Business Process Modeling Using Petri Nets

Article · January 2013

DOI: 10.1007/978-3-642-38143-0_4

3 authors:

Kees M. van Hee
Eindhoven University of Technology
205 PUBLICATIONS **6,215** CITATIONS

SEE PROFILE

Natalia Sidorova
Eindhoven University of Technology
132 PUBLICATIONS **2,885** CITATIONS

SEE PROFILE

Jan Martijn E.M. Van der Werf
Utrecht University
81 PUBLICATIONS **1,890** CITATIONS

SEE PROFILE

Business Process Modeling Using Petri Nets

Kees M. van Hee, Natalia Sidorova, and Jan Martijn van der Werf*

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee,n.sidorova,j.m.e.m.v.d.werf}@tue.nl

Abstract. Business process modeling has become a standard activity in many organizations. We start with going back into the history and explain why this activity appeared and became of such importance for organizations to achieve their business targets. We discuss the context in which business process modeling takes place and give a comprehensive overview of the techniques used in modeling. We consider bottom up and top down approaches to modeling, also in the context of developing correct-by-construction models of business processes. The correctness property we focus on is soundness, or weak termination, basically meaning that at every moment of its execution, a process has an option to continue along an execution path leading to termination, which is an important sanity check for business processes. Finally, we discuss analogies between business processes and software services and their orchestrations and argue the applicability of the described modeling techniques to the world of services.

1 Introduction

The concept of a *business process* (BP) is as old as humanity. A BP is the set of interdependent tasks and resources needed to produce some service or product. On top of this set there are constraints or business rules that have to be met. Business processes form the heart of organizations: they should make possible that organizations can realize their goals. Although business processes always have existed, the description, or modeling, of BPs started only recently. In the twentieth century auditors and accountants started working on BP specifications in their field and later it became a hot topic in quality management. In these early days process descriptions were used for documentation, and for facilitating communication between persons. As a result, these descriptions were informal, often in a natural language enriched with some diagrams.

Around the year 1990 business processes became a hot topic in industry, as people became aware of the fact that we were not fully exploiting the power of computer systems. Information systems supporting business processes were

* Supported by the PoSecCo project (project no. 257129), partially co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7).

K. Jensen et al. (Eds.): ToPNoC VII, LNCS 7480, pp. 116–161, 2013.

data-oriented, meaning they were targeted in recording the status of objects that played a role in business processes, in a database. Consequently, database technology was the main technology at the beginning of the nineties. Moreover, information systems were built to support existing business processes by automating their tasks by imitation of the human work by a computer. The papers and books of Hammer and Champy [32, 33] gave very convincing examples of inefficient use of computers and they advocated the re-engineering of business processes before the development of supporting information systems. This was also the first time that the term "engineering" was applied to business processes. Indeed computers can process information in completely different ways from people and it is logical to exploit these possibilities to make business processes more effective and more efficient.

The awareness of the importance of business processes has triggered the introduction of the concept of *process-aware information systems*. Information systems became more than recorders of the status of objects—they started to also focus on business-relevant *events*. The information systems became proactive in the sense that they, for instance, started to control the right order of task execution, keep track of deadlines and distribute the work between resources.

The most notable implementations of the concept of process-aware information systems are *workflow management systems*, a class of generic components for the construction of information systems. Workflow management systems have become the counterparts of database management systems. While a database management system is configured by a database schema and a set of constraints, a workflow management system is configured with a process model. A workflow engine can be embedded in a larger information system the same way as database engines can. From ca. 1995 up to around 2005, there was a strong focus on the support of single business processes with workflow engines. After that, the interest shifted to *cooperating business processes*, as encountered in supply chains and in BP outsourcing.

One of the most recent forms of composition trend in the last decade is the Service Oriented Computing (SOC) [14, 68]. In this paradigm for systems development, closely related to the paradigm of Service Oriented Architecture (SOA) [18, 68], systems are considered as components that deliver services to each other, like businesses in a supply chain. Each component runs a process, to orchestrate the service and in fact such a processes can be seen as a business process. The business processes in the real world are in a way mirrored in the components of the information systems. So here we also encounter the cooperation of business processes, which generates new scientific challenges to develop correct working systems.

The focus of this article is to give insight in the role of business processes, the modeling of business processes and the use of these models in BP management. Instead of presenting new theoretical results we try to give insight as well as an overview of theoretical results. In Section 2 we study the context of business processes, i.e. the world in which business processes play their role. Then in Section 3, we study the modeling of business processes, in particular a bottom

up and a top down approach, and we will analyse the correctness of BP models. We only focus on a generic property: weak termination, which is the ability of always being able to reach a final state. This turns out to be a very important sanity check for business processes. We focus on correctness by construction principles. Finally in Section 4, we move from business processes to services and in particular to computerized services as we can find them in modern web-based information systems. Services incorporate or emulate business processes. Services cooperate with each other and so we have to study the cooperation of two or more business processes which provides new challenges. We conclude the article in Section 5.

2 Context

We start with the informal introduction of a set of related concepts from the world in which BPs play their roles. Many of these concepts will be formalized or modeled in the next section. However modeling always has to serve some purpose. In order to do so, we first need to understand the context of the part of the world we are modeling. This means that we have to classify the real world entities, either abstract or physical, and map them to the concepts; see also [5].

2.1 Business Process Concepts

An *organization* is a system consisting of humans, machines, materials, buildings, data, knowledge, rules and other means, with a set of *goals* to be met. Typical examples of organizations are companies, factories, hospitals, schools and governmental institutions. We also consider *business units* or departments within a larger organization as organizations, and similarly, we also consider two or more cooperating organizations as one organization. Most organizations have, as one of their main goals, the *creation* or *delivery* of (physical) *products* or (abstract) *services*.

The creation of services and products is performed in *business processes* (BP). A BP is a set of *tasks* with *causal dependencies* between tasks. The five basic *task ordering principles* are

- *Sequence* pattern: putting tasks in a linear order;
- *Or-split* pattern: selecting one branch to execute;
- *And-split* patterns: all branches will be executed;
- *Or-join* patterns: one of the incoming branches should be ready in order to continue; and
- *And-join* pattern: all incoming branches should be ready in order to continue.

As will be shown in the next section, these basic patterns are closely related to the well-known four *control flow* constructs from programming:

- *Sequence* construct;
- *Choice* construct;

– *Iteration* constructs, like "repeat until" and "while do"; and
– *Parallel* construct.

Actually we can make the above programming constructs with the five task ordering patterns.

For the execution of tasks *resources* are required. Resources can either be *durable* or *consumable*. The first kind is available again after execution of one or more tasks, like a catalyst in a chemical process. Typical examples of this kind are humans, machines, computer systems, tools, information and knowledge. Consumable resources disappear during the task execution. Examples are energy, money, materials, components and data. The results or output of a task can be considered as resources for subsequent tasks or as final products or services. Two kinds of durable resources are of particular importance: the humans as a resource, called *human resources*, and information and knowledge, which we will call *data resources*. Since human activities are sometimes replaced by computer systems, we use the term *agents* as a generic term for human and system resources.

A BP is an abstract notion: no physical object exists that can be identified as a BP. A BP has many different forms of appearance and different names, depending on the context. For example, an administrative *procedure* for handling claims in an insurance company is a BP, and a medical protocol to treat a form of cancer describes another BP. Even a *recipe* for cooking or an *algorithm* for computing can be seen as BPs. Thus, we consider any form of task structuring in order to create a product or a service as a BP.

Tasks are viewed as *atomic* pieces of work. Atomic means that the task is executed without breaks and that the agents and durable resources needed for the execution are available at the beginning and are released at the end. We do not consider the content of a task. Later we will see that task atomicity is a view or a modeling decision which can be adjusted at a later stage of modeling: a task can be substituted by another process, which becomes a *subprocess* of the original process.

An important feature of a BP is that it can be repeated, meaning a BP has multiple *instances*. An instance can be seen from two different view points: (1) the entity that is in progress of being created, called a *case*, and (2) the process of creation which can be seen as a *project*. We will use the term case for both views. A case is at each moment in time in some *state*. In the initial state the process is ready to start and in the final state the product or service is completed. The state of a case is determined by (1) *case conditions*, that are true or false for the case at that state and (2) *case variables*, data connected to the case. There are two kinds of case variables: *routing parameters* used for routing through the process and a *case document*, which is a file with all relevant data of the case.

It is possible to have parallel task executions for a case, which means that *concurrently* different resources can be busy with different tasks for the same case. Besides the concurrency within a single case, BPs can handle several cases concurrently. However if two or more cases are executed concurrently, we assume they are *independent* of each other: they do not influence each others final result.

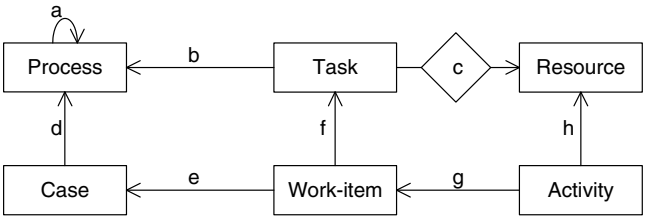


Fig. 1. Business process concepts and their relations

Cases can share resources, which may influence the *scheduling* of tasks, respecting the task ordering requirements of the BP. Sharing of resources is often a form of competition between the cases. In this context, it is useful to distinguish the notions of a *work item* and an *activity*: a work item is the combination of a task and a case, while an activity is the combination of a work item and resources. A work item can be seen as a task instance while an activity is the execution of a work item. Figure 1 shows the concepts in a BP and their relationship.

Besides the causal dependencies expressed by the task ordering principles, there are often many more *constraints* on the execution of BPs. These constraints are called *business rules* and there are special languages to express them (cf. [9,36,67]). An example of a business rule is the four-eyes-principle that says that certain tasks in one case should not be executed by the same human resource. Such rules cannot be expressed by task ordering principles and have to be guaranteed by using other means. It is of course very important to be able to verify if certain BP satisfies a set of business rules.

BPs can be classified according to their function within the organization:

- *Primary* processes: they are dedicated to the primary goals of the organization, namely the creation of the products and services using resources;
- *Secondary* processes, also called *supporting* or *enabling* processes: they are dedicated to providing and maintaining the resources for the primary processes as well as maintaining relationships with the customers and suppliers; and
- *Tertiary* processes, also called *management* processes: they are dedicated to the *control* of the organization as a whole and the *coordination* of the primary and secondary processes.

Here we encounter again a dilemma: the chosen perspective determines the classification. For example, when a business unit of a department performs a supporting process for the department, this supporting process can be a primary process for the business unit. Similarly, a tertiary process of an organization can be a primary process of a business unit. On the one hand this might be confusing, on the other hand, it gives an opportunity to define clear goals, customers and suppliers for all BPs. A typical example concerns *authorization* of human resources, i.e. the process of delegation of rights to perform functions. On the one hand this can be considered as a kind of special task within a primary process

itself but it can also be seen as a secondary process, as it concerns (human) resource management.

