The Application of Petri Nets to Workflow Management

W.M.P. van der Aalst
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, telephone: -31 40 2474295, e-mail: wsinwa@win.tue.nl

Abstract
Workflow management promises a new solution to an age-old problem: controlling, monitoring, optimizing and supporting business processes. What is new about workflow management is the explicit representation of the business process logic which allows for computerized support. This paper discusses the use of Petri nets in the context of workflow management. Petri nets are an established tool for modeling and analyzing processes. On the one hand, Petri nets can be used as a design language for the specification of complex workflows. On the other hand, Petri net theory provides for powerful analysis techniques which can be used to verify the correctness of workflow procedures. This paper introduces workflow management as an application domain for Petri nets, presents state-of-the-art results with respect to the verification of workflows, and highlights some Petri-net-based workflow tools.

1 Introduction

In former times, information systems were designed to support the execution of individual tasks. Today's information systems need to support the business processes at hand. It no longer suffices to focus on just the tasks. The information system also needs to control, monitor and support the logistical aspects of a business process. In other words, the information system also has to manage the flow of work through the organization. Many organizations with complex business processes have identified the need for concepts, techniques, and tools to support the management of workflows. Based on this need the term *workflow management* was born (cf. [HL91, Kou95]).
Until recently there were no generic tools to support workflow management. As a result, parts of the business process were hard-coded in the applications. For example, an application to support task X triggers another application to support task

Y. This means that one application knows about the existence of another application. This is undesirable, because every time the underlying business process is changed, applications need to be modified. Moreover, similar constructs need to be implemented in several applications and it is not possible to monitor and control the entire workflow. Therefore, several software vendors recognized the need for *workflow management systems*. A workflow management system (WFMS) is a generic software tool which allows for the definition, execution, registration and control of workflows (cf. [Law97]). At the moment many vendors are offering a workflow management system. This shows that the software industry recognizes the potential of workflow management tools.

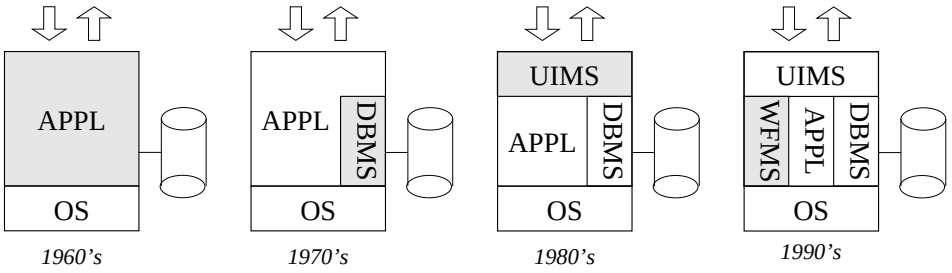

Figure 1: Workflow management systems in a historical perspective.

In order to become aware of the impact of workflow management in the near future, it is useful to consider the evolution of information systems over the last four decades (cf. [Aal96b]). Figure 1 shows the phenomenon of workflow management in a historical perspective. The figure illustrates the evolution of information systems in the last four decades by describing the architecture of a typical information system in terms of its components. In the sixties an information system was composed of a number of stand-alone applications. For each of these applications an application-specific user interface and database system had to be developed, i.e., each application had its own routines for user interaction and data storage and retrieval. In the seventies data was pushed out of the applications. For this purpose database management systems (DBMSs) were developed. By using a database management system, applications were freed from the burden of data management. In the eighties a similar thing happened for user interfaces. The emergence of user interface management systems (UIMSs) enabled application developers to push the user interaction out of the applications. In our opinion workflow management systems are the next step in pushing generic functionality out of the applications. The nineties will be marked by the emergence of workflow software, allowing application developers to push the business procedures out of the applications.
Figure 1 clearly shows that, in essence, the workflow management system is a generic building block to support business processes. Many information systems

could benefit from such a building block, because many organizations are starting to see the need for advanced tools to support the design and execution of business processes. There are several reasons for the increased interest in business processes. First of all, management philosophies such as Business Process Reengineering (BPR) and Continuous Process Improvement (CPI) stimulated organizations to become more aware of the business processes. Secondly, today's organizations need to deliver a broad range of products and services. As a result the number of processes inside organizations has increased. Consider for example mortgages. A decade ago there were just a few types of mortgages, at the moment numerous types are available. Not only the number of products and services has increased, also the lifetime of products and services has decreased in the last three decades. As a result, today's business processes are also subject to frequent changes. Moreover, the complexity of these processes increased considerably. All these changes in the environment of the information system in an average organization, have made business processes an important issue in the development of information systems. Therefore, there is a clear need for a building block named 'workflow management system'.

The main purpose of a workflow management system is the support of the definition, execution, registration and control of *processes*. Because processes are a dominant factor in workflow management, it is important to use an established framework for modeling and analyzing workflow processes [HL91, Kou95, Law97]. In this paper we use a framework based on Petri nets. Petri nets are a well-founded process modeling technique. The classical Petri net was invented by Carl Adam Petri in the sixties ([Pet62]). Since then Petri nets have been used to model and analyze all kinds of processes with applications ranging from protocols, hardware, and embedded systems to flexible manufacturing systems, user interaction, and business processes. In the last two decades the classical Petri net has been extended with color, time and hierarchy ([Aal94, Jen96]). These extensions facilitate the modeling of complex processes where data and time are important factors. There are several reasons for using Petri nets for workflow modeling:

- *formal semantics*
  A workflow process specified in terms of a Petri net has a clear and precise definition, because the semantics of the classical Petri net and several enhancements (color, time, hierarchy) have been defined formally.

- *graphical nature*
  Petri nets are a graphical language. As a result, Petri nets are intuitive and easy to learn. The graphical nature also supports the communication with end-users.

- *expressiveness*
  Petri nets support all the primitives needed to model a workflow process. All the routing constructs present in today's workflow management systems can be modeled. Moreover, the fact that states are represented explicitly, allows for the modeling of milestones and implicit choices.

- *properties*
  In the last three decades many people have investigated the basic properties of Petri nets. The firm mathematical foundation allows for the reasoning about these properties. As a result, there is a lot of common knowledge, in the form of books and articles, about this modeling technique.

- *analysis*
  Petri nets are marked by the availability of many analysis techniques. Clearly, this is a great asset in favor of the use of Petri nets for workflow modeling. These techniques can be used to prove properties (safety properties, invariance properties, deadlock, etc.) and to calculate performance measures (response times, waiting times, occupation rates, etc.). In this way it is possible to evaluate alternative workflows using standard Petri-net-based analysis tools.

- *vendor independent*
  Petri nets provide a tool-independent framework for modeling and analyzing processes. Petri nets are not based on a software package of a specific vendor and do not cease to exist if a new version is released or when one vendor takes over another vendor.

Other references where the use of Petri nets for workflow modeling is advocated are [Aal96b, WR96, MEM94, EKR95, EN93, AH97]. In the remainder of this paper we will show how Petri nets can be applied to the domain of workflow management. To do this, we first introduce the basic concepts of workflow management and workflow management systems. Then we introduce the basic Petri-net terminology. In section 4 we show how the workflow management concepts can be mapped onto Petri nets. Section 5 is concerned with the analysis of workflow processes specified in terms of a Petri net. Before we conclude, we describe a number of Petri-net-based workflow tools to illustrate the use of Petri nets in this domain.

2 Workflow management (systems)

The term workflow management refers to the domain which focuses on the logistics of business processes. There are also people that use the term *office logistics*. The ultimate goal of workflow management is to make sure that the proper activities are executed by the right person at the right time. Although it is possible to do workflow management without using a workflow management system, most people associate workflow management with workflow management *systems*. The *Workflow Management Coalition* (WfMC) defines a workflow management system as follows ([WFM96]): *A system that completely defines, manages, and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic*. Other terms to characterize a workflow management system are: 'business operating system', 'workflow manager', 'case manager' and 'logistic control system'. On the one hand, it is a pity that workflow management is often associated with workflow management systems, because it limits the application domain of workflow management in an unnecessary manner. (It is possible to do workflow management without using a workflow management system.) On the other hand, workflow management systems give concrete form to the essential concepts, techniques, and methods for workflow management.

Workflows are *case-based*, i.e., every piece of work is executed for a specific *case*. Examples of cases are a mortgage, an insurance claim, a tax declaration, an order, or a request for information. Cases are often generated by an external customer. However, it is also possible that a case is generated by another department within the same organization (internal customer). The goal of workflow management is to handle cases as efficiently and effectively as possible. A workflow process is designed to handle similar cases. Cases are handled by executing *tasks* in a specific order. The *workflow process definition* specifies which tasks need to be executed and in what order. Alternative terms for workflow process definition are: 'procedure', 'flow diagram' and 'routing definition'. Since tasks are executed in a specific order, it is useful to identify *conditions* which correspond to causal dependencies between tasks. A condition holds or does not hold (true or false). Each task has pre- and postconditions: the preconditions should hold before the task is executed, and the postconditions should hold after execution of the task. Many cases can be handled by following the same workflow process definition. As a result, the same task has to be executed for many cases. A task which needs to be executed for a specific case is called a *work item*. An example of a work item is: execute task 'send refund form to customer' for case 'complaint sent by customer Baker'. Most work items are executed by a *resource*. A resource is either a machine (e.g. a printer or a fax) or a person (participant, worker, employee). In most offices the resources

are mainly human. However, because workflow management is not restricted to offices, we prefer the term resource. Resources are allowed to deal with specific work items. To facilitate the allocation of work items to resources, resources are grouped into classes. A *resource class* is a group of resources with similar characteristics. There may be many resources in the same class and a resource may be a member of multiple resource classes. If a resource class is based on the capabilities (i.e. functional requirements) of its members, it is called a *role*. If the classification is based on the structure of the organization, such a resource class is called an *organizational unit* (e.g. team, branch or department). A work item which is being executed by a specific resource is called an *activity*. If we take a photograph of a workflow, we see cases, work items and activities. Work items link cases and tasks. Activities link cases, tasks, and resources.

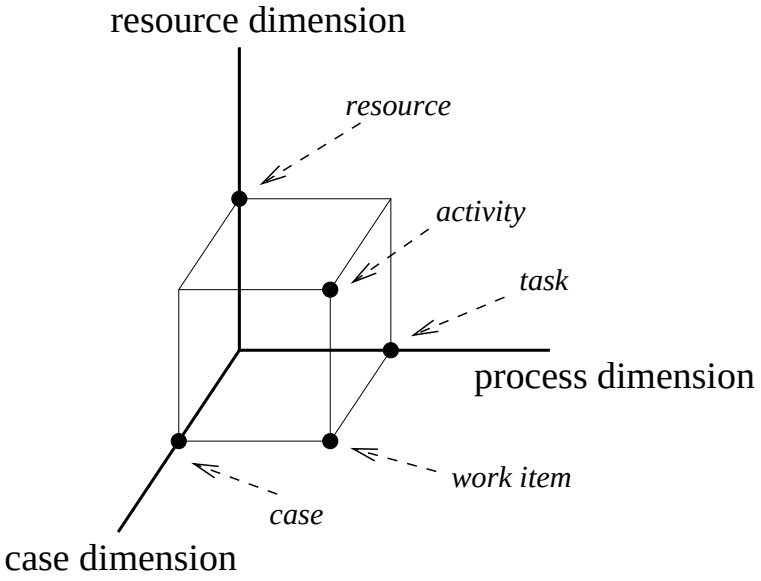

Figure 2: A three dimensional view of a workflow.

Figure 2 shows that a workflow has three dimensions: (1) the case dimension, (2) the process dimension and (3) the resource dimension. The case dimension signifies the fact that all cases are handled individually. From the workflow point of view, cases do not directly influence each other. Clearly they influence each other indirectly via the sharing of resources and data. In the process dimension, the workflow process, i.e., the tasks and the routing along these tasks, is specified. In the resource dimension, the resources are grouped into roles and organizational units. We can visualize a workflow as a number of dots in the three dimensional view shown in Figure 2. Each dot represents either a work item (case + task) or an activity (case + task + resource). Figure 2 shows that workflow management is the glue between the cases, the tasks, and the organization.

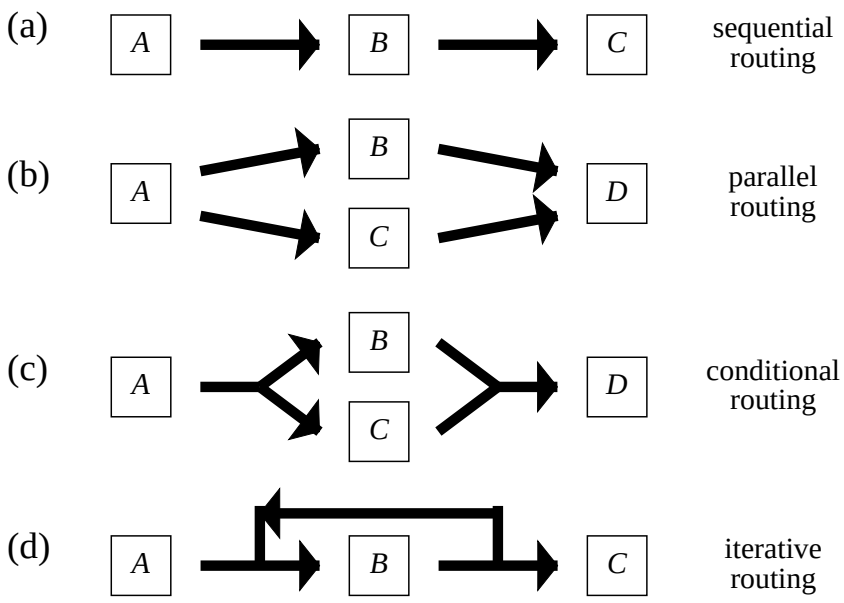

Figure 3: Four routing constructs.

In this paper we focus on the first two dimensions, i.e., we concentrate on the workflow process which is defined to handle cases. We will not discuss human resource aspects such as the classification of resources (organizational modeling) and the mapping of resources to work items (scheduling) in much detail. We will just show the mechanisms. A detailed discussion on resource management is beyond the scope of this paper, because we concentrate on the application of Petri nets. Since Petri nets are a process modeling technique, the application is restricted to the first two dimensions.

