Modelling and analysing workflow using a Petri-net based approach

W.M.P. van der Aalst[†] K.M. van Hee[‡†] G.J. Houben[†]

[†] Eindhoven University of Technology, Dept. of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
[‡] Bakkenist Management Consultants, Wisselwerking 46, 1112 XR Diemen, The Netherlands.

Contact author:

Dr. Ir. W.M.P. van der Aalst
Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Telephone: +31 40 2474295
Telefax: +31 40 2436685
E-mail: wsinwa@win.tue.nl

Modelling and analysing workflow using a Petri-net based approach

W.M.P. van der Aalst[†] K.M. van Hee[‡†] G.J. Houben[†]

[†] Eindhoven University of Technology, Dept. of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
[‡] Bakkenist Management Consultants, Wisselwerking 46, 1112 XR Diemen, The Netherlands.

Abstract

High-level Petri nets have been used to model systems in a variety of application domains, ranging from protocols to logistics. This paper addresses an application domain which is receiving more and more attention: workflows in offices and their management. High-level Petri nets are used to model (1) the workflow in an office environment and (2) the workflow management system to support the control of office work. In this way we will be able to formalize the main concepts. As a result we can use Petri net based analysis techniques to analyse the workflow in an office. Moreover, this formalization can be used to develop a prototype of a workflow management system.

Keywords: Application of nets to workflow management, High-level Petri nets, Business process reengineering, Groupware, CASE tools.

1 Introduction

Workflow management systems (wfms) are modern software tools to support and control administrative tasks in large organizations. It is becoming clear that wfms's are the next step in supporting office work, after other tools like database management systems, spreadsheets and electronic mail systems. There are several commercial wfms's on the market (e.g. OPEN/workflow by Wang and WorkFlo by Olivetti) and experiments have shown that they are successful (cf. Van den Berg et al. [4]). However, none of them have the functionality that is needed. Another problem is that there is no clear definition of these systems. Moreover, there is no general conceptual model for workflow management systems, like the relational data model for most database management systems.

To tackle these problems we propose an approach based on high-level Petri nets. Petri nets have been used to model a variety of systems ranging from operating systems to logistic systems. We will see that we can use high-level Petri nets to model workflows and workflow management systems, cf. Bracchi and Pernici [6]. This is not a surprise given the fact that Petri nets have been successfully applied in the two application domains just mentioned (i.e. operating systems and logistics). Note that wfms's can be considered as operating systems that control tasks outside computers. Moreover, there are many similarities between flows inside an office environment and flows of goods and information

inside a logistic system. In fact the term 'office logistics' is often used to refer to the flow of documents inside an office.

There have been other attempts to model workflow in terms of Petri nets, cf. Ellis and Nutt [8], Cindio et al. [7] and Zisman [23]. Ellis and Nutt ([8]) also emphasize the need for high-level Petri nets. However, there are several differences between our approach and the approach presented in in [8], e.g. the workflow concepts and their definitions differ and another Petri net model is used. Our approach integrates in a common formal framework three different aspects of an office: the office functional specification, the organizational structure and the communication aspects.
We will use the results reported in this paper to build a prototype of a workflow management system. In fact, at the moment we are using *ExSpect* ([14]), a software package based on high-level Petri nets, to develop such a wfms prototype.

In section 2 we will discuss the main concepts encountered in the workflow application domain. These concepts will be mapped onto high-level Petri nets in section 3. Both workflows and workflow management systems are modelled in terms of high-level Petri nets. In the appendix a summary of the used Petri net formalism is given. A simple example is given to illustrate our approach.

2 Workflow concepts

In this section we introduce the concepts of workflow management informally and in the next section we give their formalisation in terms of (high-level) Petri nets.

Definition *task, resource, resource class, document*
The key concept of workflow management is *task*. A *task* is a piece of work to be done by one or more *resources* in a pre-determined time interval. A *task* is atomic which means that it is not considered to be split into smaller *tasks*. A *resource* can perform *tasks* (on its own or together with other *resources*). The *resources* performing a *task* are occupied doing the *task* from the start until the end of the time-interval. A *resource class* is a set of *resources*. Each *task* will require *resources* from given *resource classes*. A *document* is the input or output of a *task*, as far as this is relevant for the *wfms*.

A task can be any piece of work, for example the examination of a patient by a physician, the typing of a letter by a secretary or the computing of the roots of a numerical equation by a computer. From the point of view of the *wfms* the task is performed externally, i.e. the *wfms* is only concerned with the management issues of the task, such as: which resource will perform the task and what documents are needed for the resource to perform the task. Some resources are automated (e.g. printers, software), others are 'human' (e.g. clerks). In the specification of a task one is generally not interested in the actual resource performing the task, but only in the class of all resources that can perform the task: for this reason resource classes are tied to tasks, so the *wfms* can choose any of the resources from a resource class to perform a task. The documents that are considered within the *wfms* are only

those messages that are needed in the communication between different tasks: the documents that are part of the object system for which the *wfms* is used are seen as external to the *wfms* and are treated in the same way as any other physical item involved in executing a task.

Definition *resource manager*
The *resources* needed for *tasks* are managed by one or more *resource managers*. They control the allocation of *resources* to *tasks*. A *resource manager* can be a person, a computer system or a combination of the two. A *resource* may be assigned to two or more *tasks* at the same time.

When a task requires several resources resource management is not trivial. Resource management happens outside of the *wfms*, i.e. the actual allocation process itself is not part of the *wfms*.

Definition *procedure, control activity*
A *procedure* is a (partially) ordered set of *control activities*, pairs of *tasks* and sets of *resource classes*, and (sub)*procedures*. The ordering models the precedence relations for the *control activities*, *tasks* and *procedures* that have to be done. The pairs of *tasks* and sets of *resource classes* represent the *tasks* to be done and the classes of *resources* required for those *tasks*. A *control activity* specifies the routing of the work within the *procedure* and the synchronisation of *tasks*.

A procedure has one input and one output: the entities serving as input and output of procedures are documents, just as the entities communicated between tasks. If multiple inputs or outputs are needed a control activity can be used to compose or decompose documents.