A subclass of the tertiary processes consists of so-called *inter-organizational* BPs. These processes exchange information, resources and cases between different, but *cooperating* organizations. In one organization the control can be hierarchical, which is normally not the case in cooperating independent organizations. There, coordination is the key activity. Examples of inter-organizational BPs occur in *supply chains*, where the BPs of the participating organizations work together as co-makers. Another example is an electronic market place where the coordination between supply and demand is organized by means of market mechanisms. Here we enter the field of *communicating* BPs which is the topic of Section 4.

2.2 Supporting Information Systems

Process aware information systems support organizations by means of three basic functions: *monitoring*, *planning* and *execution* of tasks in BPs. The monitoring concerns the recording of all *events* in the processes. This also enables providing management information: not only the status of each of the cases, like running or completed, but also aggregated information like the total number of cases running currently, average processing time and waiting time of all cases.

The planning function of a process aware information system is more proactive, as it concerns the selection of tasks that are ready to be executed, the allocation of resources for a task and preparation of the execution by transferring the right data resources to the agents (human resources or computer systems). More and more tasks of BPs are executed autonomously by information systems. Specifically in financial and governmental institutions where BPs consist of information processing only, e.g. the transfer of money, granting a mortgage or the registration of a marriage. This kind of BPs only require information processing. Sometimes human judgement is essential, but in many situations there are formalized rules for making decisions automatically.

In principle active databases (cf [69]) support these three functions. However, the set of rules involved becomes very complex, so that people lose overview, with an inconsistent set of rules as a consequence. This led to the development of a new class of software components: *workflow management systems* [91], as a counterpart of database management systems. Workflow management systems are configured by means of a *process model* like a database management system is configured by a database schema or an entity-relationship diagram. The term "workflow" is often considered to be a synonym for "BP".

An important component within a workflow management system is the *workflow engine* which takes care of the distribution of tasks over the agents. This provides flexibility: if a BP has to be changed, only the process model has to be adapted, without changing the rest of the information system. Another part of the workflow management system concerns the authorization of human resources. This function is similar to the authorization of users of databases. The support for handling data resources is very limited in workflow management

systems. Only some case parameters are considered and in particular used for routing decisions. This has been a conscious design choice: workflow management systems should only be concerned with the distribution of work and not with the content. There is another class of information systems, called *case handling systems* [13, 37], that combine workflow management with data handling. There exists a standard architecture for worfklow management systems (cf [91]). However most workflow management systems have their own process modeling language, although there are some standards as well [66, 92].

In many organizations information systems are realized by means of *standard application packages* like ERP-systems (Enterprise Resource Planning systems). These systems are actually *configurable information systems*. In the past they had support for a predefined set of possible BPs. Today there is a trend to migrate from monolithic systems with a finite set of predefined options, to more flexible component-based systems with one or more workflow engines inside.

Another, but related, trend in the world of information systems is called Service Oriented Architecture (SOA). According to this paradigm, information systems are build as *loosely coupled components* that deliver *services*, using other services. The "loose coupling" has three characteristics: (1) components communicate *asynchronously* by exchanging messages via *ports*, (2) only the port *protocols* have to be known in order to couple components and (3) the coupling can be done at runtime via a *service broker*, which is in fact dynamic binding of services. Components within such a system have internally a process that defines the service, called the *orchestration* of the service. An orchestration process is actually a workflow and each delivery of a service is a case. For instance, the Web Service Business Process Execution Language (WS-BPEL, [15]) is a language belonging to the web technology family, to orchestrate components.

The cooperation between components is sometimes called the *choreography*. There is a great analogy between BPs and service in a SOA. First, many BPs are supported by or even replaced by a service component. Specifically in retail, financial institutions and government we see that BPs involving the customers are replaced by web services where the customer is in direct contact with the information system of the organization without interference of employees unless an exception occurs. In those cases, the original BPs are in a way simulated by the service components. Secondly, service components in a SOA behave more or less like independent business units in one organization. In particular the tendency in organizations to outsource non-key sub-processes is mirrored in the execution of a service by calling other service components for particular tasks.

2.3 Role of Models

Organizations are not isolated: they create products or deliver services to their *customers* and obtain the resources needed from *suppliers*. Both suppliers and consumers can be persons or organizations themselves. All persons or organizations that have some interest in an organization are called *stakeholders*. Beside customers and suppliers, also employees (human resources), management, share holders and in many cases the government (e.g., the tax department) are

stakeholders. Stakeholders have different objectives within an organization. Therefore, in modeling BPs, it is essential to know in advance for which stakeholders a problem is being solved.

Modeling a system is a way of *understanding* the system. There is empirical evidence that making a precise model of a complex entity, such as a BP, reveals all kind of details that we are overlooking otherwise. Formalizing an informal description and then constructing a new informal description from the formalization results in a better description. There are plenty of reasons to *document* a BP. For example as teaching material for new employees, for quality control and for the development of supporting information systems. We distinguish two kind of models: *descriptive models*, also called "as is" models, that describe how a system or processes actually is, and *normative models*, also called "to be" models, that describe how the process should work. In fact, models exist for many reasons:

1. to understand processes
2. to document processes
3. to analyse processes
4. to monitor and audit processes
5. to improve or optimize processes
6. to outsource processes
7. to construct or redesign processes
8. to execute processes

Models help to analyze processes. There are two kinds of *analysis* of BPs:

- to verify *conformance*, i.e. to check if a BP satisfies a set of business rules (cf [36]);
- to determine the *performance*, i.e. to compute performance characteristics, like the time or the number of resources needed to reach a certain state.

For conformance analysis we use general purpose model checking techniques (e.g. [50, 89]) or dedicated techniques like structural analysis. Business rules typically concern the ordering of tasks and the use of resources and combinations of these. For performance analysis, simulation is the most used technique: the process model is used as simulation model (e.g. [39, 72]).

Monitoring a process is recording an execution path, which may include several cases of a BP in order to be able to reconstruct parts of the process. The most important function of monitoring is to produce reports of the past, either periodically, event-driven or just on demand. *Auditing* a process is checking if an execution path of a BP satisfies a set of business rules. Whereas conformance is checking the whole process, auditing is checking only the execution paths of the process. Of course if we are sure that a certain process is executed, then it is sufficient to check if all the business rules are satisfied, but if we are not sure a "to be" BP is really executed, then we have to audit the runtime system.

Based on performance analysis one may try to *improve* the performance of a BP by restructuring it or by reallocating resources. *Optimization* is improving

until some optimality criterion is reached. In practice often the term optimization is used in cases where it actually refers to improvement. The choice of a good optimality criterion is far from easy and since stakeholders do not know the optimal performance they are already satisfied if the performance has improved significantly.

Outsourcing is a kind of restructuring a BP by isolating a subprocess and replacing it by a process of another organization, called the supplier. This way, organizations can put their efforts in their core business, while other organizations can better perform in some tasks or sub-processes, because they have specialized skills or they can exploit economy or scale effects. By outsourcing, a particular kind of inter-organizational BP is created: the original BP is communicating with the subprocess of the supplier. This way, a supply chain is formed.

To *construct* a new BP or to *redesigning* an existing BP, one first makes a model of the process. Then we analyse this model to verify conformance and performance properties. After this, we create or implement the BP. Most of the time, implementation of a new or changed BP within an organization involves more details than the model expresses, like the location(s) where the process will be executed, the durable resource to be used and training human resources. In many cases implementation also involves the development of supporting information systems. However, the model plays an essential role in these activities.

In the *execution* of a BP the model is used to determine the *order of tasks*, the *authorization* of human resources, and the *allocation* of resources. Either this is done by a supporting information system or by human activities. In the first case the process model is used as a configuration parameter of the workflow management system, in the second case as manual or handbook.

2.4 Modeling Languages and Tools

In the nineties, various industries developed different workflow management systems. As a workflow management system needs a process model as configuration parameter, industry created many different languages to model BP. Most of these languages have a graphical syntax. Although it looks easy to define such a language, it turns out to be a difficult job, because of the dynamical semantics of processes.

Before the nineties there were already process modeling languages, mostly without formal semantics, like DFD (dataflow diagrams diagrams, cf [82]). However, these were not used for modeling of BPs. The most important examples of industrial process formalisms are: EPCs (Event-driven Process Chains) (cf [53]) with supporting tool ARIS (cf [75]), BPEL (cf [15]), BPMN (cf [66]), and UML-Activity Diagrams (cf [31]). The last three are industry standards. They all have still flaws in they formal semantics, although new releases become better. BPEL has no graphical representation.

Many of the industrial languages do not have formal semantics. Some have only an operational semantics in the form of a simulation tool that can simulate the behavior of a given model. If a formal semantics exists, it is mostly in terms of a (labeled) transition system or a (labeled) Petri net. Labeled transition

systems provide *interleaving semantics*, which means that concurrency of tasks is expressed as different possible orderings. Petri nets allow for *partial order semantics*. Formal semantics are essential for analysis purposes.

There were already very good process formalisms available from academia: the many types of Petri nets (cf [27, 70, 73]) and process algebras, e.g. CSP [48] and CCS [62]. However these formalisms are general purpose process formalisms and not tuned to BP, which means that direct support for some useful constructs like an implicit Or-split and Or-join is missing. Process algebras have no graphical syntax, and are difficult to understand by most practitioners. Petri nets, with its diagram technique is a much better candidate and indeed some workflow managements system (e.g. COSA [78] and YAWL [49]) are based on Petri nets. There is also an ISO standard for Petri nets [47]. A special class of Petri nets was developed, called workflow nets [1] and they are used frequently. The semantics of UML-Activity Diagrams and BPMN are converging to Petri nets. In this article we will use workflow nets for modeling BPs.

Many different *software tools* exist to support the modeling process. So we have *editors* to develop and publish models (cf [23, 30, 39, 57, 72, 75]), and tools to analyze models, either by model checking, structural analysis (e.g. [4, 50, 76, 87]) or simulation. Most modeling tools have simulation facilities (cf [39, 72]).

3 Modeling Business Processes

In the remainder we only consider Petri nets and in particular the class of workflow nets, to model BPs. For many purposes it is sufficient to consider classical Petri nets, i.e. with "black" tokens. However sometimes it is important to have the modeling power of colored Petri nets. For a formal definition of classical Petri nets as used in this paper see Appendix A. For more information about classical Petri nets, we refer the user to e.g. [73, 86]. For a formal definition of colored Petri nets see [51]. Here we give an informal introduction only.

A *classical Petri net* is a triple (P, T, F) where P is the set of *places*, T the set of *transitions* and F a function, called the *flow function*, that assigns to each pair of nodes of $(P \times T) \cup (T \times P)$ a natural number, possibly zero. There is a graphical notation for Petri nets where places are displayed as circles and transitions as rectangles. If $F(x, y) > 0$ for a pair of nodes (x, y) then we say that there is a directed arc from x to y and $F(x, y)$ is the arc weight. We say that x is an input node for y and that y is an output node for x.

Places may contain tokens. A distribution of tokens over places is called a *state*. In classical nets the state can be expressed as a function that assigns to each place the number of tokens in that place; this function is called a *marking* of the net. A transition is *enabled* if for all input places the number of tokens in each input place is at least the arc weight. An enabled transition can fire. If it fires, the marking changes: The number of tokens consumed from the input places of the transition and the number of tokens produced to the output places is defined by the weights of the corresponding arcs. Thus, when a transition fires, the Petri net "moves" from one state to another. In this way, a *labeled transition*

system is formed in which the Petri net transitions are the labels of the state transitions and the markings are the states. We call this the *reachability graph* of the Petri net. The combination of a Petri net and an initial marking is called a Petri system of system for short.