For the first two dimensions, the routing of cases is one of the main issues. A workflow process definition specifies how the cases are routed along the tasks that need to be executed. Figure 3 shows the routing constructs identified by the Workflow Management Coalition (WfMC). The WfMC is an international organization whose mission is to promote workflow and establish standards for workflow management systems. The WfMC was founded in 1993 and in January 1995 the WfMC released a glossary which provides a common set of terms for workflow vendors, end-users, developers, and researchers ([WFM96]). In this glossary four types of routing are identified:

- *sequential*
  Tasks are executed sequentially if the execution of one task is followed by the next task. In Figure 3(a) task *B* is executed after task *A* has been completed and before task *C* is started.

- *parallel*
  In Figure 3(b) task *B* and task *C* are executed in parallel. This means that *B* and *C* are executed at the same time or in any order. To model parallel routing, two building blocks are identified: (1) the *AND-split* and (2) the *AND-join*. The AND-split in Figure 3(b) enables *B* and *C* to be executed after *A* has been completed. The AND-join synchronizes the two parallel flows, i.e., task *D* may start after *B and C* have been completed.

- *conditional*
  In Figure 3(c) either task *B* or task *C* (exclusive OR) is executed. To model a choice between two or more alternatives we use two building blocks: (1) the *OR-split* and (2) the *OR-join*. If task *A* is executed, a choice is made between *B* and *C*. Task *D* may start after *B or C* is completed.

- *iteration*
  Sometimes it is necessary to execute a task multiple times. In Figure 3(d) task *B* is executed one or more times.

In Section 4, these routing constructs are mapped onto Petri nets. This way concepts such as case, task, work item, activity and workflow process are defined in a much more explicit manner. The issue of *triggering* is also discussed in Section 4. The triggering concept is very important in the context of workflow management ([Joo94]).

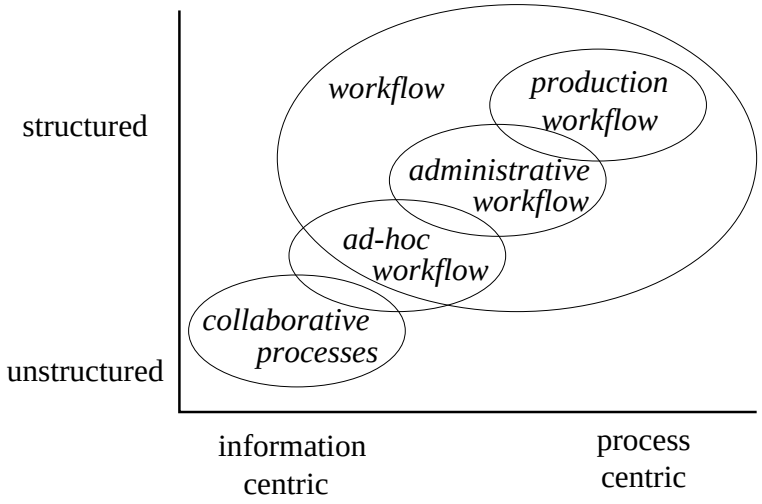

Figure 4: Workflow processes versus collaborative processes.

Not every business process is a workflow process. In our opinion, a workflow pro-

cess is characterized by three qualities. First of all, a workflow process is case-driven. Secondly, the process itself is considered to be essential. Thirdly, the process can be defined in an explicit manner. Many people use the term workflow management in a broader sense. For example, sometimes groupware software tools such as Lotus Notes and Microsoft Exchange are called workflow management systems. This is not correct since these products do not support the workflow process itself, they just allow people to collaborate by sending messages and sharing information. Figure 4 shows the spectrum of case-based business processes. *Production workflow* is concerned which highly structured processes with almost no variations. This type of workflow needs to cope with many cases per day. The processing of insurance claims is a typical example of production workflow. *Administrative workflow* corresponds to case-driven processes which follow a well-defined procedure. Alternative routing of a case is possible, but needs to be predefined. *Ad-hoc workflow* relates to processes where the procedure is not defined (completely) in advance. Per case the procedure needs to be defined or an existing procedure needs to be modified. *Collaborative processes* are outside the scope of our definition of workflow. In a collaborative process the emphasis is on communication and the sharing of information rather than the definition of processes. For collaborative processes it is not possible or not necessary to make the workflow processes explicit. Today's workflow management systems such as COSA (COSA Solutions), Flowmark (IBM), OPEN/Workflow (Eastman Software), Staffware (Staffware) and Visual Workflow (FileNet) support production/administrative workflow. Just a few products support ad-hoc workflow, e.g., Ensemble (FileNet). Collaborative processes can be supported by groupware tools such as Lotus Notes and Microsoft Exchange. Although these groupware tools do not support the logistic control of workflow processes, they can be used as a communication layer in a proprietary workflow system.

In this paper we restrict ourselves to real workflow management systems, i.e., systems to support production workflow, administrative workflow and/or ad-hoc workflow. The WfMC also focuses on this type of software tools. Many vendors and users of these real workflow management systems have joined the WfMC to identify the common characteristics of these tools, to standardize terminology, and define standard architectures and interfaces. One of the first results achieved by the WfMC was the definition of a reference model for the architecture of a workflow system. However, before we describe the reference model, we stop at the subtle difference between the term 'workflow management system' and the term 'workflow system'. A workflow management system is a generic software product which can be applied in many organizations, e.g., Staffware or COSA. However, if the workflow management system is not installed, configured, and filled with data on process definitions and applications, it cannot be used. Therefore, we reserve the

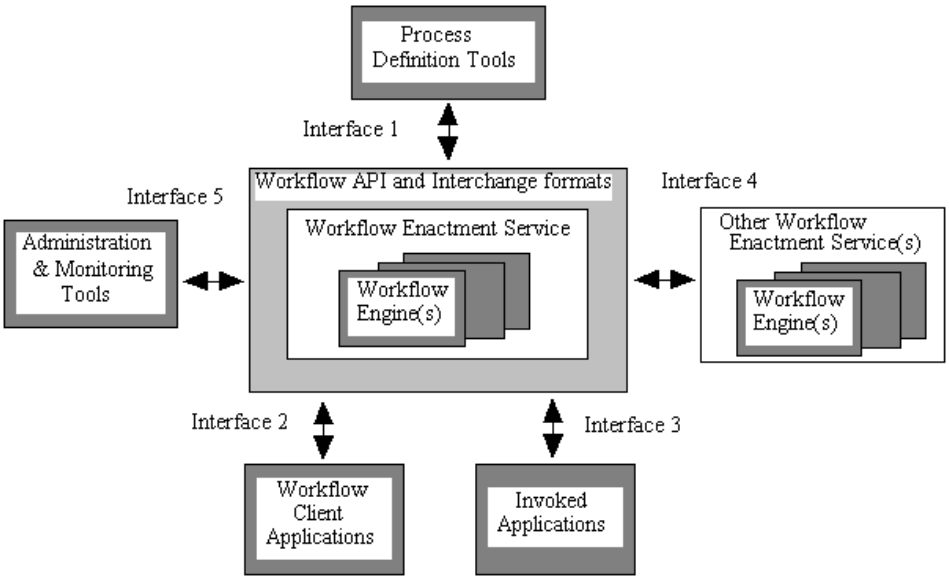

Figure 5: Reference model of the Workflow Management Coalition (WfMC).

term workflow system for the whole of the installed workflow management system, the process definition data, the organizational data, the applications, the application data, the database management system, the configuration files, and other software components in the periphery of the actual workflow management system. Figure 5 shows an overview of this reference model. The reference model describes the major components and interfaces within a workflow architecture. The core of any workflow system is the *workflow enactment service*. The workflow enactment service provides the run-time environment which takes care of the control and execution of the workflow. For technical reasons or managerial reasons the workflow enactment service may use multiple *workflow engines*. A workflow engine handles selected parts of the workflow and manages selected parts of the resources. The *process definition tools* are used to specify and analyze workflow process definitions and/or resource classifications. These tools are used at design time. In most cases, the process definition tools can also be used as a BPR-toolset. Most workflow management systems provide three process definition tools: (1) a tool with a graphical interface to define workflow processes, (2) a tool to specify resource classes (organizational model), and (3) a simulation tool to analyze a specified workflow. The end-user communicates with the workflow system via the *workflow client applications*. An example of a workflow client application is the well-known *in-basket*. Via such an in-basket work items are offered to the end user. By selecting a work item, the user can execute a task for a specific case. If necessary, the workflow engine invokes applications via interface 3. The *adminis-*

tration and monitoring tools are used to monitor and control the workflow. These tools are used to register the progress of cases and to detect bottlenecks. Moreover, these tools are used to set parameters, allocate people and handle abnormalities. Via interface 4 the workflow system can be connected to other workflow systems. To standardize the five interfaces shown in Figure 5, the WfMC aims at a common *Workflow Application Programming Interface* (WAPI). The WAPI is envisaged as a common set of API calls and related interchange formats which may be grouped together to support each of the five interfaces (cf. [Law97]).

The architecture shown in Figure 5 has been adopted by most vendors. For most of the interfaces, standards have been proposed. Unfortunately, these proposed standards are at a technical level with the emphasis on syntax instead of semantics. There is no consensus at a conceptual level. Consider for example interface 1. The syntax of the exchange format has been defined without formalizing the meaning of states and essential building blocks such as the AND/OR-split/join. Therefore, there is a need for a conceptual standard which clearly specifies the semantics of the basic constructs needed. We claim that a workflow framework based on Petri nets would be a good candidate for such a standard. In the remainder of this paper we will substantiate this claim.

3 Petri nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.[1]
Historically speaking, Petri nets originate from the early work of Carl Adam Petri ([Pet62]). Since then the use and study of Petri nets have increased considerably. For a review of the history of Petri nets and an extensive bibliography the reader is referred to [Mur89].

3.1 Classical Petri net

The classical Petri 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.

[1] Note that states are represented by weighted sums and note the definition of (elementary) paths.

Definition 1 (Petri net) *A Petri net is a triple* (P, T, F)*:*

- P *is a finite set of places,*
- T *is a finite set of transitions* ($P \cap T = \emptyset$)*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place. Note that we restrict ourselves to arcs with weight 1. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

At any time a place contains zero of more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token in each output place p of t.

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

- $M_1 \stackrel{t}{\rightarrow} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \stackrel{t}{\rightarrow} M_2$
- $M_1 \stackrel{\sigma}{\rightarrow} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n, i.e., $M_1 \stackrel{t_1}{\rightarrow} M_2 \stackrel{t_2}{\rightarrow} \ldots \stackrel{t_{n-1}}{\rightarrow} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

We use (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M \xrightarrow{*} M'$. Let us define some properties for Petri nets.

Definition 2 (Live) *A Petri net (PN, M) is live iff for every reachable state M' and every transition t, there is a state M'' reachable from M' which enables t.*

Definition 3 (Bounded, safe) *A Petri net (PN, M) is bounded iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less than n. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

Paths connect nodes by a sequence of arcs. A path is elementary if each node is unique. For convenience, we also introduce the alphabet operator α on paths.

Definition 4 (Path, elementary, alphabet) *Let PN be a Petri net. A path C from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \ldots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$. $\alpha(C) = \{n_1, n_2, \ldots, n_k\}$ is the alphabet of C. C is elementary iff for any two nodes n_i and n_j on C, $i \neq j \Rightarrow n_i \neq n_j$.*

Definition 5 (Strongly connected) *A Petri net is strongly connected iff for every pair of nodes (i.e. places and transitions) x and y, there is a path leading from x to y.*

3.2 High-level Petri nets

The classical Petri net allows for the modeling of states, events, conditions, synchronization, parallelism, choice, and iteration. However, Petri nets describing real processes tend to be complex and extremely large. Moreover, the classical Petri net does not allow for the modeling of data and time. To solve these problems, many extensions have been proposed. Three well-known extensions of the basic Petri net model are: (1) the extension with color to model data, (2) the extension with time, and (3) the extension with hierarchy to structure large models. A Petri net extended with color, time, and hierarchy is called a *high-level Petri net*. In this paper we will use these extensions to model specific aspects. However, a formalization of these aspects is beyond the scope of this paper. Therefore, we introduce the three extensions in an informal manner.

- *extension with color*
  Tokens often represent objects (e.g. resources, goods, humans) in the modeled system. Therefore, we often want to represent attributes of these objects. If an insurance claim is modeled by a token in the Petri net, we want to represent attributes such as the name of the claimant, identification number, date, and amount. Since these attributes are not easily represented by a token in a classical Petri net, we extend the Petri net model with *colored* or *typed tokens*. In a colored Petri net each token has a value often referred to as 'color'. Transitions determine the values of the produced tokens on the basis of the values of the consumed tokens, i.e., a transition describes the relation between the values of the 'input tokens' and the values of the 'output tokens'. It is also possible to specify 'preconditions' which take the colors of tokens to be consumed into account.

- *extension with time*
  For real systems it is often important to describe the *temporal behavior* of the system, i.e., we need to model durations and delays. Since the classical Petri net is not capable of handling quantitative time, a timing concept is added. There are many ways to introduce time into the Petri net. Time can be associated with tokens, places, and/or transitions.

- *extension with hierarchy*
  Although timed colored Petri nets allow for a succinct description of many business processes, precise specifications for real systems have a tendency to become large and complex. This is the reason we provide a hierarchy construct, called *subnet*. A subnet is an aggregate of a number of places, transitions, and subsystems. Such a construct can be used to structure large processes. At one level we want to give a simple description of the process (without having to consider all the details). At another level we want to specify a more detailed behavior. The extension with hierarchy allows for such an approach.

For a more elaborate discussion on these extensions and other kinds of high-level Petri nets, the reader is referred to [Jen96, Aal94, Hee94].

