Journal of Universal Computer Science, vol. 6, no. 11 (2000), 1109-1130
submitted: 9/6/00, accepted: 19/9/00, appeared: 28/11/00

Modeling Information System Behavior with Dynamic Relations Nets

Laurent Allain
(Institut Supérieur d'Electronique du Nord, Département Informatique, Lille, France
laurent.allain@isen.fr)

Pascal Yim
(Ecole Centrale de Lille, LAIL UPRESA8021, Villeneuve d'Ascq, France
pascal.yim@ec-lille.fr)

Abstract: In this paper we highlight three main qualities for a processing model: processing abstraction, dynamic behavior and graphical representation. We define a model closely related to high-level Petri Nets. Dynamic Relations Nets (*DRN*) allow the specification of data, processing, events and constraints within a unique graphical representation. Annotations of the net use a set based abstract language. Constraints arise from three levels: from places (related to the notion of abstract type), from markings (we can then express global constraints between places), and from transitions (in order to specify processing as state transformations). The *DRN* formalism has been successfully applied to a number of case studies. In this paper, we develop the standard 'IFIP case', which has been handled with a lot of modeling methods. A *DRN* specification has a well defined operational semantics. Therefore a *DRN* can also be viewed as an executable specification of information systems. We briefly introduce a tool designed to operate an application developed with *DRNs*, namely *NetSpec*, based on the use of an active database management system. This tool allows an automated code generation (C/SQL) from a *DRN* specification.

Key Words: Petri nets, constraints, dynamic behavior, code generation, information systems.
Categories: D.2.2, D.3.4, H.4.2

1 Introduction

Information system specifications and design are a central problem in computer engineering today. Most usual methods (such as UML, HOOD, OMT, ...) propose different models to specify data (conceptual data model, class diagrams, ...), events (control automaton, Petri Nets, ...) and processing (pseudo-code, object description, ...). As a matter of fact, processing description is often closer to the design phase than to the specification phase. On the other hand, formal languages such as VDM, Z, or B allow the making of an abstract description of processing, generally based on logic and set theory. Moreover, these formal languages have proofs (types, invariants, ...) related to specifications. Unfortunately, it is often difficult to have a global view from such a specification since every processing is specified separately from one another. In

addition, integration of such formalisms into other models (for example graphical ones) means hard work on adaptation problems.

Petri Nets [Murata 1989] allow to take into account the dynamic behavior of a system, and are used in models such as Merise. Petri Nets also provide a graphical formalism (more precise and formal than Data Flow Diagrams for example). In order to bring more expressive power, Predicate-Transitions Nets are often preferred in database modeling (especially distributed databases [Jensen, Rosenberg 1991]). [Bastide et al. 1993] describes an object-oriented Petri Net, closely related to object-oriented, and leads to a concrete implementation. [Heuser et al. 1993] presents a conceptual modeling approach by mixing entity-relationship diagrams and high-level Petri Nets. Nevertheless, high-level Petri Nets fail to model set based transformations of information systems.

By using an homogeneous formalism, the power of the *Dynamic Relations Nets* approach lies in the integration of both static and dynamic aspects of an information system. Place constraints, markings and transitions precisely and fully describe both structure and behavior of the system. Abstraction of this new formalism is guaranteed by a purely formal description of the system behavior by using a semantics based on set theory.

2 Dynamic Relations Nets

In practice, a *Dynamic Relations Net* (*DRN*) is a graphical tool where places depicted as circles, are used to represent availability of resources, transitions depicted as bars or rectangles, model the events, and edges indicate the relationship between places and transitions. Tokens in places and their flow regulated by fired transitions add dynamics to the *DRN*.

2.1 Definition

Before starting the definition of a *DRN*, we assume to have a set of *basic types* (a basic type is a set of values, such as the set of integers or strings). Let L be a first-order language including set operators (union, intersection, set difference, ...) and set expressions, and let V (resp. Σ, F) the set of all variables (resp. expressions, formulas) defined by L. A *DRN* is defined by a tuple (P, T, E, N, D, G, W, M_0) where:

- P, T, and E are finite and disjoined sets of *places*, *transitions*, and *edges*,
- $N : E \rightarrow (P \times T) \cup (T \times P)$ is a function mapping each edge to an *input node* and to an *output node*,
- $D : E \rightarrow$ { *production*, *consumption*, *information*, *negative information* } is a function associating a sort to each edge,
- $G : T \rightarrow F$ is a function associating a formula to each transition, also called a *guard*,
- *Type* is a function mapping each place to a cartesian product of basic types,
- $W : E \rightarrow \Sigma$ is a function mapping each edge to a set expression so that:

– the set expressions $W(p, t)$ and $W(t, p)$ denote a finite set of elements in $Type(p)$
– each free variable occurring in $W(p, t)$ and $W(t, p)$ also occurs in $G(t)$
– an edge of sort *negative information* is annotated by a finite set of variables

– M_0 is a function mapping each place to a finite set of tuples (*initial marking*).

2.1.1 Markings

The marking of a place is a finite set of *tuples*, i.e. a *relation*. It is useful to note the tokens as *structured records*. We note the definition of the type bound to a place as:

$$Type(p) = < field_1 : S_1, \ldots , field_n : S_n >$$

This declaration expresses that the type of the place p is the product of basic types $S_1 \times \ldots \times S_n$. Then, if e belongs to such a type, the notation $e.field_i$ denotes the projection of e on the i^{th} component. We extend this notation to a set of expressions s, each element of which belongs to the type of p: $s.field = \{ e.field \bullet e \in s \}$.

2.1.2 Edges and transitions

Usually, the set expressions mapped to edges are a finite set of variables $\{ x_1, \ldots , x_n \}$ or a single variable s used to define a set by a formula in the guard of the transition. See [Fig. 1] for some graphical examples of edges and annotations.