Since classical Petri nets are not Turing complete (cf. [16]), we often need more advanced features in modeling. For this purpose we introduce *inhibitor arcs* and *reset arcs*. To do so, we add for each of the arc types a new relation, I and H, respectively. The net depicted in Figure 2 has an inhibitor arc, denoted by an arc with a bullet head, between transition F and place p, indicating that transition F can only fire if place p is empty. The net has a reset arc, denoted by a dashed line, between transition G and place p, thus firing transition G empties place p.

In a *colored Petri net* (CPN) tokens have a value. Each place has an associated value type, also called a *color set*, and all tokens in a place have a value that belongs to that type. Each arc has one variable and transitions have a precondition, also called a *guard*, and a postcondition. A precondition is an expression with the variables of the input arcs of the transition as the only free variables. The post condition is an expression using the variables of the input arcs as well as of the output arcs. Figure 3 depicts a transition in a colored Petri net with its specification.

A transition in a colored Petri net is enabled if and only if for each input place a token can be found such that the precondition substituted with the actual values of the tokens is satisfied. An enabled transition may fire and if it does the post condition is evaluated with the input variables bound to the input tokens and then the output variables obtain a value. Tokens with these values are produced for each output place. In CPN tools (see [72]) and with CPNs as defined in [51] there is more freedom in modeling and more tokens can be consumed or produced for one place in one firing. In fact arcs may have expressions instead of only variables.

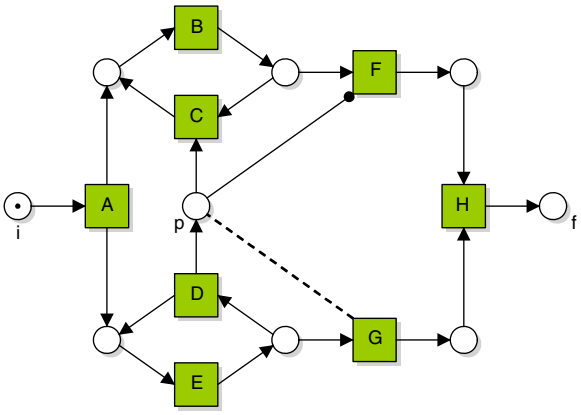


Fig. 2. Workflow net with reset arc and inhibitor arc

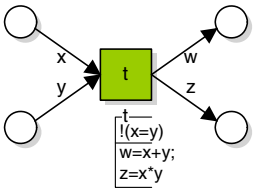


Fig. 3. Transition in a colored net

Token values are entities with one or more attributes, representing different properties. These attributes can be used to express arbitrary data. We identify two key attributes: *token identity* and *time stamps* that require a special treatment (see [35]). In case of token identity, each token carries and identifier. The precondition of each transition requires that all tokens consumed from the input places should have the same identifier [43, 74]. The tokens produced will obtain this same identifier.

The intuitive meaning of a time stamp of a token is the minimal time after which the token may be consumed by some transition. The firing rule for tokens with a time stamp is as follows. In order to determine which transition may fire next, all combinations of input tokens are considered for all transitions and a transition is selected for firing if its maximal time stamp of all its input tokens, is minimal over all enabled transitions. This time is called the *transition time* of the marking. We assume timed Petri nets to be *eager*, which means any transition fires as soon as possible. The tokens produced when a transition fires, obtain a time stamp which is the transition time plus some delay, depending on the inscription of the specific output arcs. Also for colored Petri nets we allow inhibitor arcs and reset arcs with the same semantics as for classical Petri nets.

In the remainder of this article, when we consider classical or colored Petri nets we will assume they do not have inhibitor or reset arcs, unless we explicitly require this.

3.1 Workflow Nets

For modeling convenience later, we start with a definition that generalizes the classical definition of a workflow net. A workflow net is a Petri net with two sets of special nodes: *initial* and *final* nodes. The first have no input nodes and the last have no output nodes. In fact we only use two types: one where all the special nodes are places and one where they all are transitions.

Definition 1 (Workflow net). *A* workflow net *N is a 5-tuple (P, T, F, E, C) where (P, T, F) is a Petri net, E is the set of* initial nodes *and C is the set of* final nodes *such that $E, C \subseteq P$ or $E, C \subseteq T$, ${}^\bullet E = C^\bullet = \emptyset$, and all nodes $(P \cup T)$ of (P, T, F) are on a directed path from a node in E to a node in C.*

- *If $E, C \subseteq P$ we say N is a* place-bordered WFN *(WFN-s), also called a* multi workflow net.

- *If $E, C \subseteq T$ we say N is a* transition-bordered WFN *(WFN-t).*
- *The* completion N^+ *of a WFN-s is a WFN-s (see Figure 4(a))*

$$N^+ = (\ P \cup \{i,f\},\ T\ \cup\ \{t_i, t_f\},\\ F \cup \{(i,t_i) \mapsto 1, (t_f,f) \mapsto 1\}\\ \quad \cup \{(t_i,e) \mapsto 1 \mid e \in E\} \cup \{(c,t_f) \mapsto 1 \mid c \in C\},\\ \{i\},\ \{f\}\ )$$

where $i, f \notin P$ and $t_i, t_f \notin T$ are fresh places and transitions, respectively.

- *The* completion N^+ *of a WFN-t is a WFN-s (see Figure 4(b))*

$$N^+ = (\ P \cup \{i,f\},\ T,\\ F \cup \{(i,e) \mapsto 1 \mid e \in E\} \cup \{(c,f) \mapsto 1 \mid c \in C\},\\ \{i\},\ \{f\}\ )$$

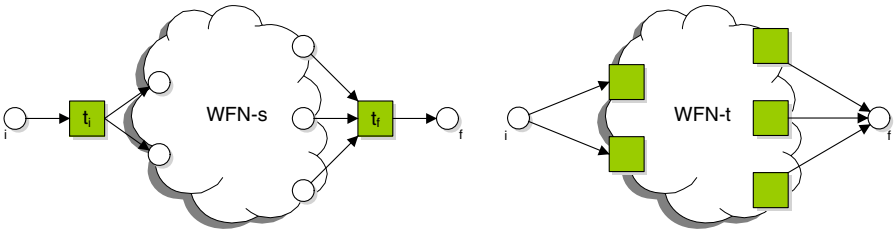
where $i, f \notin P$ are fresh places.

The completion transforms a WFN-s or a WFN-t into a WFN-s with a single input place and a single output place, as displayed in Figure 4. Such a WFN-s is a classical workflow net [1]. Actually the behavioral properties for WFN we will consider later are expressed in terms of the completions, i.e., in terms of classical workflow nets.

Definition 2 (Classical workflow net). *A WFN-s $N = (P, T, F, E, C)$ is a* classical workflow net *(WFN) iff $E = \{i\}$ and $C = \{f\}$ for some $i, f \in P$.*

- *Its workflow system is defined by $\mathcal{N} = (N, [i], \{[f]\})$.*
- *Its* closure *is defined by $\overline{N} = (P, T \cup \{t^*\}, F \cup \{((f,t^*),1),((t^*,i),1)\}$ where $t^* \notin T$ is a fresh transition.*
- *Its* fused closure *is defined by $N^* = (P \setminus \{f\}, T, (F \setminus {}^\bullet f \times \{f\}) \cup ({}^\bullet f \times \{i\}))$.*

As in classical Petri nets, we identify three important workflow net classes (cf [26]). These are: (1) free choice workflow nets (FC-WFN), where the underlying Petri nets is a free choice net, (2) state machines workflow nets (S-WFN), where the underlying Petri net is a state machine, also called S-net and marked graph workflow nets (T-WFN) where the underlying Petri net is a marked graph, also called T-net.



(a) WFN-s (b) WFN-t

Fig. 4. The completion of a WFN-s and WFN-t net

In order to model resources we extend the definition of workflow nets with resources. Resources are an additional set of places from which transitions can consume and produce tokens representing the resources they need or deliver.

Definition 3 (Resource-constrained workflow net [40]). *A* Resource-constrained WFN *(RCWFN) N is a 7-tuple $(P, R, T, F_c, F_r, E, C)$ such that*

- $P \cap R = \emptyset$*, P are called the* case *places and R the* resource *places;*
- $F_c : (P \times T) \cup (T \times P) \to \mathbb{N}$ *and* $F_r : (R \times T) \cup (T \times R) \to \mathbb{N}$*;*
- $(P \cup R, T, F_c \cup F_r)$ *is a Petri net;*
- (P, T, F_c, E, C) *is a WFN and this net is called the* production net *of N.*

So we have extended a WFN with an additional set of places, called *resource places*, and in that context we call the other places *case places*, and similarly we speak of resource tokens and case tokens.

3.2 Expressing Business Process Concepts

When modeling real life systems with Petri nets we always have to make a choice between two paradigms: either we model (time consuming) activities by transitions or by places. In the first choice the marking or state indicates a situation of rest and the transitions model the activities that may lead to a new state of rest. According to that paradigm, transitions are given names that reflect actions like, "move" or "print" while places have names that express a status, like "in stock" or "at home". In this way, transitions are named with a verb and places with a noun. According to the latter paradigm, transitions stand for instantaneous events and places may reflect a status as well as an activity. A place that represents an activity should have an input transition that represents the start event and an output transition that represents the stop event. In this section we will express the concepts of task, task ordering, case and resource in terms of workflow nets.

Tasks and Their Ordering. A *task* represents an activity and so it can be modeled in two ways: either a task is modeled as one transition or as a pair of transitions, where the first one is the *start event* of the task and the second one the *stop event* of the task (see Figure 5). Note that in classical Petri nets, time is not modeled, but only the order of execution can be expressed. In colored Petri nets time can be expressed using time stamps for tokens, which we will explain later. Although it is not necessary, it is a best practice to let a task have only

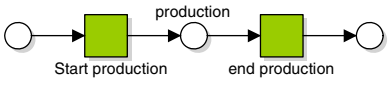


Fig. 5. Activity modeled with a start and stop event

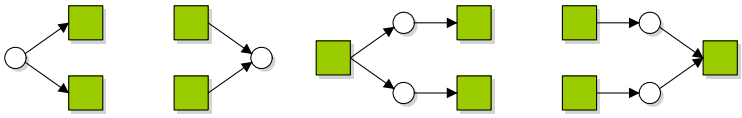
one input place and one output place. In case of start and stop events: the start event has one input and the stop event one output place.

The *ordering* of tasks, also called the causal dependencies of tasks, can be expressed by adding a place in between the two tasks, i.e. between the two transitions if we model a task with one transition, and between the stop transition of the first task and the start transition of the second task if we model according to the latter paradigm. A place in between tasks can be seen as precondition of the first task and a post-condition of the second task. In this way, places express the causal dependency between tasks and transitions.

Not only places are used for task ordering. Transitions play a role as well. For example, to express that two tasks can be executed in parallel, we need a transition with one input and two output places, and similarly if a new task can only start if two or more other tasks have completed, we need a transition with one input place for each such task and one output place to the next task. These transitions are called *AND-split* and *AND-join*, respectively. They are not considered to express tasks but *synchronization events*. Figure 6 shows the constructs. The five basic task ordering principles can be expressed in this way. In fact, most of the control flow patterns [10] can be expressed. By ordering all the tasks in this way we obtain a WFN.