4 Mapping workflow management concepts onto Petri nets

Most workflow management systems and methodologies to support workflow management separate the modeling of the workflow process (How?) from the modeling of the structure of the organization and the resources within the organization (By whom?). The distinction between these two aspects is illustrated by Figure 2 which relates the process dimension and the resource dimension. There are many reasons for decoupling these two dimensions when specifying a workflow. The complexity is reduced, reuse is stimulated, and it is possible to modify a process without changing the organizational model (and vice-versa). In this section we will first show that Petri nets are suitable for the modeling of the process dimension. Then we will show how to incorporate the resource dimension.

4.1 Workflow process definition

In the process dimension, it is specified which tasks need to be executed and in what order. Modeling a workflow process definition in terms of a Petri net is rather straightforward: *tasks* are modeled by *transitions*, *conditions* are modeled by *places*, and *cases* are modeled by tokens.
To illustrate the mapping of workflow management concepts onto Petri nets we consider the processing of complaints. First the complaint is registered (task *register*), then in parallel a questionnaire is sent to the complainant (task *send_questionnaire*) and the complaint is evaluated (task *evaluate*). If the complainant returns the questionnaire within two weeks, the task *process_questionnaire* is executed. If the questionnaire is not returned within two weeks, the result of the questionnaire is discarded (task *time_out*). Based on the result of the evaluation, the complaint is processed or not. The actual processing of the complaint (task *process_complaint*) is delayed until the questionnaire is processed or a time-out has occurred. The processing of the complaint is checked via task *check_processing*. Finally, task *archive* is executed.
Figure 6 shows a workflow process definition for the processing of complaints specified in terms of a Petri net. The tasks *register*, *send_questionnaire*, *evaluate*, *process_questionnaire*, *time_out*, *process_complaint*, *check_processing* and *archive* have been modeled by transitions. The transitions *processing_OK* and *processing_NOK* have been added to model the two possible outcomes of executing task *check_processing*. The transitions *no_processing* and *processing_required* have been added for similar reasons. To model the states between tasks, conditions have been added. Each condition is modeled by a place. For example, place *c2* corresponds to the condition 'ready to evaluate complaint'. Condition *c5* is true (i.e. place *c5* con-

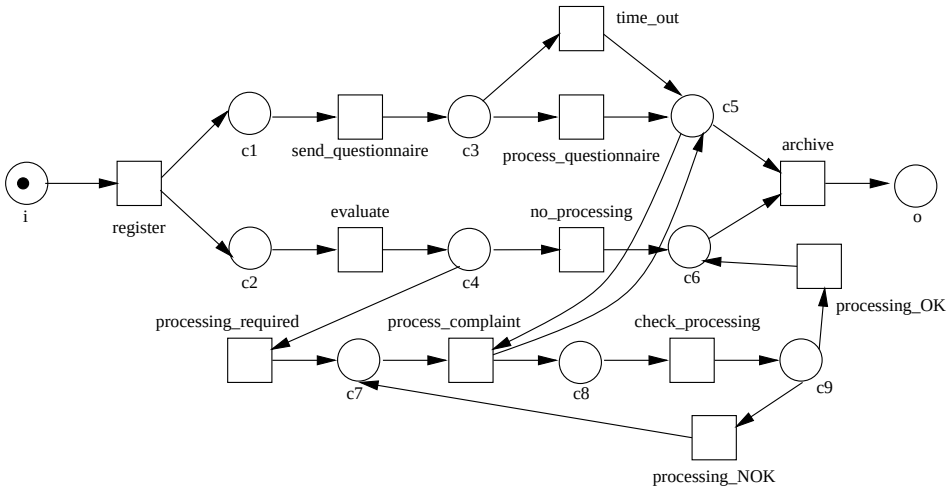

Figure 6: A Petri net for the processing of complaints.

tains a token) if the questionnaire has been processed or a time-out has occurred. Note that *c5* is a prerequisite for the task *archive* and the task *process_complaint*. Condition *i* is the start condition and condition *o* is the end condition.

The workflow process definition shown in Figure 6 models the life-cycle of a single case. In general, there are many cases which are handled according to the same workflow process definition. Each of these cases corresponds to one or more tokens. If tokens of multiple cases reside in the same Petri net, then these tokens may get mixed. For example, transition *archive* may consume two tokens which correspond to different cases. Clearly, this is undesirable. There are two ways to solve this problem. First of all, it is possible to use a high-level Petri net where each token has a value (color) which contains information about the identity of the corresponding case (case identifier). Transitions are not allowed to fire if the case identifiers of the tokens to be consumed do not match, i.e., a precondition is used which inspects and compares token values. Figure 7 shows a high-level Petri net with case identifiers. The dotted lines are used to associate tokens to cases, e.g., the token in place *c1* has value 3 thus linking it to case 3. Another way to solve this problem is the following. Each case corresponds to a unique instance of the Petri net. If there are n cases, then there are n instances of the Petri net. One can think of such an instance as a layer. If these layers are put on top of each other, it is possible to see the cases in the same diagram. The latter is interesting from a management point of view, because one gets an overview of the state of the workflow. For example, if a place contains a lot of tokens, this might indicate a bottleneck. In the remainder of this paper, we consider Petri nets which describe the life-cycle of one case in isolation.

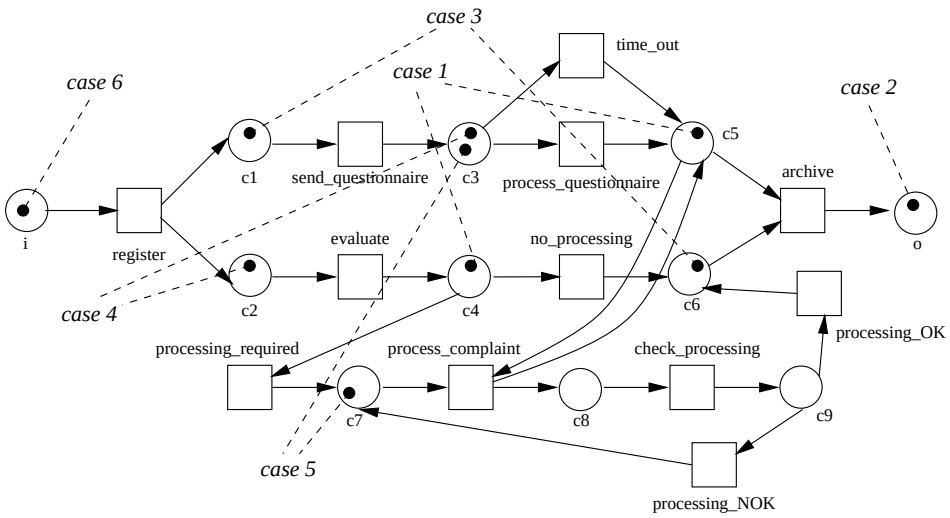

Figure 7: Tokens have a case identifier which allows for the separation of cases.

A Petri net which models a workflow process definition (i.e. the life-cycle of one case in isolation) is called a *WorkFlow net* (WF-net). A WF-net satisfies two requirements. First of all, a WF-net has one input place (i) and one output place (o). A token in i corresponds to a case which needs to be handled, a token in o corresponds to a case which has been handled. Secondly, in a WF-net there are no dangling tasks and/or conditions. Every task (transition) and condition (place) should contribute to the processing of cases. Therefore, every transition t (place p) should be located on a path from place *i* to place *o*. The latter requirement corresponds to strongly connectedness if o is connected to i via an additional transition t^* (see Figure 23).

Definition 6 (WF-net) *A Petri net $PN = (P, T, F)$ is a WF-net (WorkFlow net) if and only if:*

(i) PN has two special places: i and o. Place i is a source place: $\bullet i = \emptyset$. Place o is a sink place: $o\bullet = \emptyset$.

(ii) If we add a transition t^* to PN which connects place o with i (i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected.

Places in the set P correspond to conditions, transitions in the set T correspond to tasks. Note that the requirements stated in Definition 6 are *minimal* requirements. Even if these requirements are satisfied it is still possible to define a workflow process definition with potential deadlocks and/or livelocks.

Tokens in a WF-net represent the *workflow state* of a single case. The workflow

state contains partial information about the state of a case. In addition the case has *workflow attributes* and *application data*. The workflow state corresponds to the distribution of tokens over the places (marking). Consider for example Figure 7. The workflow state of case 5 is *c3* + *c7*. A workflow attribute is a specific piece of information used for the routing of a case. One can think of a workflow attribute as a control variable or a logistic parameter. A workflow attribute may be the age of the complainant, the department responsible for the complaint, or the registration date. We will also use the term *case attribute* to refer to the workflow attribute of a specific case. The classical Petri net $PN = (P, T, F)$ abstracts from these workflow attributes. However, in a high-level Petri net, the extension with color can be used to model workflow attributes. The application data is not used to manage the workflow, it is used to execute the task. The application data is outside the scope of the workflow process definition and the workflow management system has no knowledge of this information. (Nevertheless, often there are workflow attributes which can be derived from the application data.) Examples of application data are the address of the complainant and the evaluation report.

4.2 Routing constructs

In the process dimension, building blocks such as the AND-split, AND-join, OR-split and OR-join are used to model sequential, conditional, parallel, and iterative routing (WfMC [WFM96]). Clearly, a WF-net can be used to specify the routing of cases. In Section 2, four types of routing have been identified: sequential, parallel, conditional, and iteration.

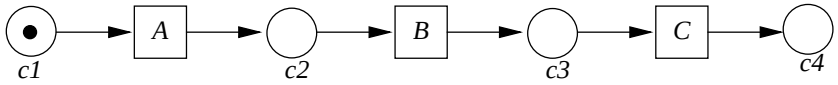

Figure 8: Sequential routing.

Sequential routing is used to deal with causal relationships between tasks. Consider two tasks *A* and *B*. If task *B* is executed after the completion of task *A*, then *A* and *B* are executed sequentially. Figure 8 shows that sequential routing can be modeled by adding places. Place *c2* models the causal relationship between task *A* and task *B*, i.e., place *c2* represents a postcondition for task *A* and a precondition for task *B*. Place *c3* models the causal relationship between task *B* and task *C*.

Parallel routing is used in situations where the order of execution is less strict. For example, two tasks *B* and *C* need to be executed but the order of execution is arbitrary. To model such a parallel routing, two building blocks are used: (1) the

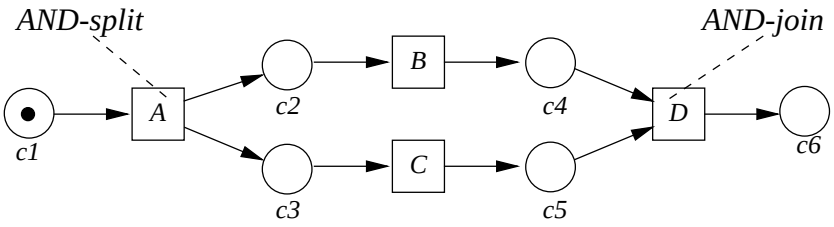

Figure 9: Parallel routing.

AND-split and (2) the AND-join. Figure 9 shows that both building blocks can be modeled by ordinary transitions. One can think of these transitions as *control tasks* that have been added for routing purposes. However, a normal task can also act as an AND-split and/or an AND-join. The execution of AND-split *A* enables both task *B* and task *C*. AND-join *D* is enabled after execution both *B* and *C*, i.e., *D* is used to synchronize two subflows. As a result, task *B* and task *C* are executed in parallel. This means that *B* and *C* are executed in arbitrary order. It is even possible that the execution of *B* and *C* coincides, i.e., for one case, tasks overlap in time. In several workflow management systems, it is not possible to execute two tasks for the same case at the same time. In this case, we use the term *semi-parallel*. If semi-parallel routing is used in Figure 9, then *B* is executed before *C* or vice versa, i.e., interleaving semantics are used to interpret the workflow process definition. Sometimes, semi-parallel routing is used to avoid serious technical problems. For example, for each case, applications and/or tasks in the workflow management system share data. In this case, the locking mechanism in the database mangement system may prevent tasks to be executed concurrently for the same case.

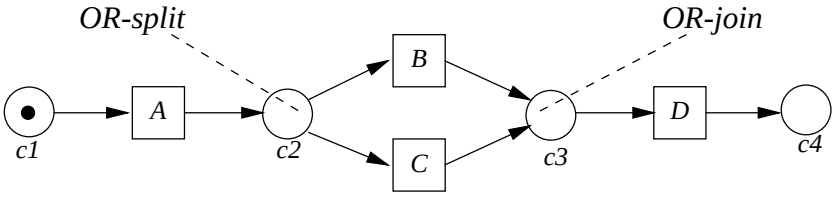

Figure 10: Conditional routing.

Conditional routing is used to allow for a routing which may vary between cases. In this way, the routing of a case may depend on the workflow attributes of a case, the behavior of the environment, or the workload of the organization. To model a choice between two of more alternatives, two building blocks are used: (1) the OR-split and (2) the OR-join (in both cases an exclusive OR). An OR-split can be modeled by a place with multiple outgoing arcs, an OR-join is modeled by a

place with multiple ingoing arcs. Figure 10 shows the situation where task *A* is followed by either task *B* or task *C*, i.e., a choice is made between *B* and *C*. The execution of one of these two tasks is followed by the execution of task *D*. Place *c2* is a precondition for both *B* and *C*. However, just one of these two tasks will be executed for the case in place *c1*.
If *c2* contains a token, a non-deterministic choice is made between *B* and *C*. However, the choice between alternatives often depends on workflow attributes. If the choice is based on workflow attributes, it is a deterministic choice. For example, the routing of an insurance claim may depend on the compensation costs. Therefore, a workflow attribute is used to take into account the compensation costs. If these costs exceed a certain amount, additional checks are necessary. The routing of a traffic violation may depend on the type of the traffic violation represented by the corresponding workflow attribute. There are two ways to model a choice based on workflow attributes. (Recall that the extension with color is used to model workflow attributes.) We can use the construct shown in Figure 10 and add a precondition (i.e. an additional enabling requirement based on one or more workflow attributes) to each of the tasks such that either *B* or *C* is enabled if *c2* contains a token. Another way to model a deterministic choice between *B* or *C* is shown in

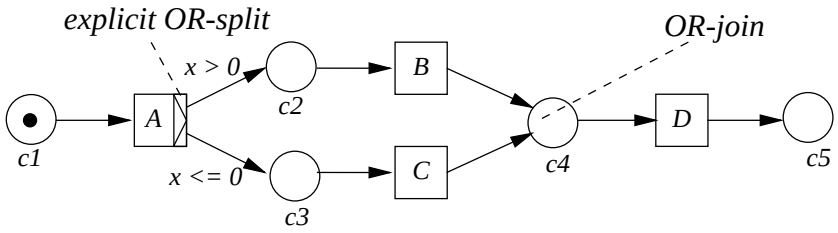

Figure 11: Explicit choice between *B* and *C* based on workflow attribute *x*.