The validation of a transition t is conditioned by some explicit constraints bound to the annotations $W(p, t)$ on edges. Indeed, each element of the set denoted by $W(p, t)$, where (p, t) is an edge of sort *information* or *consumption*, has to belong to the marking of p. If we note $\psi(p, t)$ such an implicit constraint, we have then $\psi(p, t) = (W(p, t) \subseteq M(p))$. This implicit constraint allows to ensure that any intentionally defined sets denote finite sets. On the other hand, for a *negative information* edge annotated by a singleton $\{ e \}$, we have an implicit constraint $\psi(p, t) = (e \notin M(p))$.

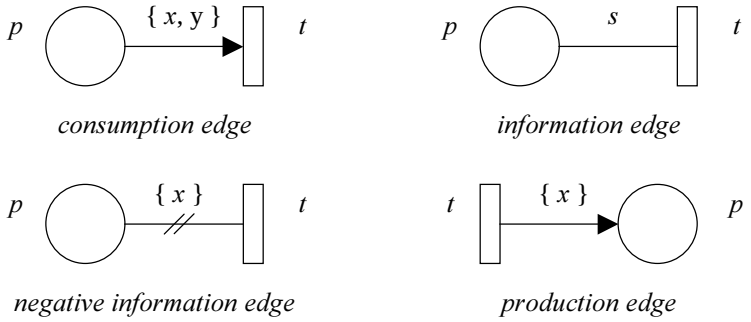

Figure 1: *Graphical representation of the edges annotated by sets of variables ({ x } or { x, y }) or by a variable denoting an intentionally defined set (s).*

The constraint $\psi(t)$ is called the firing constraint for the transition t. If a substitution θ exists, so that $\theta\psi(t)$ is valid, then the transition t is validated for θ with the marking M. The resulting marking M' after the firing of the transition becomes:

$M'(p) = M(p) - \theta E(p,t)$
if (p,t) is a consumption edge and (t,p) is not a production edge
$M'(p) = M(p) \cup \theta E(t,p)$
if (p,t) is not a consumption edge and (t,p) is a production edge
$M'(p) = (M(p) - \theta E(p,t)) \cup \theta E(t,p)$
if (p,t) is a consumption edge and (t,p) is a production edge
$M'(p) = M(p)$
otherwise

2.2 Examples and useful shortcuts

Before developing a case study discussed later in [Section 3 The IFIP case], some simple examples will clarify the behavior of a *DRN*. Hereafter we describe two examples and we introduce useful abbreviations that do not exist in the definition of a *DRN*. For these examples, we assume that all the token fields are of the basic type *integer*.

2.2.1 Example 1: Marking constraints

This example discusses the implementation of marking constraint and key constraint, which are related to markings and which must be satisfied to validate a transition.

Consider a *DRN* with three places A, B, C and one transition T [see Fig. 2] so that:

– $Type(A) = \langle x \rangle$, $Type(B) = \langle y \rangle$, $Type(C) = \langle z, t \rangle$,
– $M_0(A) = \{ 1, 2, 3 \}$, $M_0(B) = \{ 3 \}$, $M_0(C) = \varnothing$, where $\varnothing$ denotes the empty set,
– each edge is annotated by a singleton: $W(A, T) = \{ a \}$, $W(B, T) = \{ b \}$, and $W(T, C) = \{ c \}$,
– $G(T) = (a.x < 4) \wedge (b.y > 0) \wedge (c.z = b.y) \wedge (c.t = a.x)$, where $\wedge$ is the boolean *and* operator.

According to the specification of such a *DRN*, $\langle 1 \rangle$ will be removed from A and $\langle 3,1 \rangle$ will be produced into C, then $\langle 2 \rangle$ will be removed from A and $\langle 3,2 \rangle$ will be produced into C, then $\langle 3 \rangle$ will be removed from A and $\langle 3,3 \rangle$ will be produced into C. B remains unchanged because the edge between B and T is an information edge. The order of productions into C depends on the choice made in the selection of a token from A.

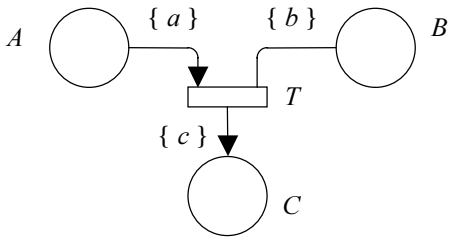

Figure 2: *A very simple example of a DRN.*

Now suppose that we don't want to produce a token into *C*, so that its two fields *z* and *t* receive the same value. We can add a *marking constraint* (*place constraint*) to *G*(*T*): ($c.z \neq c.t$), so ⟨3,3⟩ will not be produced. For convenience, this marking constraint may be specified by extending the type of *C*:

$$Type(C) = \langle z, t \rangle \bullet z \neq t.$$

Now suppose that we don't want to produce tokens into *C* with the same values for *z* only. A negative information edge from *C* to *T* must be added to the *DRN* [see Fig. 3] and *G*(*T*) must be completed (anded) with the term ($b.y = d.z$) which expresses the absence of a token with a specific value for *z* inside *C*. So if ⟨3,1⟩ has already been produced into *C* from the initial marking, the transition *T* will not be crossed any more. This constraint is called a *key constraint* and may be implicitly specified (so without adding the negative information edge to the *DRN*) by underlining the correct field in the type of *C*:

$$Type(C) = \langle \underline{z}, t \rangle$$

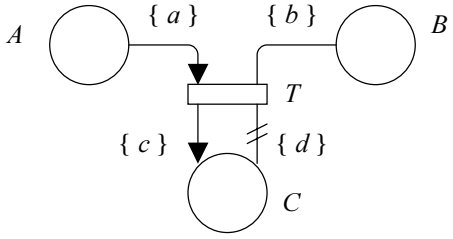

Figure 3: *Use of an explicit negative information edge to implement a key constraint.*

2.2.2 Example 2: Intentionally defined set

This example discusses the possibility to annotate an edge with a variable denoting an intentionally defined set (a finite set), so that each element satisfies a given constraint.

It is important to notice that we cannot know anything about the cardinality of such a set at design time. However, it will be the greatest possible at run time.