Cases. A *case* is a complex entity. If we consider a classical WFN net with only one token in its initial place, then that token represents the initial state of the case. After each transition the state of the case is expressed by the marking of the WFN. Note that this marking may have more than one token. Thus, in general the state of a case is expressed by multiple tokens. If a marking is reached with only a single token in the final place, then we say that the case is in its *final state*. However, it is not always the case that the final state of a case is reachable.

In case we have two or more tokens in the initial place of a classical WFN we have to deal with *batch processing*. As with one case, the initial state of the batch is a state in which all the tokens are in the initial place, and the final state of the batch is the state in which all tokens are in the final place. Any other marking reachable form the initial state is a state of the batch. An important property of batch processing is that if we start with a batch of a certain number of cases, we should reach a final state with the same number of cases.



(a) XOR (b) AND

Fig. 6. Split and join patterns

In a batch, cases can influence each other. In fact, it is not possible to identify the case to which a token belongs. An example of batch processing in which the interference of cases is irrelevant, is the production of k identical objects. (The workflow depicts the steps in the production, and each object is represented as a case.) The production process consists of the acquisition of the parts and their assembly. So, we have to order parts in multiples of k and as long as each final product has the right number of parts, it does not matter for which product (i.e., case) they were meant. We also consider situations in which we have an unbounded *stream* of incoming cases that has to be handled. If we want to avoid interference we can use colored Petri nets in which we only use the identity attribute of the token value. Then, the batch has distinguishable cases and a transition will never consume two tokens of different cases in one firing. In this way, the cases are independent of each other.

Resources. The next concept we have to model are *resources*. As seen in Section 2.1, we classify resources into durable resources and consumable resources. Durable resources are the most important resources from a modeling point of view. Consumable resources are often anonymous supplies. The only reason to model consumable resources is to analyze the number of resources needed in the execution of cases, which is a performance issue. Simulation of the process can solve this problem. For each task we identify the amount of supplies needed. We then simulate a large number of cases to determine the distribution of the occurrences of the task, which in turn can be used to determine the resource usage.

Durable resources, like human resources, are almost always identifiable and have their own rules. For example, for human resources a separation of concerns principle should often hold, like the four-eyes principle. Durable resources are often part of a secondary business process. An important requirement is that the durable resources obey a conservation law: after handling a case, the durable resources should be available for a new case. With RCWFN (see Definition 3) we are able to express these kind of properties.

Business Rules. Last, we consider the expression of *business rules*. In practice, the context of a BP defines the boundaries within which the BP should be executed. These boundaries are defined by business rules. Often, it is difficult to verify whether a given WFN satisfies these rules. Business rules are often expressed in some form of logic, like *linear time logic* (LTL, see [71]) or *computation tree logic* (CTL or CTL* see [22]). These logics lack expressing calculations. Therefore, special languages exist to express calculation as well, like *Presburger logic* or the Business Rules Language (BRL) [36].

For verification of business rules, two methods exist: (1) verification on the level of the process model, i.e. the WFN and (2) verification on the level of a *process log*, i.e. a set of case histories. The first is called *conformance checking*. In the first situation we verify that all possible cases satisfy the business rule. To check this, the reachability graph of the Petri net is constructed and *model*

checking is applied. In many situations it is possible to use methods that exploit the Petri nets structure to speed up the verification process, like in LoLA [76]. This applies for business rules that can be expressed in LTL or CTL.

The latter method is called *auditing*. In auditing, it is easier to check complex rules, as the rule should only hold for the finite set of executed cases [7, 9, 20, 36]. However, the claim is weaker than in conformance checking: we only show absense of violations in the past. As evaluation is done on a finite set of case histories, rules expressed in Presburger logic or BRL can be evaluated.

3.3 Soundness of Workflow Nets

We often require BPs or WFNs to satisfy a set of business rules, depending on the particular context. There is one rule (with some variants) that business processes should always satisfy, independent of the context. This property is called *soundness*. Soundness is a general purpose *sanity check* for workflows. Intuitively it says that once a workflow has started it should always be able to stop without leaving "garbage" in the net. It is obvious that all processes should have this property.

There are many forms of soundness [6]. The main characteristic of most soundness concepts is *weak termination* (see Appendix A) which states that from every reachable state (marking) it is possible to reach a final state. The notion of *classical soundness*, which is defined for classical WFNs, states three important properties: (1) the workflow system should be *weakly terminating*, (2) the workflow system should have the *proper completion* property: i.e., if we reach a marking that contains the final marking, it is equal to the final marking and (3) all transitions in the model should contribute to the business goals, i.e., each transition is occurring in at least one case. The last property can also be expressed as *quasi-liveness* of the WFN. Quasi liveness is not for all forms of soundness required, although it is obvious that in practice it is not useful to have transitions in a process model that are never used.

For a classical WFN the second requirement is implied by the structure of the WFN (cf [42]), although in the first definition of soundness it was required (see [1]). The first requirement of soundness is the most essential [6]. Weak termination is actually the same problem as the *home marking property*: a marking is a home marking if and only if it is reachable form every reachable marking, which is decidable for classical Petri nets [28]. In fact, the notion of classical soundness coincides with requiring the closure to be live and bounded. A (transition-bordered) WFN-t or a (place-bordered) WFN-s is classical sound if its completion is classical sound.

Theorem 4 (Classical soundness equals liveness and boundedness [1]). *Let $N = (P, T, F, \{i\}, \{f\})$ be a classical WFN. It is classical sound if and only if the system $(\overline{N}, [i], \{[f]\})$ is live and bounded.*

As this property is the oldest result in this field and the proof gives good insights, we give a sketch of the proof.

Proof. Suppose the closure $\overline{N}$ of a WFN N is live and bounded, but not classical sound. Then the closing transition is live. Hence, a marking m and firing sequence σ exist such that $(\overline{N} : [i] \xrightarrow{\sigma} m)$ with $m > [f]$, i.e., $m = m' + [f]$ with $m' > \emptyset$. Hence, the closing transition is enabled. Firing it results in marking $m' + [i]$. Now we can repeat σ to obtain marking $m' + k.[i]$ for any k which contradicts the boundedness.

Now suppose a WFN N is classical sound but its closure $\overline{N}$ is not live nor bounded. Classical soundness implies that $[f]$ is the only marking reachable in N from $[i]$ with a token in f. Since the closing transition is only moving a token from f to i, the set of reachable markings of N is the same as of $\overline{N}$. Quasi liveness is already required by definition, so the WFN should not be bounded. This means that we have an infinite set of reachable markings. By Dickson's lemma there is an infinite sequence of reachable markings $m_1 \leq m_2 \leq ....$ Thus, $m_2 = m_1 + m'$ where $m' > 0$. This would require the existence of some firing sequence σ such that $(N : m_1 \xrightarrow{\sigma} [f])$. Hence, also $(N : m_2 \xrightarrow{\sigma} [f] + m')$ which is a contradiction. □

The closure of a WFN is interesting in itself for modeling purposes, as it models a repeatable process. Therefore we also call the closure of a WFN (classical) sound if it is live and bounded. Classical soundness can be verified using inspection of the reachability graph. We first have to determine if the reachability graph is finite, which is easy to verify: either we do not find any new marking or we encounter a marking that is containing an already found marking. In the latter case the net is unbounded and due to Theorem 4, it cannot be sound. If the reachability graph is finite, then we have to find for each state in the graph a path to the final marking, which can be done in a clever way (cf [76, 87]). The check for quasi liveness of the transitions can be performed in the same steps. In the next section, construction methods are presented that guarantee classical soundness.

A second important soundness notion is *k-soundness*. A WFN N is k-sound if in the workflow system started with k tokens in the initial place, it is always possible to reach a marking with only k tokens in the final place, i.e., the system $(N, [i^k], \{[f^k]\}$ should be weakly terminating. Note that quasi liveness is not required for k-soundness, and proper completion is implied in the same way as for classical soundness. If a WFN is 1-sound, we say the WFN is *weakly sound*. Remark that k-soundness does not imply $k + 1$-soundness nor $k - 1$-soundness, as the examples of Figures 7 and 8 show.

A WFN net is *generalized sound* if it is k-sound for all $k \in \mathbb{N}$. The verification of generalized soundness is much more difficult, as it needs a check for an infinite number of k-values. In [42], it has been shown to be decidable together with an algorithm for WFNs without inhibitor or reset arcs.

A transition-bordered or place-bordered WFN is k-sound (generalized sound) if its completion is k-sound (generalized sound). Generalized soundness is in particular important for stepwise refinement as construction method. While k-soundness is important for the processing of batches of a given size, generalized soundness is important for handling infinite streams of cases. Generalized

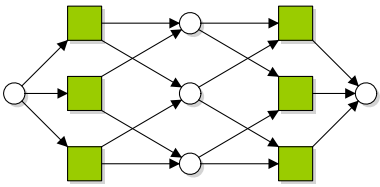


Fig. 7. 1-sound WFN, but not 2-sound

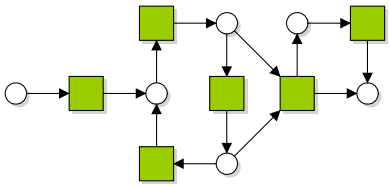


Fig. 8. 2-sound WFN, but not 1-sound

soundness states: all cases entered in the system will be handled, i.e., "what comes in will go out".

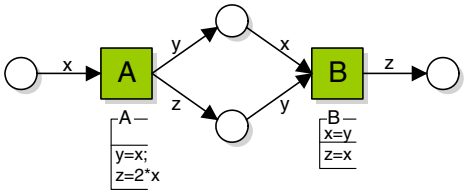
For the special classes of S-WFN and T-WFN it is proved that they are generalized sound (see [41]) by their structure. Also for FC-WFN there is a structural property: if a FC-WFN is 1-sound, then it is generalized sound (cf [46]). In the next section we will encounter more classes of WFNs that are generalized sound. We summarize these results:

Theorem 5 (Additional soundness properties). *Let* $N = (P, T, F, \{i\}, \{f\})$ *be a classical WFN.*

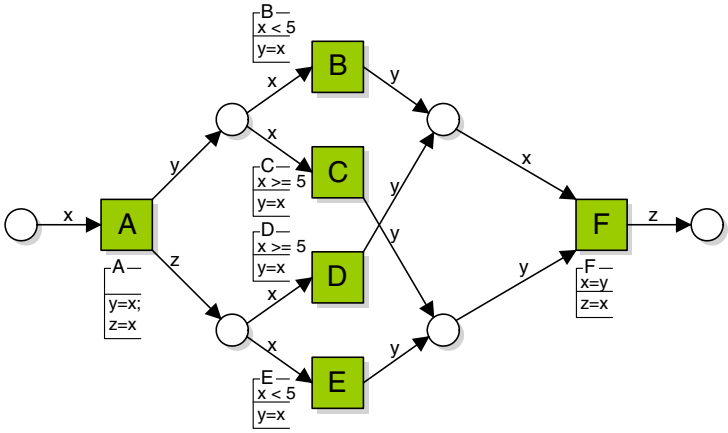
- *Generalized soundness is decidable*
- *S-WFN are generalized sound*
- *T-WFN are generalized sound*
- *If a FC-WFN is* 1*-sound, then it is generalized sound*

In literature, many different variations of soundness exist, like *up-to-k-sound*, *easy sound*, *relaxed sound* and *lazy sound*. We will discuss them briefly, for details see [6]. A WFN is up-to-k-sound if and only if it is l-sound for all $0 < l \leq k$. A WFN is easy sound if and only if it is possible that one case in isolation reaches the final marking. It is easy to verify (see [85]) that if a WFN is k-sound and easy sound, then it is up-to-k-sound. If we let a first case be handled in isolation, which is allowed by the easy soundness, then the other $k-1$ should be handled properly since the WFN is k-sound. A WFN is relaxed sound if and only if for each transition t there is a marking m reachable from an initial marking with one token in the initial place, in which t is enabled and if t fires the final marking should be reachable. Note that in a relaxed sound WFN transitions cannot be dead. There are algorithms to transform relaxed sound WFN into classical sound WFN (see [25]). Lazy soundness, considers the one case situation and requires that it must be always possible to reach a marking in which the final place has exactly one token, but there may be more tokens left (i.e., a relaxation of the proper completion property).

