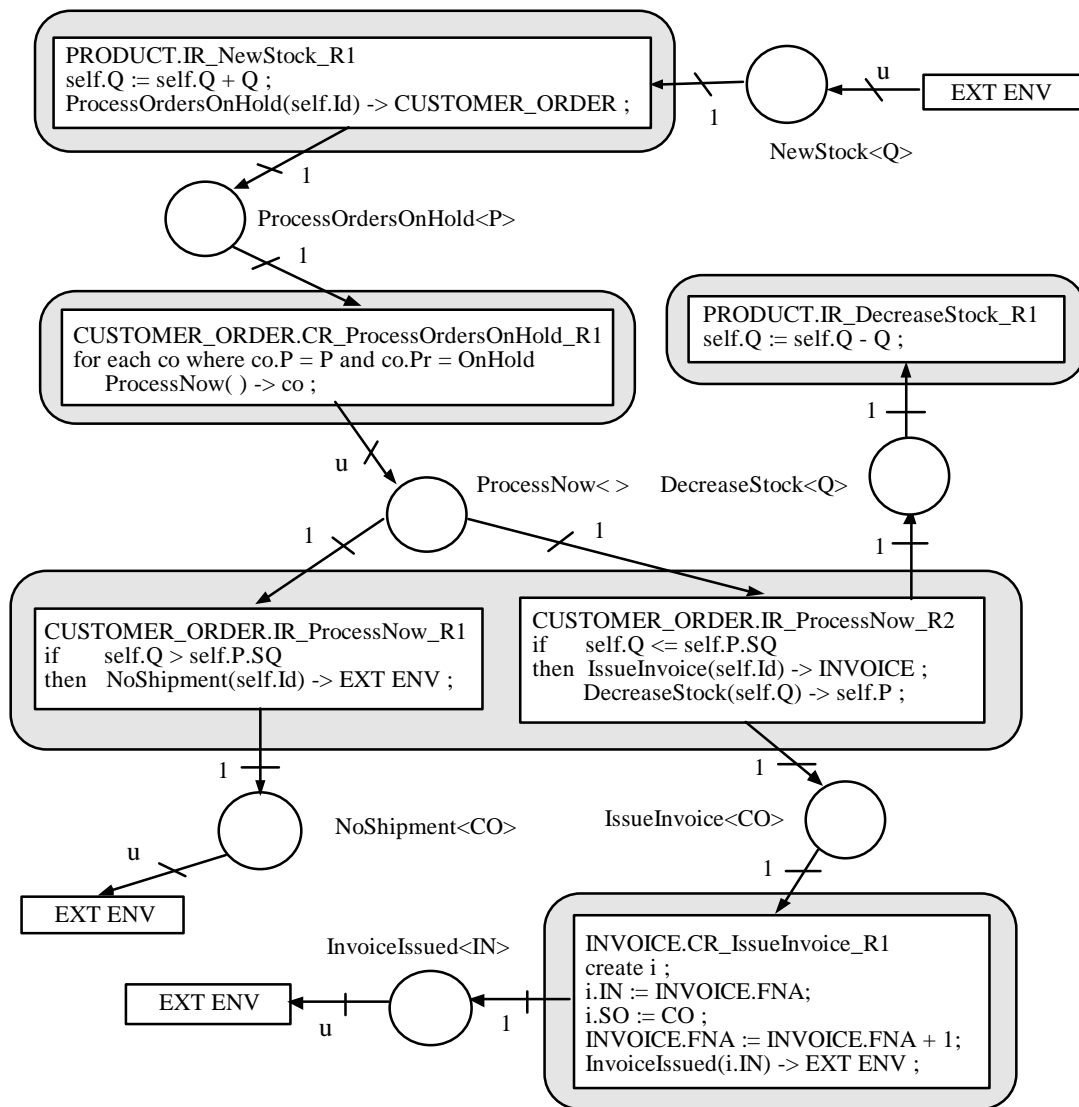


Figure 4. Detailed RBN R_d resulting from the decomposition of the low-level RBN of fig. 3(d)

It should be noted that, although animation can be carried out at all abstraction levels of RBNs, user-driven animation makes sense only at the lowest abstraction level of detailed RBNs, where the user can easily see the contents of each rule and decide which rules to fire. At all other RBN levels animation is carried out automatically and, for a given initial marking, the user may only observe communication between rules, BUs, object classes and between the system and its external environment through signals.