Consider a *DRN* with only one place *A* and one transition *T* [see Fig. 4] so that:

– $Type(A) = \langle x \rangle$,
– $M_0(A) = \{ -1, 0, 1, 2 \}$,
– the edge is annotated by the variable *a* denoting a set,
– $G(T) = (a.x > 0)$

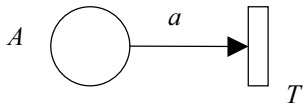

Figure 4: *Use of a variable denoting an intentionally defined set.*

According to the specification of such a *DRN*, the set of two tokens $\{ \langle 1 \rangle, \langle 2 \rangle \}$ will be removed from *A* when the transition is fired, since the field *x* of the other tokens is not strictly positive.

3 The IFIP case

The 'IFIP case' summarizes the functionality of a paper submission processing with a view to a congress organized by the International Federation for Information Processing. This case study has already been handled with a lot of modeling methods [Sibertin-Blanc 1991] [Pascot, Ridjanovic 1991].

The text below summarizes the specifications of the IFIP case:

– Clause C1: When a purpose letter or a paper is received, if the authors are not already known by the scientific organization, they are registered.
– Clause C2: Only purpose letters and papers received before the deadline for paper submission will be retained, except in the case of a derogation granted by the program committee.
– Clause C3: Paper projects (purpose letters or temporary version of a paper) are distributed to the committee members, who become referees for the papers which are assigned to them.
– Clause C4: In case of a purpose letter, if the temporary version of the paper is not received during the month following the deadline for paper submission, the communication project is canceled.
– Clause C5: Program committee members have to mark the papers which have been assigned to them: with a mark ranging from 0 to 10; for the following criteria: paper interest, paper quality, references quality.

– Clause C6: After the marking, papers are classified into 3 categories: accepted papers, rejected papers, and balloted papers. The classification rules are defined by the program committee of each conference. It is recommended that these rules are recorded by the information system, to be able to classify the papers by means of a computational process. The classification rules type is a balanced average: average mark=a_1×(average mark from criteria 1)+...+a_n×(average mark from criteria n), so that a_1+...+a_n=1. According to the paper category, this one is accepted, accepted with reserve (required corrections, and submitted to the decision of the program committee), or rejected.
– Clause C7: Accepted papers can be presented in a session. The program committee meets to: finally select the papers, plan the sessions, define the session program (assign the papers), appoint the session chairmen, plan other conference activities.
– Clause C8: The program committee will see that the paper speaker is its author.
– Clause C9: Notice that the referee of a paper cannot be the author or co-author. But the program committee members can submit communication projects.
– Clause C10: The program committee requests invited papers. The invited papers are not reviewed by the referees, however a deadline for submission is negotiated with each author. If the requirements are not met, the invited paper will not be programmed. An invited paper corresponding to the required conditions to be programmed, will be programmed according to the same rules as an accepted paper.
– Clause C11: If a paper was marked by less than 3 referees, it must be assigned to one or more new referees.
– Clause C12: If this happens to a great number of papers, the program committee meeting date could be postponed to a few days according to a decision of the program committee.
– Clause C13: If the author of an accepted paper does not send the final version of his paper before a given deadline, then the paper can be refused after the program committee decision.
– Clause C14: The final version of a paper must include corrections required by the program committee. The correction requests are established on the basis of the temporary version of a paper. Final papers might be reviewed by the program committee if necessary. A general rule is established for each conference.

A *DRN* has been designed with a graph containing 21 places, 19 transitions, 56 edges, and only four pages for the annotations [See Appendix]. In this section, we develop only two representative parts of the *DRN*, showing the power of the approach.

3.1 Mail and authors

Considering the subsystem related to mails and authors, it is obvious (clauses C1, C2, C4, and C13) that a token in place *mail* is composed of an *author name*, a *receipt date*, and a *version*, the types of which are clear (we consider here that the basic types

are integers (resp. reals, strings) and are noted ≤ (resp. ∞,)). However, as these three attributes are not sufficient to determine a given mail, we are induced to create a fourth attribute as a *reference*, which must be unique for that mail. We also have to associate a place constraint, because the version of the paper must belong to the set {'*letter*','*temporary*','*final*'}. As unknown authors must be registered, a place *author* must be created, with the unique attribute *name*, which becomes a constraint key (it is not necessary to register the same author more than once). To complete the subsystem, the transition *register_author* takes place between places *mail* and *author*, which allows the registration. Edges are of types information from *mail* and production to *author*, and the transition constraint may be *author.name = mail.author_name*, which is sufficient to take the correct action. The graph of this subsystem is summarized in [Fig. 5] and the annotations can be found in [Tab. 1].

Place	*Type(Place)*
mail	⟨ *ref* : ≤, *author_name* : , *receipt_date* : , *version* : • *version* ∈ { 'letter', 'temporary', 'final' } ⟩
author	⟨ *name* : ⟩
Transition	*G(Transition)*
register_author	*author.name = mail.author_name*

Table 1: *Annotations of the subsystem related to mail and authors.*

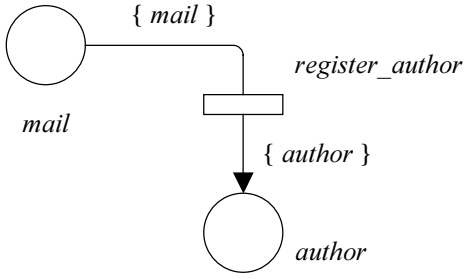

Figure 5: *Graph of the subsystem related to mail and authors.*

3.2 Reappointment of referees

A transition constraint may be more complicated, as in the example of the transition *reappoint_referee* (clause C11). When the *end of marking* is active (condition 1) and the *paper* has received less than three marks (condition 2) and a *referee* has not given his mark (condition 3) and this paper can be assigned to a *new referee* (condition 4), we have to reappoint the referee (condition 5), as shown in the subsystem described

in [Fig. 6] and in [Tab. 2], which illustrates the use of negative information edge and intentionally defined set:

– condition 1: *end_of_marking.state* = 'yes'
– condition 2: *mark'.ref* = *referee.ref* ∧ *Card*(*mark'*) < 3
– condition 3: *referee.ref* = *mark.ref* ∧ *referee.name* = *mark.referee_name*
– condition 4: *reappointment.ref* = *referee.ref* ∧
reappointment.old_referee_name = *referee.name*
– condition 5: *appointment.ref* = *reappointment.ref* ∧
appointment.referee_name = *reappointment.new_referee_name*

The meaning of the second and third conditions is: for a given set of marks for a paper (*mark'.ref* in condition 2) so that its cardinality is less than 3, we try to find a referee for the paper (*referee.ref* in condition 2), and we bind that referee (*referee.ref* and *referee.name* in condition 3) to a mark (*mark.ref* and *mark.referee_name* in condition 3) by means of a negative information edge, to see if there is no mark for him.