Up to now, we considered WFN with classical Petri net semantics, but we can also consider them with colored Petri net semantics. This gives all kind of interesting anomalies: since WFN without coloring can be sound while the colored version is unsound and vice versa, as is shown by the examples in Figure 9.



(a) Adding data makes the WFN unsound



(b) Adding data makes the WFN sound

Fig. 9. Soundness and data

Although the classical WFN of Figure 9(a) is sound, adding data makes the net unsound, as no reachable marking exist such that B is enabled in that marking. On the other hand, some behavior of the classical WFN is excluded by the data, as shown in Figure 9(b).

Similar anomalies can be created when considering time: a WFN with classical net semantics that is sound can be not sound in a timed semantics and vice versa. This means that we should be careful when we want to generalize soundness results for classical Petri nets to colored versions see [77]. When we consider only case identities then it is easier, because of the firing condition that only tokens with the same identity can be consumed and that the produced case tokens have the same identity as the consumed ones. This restricts the behavior, and in fact if a WFN is 1-sound in the classical sense then it is generalized sound if we consider case identities, since these identities make the cases independent of each other. The assumption of identifiable cases is quite natural in BP.

The last notion of soundness that we consider, is soundness for RCWFN: *resource constrained soundness* or *rc-soundness*. The intuition of soundness for RCWFNs is that the cases are handled as usual, which means that it is always

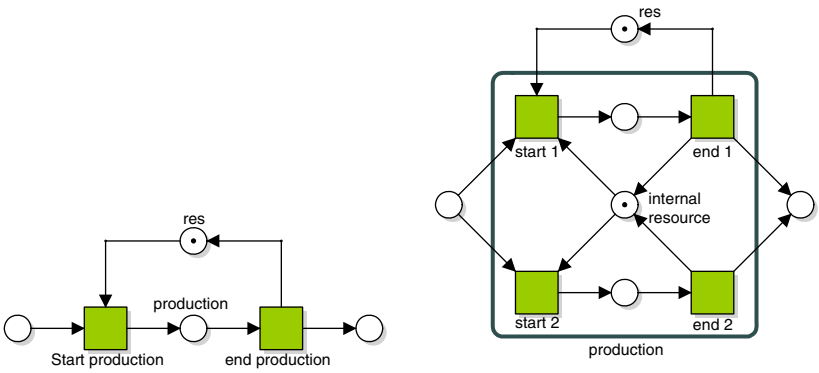
possible to reach the final marking (the marking with exactly k case tokens in the final place), if started with k cases. As we use the identity attribute, we can reformulate this requirement: the production workflow net should be 1-sound. Additionally there are (durable) resources needed to perform certain tasks, which means that transitions may consume and produce tokens for resource places. There are three important requirements: (1) when the production net reaches the final state, the resource places should have exactly the same amount of resources as at the beginning, (2) the total number of resources of any kind should not increase during the execution, and (3) if the system works properly in this sense that for a certain amount of resources per kind, then it should also work properly if we increase the number of resources of one or more kinds. Since k is an arbitrary number, we require these properties for all $k \in \mathbb{N}$. We give a formal definition:

Definition 6 (Resource constrained soundness). *Let $N = (P, R, T, F_c, F_r, \{i\}, \{f\})$ be a RCWFN with initial marking $[i^k] + r$ where $k \in \mathbb{N}$ and $r \in \mathbb{N}^R$ is a marking of the resource places only.*

- *N is (k,r)-rc-sound if for all m reachable from $[i^k] + r$, it holds that $m_R \leq r$ and marking $[f^k] + r$ is reachable.*
- *N is k-rc-sound if there is an $r \in \mathbb{N}^R$ such that N is (k, r')-rc-sound for all $r' \geq r$.*
- *N is rc-sound if it is k-rc-sound for all $k \in \mathbb{N}$.*

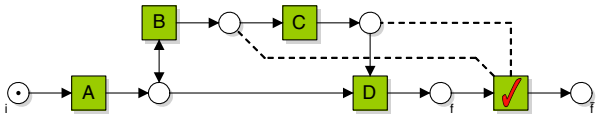
Note that if a RCWFN is sound then its production WFN is generalized sound (see [40]). For RCWFN in this form not many results are known. However for the variant with case identities, rc-soundness is decidable and there is an algorithm to verify it. Note that we only put identities to the case tokens, which make the cases independent of each other. If a RCWFN with case identities is sound then the production WFN is 1-sound. Note that the most simple form of resource modeling is when we model a task with start and stop event like in Figure 10(a) where there are one or more resource tokens dedicated for this task. It is easy to verify that any WFN where we model tasks with only one transition is branching bisimilar with the net where this transition is replaced by one of the constructs of Figure 10 when the end transitions receive the name of the task and the start transitions are silent. In this way, we can rely on the verification of (generalized) soundness of the nets without resources and one transition per task to verify rc-soundness of an RCWFN with this structure.

Up to now we did not consider inhibitor arcs or rest arcs. The verification of soundness properties when a WFN has inhibitor arcs or reset arcs is much more difficult, many questions are not decidable (see [6]). However, in the case when a WFN can always reach markings that are larger than the final marking in which only the final place is marked, the net can be made sound in a "brute force" way by adding a transition labeled ✓ that empties all places except the final place, as illustrated in Figure 11(a). Of course this is not the best way of making models!

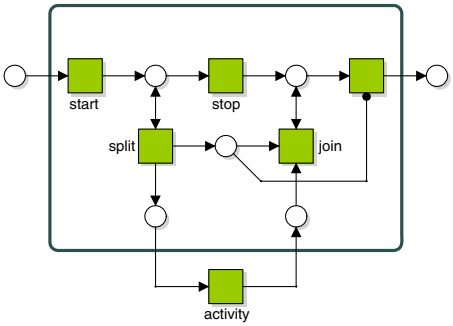


(a) Durable resource (b) Internal resource

Fig. 10. Activity with resources



(a) Reset arcs



(b) Inhibitor arcs

Fig. 11. Inhibitor arcs (bullet headed arc) and reset arcs (dashed line) repair soundness

Inhibitor arcs can also play a useful role. First note that inhibitor arcs only restrict the behavior of a system. They can destroy soundness of classical Petri nets as well as make unsound classical Petri nets sound. Consider a WFN N with inhibitor arcs and let the WFN N' be obtained from N by deleting the inhibitor arcs. If N' is bounded then N is also bounded and then we are able, by analysis of the reachability graph, to verify soundness of N. On the other hand, as inhibitor arcs only restrict behavior, it is also possible that unbounded nets become sound,

as illustrated in the example of Figure 11(b). In fact, this workflow models an important pattern to split a running case in separate instances, which in the end are combined again.

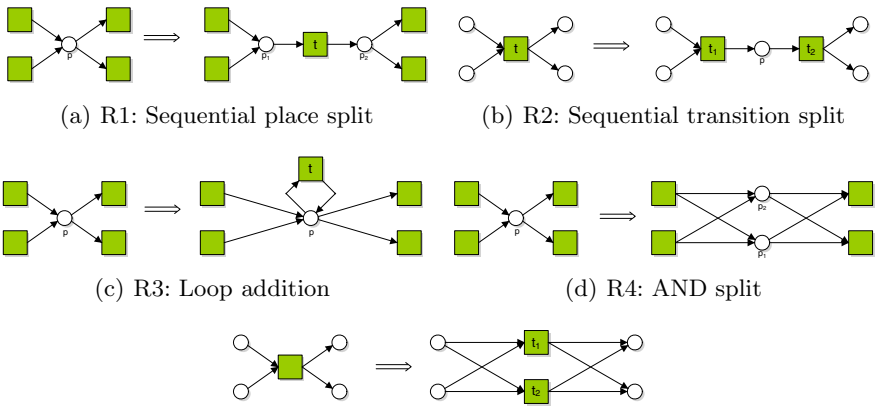
3.4 Construction Methods

Verification is an a posteriori approach: given a model, it checks whether properties like soundness hold. Although for classical Petri nets these properties are decidable, it is a very time consuming task. Therefore, we present in this section a construction method that preserves soundness: applying a rule from this method on a sound net results again in a sound net. This way, correctness of the model is guaranteed, provided that the initial model was sound.

We start with a class of WFNs that are generalized sound by their structure. First, note that a WFN consisting of a single place is generalized sound. Based on this observation, we can build a class of well-handled nets, called Jackson nets [36]. These nets are constructed with five refinement rules, as depicted in Figure 12.

The first rule (R1) is *sequential transition split*, which is shown in Figure 12(a). Given a place p in a WFN, we can split it into two new places p_1 and p_2 such that all the input transitions of p become input transitions of p_1, and, likewise, all output transitions of p become output transitions of p_2. We then add transition t such that place p_1 is the input place of t, and place p_2 is the output place of t. It is easy to verify that the refined net is generalized sound again.

The second rule (R2) is the dual of the first rule: instead of expanding a place, a transition is refined. Given a transition t in a WFN, we can split it into two new transitions t_1 and t_2 with a place in between such that the input places of t become the input places of transition t_1, place p becomes the only output place of t_1 and the only input place of transition t_2, and all output places of t in the



(a) R1: Sequential place split

(b) R2: Sequential transition split

(c) R3: Loop addition

(d) R4: AND split

(e) R5: OR split

Fig. 12. Refinement rules for Jackson nets

original net become the output places of transition t_2. The rule is depicted in Figure 12(b). We call this rule the *sequential transition split*.

To add loops in a WFN, the third rule (R3) allows to connect a new transition to an existing place in the WFN, such that this place is both the only input place and only output place of the newly added transition (see Figure 12(c)). Again, it is simple to prove that this refinement rule preserves generalized soundness. The fourth and fifth rule duplicate nodes in the WFN such that we can express AND and OR splits, respectively. The fourth rule duplicates a place (Figure 12(d)), which allows for parallelism, the fifth rule duplicates a transition (Figure 12(e)), which allows for choice.