However, although animation provides remarkable assistance in the case of small systems, larger applications necessitate tools which can automatically detect contradictions, inconsistencies and redundancies in the model. The formal foundation of RBNs offers a sound basis for the development of such tools. More specifically, the successive mappings from context to detailed RBNs are performed by means of various transformations,

formally defined in (Gouscos 1994), that decompose transitions and add some new places in each level, apart from enriching the structure of the resulting RBN with new features. The structural transformations applied during the top-down mapping towards detailed RBNs are in close analogy to the Fusion Parallel Transitions (FPT) and Fusion Series Transitions (FST) defined in (Murata 1989) which, together with some other transformations, have been shown to preserve the properties of liveness, safeness and boundedness of Petri nets. This is an important feature of our mapping, since some of these properties are desirable for RBNs, as well.

It is easy to see that usage of specialised forms of Petri nets facilitates the development of criteria for checking desirable properties. Of all the behavioural (marking-dependent) and structural (marking-independent) Petri net properties found in the literature, only a subset is of interest for RBN validation; this is due to the specific interpretation under which RBNs are used to model the dynamics of information systems. Under this interpretation, RBN places model signal types, RBN tokens model control signals, RBN markings model information system states, RBN transitions model the system dynamics and RBN firings model specific behaviour; therefore, the following properties are desirable for an RBN:

1. *Reachability*: if the information system must reach a state S corresponding to a marking M , starting from a state S_0 corresponding to a marking M_0 , then M must be reachable from M_0 .

2. *Liveness*: if the information system must perform some behaviour b corresponding to a transition t , starting from a state S_0 corresponding to a marking M_0 , and this must be repeated for a finite number of times, then transition t must be L2-live under marking M_0 .

3. *Persistence*: if the information system must perform some behaviour b_1 corresponding to a transition t_1 and some behaviour b_2 corresponding to a transition t_2 independently of one another and starting from a state S_0 corresponding to a marking M_0 , then transitions t_1 and t_2 must be in a persistence relation under marking M_0 .

4. *Structural liveness*: when there are signals available for all signal types which the information system receives from the external environment, the information system must be able to perform any specified behaviour b , depending of course on values of the signal parameters and of the object instances available in the underlying structure. Therefore, under an initial marking M_0 corresponding to the state described before, all transitions must be live; this means that the net must be structurally live.

All these properties can be checked by constructing the reachability tree of the RBN under analysis. Furthermore, specific criteria might be formulated for RBNs, taking into account the particular form of these nets. An algebraic approach may be alternatively employed by working with the incidence matrix of the net under analysis.

6. Conclusions

This paper presents an approach for explicitly modelling the dynamic aspects of information systems in terms of dynamic rules within an object-oriented framework. The construction of requirements specifications starts from a static Object Schema which is included as one constituent in an abstract RBN and then gradually refined together with the overall RBN to lower levels of abstraction, in order to include the dynamic aspects too. The outcome of this process is a detailed RBN, whose underlying structure is a full-detail dynamic Object-Rule Schema containing all static and dynamic features of the modelled information system.

Dynamic features are explicitly represented in terms of rules which are grouped in behaviour units; behaviour units are in turn grouped under object classes, and all the rules of each behaviour unit are triggered by signals of a specific signal type. Rules, behaviour units and object classes inscribe the transitions of the various RBN models. RBN places represent signals exchanged between the information system and its external environment and within parts of the information system itself. Thus, RBNs represent control, rather than data flow within a system, as is the case with many traditional approaches to information systems development. Furthermore, RBN specifications can be validated either by formal validation techniques, or by graphical animation offering the advantages of rapid prototyping.

In conclusion, the strength of the approach proposed in this paper lies in the explicit modelling of business policy at various levels of abstraction, in the construction of graphical and executable specifications and in the employment of graphical animation and formal techniques for validation purposes. Furthermore, since the behaviour of each object class is localised in a unique transition of a 'high-level' RBN that is subsequently decomposed into a number of transitions and places corresponding to the BUs, rules and signals of that object class, RBN specifications may be considered compliant to the so-called localisation principle (Rolland 1992) which recommends that each object should be described in isolation from the others.