Place	*Type(Place)*
referee	⟨ *ref* : ≤, *name* : ⟩
appointment	⟨ *ref* : ≤, *referee_name* : ⟩
reappointment	⟨ *ref* : ≤, *old_referee_name* : , *new_referee_name* : ⟩
end_of_marking	⟨ *state* : • *state* ∈ { 'yes', 'no' } ⟩
mark	⟨ *ref* : ≤, *referee_name* : , *subject* : ∞ • *subject* ∈ [0..10], *quality* : ∞ • *quality* ∈ [0..10], *references* : ∞ • *references* ∈ [0..10]⟩
Transition	*G(Transition)*
Reappoint_referee	*end_of_marking.state* = 'yes' ∧ *reappointment.ref* = *referee.ref* ∧ *reappointment.old_referee_name* = *referee.name* ∧ *referee.ref* = *mark.ref* ∧ *referee.name* = *mark.referee_name* ∧ *mark'.ref* = *referee.ref* ∧ Card(*mark'*) < 3 ∧ *appointment.ref* = *reappointment.ref* ∧ *appointment.referee_name* = *reappointment.new_referee_name*

Table 2: *Annotations of the subsystem related to the reappointment of referees.*

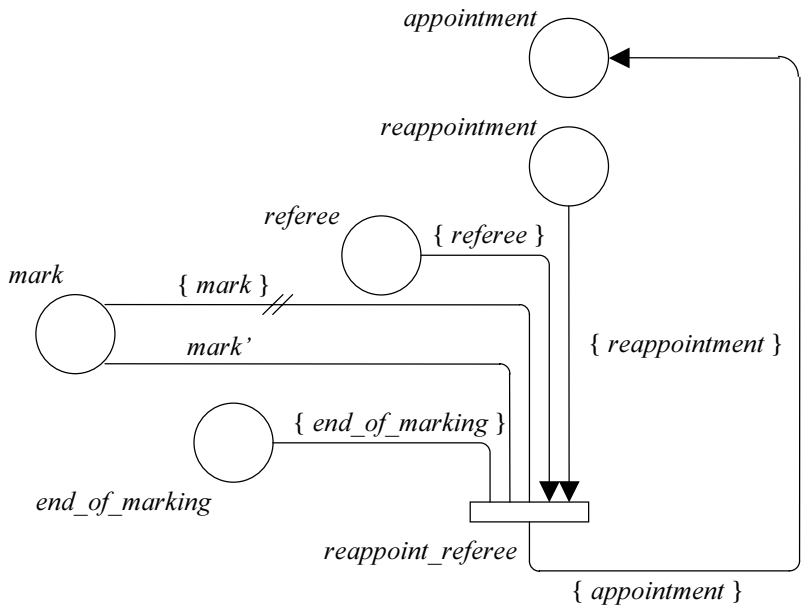

Figure 6: *Graph of the subsystem related to the reappointment of referees.*

4 Implementation issues

We have decided to implement our complete system designed to operate an application developed with *DRNs*, namely *NetSpec*, on the basis of an *active database management system* (ADBMS) layered architecture. It means, using an existing non-active DBMS (in our case DB2 from IBM) and adding a monitor layer that is responsible for providing active features.

4.1 ECA-rules in ADBMS

In this section, we recall some concepts related to ADBMS since the reader may find other explanations in the literature ([Dittrich et al. 1995] to give just one example).

In an active database system, *ECA-rules* (*Event-Condition-Action*) usually take the form:

on event
if condition
then action

An *event* happens instantaneously at specific points in time. For example, in a relational model, database events are related to actions such as *insert*, *delete*, and

update. Temporal events are related to a clock, and may be absolute or relative. Finally, explicit events are those events that are detected along with their parameters by application programs. All these types of events are *primitive* events and can be combined together with event operators to form *composite* events [Chakravarty et al. 1993].

A *condition* is a simple query over the database. In other words, a condition returns a boolean value, that is true if the query has produced a set containing at least one row of data.

An *action* is executed if the condition is satisfied. The action part of the rule usually inserts, deletes, or updates data.

If the event part of the rule does not exist, we call such a rule *pattern-based*, and if the condition part does not exist, we call such a rule *event-based*.

ECA-rules are usually processed using the following algorithm, derived from the *recognize-act* cycle of expert systems [Brownston et al. 1985]:

```
initial match    //execute rule conditions
repeat until no rule conditions produce tuples
   perform conflict resolution    //pick a triggered rule
   act    //execute the rule action for all tuples produced by the condition
   match    //test rule conditions
end
```

In the *match* phase, rule patterns are matched against data to determine which rules are triggered and for which instantiations. The entire set of triggered rule instantiations is called the *conflict set*, and one instantiation is chosen from this set using a conflict resolution strategy. In the *act* phase, the selected rule action is executed for all tuples of the selected instantiation, then the cycle repeats.

The choice of which rule to execute when multiple rules are triggered is called conflict resolution. In many active database systems this choice is made more or less arbitrarily: random [Agrawal, Gehani 1989], numeric priorities [Hanson 1992], partial order [Widom et al. 1991], based on coupling modes [Gatziu et al. 1994], concurrent execution [Chakravarthy 1989].

Each time a rule is fired, there is an *instantiation* associated with that execution: a data item, or combination of items, that matches the rule pattern. At execution time, the values of the instantiated items can be referenced in the rule action through the use of variables specified in the rule pattern. That is, at run-time, variables are *bound* in the pattern and passed to the action.