Definition *job, job state, job identification, job attributes*
A *job* is a process modelling the execution of an amount of work according to a given *procedure*. So, as a *job* is a kind of process, it can be characterized by a sequence of events, or alternatively, by a sequence of states. We use the second characterization. A *job state* is a snapshot of the *job* at some moment in time. The *job state* contains all the relevant information of the history and (known) future of the *job* at that moment; such as the work progress, i.e. the *tasks* that are currently being executed, references to *documents* that are processed for the *job*, a *job identification* and some *job attributes* that are used to determine the routing of the *job*.

A job asks for one procedure, but at any moment there may be many jobs for which the same procedure is preformed. Another term for a job is a project. A procedure is in that case a type of project. A job is active at some moment in time if this moment occurs after the initialization of the process and before it reaches is final state. The state of each active job is recorded in the *wfms*.

Definition *workflow, wfms*
A *workflow* is a partially ordered set of *jobs*. There are several ways to order the *jobs*; for example *jobs* x and y satisfy $x \prec y$ if and only if x is ready before y starts. A *wfms* is a computer system (or a software package) that manages *workflows*. It provides a number of functions:

- definition of *tasks, procedures* and *jobs,*
- processing of the information that is needed to perform the *tasks* of the *jobs,*
- the management of *resources,*
- routing of *job* information between *procedures* and *resources,*
- supervision over the *workflow*, i.e. collecting management information.

The last function is optional.

We conclude this section with an example in which the introduced concepts are illustrated. Consider a medical centre for diagnoses. A job is the check up of a client (or patient). There are three different procedures, depending on what initiative the client comes to the clinic. The client may come for a *general check up* of his own initiative. Secondly the client may come for a *fitness test* for some sports club. Thirdly, the client may be sent by his family physician for a *specific check up*. The last procedure consists of several sub-procedures: a blood test, an x-ray, cardiogram and a mri-scan. A job contains information in its job attributes that is used to select a next sub-procedure depending on the outcomes of former tasks. For example if the x-ray is all-right then there is no need for an mri-scan.

3 Mapping the concepts onto Petri nets

In this paper *high-level Petri nets* are used to model workflows and workflow management systems. We will use the term 'high-level Petri nets' to refer to Petri nets extended with 'colour', 'time' and 'hierarchy'. Commercial tools such as ExSpect, CPN/Design and Pace are based on high-level Petri nets (cf. appendix A.5). We will use ExSpect to model, analyse and prototype workflow (management) systems. Therefore, we will map the workflow concepts defined in the previous section onto the high-level Petri net model used by ExSpect. An informal introduction to this high-level Petri net model is given in appendix A. For a more formal introduction to this net model the reader is referred to Van der Aalst [2, 3] and Van Hee et al. [13, 14, 15].
Tasks, procedures and resource managers will be modelled by (Petri net) *systems*. (A system is a subnet which may communicate via *connectors*, see appendix A.) Control activities will be modelled by *transitions*. Jobs are represented by (job) *tokens*.

A *task* corresponds to a system composed of five transitions `p1`, `p2`, `p3`, `p4` and `p5` and four places `s1`, `s2`, `s3` and `s4` (see figure 1). Recall that tasks are atomic, i.e. a task cannot be decomposed into other tasks. Every task is modelled by a system similar to the one

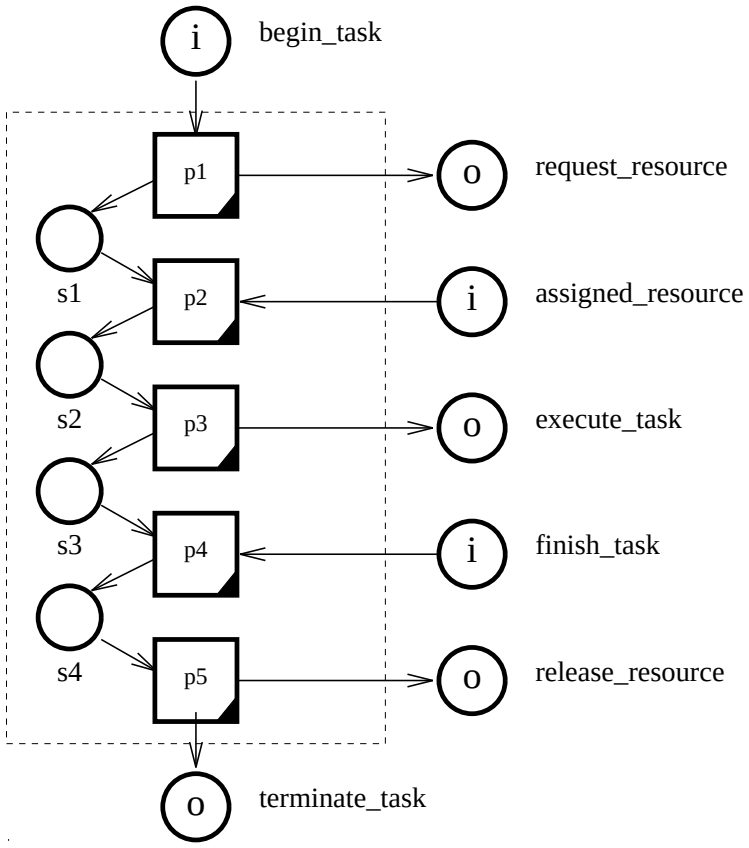

Figure 1: The structure of a task.

shown in figure 1. A task modelled by such a system interacts with a procedure (via input connector `begin_task` and output connector `terminate_task`), a resource manager (via `request_resource`, `assigned_resource` and `release_resource`) and a resource (via `execute_task` and `finish_task`). Transition `p1` is triggered by a *job token* arriving via input connector `begin_task` and sends a request to the resource manager via output connector `request_resource`.[1] The job token consumed by `p1` is put on place `s1`. Each of the places `s1`, `s2`, `s3` and `s4` marks a stage in the 'life-cycle' of the task. Transition `p2` will fire the moment the resource manager assigns a resource to a task to be performed for a specific job. The corresponding job token is moved from stage `s1` to stage `s2` (transition `p2` has a *precondition* which selects the corresponding token from place s1, see appendix A.2). The information about the assigned resource is added to the value of the job token produced for place `s2`. Transition `p3` sends all this information to the resource assigned to execute the task (via output connector `execute_task`). Note that this resource was selected by the resource manager. The resource will execute the task and trigger transition `p4` when it terminates (via `finish_task`). Transition `p4` will move the corresponding job token from `s3` to `s4`. Finally, transition `p5` fires. This transition sends a token to the resource manager to indicate that the assigned resource has

[1] In this section, we assume that each task requires precisely one resource. The extension to multiple resources is straightforward.