Definition 7 (Jackson net). *A WFN is called a* Jackson net *iff it can be constructed from the singleton WFN* $(\{i\}, \emptyset, \emptyset, \{i\}, \{i\})$ *by applying the refinement rules R1,...,R5.*

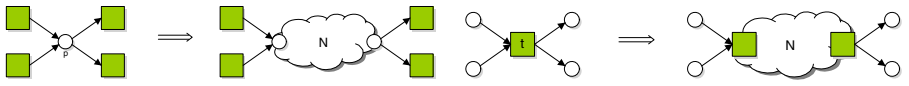
Theorem 8 (Jackson nets are generalized sound [36]). *Let N be a Jackson net. Then it is generalized sound.*

These rules are closely related to the rules of Murata [65]. In fact, when only considering refinement on nodes with a single input node and a single output node, these rules coincide. Only the rule that adds a place loop is omitted, as this rule requires the initial marking to be changed.

The nets in the class of Jackson nets are all well-formed [2], i.e., every AND split is complemented with an AND joint, and similarly for OR splits and joins. Often, this class of nets is too restrictive for modeling BP. We therefore introduce another class of workflow nets that are guaranteed to be generalized sound.

As any generalized sound net is bounded, there is an upper bound on the number of tokens in any marking reachable from the initial marking. When we refine a place by a WFN, then it should be sound for any number of tokens on the initial place of the net. Hence, if the WFN is generalized sound, we can safely refine the place with the WFN such that the refined net is generalized sound again (WP refinement, Figure 13(a)), as proven in [41]. A similar argument holds for the refinement of a transition by a WFN-t with a single initial node and a single final node (WT refinement Figure 13(b)).

Two important subclasses of WFN have been proven to be generalized sound by their structure: S-WFN and acyclic T-WFN-t with a single initial node and a single final place [41]. With these results we build a new subclass that is generalized sound by construction, which we call ST-nets. Both S-WFN and T-WFN-t are subclasses of the ST-nets. Given an ST-net, any place may be refined



(a) WP: refinement of a place by a WFN-s (b) WT: refinement of a transition by a WFN-t

Fig. 13. Place and transition refinement by a generalized WFN

by a place-bordered ST-net, and every transition may be refined by a transition-bordered ST-net. Both refinements preserve generalized soundness. Hence, any ST-net is generalized sound [41].

Definition 9 (ST-nets). *The class of* ST-nets $\mathcal{ST}$ *is recursively defined by:*

- *if* N *is an S-WFN, then* $N \in \mathcal{ST}$*;*
- *if* N *is an acyclic T-WFN-t with a single initial and final node, then* $N \in \mathcal{ST}$*;*
- *if* $N, W \in \mathcal{ST}$ *such that* W *is a WFN, and let* $p \in P_N$ *be a place of* N*. Then* $N \odot_p W \in \mathcal{ST}$*;*
- *if* $N, W \in \mathcal{ST}$ *such that* W *is a WFN-t, and let* $t \in P_N$ *be a transition of* N*. Then* $N \odot_t W \in \mathcal{ST}$*;*

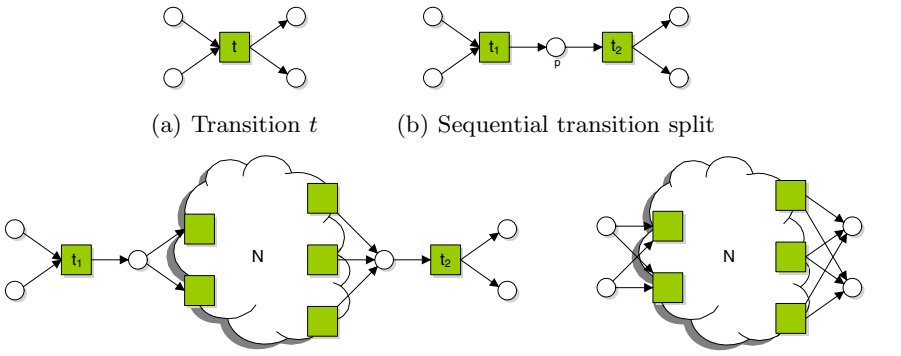
where $N \odot_n W$ *denotes the refinement of node* n *by* W*.*

Theorem 10. *Let* N *be an ST-net. Then it is generalized sound.*

It is often the case that an activity can be performed in different ways, e.g. by using different resources. To do so, we model the activity as a transition-bordered workflow net. Each initial transition represents a possible execution of the activity. We then refine the transition representing this activity by the transition-bordered workflow net. If both the original net and the refining net are generalized sound, the refined net is sound as well, which can be proven in a simple but elegant way.

Theorem 11. *Let* N *be a* k*-sound bordered WFN for some* $k \in \mathbb{N}$*. Let* W *be a generalized sound WFN-t. Then the refined net* $N \odot_t W$ *in which transition* $t \in T_N$ *is refined by* W *is* k*-sound.*

Proof. First, we refine transition t (Figure 14(a)) using the sequential place split such that we get two transitions t_1 and t_2, and a place p in between (Figure 14(b)). The resulting net is k-sound. We now refine this new place by



(a) Transition t (b) Sequential transition split

(c) Refinement with W^+ (d) Place substitution of [19]

Fig. 14. Steps in the proof of Theorem 11

the net W^+ (Figure 14(c)). By Theorem 9 of [41], the resulting net is k-sound. Places i and f of W^+ can be reduced using the "place substitution" rule of [19], which results in the net $N \odot_t W$ (Figure 14(d)). As the reduction rule preserves k-soundness, net $N \odot_t W$ is k-sound. □

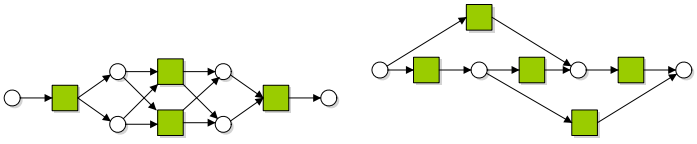
Note that the class of Jackson nets is not included in the class of ST-nets, and vice versa, as shown in Figure 15. These examples show that Jackson nets exist that are not an ST-net, and likewise, ST-nets exist that are not a Jackson net. However, the two rules, R1 and R2 are special cases of the rules WP and WT, respectively. This way, we are able to construct a new class of nets that are generalized sound by their structure.

Definition 12 (ST$^+$-Nets). *The class of* ST$^+$-Nets $\mathcal{ST}^+$ *is recursively defined by:*

- *if N is an S-WFN, then $N \in \mathcal{ST}^+$;*
- *if N is an acyclic T-WFN-t with a single initial node and a single final node, then $N \in \mathcal{ST}^+$;*
- *if $N, W \in \mathcal{ST}^+$ such that W is a WFN, and $p \in P_N$ a place of N. Then $N \odot_p W \in \mathcal{ST}^+$;*
- *if $N, W \in \mathcal{ST}^+$ such that W is a WFN-t, and $t \in P_N$ a transition of N. Then $N \odot_t W \in \mathcal{ST}^+$;*
- *if $N \in \mathcal{ST}^+$ and M can be constructed using rules R3,...,R5, then $M \in \mathcal{ST}^+$.*

Theorem 13. *Let N be an ST$^+$-net. Then it is generalized sound.*

Not all constructs needed in modeling BP can be expressed using an ST-net or a Jackson net. To guide the modeler, many patterns for modeling BP have been identified. In [10] many workflow patterns are collected based on best practices. Most of the patterns concern the control flow. We have seen already some of them. A special construct is needed for handling the multiple instance patterns, which allow for choosing the number of instances executed at runtime. One way is to model these patterns by using the splitter pattern as shown in Figure 11(b). This pattern allows to execute the activity multiple times, without specifying an upper bound at design time. It is easy to see that although the places between split and join are unbounded, the pattern is weakly terminating, i.e., placing a token in the initial place of the pattern always leads to only a single token in the final place, while all other places are empty.



(a) Jackson net, but not ST (b) ST net, but not Jackson

Fig. 15. Not all Jackson nets are ST-nets and vice versa

4 Modeling Cooperating Business Processes

Most organizations are divided into more or less autonomous cooperating business units. Each business unit can be seen as an organization. These business units cooperate to achieve the common goals of the larger organization. To realize this, the business units need to communicate. They need information from other business units to reach their goals, or on demand they can deliver requested information so that other units can accomplish their goals. As a consequence, this communication also needs to be reflected in the BPs of the different cooperating business units.

A second trend in the last decade is that organizations focus more and more on their core activities, while outsourcing all other activities. As a consequence, organizations form partnerships to achieve common goals. These organizations together form a larger organization. Again, communication is a key factor in realizing the common business goals of the larger organization. In the mean time, each organization still has its own business goals, that need to be achieved as well.

4.1 Service Oriented Approaches

Business units within an organization, or organizations in a cooperation can be seen as components of a larger system. A *component* offers services via its interfaces, and in order to deliver its services, it needs services of other components. This way, a component has two roles: a *service provider* and a *service consumer*. From a business oriented view, a component *sells* services, and in order to meet its commitments, it *buys* services of other components. In this way, a tree of components is built up to deliver a service.

Software systems have a similar component-based structure. Many different definitions exist for components. We adopt the definition of [84], which defines a *component* as a unit of composition with contractually specified interfaces and explicit context dependencies only. This definition shows two important aspects of a component: first, a component is a subsystem implementing a set of *coherent* functionalities. Its interfaces to the outside are listed explicitly in a contract, i.e., it defines how other components can access its functionality via its interfaces. Secondly, a component should be as independent as possible. A component may need other components, but these dependencies are known in advance, and they are only accessed via the specified interfaces. Note that the dependencies are on a service level, rather than defining which specific components are needed.

Current paradigms like service oriented computing (SOC) and service oriented architectures (SOA) [3,14,68] enable this business oriented view in system construction. The main principle behind SOC is to aggregate services to service compositions to implement a BP. SOA is a technology to design and execute services according to the SOC paradigm. In SOA, the network of communicating services is built at runtime, and no service knows the whole network at any point in time. An important concept in SOA is the *service broker*: a trusted third party where all components publish the services they provide (step 1 in Figure 16).

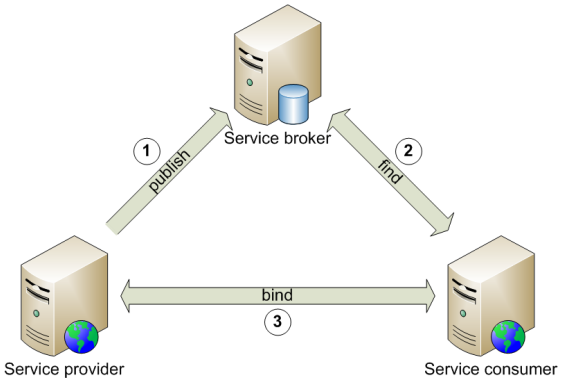


Fig. 16. SOA principle: a provider publishes it services at a broker (1). A consumer uses the broker to find a service (2) and binds to it (3).

A component that needs a service, queries the broker to find a service. The broker returns a list of possible components that deliver the requested service (step 2), after which the consumer binds to one of these components (step 3) and start communicating.