Coupling modes [McCarthy, Dayal 1989] determine how rule events, conditions, and actions relate to database transactions. Generally, rule conditions are evaluated and actions are executed in the same transaction, but it is not always the case. Associated with each rule is an *E-C* coupling mode and a *C-A* coupling mode, where *E*, *C*, *A* denote the events, conditions and actions respectively. Each coupling mode is either *immediate*, indicating immediate execution, *deferred*, indicating execution at the end of the current transaction, or *decoupled* (detached), indicating execution in a separate transaction. For each of the combinations of coupling modes, it is relatively

easy to construct an active database application for which the behavior seems most appropriate [Hanson, Widom 1992].

4.2 Special Considerations

Although *DRNs* have not primarily been created to model ECA-rules, we can easily use them to do that. However, special considerations must be understood if we want to model ECA-rules by means of *NetSpec*:

- Events are not managed as in the definition of ECA-rules because the existence (the absence) of a token, as well as its attribute values, can be specified directly in the condition. So, ECA-rules supported by *NetSpec* are pattern-based only,
- The act phase of the recognize-act cycle operates only for one tuple rather than for all of those, to avoid a set-oriented firing of transitions [Kiernan et al. 1990] as in Petri nets,
- *DRNs* do not allow to specify a conflict resolution strategy: a transition may be fired as soon as it has been triggered. So, the design of a transition constraints is very important if we want to obtain a deterministic behavior of the net. With *NetSpec*, conflict resolution is implemented by using numeric priorities of three different types: fixed priorities (the match phase checks the rule conditions in a specific order), rotating priorities (a triggered rule will receive the lowest priority for the next match phase), and iterative priorities (a triggered rule will receive the highest priority for the next match phase, allowing an equivalent of set-oriented firing. We can also specify no conflict resolution, since *NetSpec* now supports multitasking (all transitions are modeled by concurrent processes),
- Finally, the only coupling mode supported when using conflict resolution is the deferred one. In fact, immediate C-A coupling mode may cause problems (causally dependent constraints), and detached C-A coupling mode is reserved for multitasking mode. Practically, on the match and act phases, queries of types delete (for consumption) and insert (for production) are generated, but will be executed at the end of the transaction, in that order, to avoid token duplication in the case of an update of attribute values.

These restrictions are not developed in this paper because our purpose is first to show that *DRNs* have enough potential toward the design side of an information system.

4.3 Architecture of NetSpec

As shown in [Fig. 7] an application layer is provided to design a *DRN* with a graphical tool, to store its definition, to analyze and translate this definition into a set of executable programs (formally the job of the *DRN* compiler). A run-time library (the situation monitor layer) independent of the application itself, provides rule processing (transition ordering based on conflict resolution).

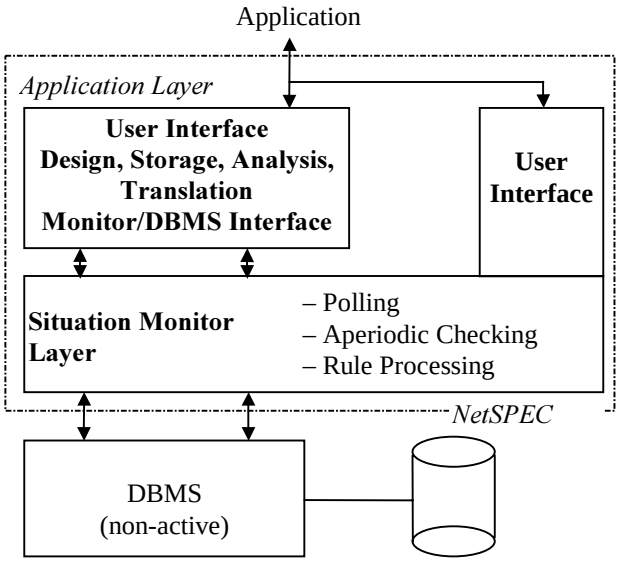

Figure 7: *The layered architecture of NetSpec.*

4.4 Code generation

The main job of the situation monitor layer is to provide rule processing. In fact, the main difficulty is for the *DRN* compiler, to order the evaluation of all constraints over the net [Allain 1999].

For a transition, the input part (from places to a transition) will be evaluated first, beginning with finite sets of variables circulating over the edges, i.e. variables well known. For one tuple of the result, variables of intentionally defined sets are evaluated. Last, these intentionally defined sets are checked for their cardinalities. The output part (from a transition to places) is then computed to generate new token attribute values. These values are optionally checked for compatibility and finally, place constraints and key constraints are evaluated. If all these constraints are satisfied, the transition can be crossed and the update of the net is done by means of insert and/or delete queries.

From a *DRN* specification, either created by means of the graphical tool or directly written in the native language of *NetSpec*, the compiler creates two executable files. The former of these two files is a script that contains commands used to create the database itself. With a relational data model, places are implemented by tables, each record of which (row) keeps one token, which attributes are columns of the row. For example, the place *mail* of the IFIP case will be created by:

```
CREATE TABLE mail ( INT PK_mail,
                    INT ref,
                    VARCHAR(40) author_name,
                    DATE receipt_date,
                    VARCHAR(40) version,
                    PRIMARY KEY(PK_mail) )
```

Notice that the primary key *PK_mail* is automatically inserted by the compiler to avoid duplication of tokens having no key constraint and receiving the same attribute values at run-time (a place contains a set, not a multiset). So, such tokens can be distinguished without ambiguity inside database queries. This primary key is invisible for the designer and is reserved for internal use only.