Figure 11. Transition *A* has two output places *c2* and *c3*. Transition *A* produces either a token in *c2* or *c3*. The choice between *c2* or *c3* is based on workflow attribute *x*. If *x* is positive, task *B* will be executed, otherwise task *C*. A special symbol is used to denote the fact that task *A* is an OR-split (exclusive OR). Note that the non-deterministic choice in Figure 10 differs from the choice in Figure 11 with respect to the *moment of choice*. In Figure 11 the choice is made the moment task *A* is completed, in Figure 10 the choice is made the moment *B* or *C* is executed. We will use the term *explicit OR-split* for a choice based on workflow attributes. The term *implicit OR-split* is reserved for the situation where the moment of choice is as late as possible. For workflow modeling it is of the utmost importance to distinguish between implicit and explicit OR-splits.

In Figure 11 we introduced a special notation for an explicit OR-split. We will also

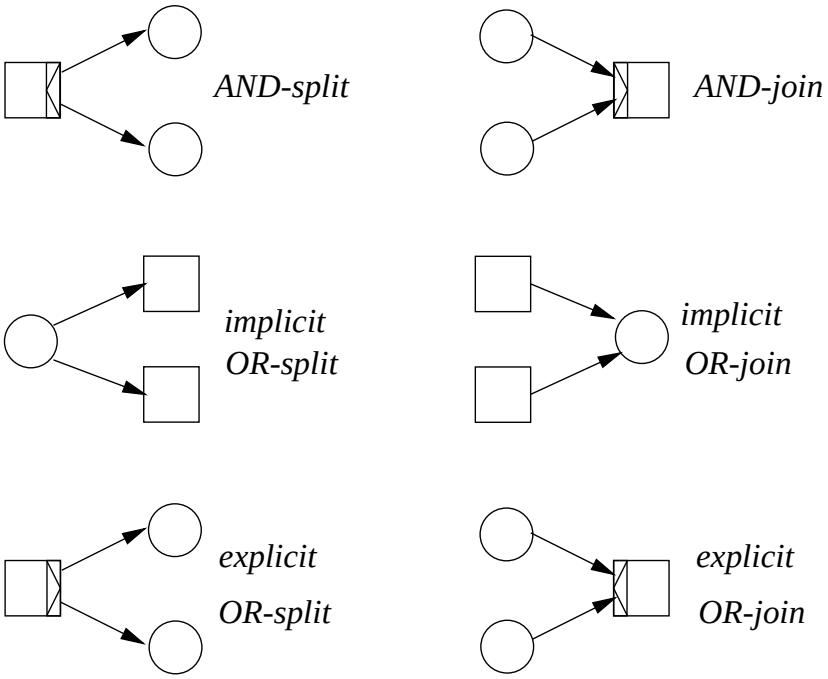

Figure 12: The building blocks for workflow modeling.

denote the fact that a task is an AND-split, an AND-join, and/or an OR-join. Figure 12 lists the main constructs. The AND-split and the AND-join correspond to the normal behavior of a transition in a classical Petri net. The implicit OR-split and OR-join are modeled by places. The explicit OR-split is modeled by a transition which produces one token in one of its output places, i.e., it is an exclusive OR (XOR). The place is selected on the basis of the workflow attributes. The explicit OR-join is modeled by a transition which is enabled if one of the input places contains a token. In general, the explicit OR-join can be modeled by an implicit OR-join (i.e. a place). There is no compelling need to distinguish between implicit and explicit OR-joins. Therefore, we will avoid the use of explicit OR-joins in this paper. However, the distinction between the implicit OR-split and the explicit OR-split is of practical relevance and will be discussed in more detail when the concept of triggering is introduced.

The last form of routing is *iteration*. Iteration can be modeled using the building blocks shown in Figure 12. Consider for example the workflow process definition shown in Figure 13. Task *C* is a control task which checks the result of task *B*. Based on this check, task *B* may be executed once more. In Figure 13 task *B* is executed one or more times. It is also possible to specify that task *B* is executed zero or more times. Iteration is often considered to be an undesirable form of routing,

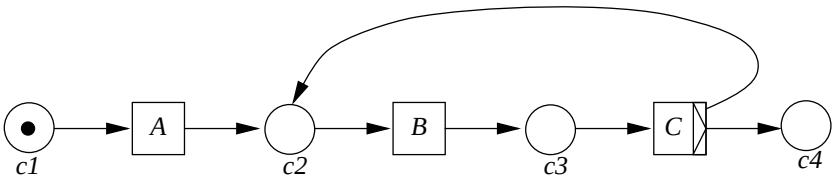

Figure 13: Iteration: *B* may be executed multiple times.

because it corresponds to the repeated execution of the same task without making any real progress. Unfortunately, there are many situations where iteration cannot be avoided. For example, the information supplied by the customer is incomplete or a request is refused.

4.3 Triggering

A workflow process definition, such as the one shown in Figure 6, specifies how a case is routed, i.e., which tasks need to be executed and in what order. For an arbitrary case in a specific state (workflow state and workflow attributes) it is specified which tasks *can* be executed. It is important to note that the fact that if a task can be executed for a specific case, then this does not mean that the task is executed directly. For example, if a task is to be executed by an employee, then the employee has to be available and willing to execute the task. If the employee is ill, on holidays or having lunch, then the task will not be executed. Another example is the processing of a form which is returned by a customer. If the customer does not return the form, the corresponding task cannot be executed. These examples show that for many tasks additional conditions (e.g. availability of resources or information) need to be satisfied. The execution of these tasks cannot be forced by the workflow management system. The workflow management system cannot force customers to return forms nor can it force employees to execute specific tasks. The workflow management system is not in complete control, it just supports the workflow. Therefore, it is important to differentiate between the enabling of a task and the execution of a task. Since the enabling of a task does not imply that the task will be executed (immediately), it is crucial to have this distinction. Therefore, we introduce the concept of *triggering*.
A *trigger* is an external condition which leads to the execution of an enabled task. The execution of a task instance for a specific case starts the moment the task instance is triggered. A task instance can only be triggered if the corresponding case is in a state which enables the execution of the task. In this paper, we distinguish between four types of tasks.

- *Automatic*: a task is triggered the moment it is enabled. This kind of triggering is used for tasks which are executed by an application which does not

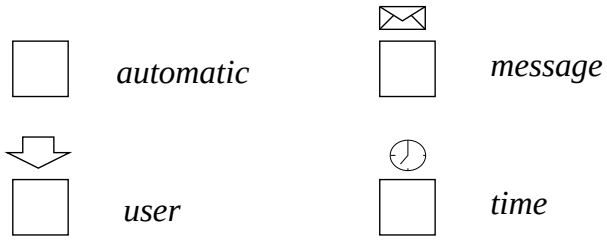

Figure 14: Four types of triggering.

require human interaction.

- *User*: a task is triggered by a human participant, i.e., a user selects an enabled task instance to be executed. In a workflow management system each user has a so-called 'in-basket'. This in-basket contains tasks instances that are enabled and may be executed by the user. By selecting a task instance the corresponding task instance is triggered.
- *Message*: an external event (i.e. a message) triggers an enabled task instance. Examples of messages are telephone-calls, fax messages, e-mails or EDI messages.
- *Time*: an enabled task instance is triggered by a clock, i.e., the task is executed at a predefined time. For example, the task 'remove document' is triggered if a case is trapped in a specific state for more than 15 hours.

Only for automatic tasks do the enabling and the actual start of the execution coincide. In the workflow process definition we will use the symbols shown in Figure 14 to denote the type of trigger required.

Figure 15 shows the workflow process definition for the processing of complaints using the building blocks introduced in Figure 12. The tasks *no_processing*, *processing_required*, *processing_OK* and *processing_NOK* (see Figure 6) are replaced by the OR-split *evaluate* and the OR-split *check_processing*. The diagram is also extended with triggering information. The tasks *register*, *evaluate*, *process_complaint* and *check_processing* require a user trigger because they are executed by human resources. The tasks *send_questionnaire* and *archive* are executed by an application which does not require human interaction. The task *process_questionnaire* can only be executed if the complainant returns the form, i.e., an external trigger is required to execute this task. The task *time_out* requires a time trigger. If the workflow state of a case is such that there is a token in *c3* (Figure 15), then there are two possibilities: either *process_questionnaire* is executed because the complainant returns the form in time or *time_out* is executed the moment a certain period of time

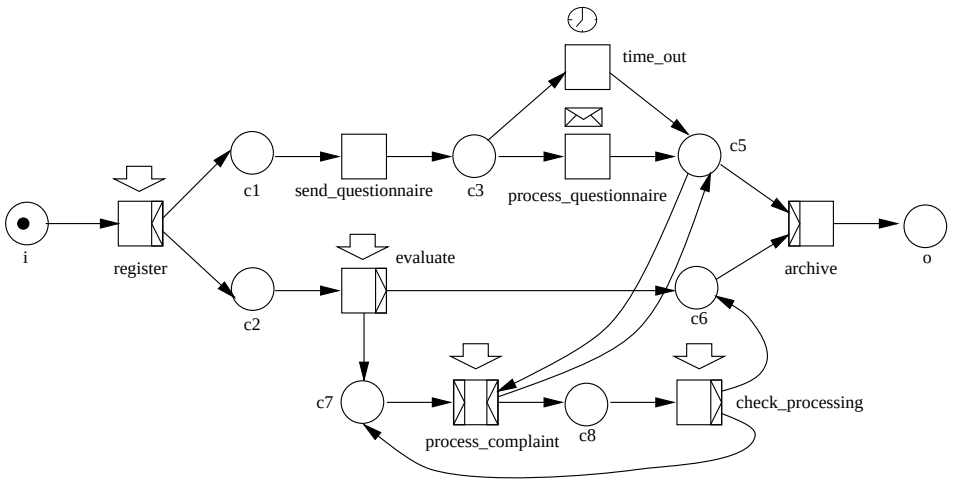

Figure 15: A workflow process definition extended with triggering information.

expires. Note that the moment of choice between *process_questionnaire* and *time_out* is as late as possible. Therefore, an implicit OR-split is used instead of an explicit OR-split.

The introduction of triggers can be used to illustrate the essence of the difference between the implicit and the explicit OR-split. If the implicit OR-split in Figure 15 is replaced by an explicit OR-split (i.e. *send_questionnaire* chooses between *process_questionnaire* and *time_out*), the behavior changes dramatically. For example, it is possible that forms which are returned in time are rejected, or cases are blocked if some of the forms are not returned at all. Another example is given in Figure 16. In both process definitions the execution of task *A* is followed by the execution of *B* or *C*. In the first process definition (a), the moment of choice is as late as possible. After the execution of *A*, there is a 'race' between *B* and *C*. If the external message required for task *C* arrives before someone starts executing task *B*, then *C* is executed, otherwise *B*. In the second workflow process definition (b), the choice is fixed after the execution of *A*. If task *B* is selected, then the arrival of the external message has no influence. If *C* is selected, then *B* cannot be used to bypass *C*. Figure 16 shows the importance of the explicit representation of triggers, states, and the moment of choice. Many workflow management systems abstract from states between subsequent tasks, therefore they have problems modeling implicit OR-splits. They also have problems modeling *milestones* such as *c5* in Figure 15. Clearly, the explicit representation of states is important. Fortunately, Petri nets allow for the explicit representation of states in a very elegant manner.

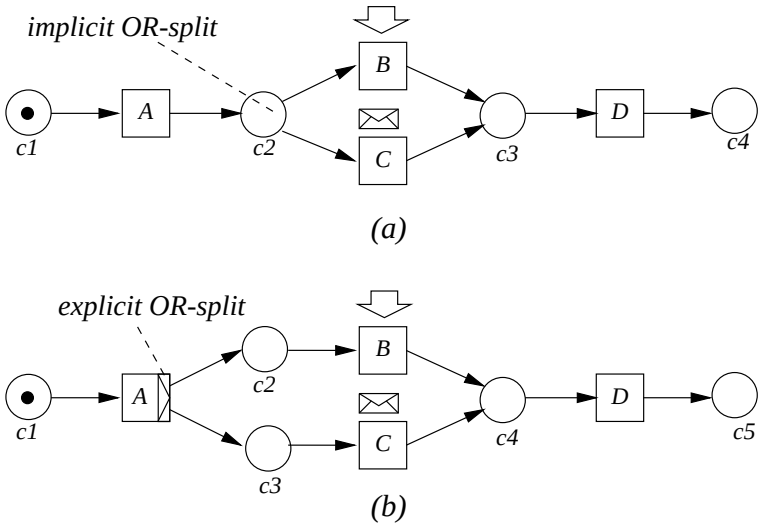

Figure 16: An example to illustrate the difference between (a) the implicit OR-split and (b) the explicit OR-split.

4.4 Tasks, work items and activities

A task corresponds to a generic piece of work. A task is not defined for a specific case, but for a type of cases, i.e., a task may be executed for many cases. To avoid confusion between (1) a task, (2) the enabling of a task, and (3) the execution of a task, we introduce two additional terms: *work item* and *activity*. A work item is a task which is enabled for a specific case, i.e., a work item corresponds to a concrete piece of work. In a sense, a work item is the combination of a task, a case and a trigger (optional). In a workflow management system, work items which correspond to interactive tasks are offered to the user via the so-called in-basket. An activity is the actual execution of a work item, i.e., a task is executed for a specific case. An activity combines a task, a case, a trigger and/or a resource (optional). It is easy to map the terms task, work item, and activity onto Petri-net terminology. A tasks corresponds to a transition, a work item corresponds to an enabled transition (binding) and an activity corresponds to a transition firing. Consider for example Figure 15. Transition *process_complaint* corresponds to a task. If both *c5* and *c7* contain a token with case-identifier 91212123, then the enabled transition *process_complaint* corresponds to a work item. If *process_complaint* fires while consuming the two tokens with case-identifier 91212123, then this firing corresponds to an activity.