Many different formalisms and techniques exist in industry to support the modeling of service oriented paradigms. Current techniques are almost all web focused. On a low level, the Simple Object Access Protocol (SOAP) [63] defines the structure of the messages sent between components. To describe the interface of a component, the Web Service Description Language (WSDL) [21] is used. It defines the different types of messages the service can send and receive.

Each component has its own internal process to *orchestrate* its services: it defines the order in which messages are sent and received by the component. One of the main languages to model the service orchestration in a component is the Web Service Business Process Execution Language (WS-BPEL) [15]. Modeling techniques for component interaction should support the dynamics of the network formed by the components, called a *choreography*. One of the main industry standards to model this is the Web Service Choreography Language (WS-CDL) [52]. In WS-CDL, one models the possible interaction patterns between so called parties. An organization implements a party, such that it exactly mimics the behavior defined in the interaction patterns. In this article, organizations can be seen as software components and vice versa.

4.2 Modeling Interaction With Petri Nets

Communication between organizations is message driven: an organization requests a service provided by another organization, and eventually, this organization delivers it. Petri nets are well suited to model message driven communication: places in the Petri net serve as message buffers, from which messages are read in *random order*. An organization can deliver multiple services, each service can be seen as an *interface* to the outside. An interface consists of places

that are either input places for the service, i.e., *requesting messages*, or output places for the service: i.e., *sending messages*. We do not allow for places in the interface that are both input and output, as such a place can be interpreted as a shared variable, which is not allowed in asynchronous communication. We call such a Petri net with interfaces an *open net* [58, 60].

In modeling cooperating business processes, places resemble status, activity, and message buffers. The business processes of each organization (or business unit) are modeled as a WFN with interface places. As business processes are repeatable, we model a component by the fused closure of the WFN. The fused initial place and final place is called the *idle place*. If a component needs a service of another component, the corresponding interfaces are connected. As a consequence, to compose two components to model their interaction, they should only share some interfaces, and these interfaces should be their complements. In the composition, the interface places are fused and become internal places of the newly created component. The interfaces of the components that are not connected become the interfaces of the newly composed component.

As an example, consider the open net N in Figure 17. This net has three interfaces: G, H and J. Interface G consists of three output places, a,c and d, and two input places: b and e. Net M has a similar port G, with two output places, b and e, and three input places, a, c and d. Interface G of M is the complement of interface G of N. Hence, the two nets are composable with respect to interface G. In the composition, the interface places of the two components are fused based on their name and become internal places of the new component. For example, place a of N is fused with place a of M, i.e., in the composition with respect to interface G, denoted by $N \oplus_G M$, place a has the input transitions of N

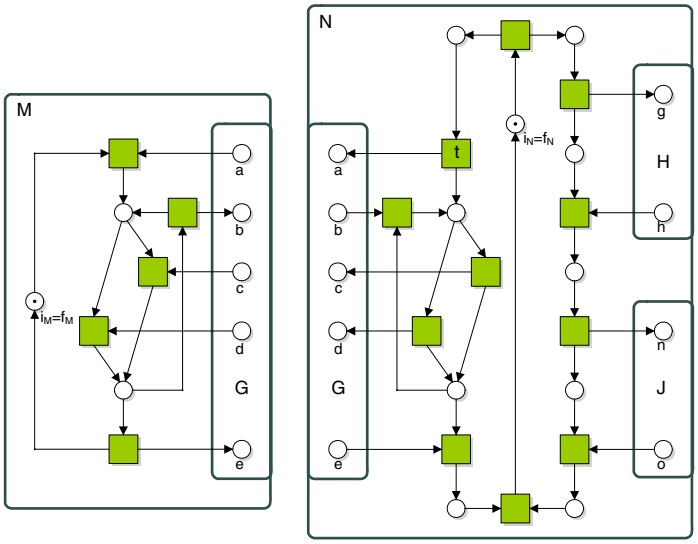


Fig. 17. Two components that share an interface

and output transitions of M. The newly created component has two interfaces, namely those of N: interface H and J. In practice, it is useful to have a rename operator in order to give places that should be fused the same name.

Many similar notions for modeling component based systems using Petri nets exist [34,54,55]. Also the Petri Net Markup Language (PNML) [47] has a module concept with interfaces. In [24], the authors propose to model choreography using Interaction Petri nets, which is a special class of Petri nets, where transitions are labeled with the source and target component, and the message type being sent. For each of the components, a Petri net with an interface is extracted. The interaction Petri net is then realizable if the composition of the behavioral interfaces is branching bisimilar related with the interaction Petri net.

4.3 Verification Methods

An important behavioral property for components is that its internal behavior should be correct: disregarding the interface, the component should be weakly terminating. Verification of asynchronously communicating components is known to be a hard problem. Because of the high degree of concurrency in such systems, model checking a complete system often becomes infeasible. A second problem is that the complete system is usually not known. Only at runtime components decide to cooperate. Therefore, *compositional verification* is needed to check these systems on conformance: given that each component is correct, and each pair of connected components satisfy some conditions, we want to conclude that the whole system is correct.

Just requiring each composition of connected pairs of components to be sound is not sufficient to guarantee the correctness of the composition, as shown in Figure 18. In this example, both $A \oplus B$ and $B \oplus C$ are sound. However the composition $A \oplus B \oplus C$ has a deadlock, since this composition introduces a cyclic dependency over the three components: after firing transition v, the system is in deadlock.

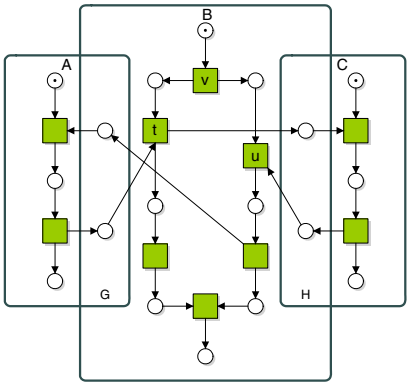


Fig. 18. The composition of three components

Behaviorally, communicating components can be seen as *partners* that together try to reach some desired state. A component cannot communicate with every partner that has the same interface. There are additional requirements on the environment based on the internal process of a component. The *operating guideline* [58] of a component is an automaton that represents all partners in a finite manner. Wolf [90] shows that if a service has a partner, a most-permissive partner exists, i.e., a partner exists such that all partners simulate this partner.

Given a service R, to check whether some service P is a partner, it should "follow" the operating guideline, i.e., partner P should simulate the operating guideline of R. However, this condition is not sufficient [90]. Therefore, each state in the operating guideline is annotated with a boolean expression stating which messages should be sent and received in each state. However, the operating guideline is currently only defined for deadlock-freedom. In [81] the authors present an extension to test whether all transitions are covered.

Within a system, if a component S is *replaced* by some other component T, the system should still function, i.e., the system should not notice the replacement. Formally, this means that every partner of S should also be a partner of T. We call this relation the *accordance pre-order* [64, 80]: T accords with S if every partner of S is a partner of T. The operating guideline can be used to decide whether a component accords to another component. In this way, the accordance pre-order can be used to decide substitutability of services.

In the setting where the network of communicating components is restricted to acyclic graphs, i.e., each component only communicate with a "parent" component, the parent component "buys" services from a child components. The accordance relation for livelock-freedom is sufficient to compositionally verify the network. However, a solution to decide accordance for weak termination is not available [79]. In [61], the authors prove that for general open nets the question whether a component has a partner is undecidable. Current research results (cf. [59, 80, 90]) are based on a message bound.

In [8, 88], the authors search for a sufficient condition to conclude weak termination based on an extra condition on the communication between each pair of communicating components. They identify several *communication patterns* that are sufficient for compositional verification of soundness. The basic communication pattern defined is the *identical communication pattern*. Given a composition of three components A, B and C, such that the composition of A and B and of B and C is sound, and A and C are disjoint, then C cannot hamper B, i.e. the composition $B \oplus C$ should mimic every firing sequence B can execute. Thus, for every firing sequence σ in B leading to its final marking, a firing sequence $\tilde{\sigma}$ should exist in the composition $B \oplus C$ that leads to the final marking of the composition, and the projection of $\tilde{\sigma}$ on the transitions of component B should be equal to σ. This way, component A does not notice whether it is communicating to component B or to the composition $B \oplus C$. The notion is closely related to simulation.

4.4 Construction Methods

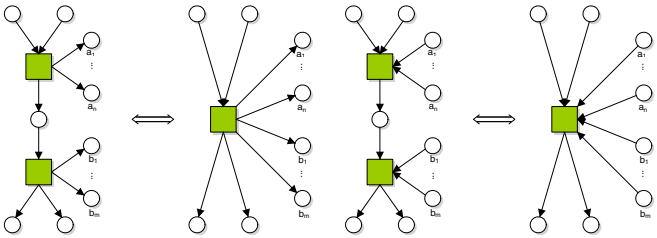
Although compositional verification of asynchronously communicating components is possible, it still is a very time consuming activity. For SOA-based architectures, i.e., in which it is unknown which components the component to be designed will cooperate with, verification needs to be done at runtime by the broker. In SOC based architectures, i.e., the components with which the component to be designed is known beforehand, construction methods exist that guarantee correctness criteria like (generalized) soundness. In this section we discuss two approaches to guarantee soundness. These construction methods allow for the construction of general networks of communicating components.

Reordering Communication. The first approach is based on public and private views of a model [11]. One first models the whole BP, and verifies soundness of the constructed WFN. Then, the WFN is divided in public views for each of the parties involved in the BP. This public view serves as a contract between the different parties. Each party implements its public view as a private view. If the private view accords to the public view, i.e., each partner of the public view is a partner of the private view, then the system composed of all the private views will be sound as well.

Like for WFN, rules exist for the refinement from a public view to a private view. The refinement rules defined in [36] for Jackson nets and the refinement rules WP and WT can be used as long as no communicating transitions, i.e., transitions that are connected to an interface, are involved. For the communicating transitions, [11] provides reordering rules that preserve accordance (see Figure 19), enabling the specialization of a public view to a private view. An overview of patterns can be found in [12].

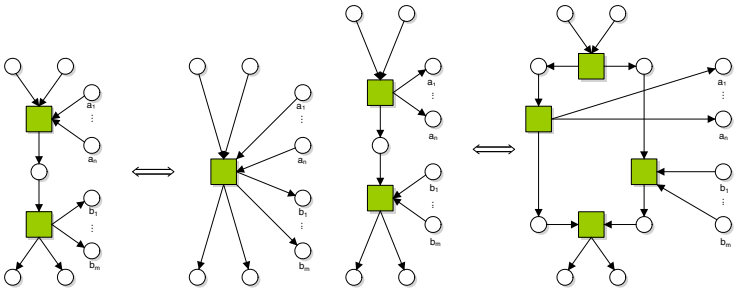
The first two rules show that a sequence of only sending transition (Figure 19(a)) or only receiving transitions (Figure 19(b)) may be refined in any order, as such a sequence can be mapped on a single transition that sends (receives) all messages at once. The third reordering rule (Figure 19(c)) shows that if first a sequence of receiving transitions occur followed by a sequence of sending transitions, the whole sequence can be replaced by a single transition. The last refinement rule (Figure 19(d)) shows that a sequence of sending transitions followed by a sequence of receiving transitions can be replaced by two parallel branches one for the sending transitions and one for the receiving transitions. In [56], the authors show that these rules can be applied to WS-BPEL, slightly relaxing the relation between abstract and executable WS-BPEL processes.