Transition constraint queries can also be created as views at this time. These views are relative to the input part of such a constraint (only those generating well known variables). For example, the view created for the transition *register_author* is:

```
CREATE VIEW view_register_author ( PK_mail,author_name )
                        AS SELECT PK_mail, author_name FROM mail
```

The latter of the two files created by the compiler provides functions used for rule processing. For the example developed in [Section 3.1], the non-optimized code of the transition *register_author* will be:

```
begin
  crossed ← FALSE
  DECLARE CURSOR C1 AS SELECT * FROM view_register_author
  OPEN C1
  while ¬EOF(C1) and ¬crossed
    FETCH C1 INTO :pk_a,:author_name
    SELECT COUNT(*) INTO :count FROM author
                                    WHERE name=:author_name
    if count = 0 then
        INSERT INTO author VALUES(:author_name)
        COMMIT WORK
        crossed ← TRUE
      else ROLLBACK WORK
    fi
  endwhile
  CLOSE C1
end
```

5 Conclusion

The definition of *DRNs* is now complete. Several systems derived from multiple scopes were modeled with our formalism: information processing (the 'IFIP case'), planning (robot and cubes [Wilkins 1988]), simulation (IBM360 pipeline execution

unit [Kogge 1981]), process control (steam-boiler [Abrial 1994]), and reverse design of a real application (data management [Cadivel 1997]). In this last study, a *DRN* restricted to one page for its graph and three pages for all annotations has been build, even though the initial program had about 50000 lines of code written in a classical programming language. All these examples had successfully passed their implementation with *NetSpec*. We have highlighted three main qualities for a processing model: processing abstraction, dynamic behavior and graphical representation. We have defined a model closely related to high-level Petri Nets. *Dynamic Relations Nets* allow within a unique graphical representation the specification of data, processing, events and constraints. Annotations of the net use a set based abstract language. Constraints arise from three levels: from places (related to the notion of abstract type), from markings (we can then express global constraints between places), and from transitions (in order to specify processing as state transformations).

We have not developed here a true method built around *DRNs*. The reverse design study gave birth to new ideas especially related to valid schema transformations, allowing a gradual materialization from an abstract model to an operational one, as with the B method. In fact, the non-deterministic behavior remains a critical aspect: a *DRN* must satisfy some properties so that the implementation (for example with *NetSpec*) fits the initial specifications.

We have not looked for the usual properties of Petri Nets and high-level Petri Nets (invariants, ...) applied to *DRNs*. We think that these properties are useful in concurrent systems but are less significant in information systems. Therefore, we will concentrate on applying proof tools related to Z and B to our model.

In this paper, we have described the implementation of our approach in active database systems, based on the *Dynamic Relation Nets*. Our system, *NetSpec*, automatically generates the imperative code of the computational part of an ADBMS based application, by means of a new modeling language that makes abstraction of classical programming.

However, *NetSpec* has enlightened some problems that do not exist in the *DRN* theory:

- First of all and the most important one, a *DRN* may have concurrent transitions the constraints of which are not exclusive. Such a configuration introduces a non-deterministic behavior of the net. *NetSpec* uses priorities to solve conflicts between triggered transitions, as many of the existing ADBMS. However, the ”programming style” has a possible influence on the model, because the designer can choose its priority type,
- Another problem is the necessity to build a coherent development tool, especially with a debugger [see Fig. 8]. In fact, the fired transitions of a *DRN* introduce the same consequences of the fired ECA-rules in ADBMS: transitions (as ECA-rules) behavior may be complex and not easily understood. The designer of an application must be able to directly influence the transition firing, to solve terminations and deadlocks. This is an objective of our future directions.

Our future works will be based on a set of tools designed to build and to validate a general method to be used on non obvious applications. We hope to create a set of consistent design tools, dedicated to processing specifications.

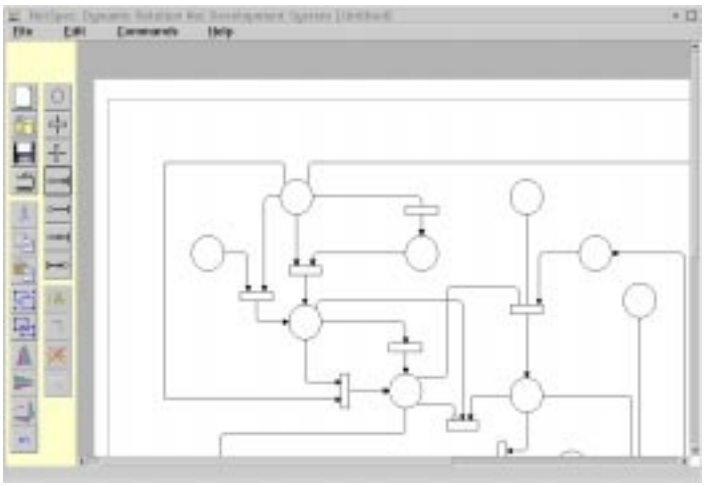

Figure 8: *A screen sample of the NetSpec prototype development tool.*

References

[Abrial 1994] Abrial, R.: ”Steam-Boiler Control Specification Problem”; Technical Report (1994).

[Agrawal, Gehani 1989] Agrawal, R., Gehani, N.H.: ”ODE (Object Database and Environment): The Language and the Data Model”; Proc. ACM-SIGMOD International Conference on Management of Data, Portland, Oregon (1989), 36-45.

[Allain 1999] Allain, L.: ”Contribution à la Modélisation et à la Spécification: Réseaux Formels et Bases de Données Actives”; Thèse de Doctorat en Productique, Automatique et Informatique Industrielle, LAIL UPRES A8021, EC-Lille, Université des Sciences et Technologies de Lille, France (1999).

[Bastide et al. 1993] Bastide, R., Sibertin-Blanc, C., Palanque, P.: ”Cooperative objects: a concurrent, Petri-net based, object-oriented language”; Proc. IEEE SMC, 1, (1993), 286-291.

[Brownston et al. 1985] Brownston, L., Farrell, R., Kant, E., Martin, N.: ”Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming”; Addison-Wesley, Reading, Massachusetts (1985).

[Cadivel 1997] Cadivel, C.: ”Contribution à la Specification des Systèmes d’Information”; Thèse de Doctorat en Informatique et Automatique Appliquées, INSA Lyon, France (1997).

[Chakravarthy 1989] Chakravarthy, S.: HiPAC: ”A Research Project in Active, Time-Constrained Database Management, Final Report”; Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, Massachussets (1989).