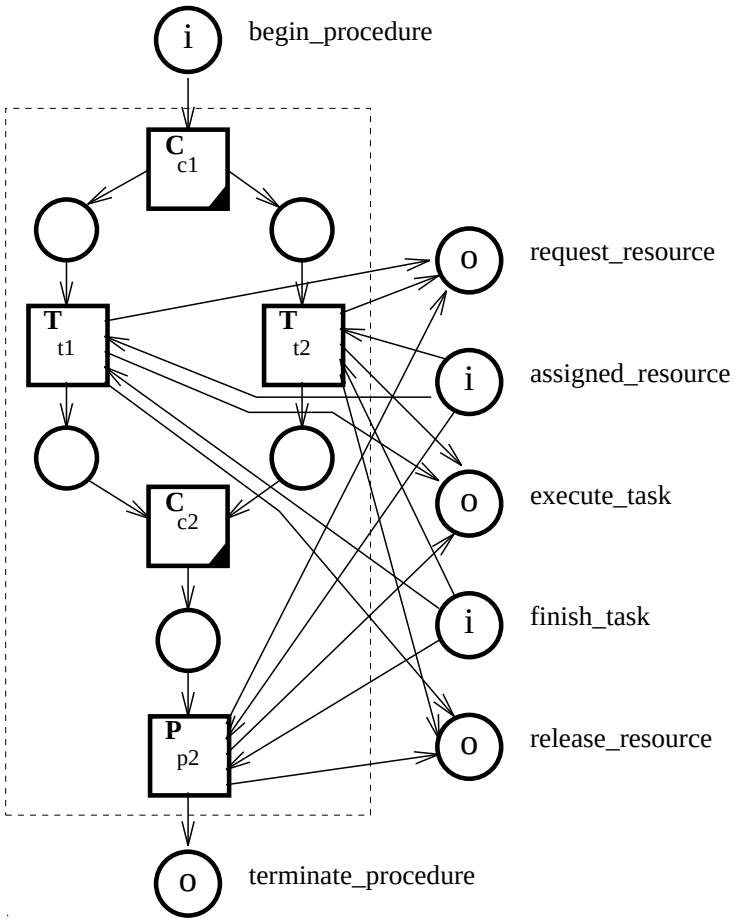

Figure 2: The procedure `p1`.

been released. Transition `p5` also returns the job token to the procedure that started up the task.

A *procedure* is also represented by a system (see figure 2). Since a task is some kind of elementary (i.e. atomic) procedure, it is not a surprise that tasks and procedures have similar connectors. However, a procedure is composed of tasks, control activities and/or subprocedures.
Control activities are represented by transitions marked with a **C**. Tasks and procedures are marked with a **T** and **P** respectively. Figure 2 shows that procedure `p1` is composed of two control activities (`c1` and `c2`), two tasks (`t1` and `t2`) and one procedure (`p2`).
Control activities are used to:

- route jobs (work) inside a procedure,
- synchronize the work in a procedure,
- duplicate job tokens.

Note that control activities are not allowed to interact with the resource manager or the resources. If a control activity requires such interaction, then it is preceded by a task. In

procedure p1 control activity c1 starts up two tasks (t1 and t2) by duplicating the job token arriving via begin_procedure. Control activity c2 starts up procedure p2 when both tasks have been executed.

Job tokens exchanged via input connector begin_procedure and output connector terminate_procedure mark the beginning and ending of a procedure preformed for a specific job. Tasks and sub-procedures inside a procedure communicate with the resource manager and the resources via the other connectors. The output connector request_resource of task t1 is connected to output connector request_resource of p1, etc. The connectors of tasks and procedures are connected in a straightforward manner. Therefore, we could have omitted the five connectors on the right side of the procedure shown in figure 2. (These connections and connectors can be added automatically.)

The only tokens allowed in the places of a procedure are *job tokens*. A job token refers to a specific job. However, inside a procedure there may be several job tokens referring to the same job. Therefore, the procedure should satisfy the following constraint; a procedure executed for a specific job terminates if and only if all job tokens referring to this job have been removed. Note that the *state* of a job j is given by the configuration and values of job tokens referring to j.
The value (colour) of a job token is composed of (1) *a job identification*, (2) *references to documents* and (3) a set of *attributes*. These attributes can be used for the routing of the job token, i.e. control activities may only inspect these attributes. Note that the document itself cannot be used to route the job.

A *resource manager* is modelled by a system with at least three connectors (see figure 3). This system is triggered by tasks sending requests via input connector request_resource. Via the output connector assigned_resource the corresponding task is informed about the resource selected to execute the task. When the task terminates, the resource manager is informed via the input connector release_resource. The resource manager shown in figure 3 is very simple. Each token in the place resources represents a free resource. Transition r1 handles each request and removes the corresponding resource descriptor from the place resources. Transition r2 returns the resource descriptor into the place resources.
In practice the resource manager will be much more complex. For example, an advanced resource manager will try to anticipate tasks in the future, critical jobs, set-up times of resources, etc. In fact, the resource manager has to schedule tasks using simple heuristics, humans or complex optimization techniques. Depending on the knowledge the resource manager has of the workflow, it can become very complex.

Tokens exchanged via request_resource have a value composed of: (1) a *job identification*, (2) a *task identification*, (3) a *resource class* and (4) some additional *attributes*. Note that each task has a unique identification, i.e. it is possible to distinguish between tasks having identical names in different procedures. The resource class indicates the type of resource that is required. Recall that a resource may belong to many resource classes.

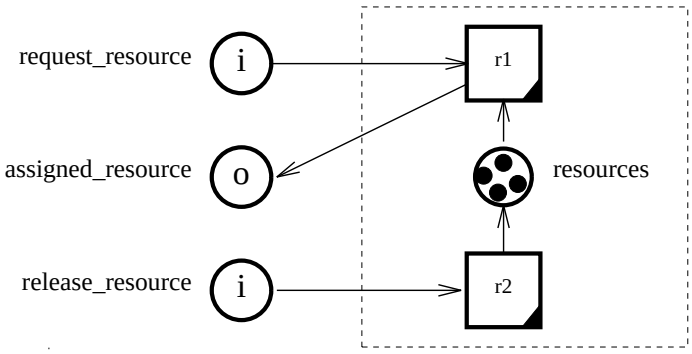

Figure 3: A resource manager.