One can think of a trigger as an additional condition which needs to be satisfied before a task can be executed. If we model this in terms of Petri nets, a trigger corresponds to a token in an additional input place of the task. Thus far, we have

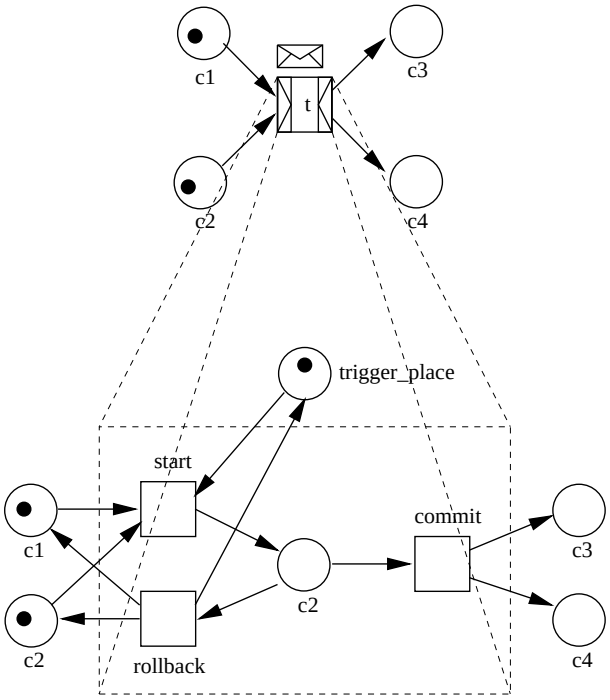

Figure 17: The low-level behavior of a task.

modeled tasks by transitions. By mapping tasks onto transitions we abstract from the internal behavior of a task. This abstraction is convenient for workflow modeling. However, if we consider a task in more detail, then it is clear that the execution of a task takes time, i.e., time elapses between start and completion. In fact, it is possible that during the execution of a task some anomalies occur, e.g., an application crashes, the database is down, or information is missing. Therefore, it is possible that the execution of a task is postponed or canceled. Figure 17 shows a simple Petri net model of the low-level behavior of a task. The start of an activity (transition *start* fires) is followed by either a commit (transition *commit*) or a rollback (transition *rollback*). Note that the trigger is modeled by a token in the place *trigger_place*.

One can think of a task as a *Logical Unit of Work* (LUW) and an activity as a *transaction*. Just like a transaction, an activity should satisfy the well-known *ACID properties*. The acronym ACID stands for:

- *atomicity*
  An activity is executed successfully (commit) or is rolled back completely, i.e., a task cannot be completed partially.

- *consistency*
  The execution of an activity leads to a consistent state.

- *isolation*
  The effect of the execution of an activity in parallel with other activities is equal to the effect of the execution of one activity in isolation.

- *durability*
  The result of a committed activity cannot get lost.

The ACID properties impose technical constraints on the size of a task. Choosing tasks which have the 'right' size is one of the main issues in workflow modeling. If the tasks are too small, then there is a lot of overhead (setup times) and the workflow becomes too complex to manage. If the tasks are too large, then it may become a problem to execute a task in one go, because it is not possible to commit a partially completed task or put parts of the task out to contract. Moreover, the management of resources, i.e., the allocation of work items to users, is not fine-grained enough if the tasks are too large to allow for specialization. Note that the granularity of the management of resources is determined by the size of each task.

4.5 Resource classification

Clearly, Petri nets are well-suited for the modeling of workflow process definitions. However, thus far we focused on the process dimension and not on the resource dimension. The resource dimension is concerned with organizational issues such as the formation of teams and the authorization of users. Most workflow management systems provide a design tool to model the organization. These tools can be used to structure the resource dimension and facilitate the specification of the rules used to allocate work items to resources. We will use the term *resource* for an actor which is able to execute specific tasks. In most environments where workflow management systems are used, the resources are mainly human. However, as indicated in Section 2, we prefer the term resource over employee or participant, because workflow management is not restricted to offices.
For each task, we need to specify *who* is allowed to execute the task. However, explicitly linking resources and tasks is considered to be harmful. If a task is linked to a specific user, then the task will be blocked if the user is absent. The use of links to specific resources reduces flexibility. Moreover, the workflow process definition needs to be modified each time a new employee is hired (or an existing employee is discharged). To avoid links to specific resources, resources are grouped into classes. A *resource class* is a group of resources with similar characteristics. Note that a resource may be a member of multiple resource classes. Consider for example Figure 18. There are four resource classes: *Complaints_department*,

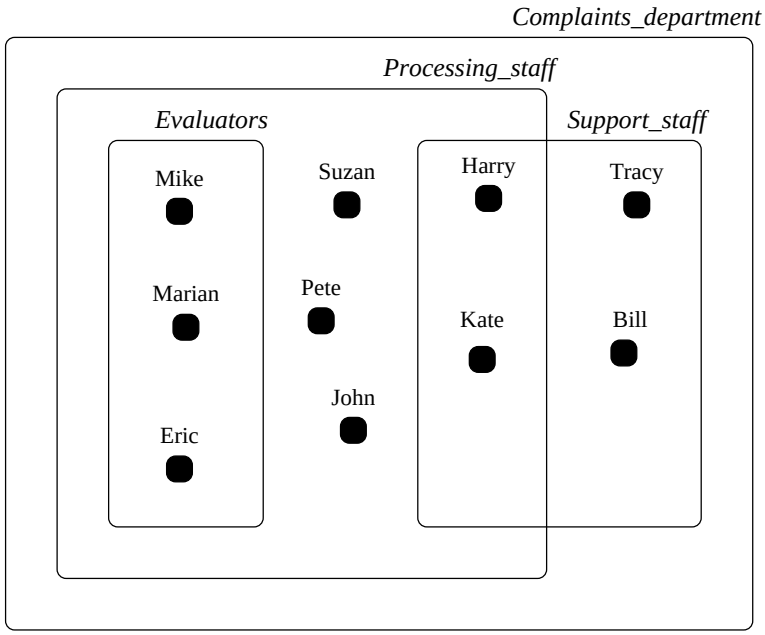

Figure 18: Resource classification.

Evaluators, *Processing_staff* and *Support_staff*. Marian is a member of *Evaluators*, *Processing_staff*, and *Complaints_department*, but she is not a member of the resource class *Support_staff*. The resource classification shown in Figure 18 can be used to specify which resources are allowed to execute the four user tasks in Figure 15. Tasks *register*, *evaluate*, *process_complaint* and *check_processing* should be executed by a member of the class *Complaints_department*. Moreover, *register* requires a member of the class *Support_staff*, *evaluate* requires a member of the class *Evaluators*, *process_complaint* requires a member of the class *Processing_staff*, and *check_processing* requires a member of the class *Evaluators*. Note that without explicitly linking tasks and resources it is specified that Marian is one of the three persons allowed to execute the task *check_processing*.

In Section 2, two types of resource classes were introduced: roles and organizational units. If a resource class is based on the capabilities (i.e. functional requirements) of its members, it is called a *role*. The resource classes *Support_staff*, *Evaluators* and *Processing_staff* are examples of roles. If the classification is based on the structure of the organization, such a resource class is called an *organizational unit* (e.g. team, branch or department). The class *Complaints_department* is an example of resource class which is based on the structure of the organization instead of the capabilities of its members. Many workflow management systems associate a role and an organizational unit with each task. This means that the task should be

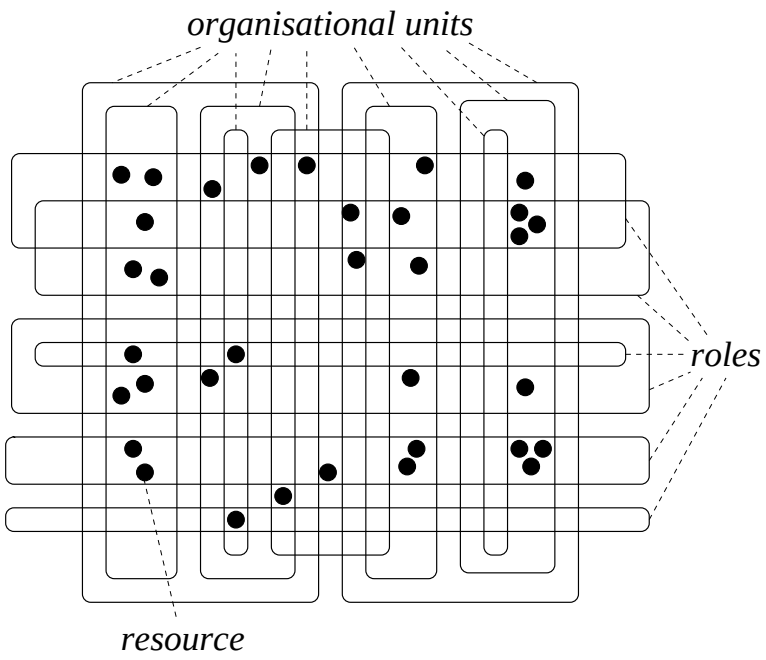

Figure 19: Resource classification: a resource is a member of organizational units and roles.

executed by a resource which is a member of both a resource class based on functional requirements (role) and a resource class based on the structure of the organization (organizational unit). Figure 19 illustrates the orthogonal relation between roles and organizational units.

By associating a role and an organizational unit with each task, the number of resources allowed to execute this task is reduced to the intersection of both classes. Sometimes more advanced concepts are needed to specify who is allowed to execute a specific work item. Sometimes the role or the organizational unit associated with a task is case-dependent. For example, insurance claims with estimated compensation costs of more than 1000 dollars are handled by a senior employee, or complaints about a specific department are handled by team A. It is also possible that specific resources are excluded because they have executed related tasks. For example, the tasks *evaluate* and *check_processing* in Figure 15 should be executed by different resources. This means that if Marian executed the task *evaluate* for a specific case, she is not allowed to execute *check_processing* for the same case (i.e. it should be executed by Mike or Eric). We use the term 'separation of functions' for this additional requirement.

4.6 Resource management

The resource classification and more advanced concepts such as the separation of functions are used to specify who is *allowed* to execute a work item. If there are multiple resources allowed to execute a work item, a choice has to be made. There are two mechanisms to resolve this choice:

- *push control*
  The workflow management system makes a choice and sends each work item to a specific user, i.e., for each work item a resource is chosen. The choice may be based on statistical information, work load, or simple heuristics.

- *pull control*
  The workflow management system sends a copy of each work item to every user who is allowed to execute it. The moment one user selects a work item, all other copies disappear.

In most situations, pull control is the preferred mechanism to distribute work items. Note that push control reduces the flexibility by linking work items to specific resources. The users of the workflow management system often dislike push control because they feel controlled by the system.
The choice between multiple resources allowed to execute a work item is not the only decision to be made. If there are many work items, the order in which these work items have to be executed needs to be decided. Well-known queuing disciplines known from operations management ([Wil89]) can be used to order pending work items: FIFO (first in first out), LIFO (last in first out), SPT (shortest processing time), SRPT (shortest remaining processing time), EDD (earliest due date) and PRIO (tasks with priority go first). If push control is used, the workflow management system will force the execution of work items in a specific order. If pull control is used, the user can deviate from the order suggested by the system. Note that the ordering of work items and the selection of resources are closely related.

Each time a user executes a work item, a setup is required both for the case and the task. To reduce setup times some workflow management systems allow for *chained execution* and *piled execution*. Chained execution corresponds to the execution of multiple subsequent tasks for the same case in one go and by one user. Consider for example the workflow process shown in Figure 15. If the tasks *register*, *send_questionnaire*, and *evaluate* can be executed by a specific resource, then chained execution would mean that if the resource starts the task *register*, the two other tasks will follow automatically (for the same case). Chained execution allows tasks to be clustered into macrotasks. Piled execution corresponds to the repeated execution of one task for multiple cases. For example, a resource executes

the task *register* for a batch of cases. Chained execution reduces the overall setup time needed to get acquainted with the case. Piled execution helps the user to build routine.
A concept which is closely related to chained execution is *case management*. If case management is used, then each case has a case manager. The case manager is responsible for the case and executes as many tasks for the case as possible. The use of case management has two benefits: setup times are reduced and the case manager serves as a contact for the customer whose case is handled. Only a few workflow management systems support advanced concepts such as chained execution, piled execution, and case management.

4.7 Summary

In this section we introduced a number of workflow concepts and showed how these concepts can be mapped onto Petri nets. The four routing constructs identified by the WfMC have been mapped onto WF-nets. The Petri-net formalism also allowed us to experience the subtle difference between the implicit OR-split and the explicit OR-split. More advanced topics such as milestones, triggers, and the difference between tasks, work items, and activities have been illustrated by using some examples. These examples show that the Petri-net formalism is well-suited to deal with the first two dimensions shown in Figure 2. To establish a link between the workflow process definition defined in terms of a WF-net and the third dimension (resource dimension), we discussed the classification of resources and allocation of resources to work items (resource management).

5 Analysis of workflows

In this section we focus on the analysis of workflows. We start with an overview of the various types of workflow analysis. Then we concentrate on a a specific type of analysis: verification. For this purpose, we will introduce a notion of correctness called ‘soundness’ and devote the rest of this section to Petri-net-based analysis techniques to establish soundness. The results presented in this section will illustrate the potential of Petri nets in the workflow domain.

5.1 Introduction

The correctness, effectiveness, and efficiency of the business processes supported by the workflow management system are vital to the organization. A workflow process definition which contains errors may lead to angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of a workflow definition

may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to *analyze* a workflow process definition before it is put into production. Basically, there are three types of analysis:

- *validation*, i.e., testing whether the workflow behaves as expected,
- *verification*, i.e., establishing the correctness of a workflow, and
- *performance analysis*, i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization.