[Chakravarthy et al. 1993] Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: ”Anatomy of a Composite Event Detector”. Technical Report UF-CIS-TR-93-039, CIS Department, University of Florida (1993).

[Dittrich et al. 1995] Dittrich, K.R., Gatziu, S., Geppert, A.: ”The Active Database Management System Manifesto”; Proc. 2nd Workshop on Rules in Databases, Athens, Greece (1995), 3-20.

[Gatziu et al. 1994] Gatziu, S., Geppert, A., Dittrich, K.R.: ”The SAMOS Active DBMS Prototype”; Technical Report 94.16, Institut für Informatik, Universität Zürich Switzerland (1994).

[Hanson, Widom 1992] Hanson, E.N., Widom, J.: ”An Overview of Production Rules in Database Systems”; Technical Report UF-CIS 92-031, CIS Department, University of Florida (1992).

[Hanson 1992] Hanson, E.N.: ”Rule Condition Testing and Action Execution in Ariel”; Proc. ACM-SIGMOD International Conference, (1992), 49-58.

[Heuser et al. 1993] Heuser, C.A., Peres, E.M., Richter, G.: ”Towards a complete conceptual Model: Petri Nets and Entity-Relationship Diagrams”; Information Systems, 5, (1993), 275-298.

[Jensen, Rosenberg 1991] Jensen, K., Rosenberg, G.: ”High-Level Petri Nets”; Springer, Berlin (1991).

[Kiernan et al. 1990] Kiernan, G., de Maindreville, C., Simon, E.: ”Making Deductive Database a Practical Technology: A Step Forward”; Proc. ACM SIGMOD International Conference on Management of Data (1990).

[Kogge 1981] Kogge, P.M.: ”The Architecture of Pipelined Computers”; McGraw-Hill Book Company (1981).

[McCarthy, Dayal 1989] McCarthy, D.R., Dayal, U.: ”The Architecture of an Active, Object-oriented Database Management System”; Proc. ACM-SIGMOD International Conference on Management of Data, 18, 2, Portland, Oregon (1989), 215-224.

[Murata 1989] Murata, T: ”Petri nets: properties, analysis and applications”; Proc. IEEE, 77, 4, (1989), 541-580.

[Pascot, Ridjanovic 1991] Pascot, D., Ridjanovic, D.: ”DATARUN: La modélisation des traitements au sein des modèles de données. Application au cas IFIP”; AFCET, Paris (1991).

[Sibertin-Blanc 1991] Sibertin-Blanc, C.: ”Cooperative Objects for the Conceptual Modeling of Organizational Information Systems”; Proc. IFIP TC8, Quebec City, Canada (1991).

[Widom et al. 1991] Widom, J., Cochrane, R.J., Lindsay, B.G.: ”Implementing Set-Oriented Production Rules As An Extension To Starburst”; Proc. 17th International Conference on Very Large Data Bases (VLDB), Barcelona, Spain (1991), 275-285.

[Wilkins 1988] Wilkins, D.: ”Practical Planning: Extending the classical AI planning paradigm”; Morgan Kaufmann (1988).

Appendix

Hereafter we show the complete graph of the *DRN* [see Fig. 9] and the complete table of annotations [see Tab. 3] for the IFIP case study. For convenience, we do not annotate the graph with all the variables but we consider that a given edge is annotated by a set { *p* } where *p* is the place linked to it. The only exception is for the variables denoting intentionally defined sets: they can be located by a '*' and they appear in the annotation table as *p'*.