The token returned via `assigned_resource` has a value composed of (1) a *job identification*, (2) a *task identification*, (3) a *resource identification*. The job identification and task identification are used to assign the resource to the proper job token.
The value of a token exchanged via `release_resource` is equal to the *resource identification* of the released resource.

Finally, we have to define a workflow management system (wfms) is terms of a high-level Petri nets. Although a wfms can be used to define tasks and procedures, we will restrict ourselves to the execution, monitoring and supervision of workflows. Adding a procedure implies an update of the high-level Petri net which represents the wfms. Under these conditions we can model a wfms as shown in figure 4.
There are n procedures that can handle jobs generated by the environment of the wfms (via `begin_procedure`). Note that inside these procedures there may be many other subprocedures. The wfms shown in figure 4 has only one resource manager (`rm1`). In general there will be multiple resource managers. Besides this resource manager there is also a system called `supervisor`. The supervisor collects management information and uses this information to control the workflow. We have omitted the connections between the supervisor system and the procedures and resource manager because of their ad-hoc nature. An advanced wfms may have a very complex supervisor that monitors the status of each job and each resource. In this case there will be many interactions. If the wfms is more simple or tailored towards a specific application area, then there will be a different interaction structure.

We have mapped the workflow concepts defined in section 2 onto the high-level Petri net model used by ExSpect. Note that we could have used any other Petri net model extended with ‘colour’, ‘time’ and ‘hierarchy’. The extension with ‘colour’ is needed to model attributes of jobs and resources. The ‘time’ extension is used to model durations. The ‘hierarchy’ concept is used to decompose procedures into subprocedures and tasks.

There are several reasons for using high-level Petri nets for the modelling of workflows and workflow management systems.
First of all, it is a way to formalize the important concepts. The mapping of concepts such

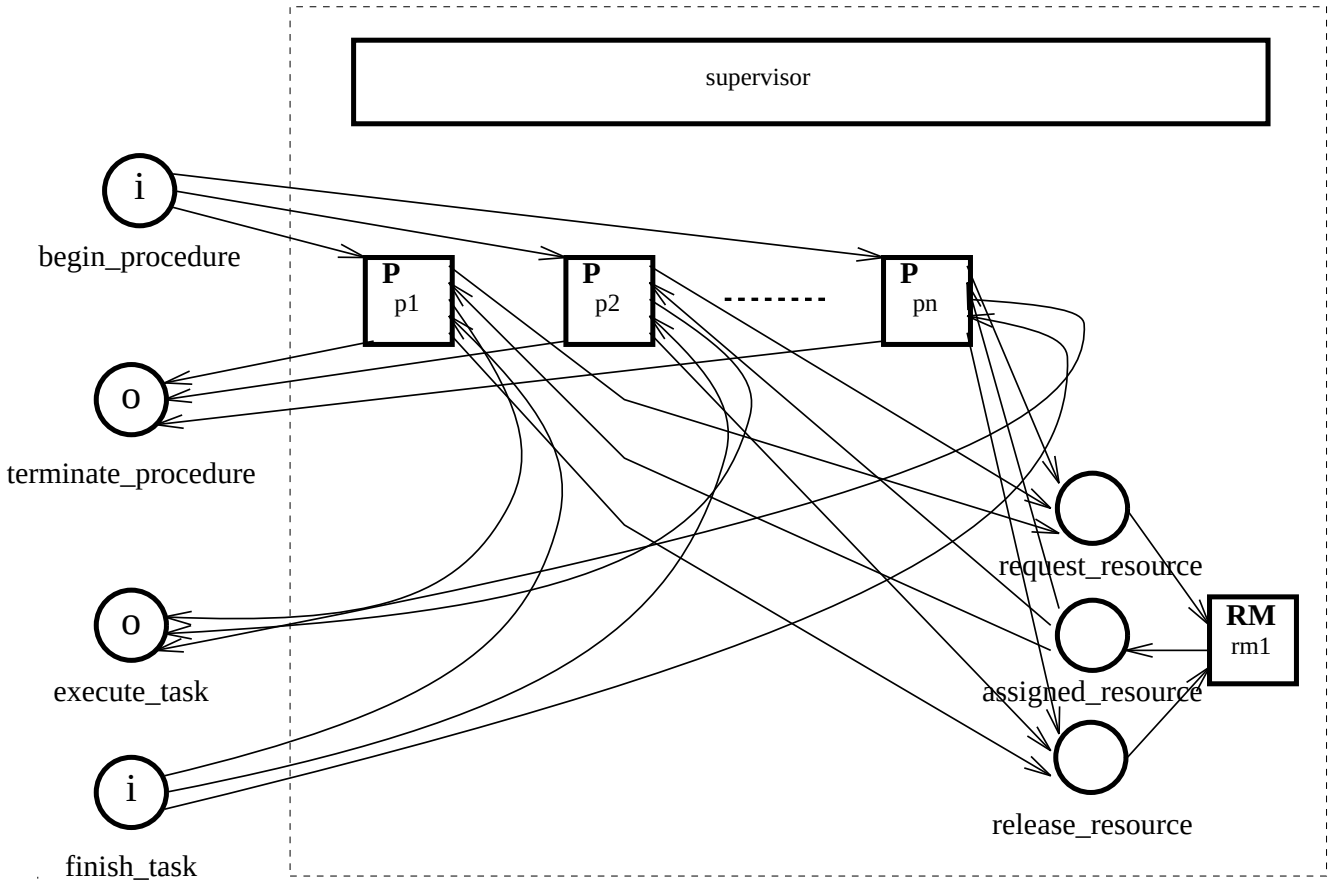

Figure 4: The architecture of the wfms.

as tasks, resources, procedures and jobs onto high-level Petri nets is used to provide the semantics of workflow and wfms's. Note that we did this in an 'object-oriented fashion'. The transitions inside a task system can be regarded as methods, the tokens exchanged between tasks and resources (or resource managers) can be seen as messages, etc. For more information on this subject the reader is referred to [13].
Another reason for using high-level Petri nets is the fact that we can use these nets for analysis purposes. We can use Petri net based analysis techniques to verify a number of properties. Interesting properties that can be verified are:

- Do all jobs terminate properly?
- Are the tasks properly synchronized?
- Are there any subprocedures that are never used?
- Etc.