The proposed approach is supported by a set of tools which constitute the VENUS environment (Tsalgaidou et al. 1994). VENUS is being developed on Sun SPARC workstations running BSD UNIX, X11R5 and OSF/Motif, in ANSI C++ and Prolog. This environment is designed to provide syntax-directed editors, graphical editors, animators and mapping tools for building entity-relationship models, mapping them to static object-oriented models, enhancing them with dynamic behaviour in terms of behaviour units, signals and dynamic rules, mapping them to RBNs at different abstraction levels, animating them etc. Using VENUS, a systems analyst may gradually construct and animate graphically RBNs which can be totally or partially folded and unfolded; partial folding and unfolding results in multi-level RBNs where the object class, behaviour unit or rule of interest can be pin-pointed. In this way, clear, concise, compact, modular and easily modifiable dynamic specifications can be formulated; the latter provide a basis for development of information systems that satisfy users' needs and are flexible enough to incorporate future changes of behaviour.

We are currently working towards the mapping of requirements specifications to design and implementation structures, so as to provide a complete methodology for information systems development. Further development of formal validation algorithms is also currently under study.

References

Ang, J.S.K.; D.W.Conrath; and V.Savolainen. 1991. "Analyzing Information Systems using Petri Nets." In *Proceedings of the 2nd International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-II)* (Washington D.C., U.S.A., July 18-19, 1991). Elsevier Science Publishers, The Netherlands, 329-352.

Conrath, D.W.; V.De Antonellis; and C.Simone. 1991. "Dynamic Modelling for Office Support System Analysis and Design." *Proceedings of the 2nd International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-II)* (Washington D.C., U.S.A., July 18-19, 1991). Elsevier Science Publishers, The Netherlands, 75-94.

DYNMOD-I. 1990. *Proceedings of the 1st International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-I)* (Noordwijkerhout, the Netherlands, April 9-10, 1990). Participants edition. Delft University of Technology, The Netherlands.

DYNMOD-II. 1991. *Proceedings of the 2nd International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-II)* (Washington D.C., U.S.A., July 18-19, 1991). Elsevier Science Publishers, The Netherlands.

DYNMOD-III. 1992. *Proceedings of the 3rd International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-III)* (Noordwijkerhout, the Netherlands, June 9-10, 1992). Participants edition. Delft University of Technology, The Netherlands.

Genrich, H.J. and K.Lautenbach. 1981. "System Modelling with High-Level Petri Nets." *Theoretical Computer Science* 13: 109-136.

Gouscos, D. 1994. "Rule-Based Nets: Model Definitions and Mapping Algorithms". Internal Report. Department of Informatics, University of Athens, Athens, Greece. (Feb.).

Guha, R.K.; M.Bassiouni; A.Dickinson; D.Roy; and M.C.Fischer. 1991. "Stochastic Petri Net Models of Distributed Simulation Systems." In *Proceedings of the 2nd International Working Conference on Dynamic Modelling and Information Systems (DYNMOD-II)* (Washington D.C., U.S.A., July 18-19, 1991). Elsevier

Science Publishers, The Netherlands, 109-130.

Murata, T. 1989. "Petri Nets: Properties, Analysis and Applications." *Proceedings of the IEEE* 77, no.4 (Apr.): 541-580.

Rolland, C. 1992. "Trends and Perspectives in Conceptual Modelling." In *Conceptual Modelling, Databases and CASE*, P.Loucopoulos and R.Zicari, Eds. John Wiley & Sons, New York, 27-48.

Tsalgaidou, A.; D.Gouscos; and C.Halatsis. 1993. "Rule-Based Behaviour Modelling of Information Systems." In *Proceedings of the 26th Hawaiian International Conference on System Sciences (HICSS-26)* (Hawaii, January 1993). IEEE Computer Society Press, IV: 409-418.

Tsalgaidou, A.; D.Gouscos; and C.Halatsis. 1994. "Specifying and Validating Requirements: The VENUS System." In *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS-EUROPE '94)* (Versailles, France, March 7-10, 1994). Prentice Hall International, 89-102.