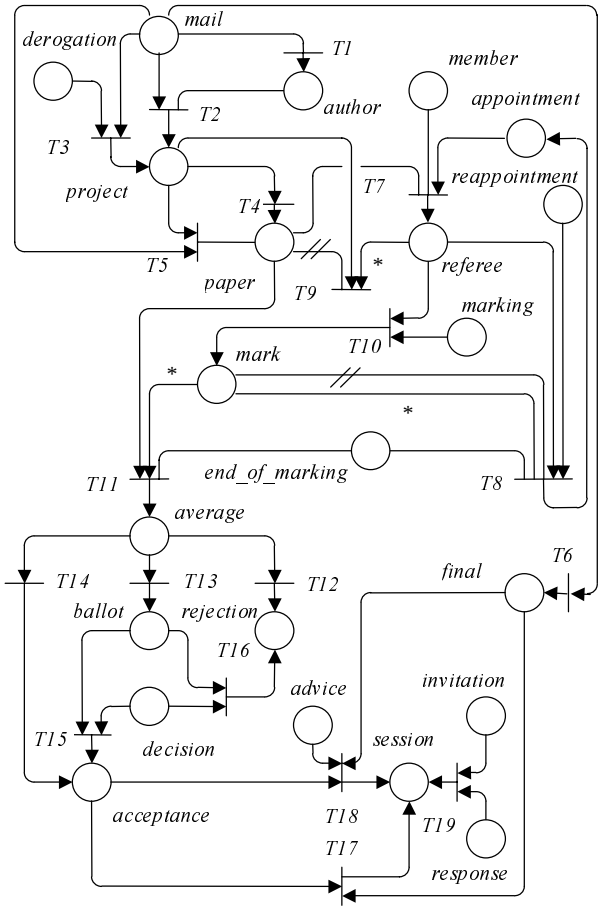

Figure 9: *Complete graph of the DRN.*

Place	*Type(Place)*
mail	⟨ *ref* : ≤, *author_name* : , *receipt_date* : , *version* : • *version* ∈ { 'letter', 'temporary', 'final' } ⟩
author	⟨ *name* : ⟩
derogation	⟨ *ref* : ≤, *agreement* : • *agreement* ∈ { 'yes', 'no' } ⟩
project	⟨ *ref* : ≤, *author_name* : , *receipt_date* : , *version* : • *version* ∈ { 'letter', 'temporary' } ⟩
paper	⟨ *ref* : ≤, *author_name* : ⟩
member	⟨ *name* : ⟩
appointment	⟨ *ref* : ≤, *referee_name* : ⟩
referee	⟨ *ref* : ≤, *name* : ⟩
reappointment	⟨ *ref* : ≤, *old_referee_name* : , *new_referee_name* : ⟩
end_of_marking	⟨ *state* : • *state* ∈ { 'yes', 'no' } ⟩
marking	⟨ *ref* : ≤, *referee_name* : , *subject* : ∞ • *subject* ∈ [0..10], *quality* : ∞ • *quality* ∈ [0..10], *references* : ∞ • *references* ∈ [0..10] ⟩
mark	⟨ *ref* : ≤, *referee_name* : , *subject* : ∞ • *subject* ∈ [0..10], *quality* : ∞ • *quality* ∈ [0..10], *references* : ∞ • *references* ∈ [0..10] ⟩
average	⟨ *ref* : ≤, *author_name* : , *value* : ∞ ⟩
rejection	⟨ *ref* : ≤, *author_name* : ⟩
ballot	⟨ *ref* : ≤, *author_name* : ⟩
acceptance	⟨ *ref* : ≤, *author_name* : ⟩
final	⟨ *ref* : ≤, *author_name* : , *receipt_date* : ⟩
decision	⟨ *ref* : ≤,

	agreement : • *agreement* ∈ { 'yes', 'no' } ⟩
advice	⟨ *ref* : ≤, *agreement* : • *agreement* ∈ { 'yes', 'no' } ⟩
session	⟨ *ref* : ≤, *speaker_name* : ⟩
invitation	⟨ *ref* : ≤, *author* : , *deadline* : ⟩
response	⟨ *ref* : ≤, *receipt_date* : ⟩

Transition	*G(Transition)*
register_author (T1)	*author.name = mail.author_name*
receive_mail (T2)	*mail.receipt_date* ≤ '06/09/2000' ∧ (*mail.version* = 'letter' ∨ *mail.version* = 'temporary') ∧ *mail.author_name = author.name* ∧ *project.ref = mail.ref* ∧ *project.author_name = mail.author_name* ∧ *project.receipt_date = mail.receipt_date* ∧ *project.version = mail.version*
Derogate_delay (T3)	*mail.receipt_date* > '06/09/2000' ∧ (*mail.version* = 'letter' ∨ *mail.version* = 'temporary') ∧ *mail.author_name = author.name* ∧ *derogation.ref = mail.ref* ∧ *derogation.agreement* = 'yes' ∧ *project.ref = mail.ref* ∧ *project.author_name = mail.author_name* ∧ *project.receipt_date = mail.receipt_date* ∧ *project.version = mail.version*
receive_temporary (T4)	*project.version* = 'temporary' ∧ *paper.ref = project.ref* ∧ *paper.author_name = project.author_name*
receive_temp_letter (T5)	*mail.version* = 'temporary' ∧ *project.version* = 'letter' ∧ *project.ref = mail.ref* ∧ *paper.ref = mail.ref* ∧ *paper.author_name = mail.author_name*
receive_final (T6)	*mail.version* = 'final' ∧ *final.ref = mail.ref* ∧ *final.author_name = mail.author_name* ∧ *final.receipt_date = mail.receipt_date*

appoint_referee (T7)	*appointment.ref = paper.ref* ∧ *appointment.referee_name = member.name* ∧ *appointment.referee_name ≠ paper.author_name* ∧ *referee.ref = appointment.ref* ∧ *referee.name = appointment.referee_name*
Reappoint_referee (T8)	Please see [Tab. 2]
cancel (T9)	*project.ref = referee'.ref* ∧ *project.ref = paper.ref* ∧ *project.version* = 'letter' ∧ CurrentDate – *project.receipt_date* > 30 days
register_mark (T10)	*marking.ref = referee.ref* ∧ *marking.referee_name = referee.name* ∧ *mark.ref = marking.ref* ∧ *mark.referee_name = marking.referee_name* ∧ *mark.subject = marking.subject* ∧ *mark.quality = marking.quality* ∧ *mark.references = marking.references*
compute_average (T11)	*end_of_marking.state* = 'yes' ∧ *mark'.ref = paper.ref* ∧ Card(*mark'*) ≥ 3 ∧ *average.ref = paper.ref* ∧ *average.author_name = paper.author_name* ∧ *average.value* = Avg(*mark'.subject*) × 0.5 + Avg(*mark'.quality*) × 0.3 + Avg(*mark'.references*) × 0.2
register_rejection (T12)	*average.value* < 8.0 ∧ *rejection.ref = average.ref* ∧ *rejection.author_name = average.author_name*
register_ballot (T13)	*average.value* ≥ 8.0 ∧ *average.value* < 10.0 ∧ *ballot.ref = average.ref* ∧ *ballot.author_name = average.author_name*
register_acceptance (T14)	*average.value* ≥ 10.0 ∧ *acceptance.ref = average.ref* ∧ *acceptance.author_name = average.author_name*
accept_ballot (T15)	*ballot.ref = decision.ref* ∧ *decision.agreement* = 'yes' ∧ *acceptance.ref = ballot.ref* ∧ *acceptance.author_name = ballot.author_name*
reject_ballot (T16)	*ballot.ref = decision.ref* ∧ *decision.agreement* = 'no' ∧ *rejection.ref = ballot.ref* ∧ *rejection.author_name = ballot.author_name*
organize_final_session (T17)	*acceptance.ref = final.ref* ∧ *final.receipt_date* ≤ '06/30/2000' ∧ *session.ref = final.ref* ∧ *session.speaker_name = final.author_name*

organize_delayed_session (T18)	*acceptance.ref* = *final.ref* ∧ *final.receipt_date* > ’06/30/2000’ ∧ *final.ref* = *advice.ref* ∧ *advice.agreement* = ’yes’ ∧ *session.ref* = *final.ref* ∧ *session.speaker_name* = *final.author_name*
invite (T19)	*invitation.ref* = *response.ref* ∧ *response.receipt_date* ≤ *invitation.deadline* ∧ *session.ref* = *invitation.ref* ∧ *session.speaker_name* = *invitation.author_name*

Table 3: *Complete annotations of the DRN.*