It is also possible to evaluate the 'performance' of alternative workflows.
Finally, there are there are several software tools available (see appendix A.5) that can be used to model and analyse high-level Petri nets. If fact, by modelling a wfms we obtain a *prototype* of the wfms. This prototype can be used to evaluate the design of a wfms. At the

moment, we are using the software package *ExSpect* ([14]) to develop such a prototype. ExSpect is used to interpret high-level Petri nets representing procedures, tasks, resources managers etc. However, there are other approaches possible to build a (prototype) wfms, e.g. an active database which manages the execution of procedures.

4 Example

In the second section we introduced the example of the medical center for diagnoses. The work that needs to be done in that center consists of different tests performed for the center's clients (patients). In this section we will consider the workflow in this center using the Petri nets from the previous section.
A job in this medical center represents the tests performed for a given patient visiting the center. Such a job can match one of three procedures: a procedure modelling a general check up on the client's own initiative, a procedure modelling a fitness test for a sports club, or a procedure modelling a specific check up on the initiative of the client's family physician. The latter procedure consists of subprocedures modelling a blood test, an x-ray, a cardiogram or an mri-scan.
In the wfms this procedure could look like shown in the figure 5. (Note that we have omitted the five connectors used to interact with the resource manager and the resources.)
In this procedure four subprocedures are used for the different tests to be performed for this check up. The control activities are used to evaluate the tests done so far and to determine whether the diagnosis is final or whether further tests are necessary. Depending on the outcome of the blood test the diagnosis is ready or other tests need to be performed starting with an x-ray. Again depending on the outcome the diagnosis is ready or a cardiogram needs to be made. Eventually, if the diagnosis is not ready yet, an mri-scan is the ultimate test on which the final diagnosis is based.
The (sub)procedure for the blood test is itself a procedure. Figure 6 shows the corresponding net.
In this procedure three tasks need to be executed in a predefined sequence. First, blood has to be taken from the client. Subsequently, this blood has to be analysed and then a report has to be produced confirming the result of the analysis. The resources that are necessary for these tasks come from three different classes (modelled as types of resource tokens). A nurse is needed for taking the blood from the client. The analysis has to be performed by a recognized analyst, while the production of the result of the test needs work by the chief analyst.
The job tokens that flow through these procedure nets contain besides a job identification:

- a reference to the client, as the patient is an essential part of the work to be managed;
- a reference to the client's file, as the medical data need to be examined in order to produce the correct diagnosis;
- an attribute *state_of_test* stating the result of the test so far, which can be used to route the job through the entire collection of tests (as soon as the result stands, the test is ended).

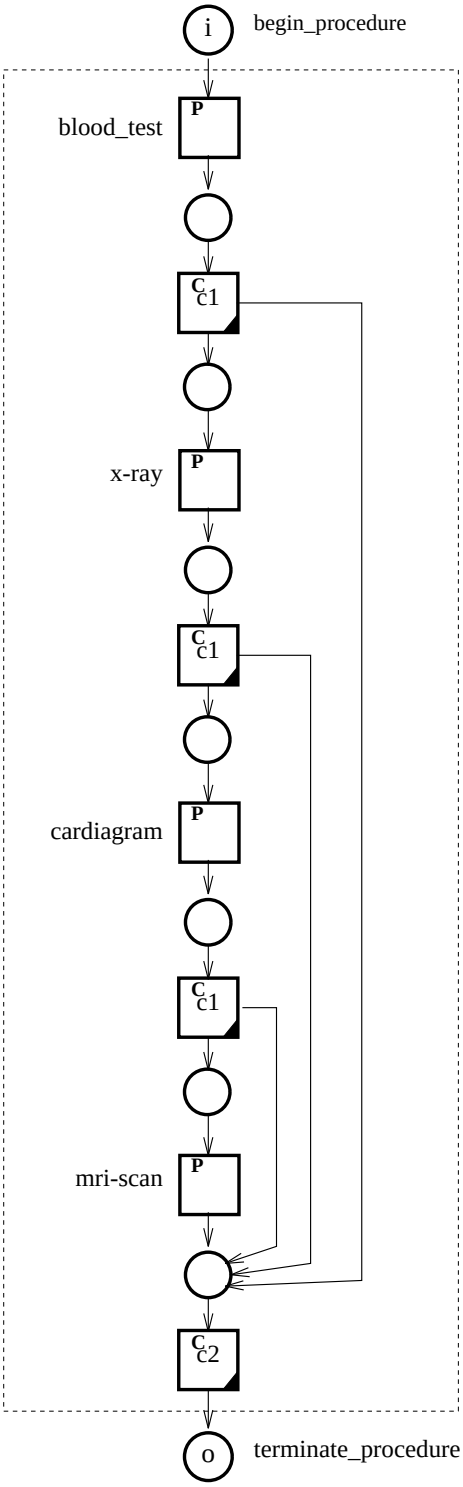

Figure 5: The procedure `check_up`.

The procedure `check_up` contains three instances of the same control activity `c1`. The control activities `c1` in the check up procedure basically verify whether the result is known or not. For this they redirect their input to either the end of the procedure (*output1*) or the

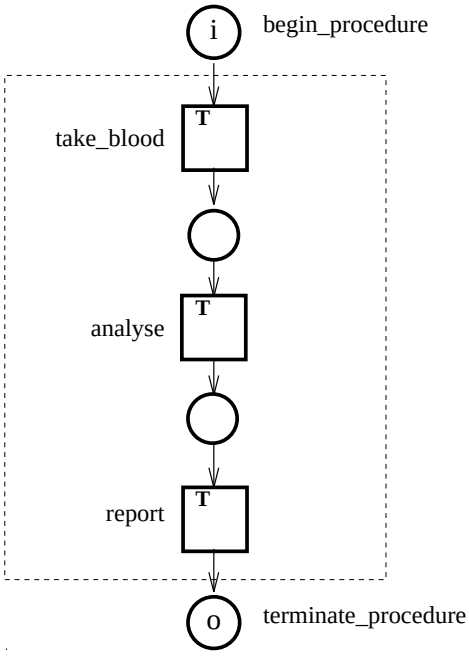

Figure 6: The subprocedure `blood_test`.

next test (*output2*) based on the value of the *state_of_test* attribute:
if ready(state_of_test) then output1 < − input else output2 < − input fi
Note that control activity `c1` operates as an 'OR' gate, i.e. precisely one of the output places obtains a token. (This is possible because of the net model used in this paper, see appendix A.2.)
The control activity `c2` collects the information from the previous tests and produces the final output. For this it uses the documents (tokens) communicated between these tasks.