Validation can be done by interactive simulation: a number of fictitious cases are fed to the system to see whether they are handled well. For verification and performance analysis more advanced analysis techniques are needed. Fortunately, many powerful analysis techniques have been developed for Petri nets ([Mur89]). Linear algebraic techniques can be used to verify many properties, e.g., place invariants, transition invariants, and (non-)reachability. Coverability graph analysis, model checking, and reduction techniques can be used to analyze the dynamic behavior of a Petri net. Simulation and Markov-chain analysis can be used for performance evaluation (cf. [MBea95]). The abundance of available analysis techniques shows that Petri nets can be seen as a solver independent medium between the design of the workflow process definition and the analysis of the workflow.

Today's workflow management systems give limited support to performance analysis. Most workflow management systems provide a rudimentary simulator or provide a gateway to a simulation tool. Simulation can be used to estimate key performance indicators by experimenting with the specified workflow under the assumption of a specific behavior of the environment. Examples of key performance indicators are: average throughput time of cases, average waiting time, occupation rates of resources, service levels, and the average number of pending cases.
Most workflow management systems do not give any support for the verification of workflows. As a result, workflow process definitions become operational before they are thoroughly checked for correctness. This often results in runtime errors which need to be repaired on-the-fly at high costs. Both manufacturers and users of workflow management systems see the need for analysis tools which take care of the verification of workflows. Unfortunately, most manufacturers do not have the technology to build such tools. In the remainder of this section we will show that workflows specified in terms of Petri nets can be verified using state-of-the-art analysis techniques.

5.2 Soundness property

Consider the WF-net shown in Figure 20. It is easy to see that this workflow process definition contains several deficiencies. If *time_out_1* and *processing_2* fire, or *time_out_2* and *processing_1* fire, then the WF-net will not terminate properly because a token gets stuck in *c5* or *c4*. If *time_out_1* and *time_out_2* fire, then the task *processing_NOK* will be executed twice and because of the presence of two tokens in *o* the moment of termination is not clear.

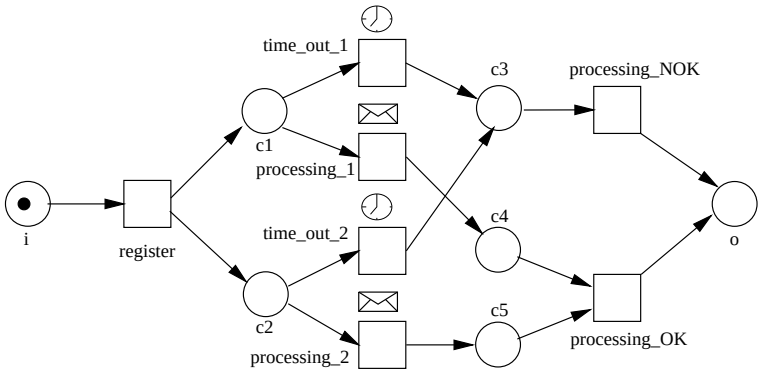

Figure 20: Another WF-net for the processing of complaints.

Clearly the WF-net shown in Figure 20 is not correct, but what is the definition of correctness? The correctness criterion we use in this paper is a set of minimal requirements any workflow process definition should satisfy. First of all, the workflow process definition should satisfy the two requirements listed in Definition 6: (1) a WF-net has a source place i (start condition) and a sink place o (end condition), and (2) each task/condition is on a path from i to o. These requirements can verified statically, i.e., they only relate to the structure of the Petri net. However, there is a third requirement which should be satisfied:

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there is a fourth requirement:

> *There should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net.*

These two additional constraints correspond to the so-called *soundness property*.

Definition 7 (Sound) *A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:*

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:[2]

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \ \wedge \ M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \exists_{M,M'} \ i \xrightarrow{*} M \xrightarrow{t} M'$$

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 7 states that starting from the initial state (state *i*), it is always possible to reach the state with one token in place *o* (state *o*). If we assume fairness (cf. [Val87, Mur89]) then the first requirement implies that eventually state o will be reached. The fairness assumption is reasonable in the context of workflow management: all choices are made (implicitly or explicitly) by applications, humans, or external actors. Clearly, they should not introduce an infinite loop. (We will come back to issue of fairness.) The second requirement in Definition 7 states that the moment a token is put in place *o*, all the other places should be empty. Sometimes the term *proper termination* is used to describe the first two requirements [GCEV72]. The third and last requirement states that there are no dead transitions (tasks) in the initial state *i*, i.e., for each transition t it is possible to reach (starting from i) a state where t is enabled.

Definition 7 assumes that the workflow is specified in terms of a WF-net. Recall that a WF-net is a classical Petri net which satisfies certain properties. However, we have enhanced the workflow process definition with triggers and workflow attributes. The analysis techniques presented in the remainder of this paper will abstract from this additional information. Triggers are simply removed. An AND-split/AND-join corresponds to a normal transition in the Petri net. Tokens have no value (uncolored). However, we cannot map an OR-split and/or OR-join task onto a single transition. An OR-split is unfolded into a small network with a transition for each alternative. An OR-join is also replaced by two or more transitions.

[2] Note that there is an overloading of notation: the symbol *i* is used to denote both the *place i* and the *state* with only one token in place *i* (see Section 3).

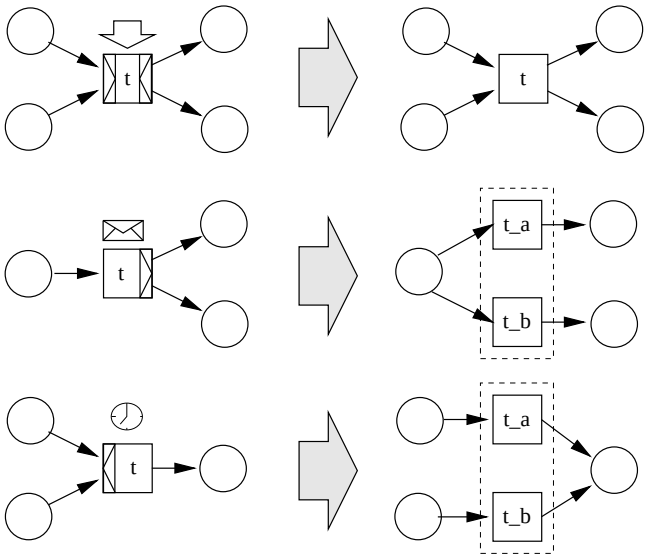

Figure 21: Removing triggers and unfolding tasks to obtain a WF-net.

Figure 21 shows the translation into a WF-net by removing triggers and unfolding tasks.
There are several reasons for abstracting from triggers and workflow attributes. First of all, the behavior of the environment cannot be modeled completely. The environment (e.g. employees and customers) generates triggers and sets workflow attributes. For simulation purposes it is necessary to model the behavior of the environment to a certain extent. However, for verification purposes it is very dangerous to use an explicitly modeled environment. If the real environment differs from the modeled environment, verification is pointless. Incorrect assumptions about the environment, may hide errors in the design of a workflow process definition. Therefore, we abstract from triggers and workflow attributes. We just assume that the environment is willing to generate triggers and each choice is considered to be non-deterministic. Consider for example the task *evaluate* in Figure 15. This task is replaced by two transitions to model the two possible outcomes of the evaluation. In reality, the choice is based on workflow attributes, application data and/or the behavior of the evaluator. Clearly, this cannot be modeled completely. Therefore, we consider the choice to be a non-deterministic one. There are other reasons for abstracting from the triggers and workflow attributes. If we are able to prove soundness for the situation without triggers and workflow attributes, it will also hold for the situation with triggers and workflow attributes. Last but not least, we abstract from triggers and workflow attributes because it allows us to use classical Petri nets instead of high-level Petri nets. From an analysis point of view, the classical Petri net is preferable because of the availability of efficient algorithms and

powerful analysis tools.

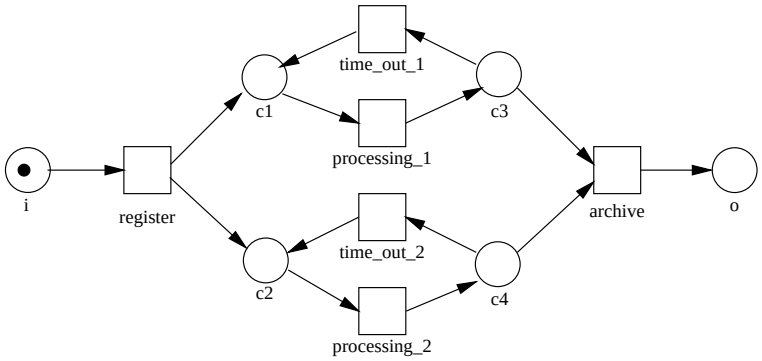

Figure 22: A sound WF-net to illustrate the fairness assumption.

Definition 7 states the requirements to guarantee that it is possible to terminate, i.e., staring in state i it is always possible to reach the final state o. Consider for example the WF-net shown in Figure 22. From any state reachable from state i, there is a firing sequence that leads to state o. However, this does not guarantee that the firing sequence will occur. For example, it is possible that always one of the time-outs occurs before the task *archive* occurs. Therefore, we assume *strong fairness* (cf. [Val87, Mur89]). This means that if we consider a arbitrary infinite firing sequence of the extended net ($\overline{PN}$, see Figure 23), each transition occurs infinitely often. (In the extended net an extra transition t^* is added which connects o and i, thus making the net cyclic.) Strong fairness corresponds to the probablistic interpretation of fairness, e.g., the enabling time of each task is negative exponentially distributed, or enabled tasks are selected in random order (cf. [MBea95]). Assuming soundness and strong fairness, it is guaranteed that each case will end up in place o eventually. A weaker form of fairness is often not sufficient. Consider for example the following notion of fairness: a transition that is enabled infinitely often, will fire eventually. If we assume this notion of fairness, it is not guaranteed that state o will be reached in Figure 22. If the tasks *processing_1*, *time_out_1*, *processing_2*, and *time_out_2* fire alternately, then task *archive* will never become enabled and, in spite of the weak-fairness assumption, termination is not guaranteed. However, it is reasonable to assume strong fairness in the context of workflow management. Moreover, it is quite easy to detect behavior which violates the assumption of strong fairness, e.g., count the number of times each task is executed for a specific case.

5.3 A necessary and sufficient condition for soundness

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. For this purpose we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$. Figure 23 illustrates the relation between PN and $\overline{PN}$. The extended net $\overline{PN}$ can be used to facilitate the verification of the soundness property. The following theorem shows that the extended net allows for the formulation of the soundness property in terms of well-known Petri net properties.

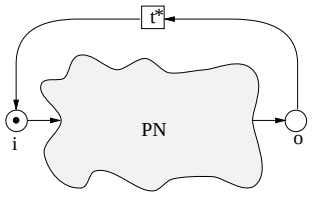

Figure 23: $\overline{PN} = (P, T \cup \{t^*\}, F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\})$.

Theorem 1 *A WF-net PN is sound if and only if $(\overline{PN}, i)$ is live and bounded.*

Proof.
See [Aal97] or [Aal96a]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness. Consider for example the workflow process definition shown in Figure 15. We can use one of the many standard Petri-net-based analysis tools to verify that the extended net is live and bounded. Therefore, the workflow process specified in Figure 15 is guaranteed to behave properly (cf. Definition 7). Theorem 1 is an extension of the results presented in [Aal95, SH95]. In [Aal95] we restrict ourselves to free-choice WF-nets. In free-choice Petri nets it is not allowed to mix choice and synchronization (see Definition 8). Independently, Straub and Hurtado [SH95] found necessary and sufficient conditions for the soundness of COPA nets. (COPA nets correspond to a subclass of free-choice Petri nets.)

5.4 Structural characterizations of soundness

Theorem 1 gives a useful characterization of the quality of a workflow process definition. However, there are a number of problems:

- For a complex WF-net it may be intractable to decide soundness. For arbitrary WF-nets soundness is decidable but also expensive in terms of time and space complexity. If fact, the problem of deciding liveness and boundedness is EXPSPACE-hard (cf. [CEP93]).
- Soundness is a minimal requirement. Readability and maintainability issues are not addressed by Theorem 1.
- Theorem 1 does not show how a non-sound WF-net should be modified, i.e., it does not identify constructs which invalidate the soundness property.

These problems stem from the fact that the definition of soundness relates to the dynamics of a WF-net while the workflow designer is concerned with the static structure of the WF-net. Therefore, it is interesting to investigate structural characterizations of sound WF-nets. For this purpose we introduce three interesting subclasses of WF-nets: free-choice WF-nets, well-structured WF-nets, and S-coverable WF-nets.

5.4.1 Free-choice WF-nets

Most of the workflow management systems available at the moment, abstract from states between tasks, i.e., states are not represented explicitly. These workflow management systems use building blocks such as the AND-split, AND-join, OR-split and OR-join to specify workflow procedures. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. Because these systems abstract from states, every choice is made *inside* an OR-split building block. If we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places (see Figure 21). This means that for these workflow management systems, a workflow procedure corresponds to a *free-choice Petri net.*

Definition 8 (Free-choice) *A Petri net is a free-choice Petri net iff for every two transitions* t_1 *and* t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ *implies* $\bullet t_1 = \bullet t_2$.

It is easy to see that a process definition composed out of AND-splits, AND-joins, OR-splits and OR-joins is free-choice ([Aal97]). If two transitions t_1 and t_2 share an input place ($\bullet t_1 \cap \bullet t_2 \neq \emptyset$), then they are part of an OR-split, i.e., a 'free choice' between a number of alternatives. Therefore, the sets of input places of t_1 and t_2 should match ($\bullet t_1 = \bullet t_2$). Figure 20 shows a free-choice WF-net. The WF-net shown in Figure 15 is not free-choice: *archive* and *process_complaint* share an input place but the two corresponding input sets differ.