5 Conclusions and research issues

In this paper we presented an approach to model workflows and workflow management systems in terms of high-level Petri nets. This way we have been able to formalize the main concepts. Moreover, the use of high-level Petri nets gives us the opportunity to analyse and prototype both workflow and workflow management systems. Nevertheless, there remains a lot of work to be done.
First of all, we need to provide a user-friendly interface to define workflows, i.e. a customized editor to define procedures, control activities and tasks.
Secondly, we have to do some research on the scheduling in the wfms, i.e. the assignment of resources to tasks in a workflow.
Finally, we have to build a prototype wfms. We will use the software package ExSpect (see appendix A.5) to do this. By applying this prototype to the management of a variety of workflows, we will be able to evaluate a number of designs. This knowledge will support the successful implementation of future workflow management systems.

References

[1] W.M.P. VAN DER AALST, *Modelling and Analysis of Complex Logistic Systems*, in Integration in Production Management Systems, H.J. Pels and J.C. Wortmann, eds., vol. B-7 of IFIP Transactions, Elsevier Science Publishers, Amsterdam, 1992, pp. 277–292.

[2] ——, *Timed coloured Petri nets and their application to logistics*, PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.

[3] W.M.P. VAN DER AALST AND A.W. WALTMANS, *Modelling logistic systems with EXSPECT*, in Dynamic Modelling of Information Systems, H.G. Sol and K.M. van Hee, eds., Elsevier Science Publishers, Amsterdam, 1991, pp. 269–288.

[4] A. VAN DEN BERG, A. BOUWMANS, AND G. SPANHOFF, *Workflow management*, Computable, 21 (1993), pp. 26–32.

[5] B. BERTHOMIEU AND M. DIAZ, *Modelling and verification of time dependent systems using Time Petri Nets*, IEEE Transactions on Software Engineering, 17 (1991), pp. 259–273.

[6] G. BRACCHI AND B. PERNICI, *Trends in office modelling*, in Office Systems, A.A. Verrijn-Stuart and R.A. Hirschheim, eds., IFIP, Elsevier Science Publishers, Amsterdam, 1986, pp. 77–92.

[7] F. DE CINDIO, C. SIMONE, R. VASSALLO, AND A. ZANABONI, *CHAOS: a knowledge-based system for conversing within offices*, in Office Knowledge: Representation, Management and Utilization, W. Lamersdorf, ed., IFIP, Elsevier Science Publishers, Amsterdam, 1988, pp. 257–275.

[8] C.A. ELLIS AND G.J. NUTT, *Modelling and Enactment of Workflow Systems*, in Application and Theory of Petri Nets 1993, M. Ajmone Marsan, ed., vol. 691 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1993, pp. 1–16.

[9] H.J. GENRICH, *Predicate/Transition-Nets*, in Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties, W. Brauer, W. Reisig, and G. Rozenberg, eds., vol. 254 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1987, pp. 207–247.

[10] H.J. GENRICH AND K. LAUTENBACH, *System modelling with high level Petri nets*, Theoretical Computer Science, 13 (1981), pp. 109–136.

[11] M. HAMMER AND J. CHAMPY, *Reengineering the corporation*, Nicolas Brealey Publishing, London, 1993.

[12] K. HAYES AND K. LAVERY, *Workflow management software: the business opportunity*, Ovum, 1991.

[13] K.M. VAN HEE, *Information System Engineering: a Formal Approach*, Cambridge University Press, 1994.

[14] K.M. VAN HEE, L.J. SOMERS, AND M. VOORHOEVE, *Executable specifications for distributed information systems*, in Proceedings of the IFIP TC 8 / WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis, E.D. Falkenberg and P. Lindgreen, eds., Namur, Belgium, 1989, Elsevier Science Publishers, Amsterdam, pp. 139–156.

[15] K.M. VAN HEE AND P.A.C. VERKOULEN, *Integration of a Data Model and High-Level Petri Nets*, in Proceedings of the 12th International Conference on Applications and Theory of Petri Nets, Gjern, June 1991, pp. 410–431.

[16] K. JENSEN, *Coloured Petri Nets: A High Level Language for System Design and Analysis*, in Advances in Petri Nets 1990, G. Rozenberg, ed., vol. 483 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1990, pp. 342–416.

[17] ——, *Coloured Petri Nets. Basic concepts, analysis methods and practical use.*, EATCS monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1992.

[18] M. AJMONE MARSAN, G. BALBO, AND G. CONTE, *A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems*, ACM Transactions on Computer Systems, 2 (1984), pp. 93–122.

[19] D. MORRIS AND J. BRANDON, *Reengineering your business*, McGraw-Hill, New York, 1993.

[20] T. MURATA, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, 77 (1989), pp. 541–580.

[21] C.A. PETRI, *Kommunikation mit Automaten*, PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[22] R. ZICARI, M.G. FUGINI, AND R. MAIOCCHI, *Time management in the office-net system*, in Office Knowledge: Representation, Management and Utilization, W. Lamersdorf, ed., IFIP, Elsevier Science Publishers, Amsterdam, 1988, pp. 163–176.

[23] M.D. ZISMAN, *Representation, Specification and Automation of Office Procedures*, PhD thesis, University of Pennsylvania, Warton School of Business, 1977.

A High-level Petri nets

In this paper we use high-level Petri nets to model workflows and workflow management systems (wfms). A high-level Petri net is a Petri net extended with ‘colour’, ‘time’ and ‘hierarchy’. In this appendix, we give an informal introduction to the classical Petri net, followed by a short description of each of the extensions.

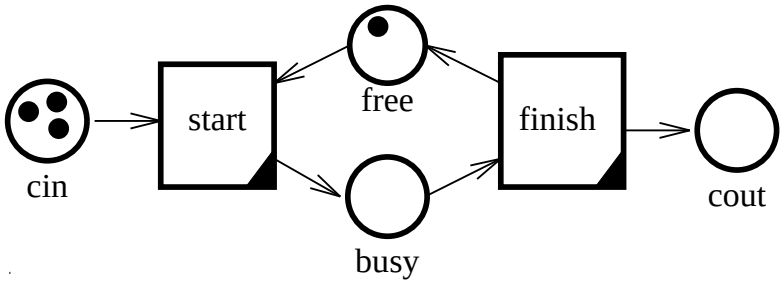

Figure 7: A classical Petri net which represents a machine.

A.1 The classical Petri net model

Historically speaking, Petri nets originate from the early work of Carl Adam Petri ([21]). Since then the use and study of Petri nets has increased considerably. For a review of the history of Petri nets and an extensive bibliography the reader is referred to Murata [20].

The classical Petri net (often called place/transition net) is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles with a marked corner. (In literature transitions are often displayed as bars.) Places may contain zero or more *tokens*, drawn as black dots. The number of tokens may change during the execution of the net. A place p is called an *input place* of a transition t if there exists a directed arc from p to t, p is called an *output place* of t if there exists a directed arc from t to p.
We will use the net shown in figure 7 to illustrate the classical Petri net model. This figure models a resource (e.g. a machine) which processes jobs and has two states (free and busy). There are four places (`cin`, `free`, `busy` and `cout`) and two transitions (`start` and `finish`). In the state shown in figure 7 there are four tokens; three in place `cin` and one in place `free`. The tokens in place `cin` represent jobs to be processed by the machine. The token in place `free` indicates that the machine is free and ready to process a job. If the machine is processing a job, then there are no tokens in `free` and there is one token in `busy`. The tokens in place `cout` represent jobs which have been processed by the machine. Transition `start` has two input places (`cin` and `free`) and one output place (`busy`). Transition `finish` has one input place (`busy`) and two output places (`cout` and `free`).

A transition is called *enabled* if each of its input places contains 'enough' tokens. An enabled transition can *fire*. Firing a transition t means consuming tokens from the input places and producing tokens for the output places, i.e. t 'occurs'.
Transition `start` is enabled in the state shown in figure 7, because each of its input places (i.e. `cin` and `free`) contains a token. Transition `finish` is not enabled because there are no tokens in place `busy`. Therefore, transition `start` is the only transition that can fire. Firing a transition `start` means consuming two tokens, one from `cin` and one from `free`, and producing one token for `busy`. The resulting state is shown in figure 8. In this state only transition `finish` is enabled, i.e. transition `finish` fires and the token in

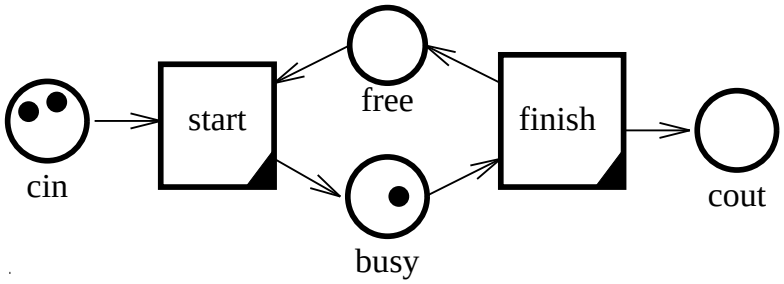

Figure 8: The state after firing `start`.

place `busy` is consumed and two tokens are produced, one for `cout` and one for `free`. Now transition `start` is enabled, etc. Note that as long as there are jobs waiting to be processed, the two transitions fire alternately, i.e. the machine modelled by this net can only process one job at a time.

The classical Petri net model has been used in many application areas, e.g. communication protocols, flexible manufacturing systems and distributed information systems (see Murata [20]). However, Petri nets describing real systems tend to be complex and extremely large. To solve these problems, many authors propose extensions of the basic Petri net model. We propose three extensions; 'colour', 'time' and 'hierarchy'. Such extensions are a necessity for the successful application of Petri nets to the modelling of workflows and workflow management systems.

A.2 Adding colour

Tokens often represent objects (e.g. resources, goods, humans) in the modelled system. Therefore, we often want to represent attributes of these objects. If a truck is modelled by a token in the Petri net, then we may want to represent the capacity, registration number, location, etc. of the truck. Since these attributes are not easily represented by a token in a classical Petri net, we extend the Petri net model with *coloured* or *typed tokens*. In a coloured Petri net each token has a value often referred to as 'colour'. Many coloured Petri net models have been proposed in literature ([2, 10, 9, 13, 16, 17]). One of the main reasons for such an extension is the fact that uncoloured nets tend to become too large to handle.
We will use the machine modelled in figure 7 to clarify this concept. Tokens in the places `cin` and `cout` represent jobs. These jobs may have attributes like an identification number, a description and a due-date. We can model this by giving the tokens in `cin` and `cout` a *value* (colour) which corresponds to these attributes. In figure 9 we see that the job in `cin` has an identification number 3241 and a due-date 29-01-94. The token in place `free` represents a machine and its value contains information about this machine (type and mode).
Transitions determine the values of the produced tokens on the basis of the values of the consumed token, i.e. a transition describes the relation between the values of the 'input tokens' and the values of the 'output tokens'. Moreover, the *number of tokens* produced

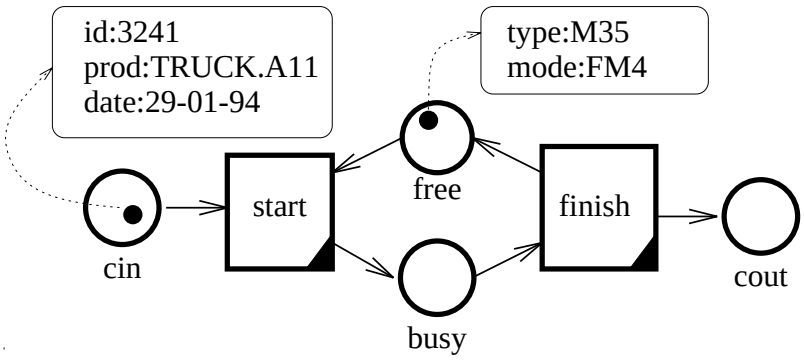

Figure 9: Adding colour.