We have evaluated many workflow management systems and only a few of these

systems (e.g. COSA, INCOME, and LEU) allow for a construction which is comparable to a non-free choice WF-net. Therefore, it makes sense to consider free-choice Petri nets. Clearly, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property. Another reason for restricting WF-nets to free-choice Petri nets is the following. If we allow non-free-choice Petri nets, then the choice between conflicting tasks *may* be influenced by the order in which the preceding tasks are executed. The routing of a case should be independent of the order in which tasks are executed. A situation where the free-choice property is violated is often a mixture of parallelism and choice. Figure 24 shows such a situation. Firing transition *t1* introduces parallelism. Although there is no real choice between *t2* and *t5* (*t5* is not enabled), the parallel execution of *t2* and *t3* results in a situation where *t5* is not allowed to occur. However, if the execution of *t2* is delayed until *t3* has been executed, then there is a real choice between *t2* and *t5*. In our opinion parallelism itself should be separated from the choice between two or more alternatives. Therefore, we consider the non-free-choice construct shown in Figure 24 to be improper. In literature, the term *confusion* is often used to refer to the situation shown in Figure 24.

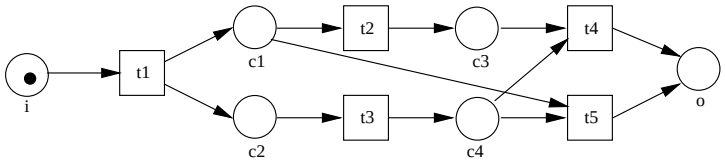

Figure 24: A non-free-choice WF-net containing a mixture of parallelism and choice.

Free-choice Petri nets have been studied extensively (cf. [Bes87, DE95, Des92, Esp90, Hac72]) because they seem to be a good compromise between expressive power and analyzability. It is a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. For example, the well-known Rank Theorem ([DE95]) allows us to formulate the following corollary.

Corollary 1 *A free-choice WF-net can be checked for soundness in polynomial time.*

Proof.
See [Aal96a]. □

Corollary 1 shows that, for free-choice nets, there are efficient algorithms to decide soundness. It can also be shown that sound free-choice WF-nets are safe ([Aal96a]).

Although most workflow management systems only allow for free-choice workflows, free-choice WF-nets are not a completely satisfactory structural characterization of 'good' workflows. On the one hand, there are non-free-choice WF-nets which correspond to sensible workflows (cf. Figure 15). On the other hand there are sound free-choice WF-nets which make no sense. Nevertheless, the free-choice property is a desirable property. If one can model a workflow as a free-choice WF-net, one should do so. A workflow specification based on a free-choice WF-net can be enacted by most workflow systems. Moreover, a free-choice WF-net allows for efficient analysis techniques and is easy to understand. Non-free-choice constructs such as the construct shown in Figure 24 are a potential source of anomalous behavior (e.g. deadlock) which is difficult to trace.

5.4.2 Well-structured WF-nets

Another approach to obtain a structural characterization of 'good' workflows, is to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. As shown in Figure 25, an AND-split should be complemented by an AND-join and an OR-split should be complemented by an OR-join.

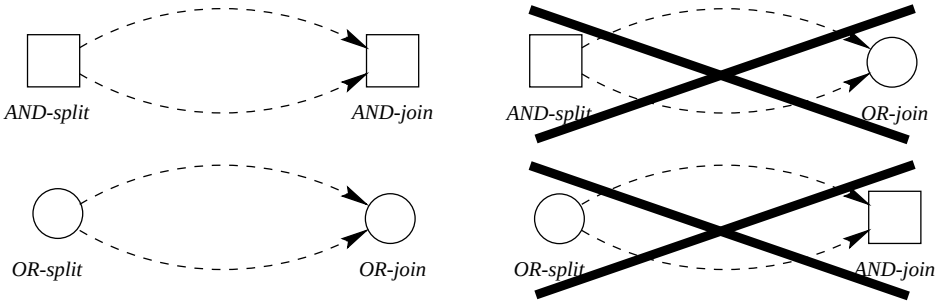

Figure 25: Good and bad constructions.

One of the deficiencies of the WF-net shown in Figure 20 is the fact that the AND-split *register* is complemented by the OR-join *c3* or the OR-join *o*. To formalize the concept illustrated in Figure 25 we give the following definition.

Definition 9 (Well-handled, well-structured) *A Petri net PN is well-handled iff for any pair of nodes x and y such that one of the nodes is a place and the other a transition and for any pair of elementary paths C_1 and C_2 leading from x to y, $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$. A WF-net PN is well-structured iff the extended net $\overline{PN}$ is well-handled.*

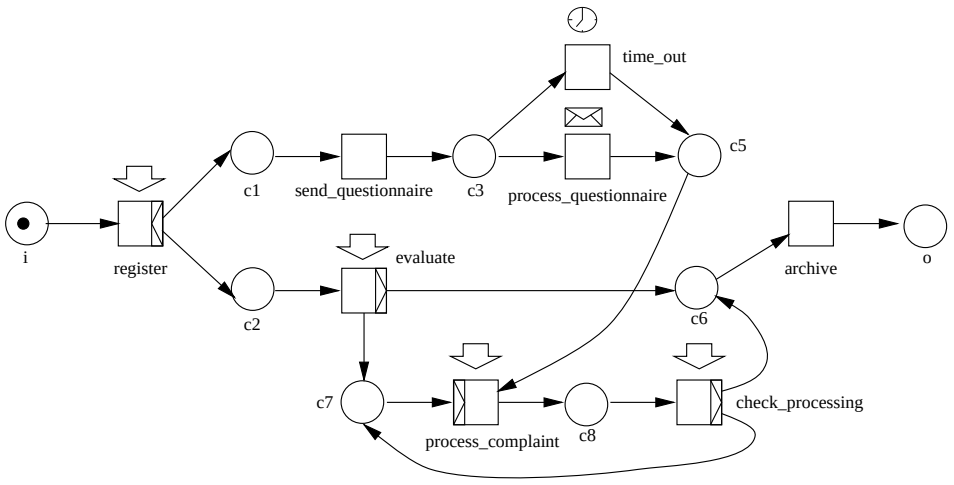

Figure 26: A workflow process definition which is not well-structured.

Consider for example the alternative complaints handling process shown in Figure 26. The WF-net is not well-structured because the AND-split *register* is complemented by the OR-join *c6*: there are two disjunct paths leading from *register* to *c6*, one via *c2* and *evaluate*, and one via *c1*, *send_questionnaire*, *c3*, *process_questionnaire*, *c5*, *process_complaint*, *c8* and *check_processing*. These two paths highlight the fact that the workflow process shown in Figure 26 is not sound. If task *evaluate* puts a token in *c6*, then a token gets trapped in the second path (e.g. in place *c5*). If the arc from *c5* to *process_complaint* is replaced by an arc from *c5* to *archive*, then the AND-split *register* is complemented by the AND-join *archive* and the WF-net is sound.
Figure 20 shows another example of a WF-net that is not well-structured. The WF-net is not well-structured because AND-split *register* is complemented by *c3*.

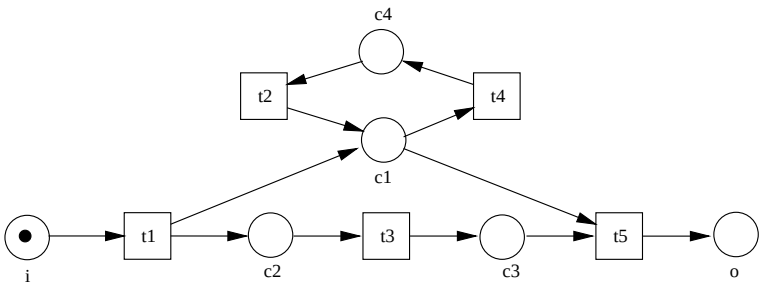

Figure 27: A well-structured WF-net.

Figure 27 shows a typical example of a well-structured WF-net which is not free-choice. The WF-net shown in Figure 22 is another example of well-structured net

which is not free-choice. Although both WF-nets are not free-choice, it is possible to verify soundness for such a WF-net very efficiently.

Corollary 2 *A well-structured WF-net can be checked for soundness in polynomial time.*

Proof.
See [Aal96a]. The proof uses the results presented in [ES90] and [BCD95]. □

Well-structured WF-nets and free-choice WF-nets have similar properties. In both cases soundness can be verified very efficiently and soundness implies safeness. In spite of these similarities, there are sound well-structured WF-nets which are not free-choice (Figure 27) and there are sound free-choice WF-nets which are not well-structured. In fact, it is possible to have a sound WF-net which is neither free-choice nor well-structured (Figures 15 and 24).

5.4.3 S-coverable WF-nets

What about the sound WF-nets shown in Figure 15 and Figure 24? The WF-net shown in Figure 24 can be transformed into a free-choice well-structured WF-net by separating choice and parallelism. The WF-net shown in Figure 15 cannot be transformed into a free-choice or well-structured WF-net without yielding a much more complex WF-net. Place *c5* acts as some kind of milestone which is tested by the task *process_complaint*. Traditional workflow management systems which do not make the state of the case explicit, are not able to handle the workflow specified by Figure 15. Only workflow management systems such as COSA ([SL96]) have the capability to enact such a state-based workflow. Nevertheless, it is interesting to consider generalizations of free-choice and well-structured WF-nets: *S-coverable WF-nets* can be seen as such a generalization.

Definition 10 (S-coverable) *A WF-net PN is S-coverable iff the extended net $\overline{PN} = (P, T, F)$ satisfies the following property. For each place p there is subnet $PN_s = (P_s, T_s, F_s)$ such that: $p \in P_s$, $P_s \subseteq P$, $T_s \subseteq T$, $F_s \subseteq F$, PN_s is strongly connected, PN_s is a state machine (i.e. each transition in PN_s has one input and one output arc), and for every $q \in P_s$ and $t \in T$: $(q,t) \in F \Rightarrow (q,t) \in F_s$ and $(t,q) \in F \Rightarrow (t,q) \in F_s$.*

This definition corresponds to the definition given in [DE95]. A subnet PN_s which satisfies the requirements stated in Definition 10 is called an *S-component*. PN_s is a strongly connected state machine such that for every place q: if q is an input (output) place of a transition t in $\overline{PN}$, then q is also an input (output) place of t in PN_s.

The WF-nets shown in Figure 15 and Figure 24 are S-coverable. The WF-net shown in Figure 26 is not S-coverable because *c5* is not in any S-component. The following theorem shows that a S-coverability is a generalization of the free-choice property and well-structuredness.

Theorem 2 *A sound free-choice WF-net is S-coverable. A sound well-structured WF-net is S-coverable.*

Proof.
See [Aal96a]. □

All the sound WF-nets presented in this paper are S-coverable. Every S-coverable WF-net is safe. The two WF-nets which are not sound, see Figure 26 and Figure 20, are not S-coverable. These examples show that there is a high correlation between S-coverability and soundness. It seems that S-coverability is one of the basic requirements any workflow process definition should satisfy. From a formal point of view, it is possible to construct WF-nets which are sound but not S-coverable. Typically, these nets contain places which do not restrict the firing of a transition, but which are not in any S-component. (See for example Figure 65 in [Rei85].) From a practical point of view, these WF-nets are to be avoided. WF-nets which are not S-coverable are difficult to interpret because the structural and dynamical properties do not match. For example, these nets can be live and bounded but not structurally bounded. There is no practical need for using constructs which violate the S-coverability property. Therefore, we consider S-coverability to be a basic requirement any WF-net should satisfy.

S-coverability can be verified in polynomial time. Unfortunately, in general it is not possible to verify soundness of an S-coverable WF-net in polynomial time. The complexity of deciding soundness for an S-coverable WF-net is PSPACE-complete. For most applications this is not a real problem. In most cases the number of tasks in one workflow process definition is less than 100 and the number of states is less than 200.000. Tools using standard techniques such as the construction of the coverability graph have no problems in coping with these workflow process definitions.

5.5 Summary

The three structural characterizations (free-choice, well-structured, and S-coverable) turn out to be very useful for the analysis of workflow process definitions. S-coverability is a desirable property any workflow definition should satisfy. Constructs violating S-coverability can be detected easily and tools can be build to help the

designer to construct an S-coverable WF-net. S-coverability is a generalization of well-structuredness and the free-choice property (Theorem 2). Both well-structuredness and the free-choice property also correspond to desirable properties of a workflow. A WF-net satisfying at least one one of these two properties can be analyzed very efficiently. However, we have shown that there are workflows that are not free-choice and not well-structured. Consider for example Figure 15. The fact that task *process_complaint* tests whether there is a token in *c5* prevents the WF-net from being free-choice or well-structured. Although this is a very sensible workflow, most workflow management systems do not support such an advanced routing construct. Even if one is able to use state-based workflows (e.g. COSA) allowing for constructs which violate well-structuredness and the free-choice property, then the structural characterizations are still useful. If a WF-net is not free-choice or not well-structured, one should locate the source which violates one of these properties and check whether it is really necessary to use a non-free-choice or a non-well-structured construct. If the non-free-choice or non-well-structured construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors.

6 Toolset

Workflow management is a very rewarding application domain for Petri nets. On the one hand, Petri nets allow for the design of complex workflows using advanced routing constructs. On the other hand, powerful analysis techniques can be used to verify the correctness of a workflow process definition. Unfortunately, most workflow management systems are not based on Petri nets. There are just a few products which use Petri nets as a design language, e.g., COSA (Software Ley/COSA Solutions, Pullheim, Germany), INCOME (Promatis, Karlsbad, Germany), and LEU (Vebacom, Bochum, Germany).

We have a lot of practical experience with the application of COSA. COSA is one of the leading products at the Dutch workflow market. The COSA workflow management system consists of a number of components:

- *COSA Network Editor (CONE)*
  CONE is a tool to design workflow process definitions. Figure 28 shows a window of CONE.

- *COSA User Editor (COUE)*
  COUE is used for the classification of resources in terms of roles and organizational units.