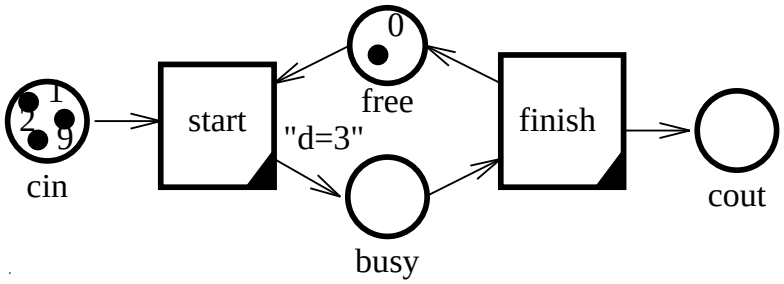

Figure 10: Adding time.

for each output place may depend upon the values of the input tokens. It is also possible to specify 'preconditions', e.g. transition `start` may have a precondition which specifies that jobs require a machine of a specific type.

A.3 Adding time

For real systems it is often important to describe the *temporal behaviour* of the system, i.e. we need to model durations and delays. Since the classical Petri net is not capable of handling quantitative time, we add a timing concept. There are may ways to introduce time into the classical Petri net ([2]). We use a timing mechanism where time is associated with tokens and transition determine delays.

Consider the net shown in figure 10. Each token has a *timestamp* which models the time the token becomes available for consumption. The token in `free` has timestamp 0, the tokens in `cin` have timestamps ranging from 1 to 9. Since these timestamps indicate when tokens become available, transition `start` becomes enabled at time 1. (Time 1 is the earliest moment that each of the input places contains a token which is available.) Therefore, transition `start` fires at time 1, thereby producing a token for `busy` with delay 3. The timestamp of this token is equal to 1+3=4. Transition `finish` will be the next to fire (at time 4), etc. The delay of a produced token can be described by a fixed value, an interval or a probability distribution (cf. [2, 5, 18, 20]).

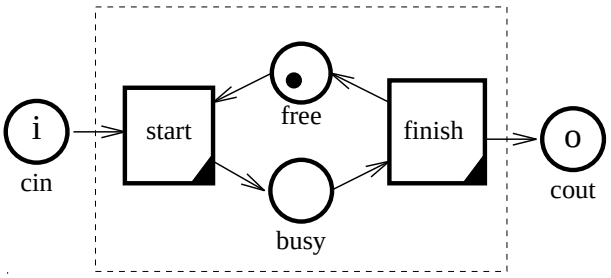

Figure 11: The definition of the `machine` system.

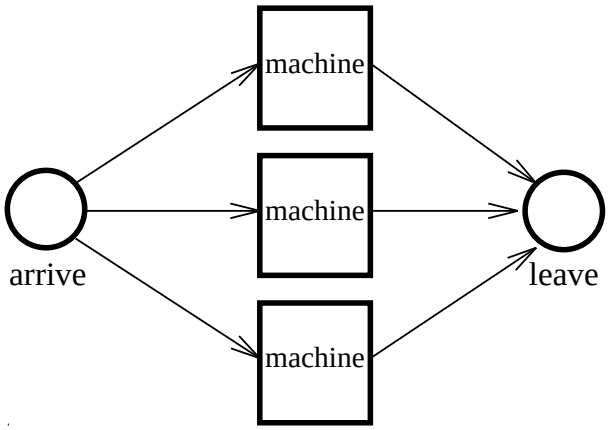

Figure 12: Three parallel machines modelled in terms of the `machine` system.

A.4 Adding hierarchy

Although timed coloured Petri nets allow for a succinct description of many workflows (and wfms), precise specifications for real workflows have a tendency to become too large and complex. This is the reason we provide a hierarchy construct, called *system*. A system is an aggregate of places, transitions and (possibly) subsystems.

Figure 11 shows the definition of the `machine` system. This system is composed of two places (`free` and `busy`) and two transitions (`start` and `finish`) and two *connectors* (`cin` and `cout`). These connectors provide an interface with the environment of the `machine` system. The `cin` connector is an *input connector* (i.e. tokens may enter the system via this connector), `cout` is an *output connector* (i.e. tokens may leave the system via this connector). If a system is used then the connectors are connected to places at a 'higher level'. Consider for example the net shown in figure 12. In this net the same definition is 'installed' three times. In this case, for each of these 'installations' the `cin` connector is connected to the place `arrive` and the `cout` connector is connected to the place `leave`, i.e. the connectors inside the `machine` system are 'glued' on top of places at a higher level.

The system concept allows for hierarchical modelling, i.e. it is possible to decompose complex systems into smaller subsystems. (Note that it is possible to have an arbitrary number of levels.) For practical applications of Petri nets, the system concept is of the utmost importance. The system concept can be used to structure large specifications. At one level we

want to give a simple description of the system (without having to consider all the details). At another level we want to specify a more detailed behaviour.
For a more elaborate discussion on hierarchy constructs, the reader is referred to Jensen [16], van der Aalst [1, 2] and van Hee [13].

A.5 Language and tools

This paper shows that Petri nets extended with 'colour', 'time' and 'hierarchy' (i.e. *high-level Petri nets*) are suitable for the modelling of workflows and wfms's.
Only a few high-level Petri net models (i.e. hierarchical timed coloured Petri net models) have been proposed in literature. Even fewer high-level Petri net models are supported by software tools. Nevertheless, there are at least two software products, *ExSpect* ([2, 14]) and *Design/CPN* ([16]), that are being distributed on a commercial basis. Both software products provide a graphical interface to create, modify and simulate high-level Petri nets. Moreover, they provide analysis tools and reporting facilities.
To specify the behaviour of each transition (i.e. the number of tokens produced and the value and delay of each produced token), Design/CPN provides an 'inscription language' (expressions on the input and output arcs of a transition). ExSpect (Executable Specification Tool) uses a specification language to describe the behaviour of a transition. Both languages originate from 'pure' functional languages.
The approach described in this paper uses the software package ExSpect. ExSpect has been developed by the information systems department of Eindhoven University of Technology and is being marketed by Bakkenist.