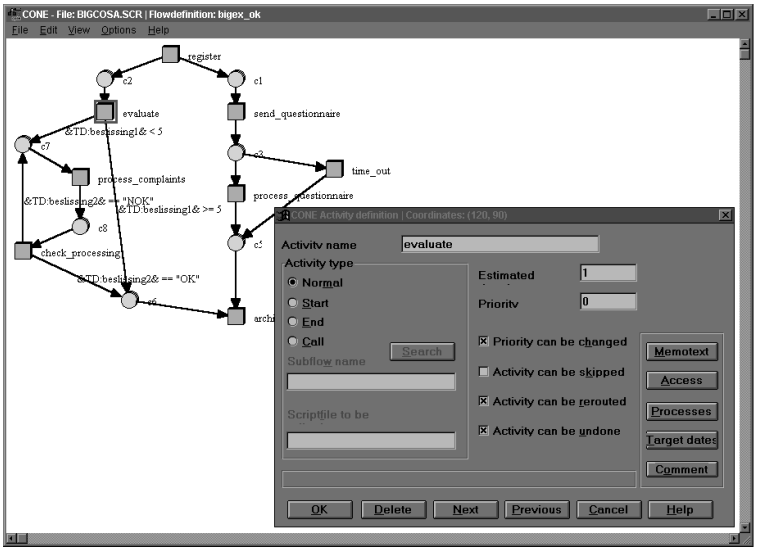

Figure 28: A COSA workflow process definition made with CONE.

- *COSA Simulator (COSI)*
  COSI is a primitive simulation tool which can be used to validate workflow process definitions.

- *COSA Runtime Service (CORS)*
  CORS is a workflow enactment service which consists of one or more workflow engines.

- *COSA MemoBox (COMB)*
  COMB is a workflow client application (in-basket) which offers work items to the end user.

- *COSA Networkstatus Displayer (COND)*
  COND is an extension of COMB which allows the end user to see the workflow state of a case.

- *COSA Administrator (COAD)*
  COAD is an administration and monitoring tool which can be used to handle abnormalities, execute changes, detect problems, and collect management information.

COSA is based on Petri nets: CONE uses Petri nets as a design language, COSI and COND visualize the workflow in terms of a Petri net, and CORS uses the firing rule to enact the workflow.

COSA allows for the modeling and enactment of complex workflow processes which use advanced routing constructs. However, COSA does not support verification, it does not allow for detailed simulation, and it is difficult to use for users which are just interested in business process modeling. We use COSA in combination with three other products to overcome these problems: *Protos*, *ExSpect*, and *Woflan*. These products can interface with each other. Figure 29 shows the relations between COSA, Protos, ExSpect, and Woflan.

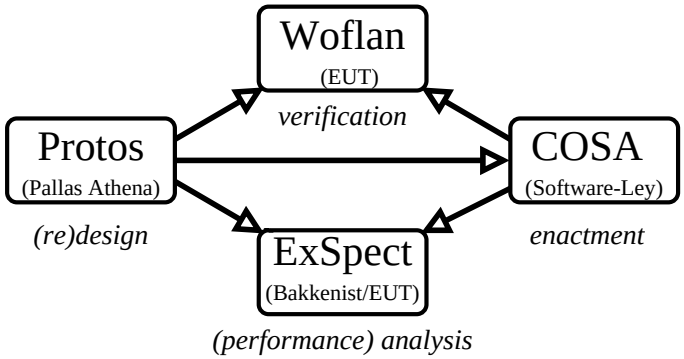

Figure 29: An integrated set of tools for the design, analysis and enactment of workflows.

Protos (Pallas Athena, Plasmolen, The Netherlands) is a so-called BPR-tool. Protos supports Business Process Reengineering (BPR) efforts and can be used to model and analyze business processes ([Pal97]). The tool is very easy to use and is based on Petri nets. To facilitate the modeling of simple workflows by users not familiar with Petri nets, it is possible to abstract from states. Another example of a BPR-tool based on Petri nets is Structware/BusinessSpecs (IvyTeam, Zug, Switzerland). This tool can be used to model and simulate business processes which can be exported to Staffware (Staffware, London, United Kingdom). Staffware is one of the leading workflow products in the world.

ExSpect (Bakkenist Management Consultants, Diemen, The Netherlands) can be used to model and simulate processes modeled in terms of high-level Petri nets ([Bak96]). (The first versions of ExSpect have been developed by members of the department of Mathematics and Computing Science of Eindhoven University of Technology.) Workflow processes modeled with Protos or COSA can be imported into ExSpect. During the simulation of a workflow process with ExSpect, the workflow is animated and key performance indicators (e.g. average throughput time and occupation rate) are measured.

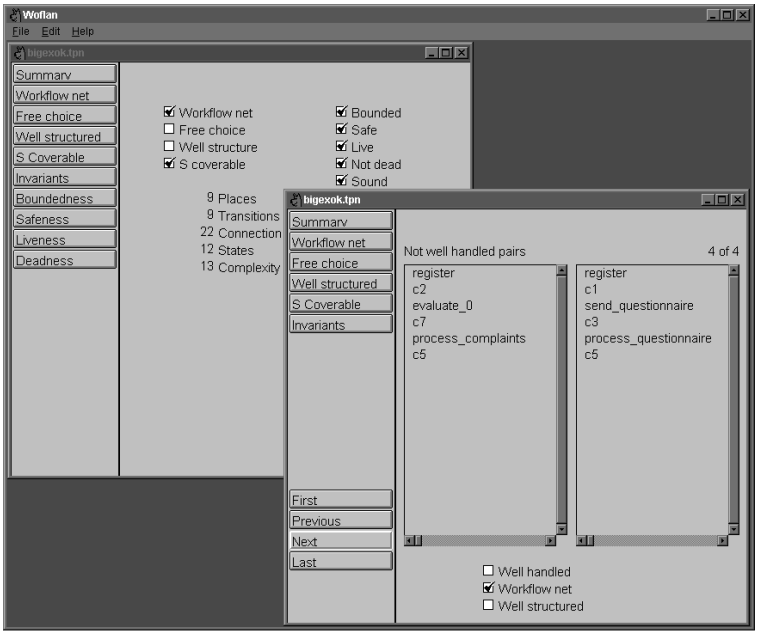

Figure 30: One of the many analysis windows of Woflan.

Woflan (Eindhoven University of Technology, Eindhoven, The Netherlands) is a Petri-net-based workflow analyzer ([HVA97, AHV97]). Woflan (WOrkFLow ANalyzer) can be used to verify the correctness of a workflow. The tool is workflow management system independent, i.e., a number of import functions to download workflow scripts in Woflan (e.g. from COSA and Protos) are provided. Woflan uses standard Petri-net-based analysis techniques. However, the analysis results are presented in such a manner that end-users are able to understand the output of Woflan. Moreover, Woflan guides the end-user in correcting an erroneous workflow. One of the central issues which is analyzed by Woflan is the soundness property. Deciding whether the workflow definition is sound is not sufficient. In many cases more requirements need to be satisfied. Moreover, if the workflow definition is not sound, then the user should be guided in detecting the source of the error and support should be given to repair the error. This is why Woflan offers a large selection of analysis methods including the methods discussed in this paper. For example, Woflan detects constructs which violate the free-choice property, well-structuredness or S-coverability, calculates invariants, detects unbounded places, and reports dead transitions.

7 Conclusion

In this paper we discussed the application of Petri nets to the workflow domain. Workflow management turns out to be an application domain which could benefit from the features of Petri nets. There are at least three good reasons for using Petri nets for workflow modeling and analysis ([Aal96b]):

- *Formal semantics despite the graphical nature*
  On the one hand, Petri nets are a graphical language which allows for the modeling of the workflow primitives identified by the WfMC. On the other hand, the semantics of Petri nets (including most of the extensions) have been defined formally. Many of today's available workflow management systems provide ad-hoc constructs to model workflow procedures. Moreover, there are workflow management systems that impose restrictions on the workflow primitives shown in Figure 12. Some workflow management systems also provide exotic constructs whose semantics are not 100% clear. Because of these problems it is better to use a well-established design language with formal semantics.

- *State-based instead of event-based*
  In contrast with many other process modeling techniques, the state of case can be modeled explicitly in a Petri net. Process modeling techniques ranging from informal techniques such as dataflow diagrams to formal techniques such as process algebra's are *event-based*, i.e., transitions are modeled explicitly and the states between subsequent transitions are modeled implicitly. Today's workflow management systems are typically event-based, i.e., tasks are modeled explicitly and states between subsequent tasks are suppressed. The distinction between an event-based and a state-based description seems to be very subtle, but examples in this paper (e.g. Figure 16) show that this is of the utmost importance for workflow modeling. For example, in most workflow management systems which abstract from states it is not possible to use the implicit OR-split.

- *Abundance of analysis techniques*
  Petri nets are marked by the availability of many analysis techniques. Clearly, this is a great asset in favor of a Petri nets. In this paper, we focused on the verification of workflows. We have seen that Petri-net-based analysis techniques can be used to determine the correctness of a workflow process definition. The availability of these techniques illustrates that Petri-net theory can be used to add powerful analysis capabilities to the next generation of workflow management systems.

Today's situation with respect to workflow management software is comparable to the situation as regards to database management software in the early seventies. In the beginning of the seventies most of the pioneers in the field of DBMSs were using their own ad-hoc concepts. This situation of disorder and lack of consensus resulted in an incomprehensive set of DBMSs. However, emerging standards such as the Relational Data Model [Cod70] and the Entity-Relationship Model [Che76] lead to a common formal basis for many DBMSs. As a result, the use of these DBMSs boosted. There are many similarities between today's workflow management systems and the DBMSs of the early seventies. Despite the efforts of the Workflow Management Coalition a real conceptual standard is missing. As a result, many organizations are reluctant to use existing workflow management software. In our opinion Petri nets constitute a good basis for standardization. We have just given three solid reasons for using a Petri-net-based workflow management system. Inspired by practical experiences, we have come to realize that many of the features of the Petri net formalism are useful in the context of workflow management.

References

[Aal94] W.M.P. van der Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[Aal95] W.M.P. van der Aalst. A class of Petri net for modeling and analyzing business processes. Computing Science Reports 95/26, Eindhoven University of Technology, Eindhoven, 1995.

[Aal96a] W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996.

[Aal96b] W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, Nov 1996.

[Aal97] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.

[AH97] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Modellen, Methoden en Systemen (in Dutch)*. Academic Service, Schoonhoven, 1997.

[AHV97] W.M.P. van der Aalst, D. Hauschildt, and H.M.W. Verbeek. A Petri-net-based Tool to Analyze Workflows. In B. Farwer, D. Moldt, and M.O. Stehr, editors, *Proceedings of Petri Nets in System Engineering (PNSE'97)*, pages 78–90, Hamburg, Sept 1997. University of Hamburg (FBI-HH-B-205/97).

[Bak96] Bakkenist Management Consultants. *ExSpect 5.0 User Manual*, 1996.

[BCD95] K. Barkaoui, J.M. Couvreur, and C. Dutheillet. On liveness in Extended Non Self-Controlling Nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, Berlin, 1995.

[Bes87] E. Best. Structure theory of Petri nets: the free choice hiatus. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 168–206. Springer-Verlag, Berlin, 1987.

[CEP93] A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In R.K. Shyamasundar, editor, *Foundations of software technology and theoretical computer science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, 1993.

[Che76] P.P. Chen. The Entity-Relationship Model: Towards a unified view of Data. *ACM Transactions on Database Systems*, 1:9–36, Jan 1976.

[Cod70] E.F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[DE95] J. Desel and J. Esparza. *Free choice Petri nets*, volume 40 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1995.

[Des92] J. Desel. A proof of the Rank theorem for extended free-choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, Berlin, 1992.

[EKR95] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C. Ellis, editors, *Conf. on Organizational Computing Systems*, pages 10 – 21. ACM SIGOIS, ACM, Aug 1995. Milpitas, CA.

[EN93] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.

[ES90] J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.

[Esp90] J. Esparza. Synthesis rules for Petri nets, and how they can lead to new results. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 182–198. Springer-Verlag, Berlin, 1990.

[GCEV72] K. Gostellow, V. Cerf, G. Estrin, and S. Volansky. Proper Termination of Flow-of-control in Programs Involving Concurrent Processes. *ACM Sigplan*, 7(11):15–27, 1972.

[Hac72] M.H.T. Hack. Analysis production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1972.

[Hee94] K.M. van Hee. *Information System Engineering: a Formal Approach*. Cambridge University Press, 1994.

[HL91] K. Hayes and K. Lavery. Workflow management software: the business opportunity. Technical report, Ovum Ltd, London, 1991.

[HVA97] D. Hauschildt, H.M.W. Verbeek, and W.M.P. van der Aalst. WOFLAN: a Petri-net-based Workflow Analyzer. Computing Science Reports 97/12, Eindhoven University of Technology, Eindhoven, 1997.

[Jen96] K. Jensen. *Coloured Petri Nets. Basic concepts, analysis methods and practical use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.

[Joo94] S. Joosten. Trigger Modelling for Workflow Analysis. In G. Chroust and A. Benczur, editors, *Proceedings CON'94: Workflow Management, Challenges, Paradigms and Products*, pages 236–247, Vienna, Oct 1994.

[Kou95] T.M. Koulopoulos. *The Workflow Imperative.* Van Nostrand Reinhold, New York, 1995.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.

[MBea95] M. Ajmone Marsan, G. Balbo, G. Conte, et al. *Modelling with Generalized Stochastic Petri Nets.* Wiley series in parallel computing. Wiley, New York, 1995.

[MEM94] G. De Michelis, C. Ellis, and G. Memmi, editors. *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, Zaragoza, Spain, June 1994.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[Pal97] Pallas Athena. *Protos User Manual.* Pallas Athena BV, Plasmolen, The Netherlands, 1997.

[Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[Rei85] W. Reisig. *Petri nets: an introduction*, volume 4 of *Monographs in theoretical computer science: an EATCS series*. Springer-Verlag, Berlin, 1985.

[SH95] P.A. Straub and C. Hurtado. The Simple Control Property of Business Process Models. In *XV International Conference of the Chilean Computer Science Society*, 1995.

[SL96] Software-Ley. *COSA User Manual.* Software-Ley GmbH, Pullheim, Germany, 1996.

[Val87] R. Valk. Infinite Behaviour and Fairness. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 377–396. Springer-Verlag, Berlin, 1987.

[WFM96] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.

[Wil89] R. Wild. *Production and Operations Management : Principles and Techniques.* Cassell, London, 1989.

[WR96] M. Wolf and U. Reimer, editors. *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow,* Basel, Switzerland, Oct 1996.