Refinement Rules. The rules presented in [11] show that accordance is preserved under the reordering rules. However, these rules do not extend the behavior of the components. In the remainder of this section, we will present three refinement rules that extend the behavior of a network of communicating components [44,88]: The first rule refines a single component, the second rule refines the communication between two components, and the last rule introduces a new component in the network.



(a) Rule C1: reordering send actions

(b) Rule C2: reordering receiving actions



(c) Rule C3: Sequence of first receive then send

(d) Rule C4: Sequence of first send then receive

Fig. 19. Reordering communication

Within a business process modeled as a Petri net, a place can resemble both an activity or a state. The place refinement defined in Section 3.4 allows us to refine a place with another WFN that is generalized sound. As for open nets the concept of generalized soundness is not defined, we require the place that will be refined to be safe: if the place is safe, it may be refined by an open net whose inner structure is a WFN. The resulting net has both the interfaces of the original net as well as the interfaces of the refining net, as shown in Figure 20.

This definition of place refinement propagates the ports of the refining component to the original component. At a first glance, this definition seems to contradict the paradigm of information hiding. However, the definition allows for the refinement of a component by a composed component, as long as this composition remains a workflow component. This way, the ports remain invisible to the environment of the original component.

In the refinement of a system of asynchronously communicating components, different components can contain places that together represent a single procedure. To refine the system with the actual procedure, which mostly includes communication between the different components, these places need to be simultaneously refined by the actual procedure.

Not every subset of places in such a system can be refined while preserving soundness. First, a marking in the system should exist in which all places to be

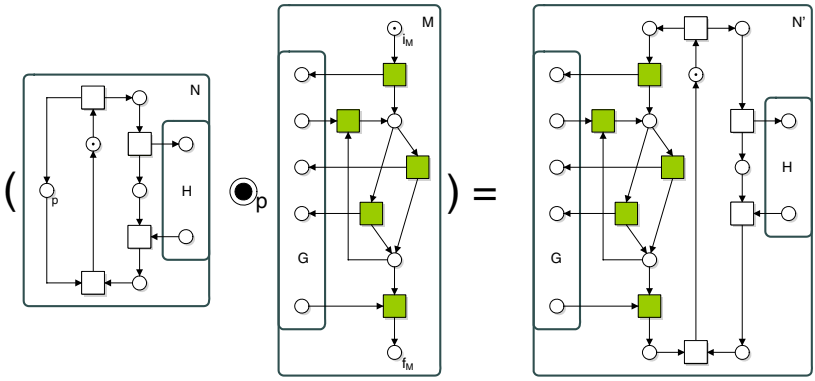


Fig. 20. Refinement of place p in N by component M

refined are marked. Second, such a place can only become marked again after all other places to be refined have been marked, i.e., when refining these places by rule R1 (Figure 12(a)) the synchronic distance [83] of the newly added transitions should be 1. Thirdly, from any marking reachable a path exists that ensures that all places to be refined are marked before one of these places become unmarked again.

Figure 21 depicts a May/Exit transition system that expresses the three conditions. Solid transitions are exit transitions, and from every state in the system it should be possible to reach the final marking using only exit transitions. Each state is annotated by two sets of places: the set of places that has been marked in the current cycle and the set of places that already have become unmarked. The transitions p and q in this system are mapped onto the transitions in the preset of the places p and q, respectively, and the transitions p' and q' are mapped onto the transitions in the preset of the places p and q, respectively. All other transitions are mapped to the silent transition. If every firing sequence in the Petri net can be replayed in the May/Exit transition system, and for each marking a firing sequence to the final marking exist using only the solid transitions of the May/Exit transition system, then the places p and q are synchronizable [38, 45].

Refinement of a set of places by an actual procedure can be seen as the refinement by a *communication protocol* between the components. A communication protocol is a set of open nets whose inner structure is a WFN, and each open net has at most one interface for each of the other open nets in the protocol, and no other interfaces exist. In this way, a communication protocol models the cooperation between the different parties.

Using the concept of synchronizable places and the communication protocol, we can define a refinement rule for the communication between components. Given a system of communicating components, a set of synchronizable places and a communication protocol between as many parties as there are synchronizable places. In the refinement, we refine each synchronizable place with a party from

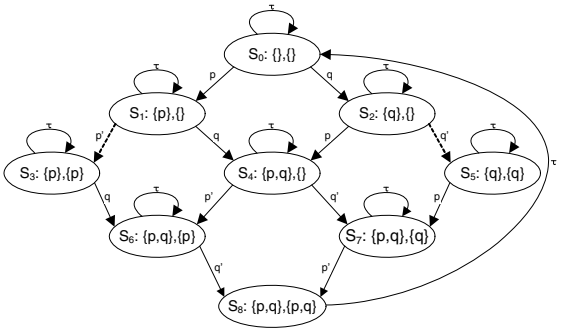


Fig. 21. May/Exit transition system for synchronizable places p and q

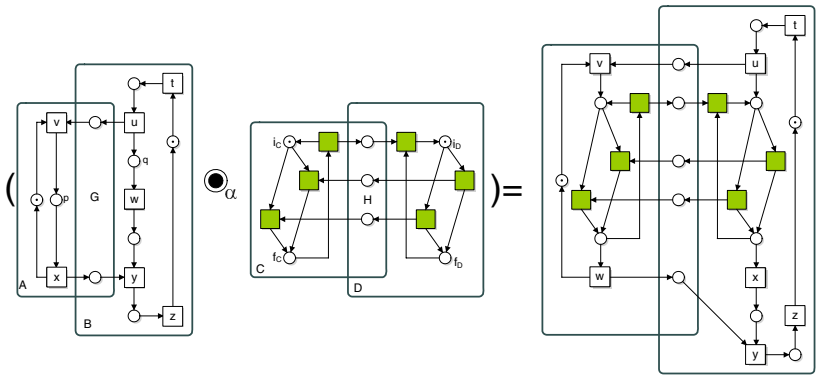


Fig. 22. Refinement of synchronizable places p and q with $\alpha = \{p \mapsto C, q \mapsto D\}$: place p is refined by C and place q simultaneously by D

the communication protocol. If the communication protocol is sound, the refined net is sound again.

Whereas the previous two rules focused on the extension of existing components, these rules do not allow for expansion of the network by adding new components. With the next rule, it is possible to connect new components in a system such that the system remains sound. The rule is based on the principle of *outsourcing*.

Consider Figure 23. For example, if place p has the meaning that when a token resides in it, "an item is produced", and the decision is taken to outsource the production activity, we can add two transitions: a "start producing item" and a "finish producing item". Then the start transition initiates the component producing the item, and the finish transition fires if the item is produced. In this way, we create a new interface for outsourcing the activity, which allows us to connect it to a new component. In fact, we replace place p by a communication protocol between two parties: the existing component and a new component.

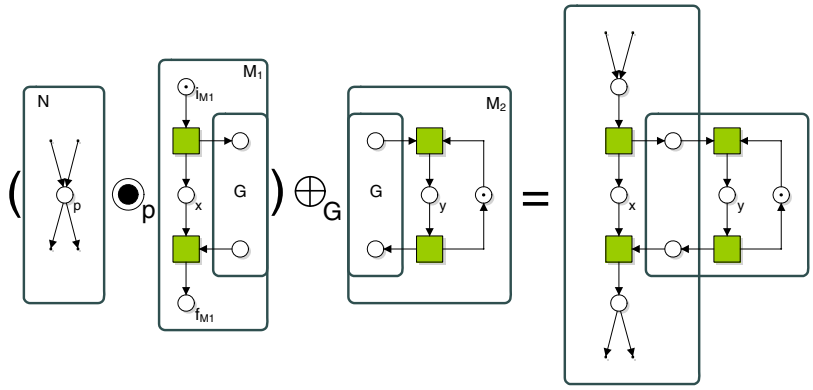


Fig. 23. Extending the composition with a new coupled component

The new component is formed by taking the fused closure of the open net in the communication protocol. If every firing sequence from the idle state of the new component back to this idle state starts with receiving a message and ends with sending a message, then the refinement preserves soundness.

4.5 Communication Protocols

The last two refinement rules presented in the previous section are based on sound communication protocols. Two special subclasses of WFNs are the class of acyclic marked graphs and the class of state machines. These nets are generalized sound by their structure [41]. In this section, we will discuss a subclass of communication protocols based on these classes of WFNs that are sound by their structure. We focus here on sound communication protocols between two parties.

Soundness of a communication protocol means that all parties should always be able to reach their final marking, and no messages are pending in the interfaces. Every communication protocol is a WFN-s net. As the completion of a WFN-s is bisimilar to that WFN-s when hiding the initial and final transitions, soundness of the WFN implies soundness of the communication protocol.

As proven in [41], acyclic marked graphs are generalized sound and safe by their structure. Now let us consider an open net whose inner structure is a T-WFN, and each interface place is connected to exactly one transition, then the composition of two of these nets is again a marked graph. Hence, if the composition is acyclic, the WFN of the communication protocol is sound, and thus also the communication protocol. As a result, the class of communication protocols whose inner structure is an acyclic marked graph are sound by their structure. We name this class AT-nets.

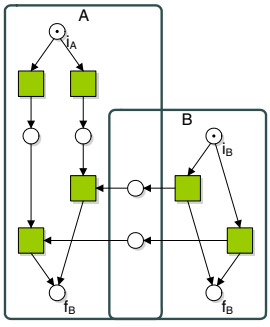
As communication protocols are also asynchronously communicating systems, we can apply the refinement rules of the previous section on them. However,

as the refinement over components requires synchronizability of the places to be refined, we still need to check which pairs of places are synchronizable. In an acyclic marked graph, every place will be marked exactly once. Hence, if a marking exists in which two places are marked at the same time, these places are synchronizable. For this subclass of communication protocols, we can express this as a graph property: two places are synchronizable if there does not exist a directed path between the two places. Hence, the set of all synchronizable places can be computed from the structure of the Petri net.

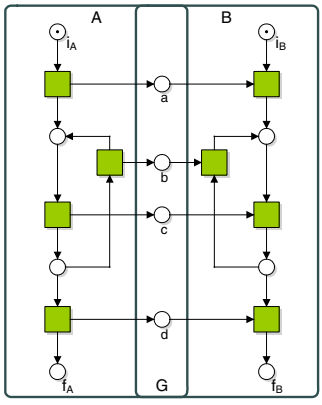
A second class of WFNs that is generalized sound by structure is the class of state machine WFNs. Consider an open net whose inner structure is a S-WFN. Then unlike the composition of open nets whose inner structure is T-WFN, the composition of two of these open nets is not a state machine. Composing two state machines introduces concurrency; it is very simple to compose two open nets with an inner S-WFN structure resulting in a composition that is not sound.

For example, consider the example of Figure 24(a). In this net, first component A makes a decision, then component B makes a decision. As both decisions are independent, if B makes the wrong choice, the composition reaches a deadlock. A solution to overcome this problem is to only connect isomorphic open nets with an inner S-WFN structure such that the interface places are determined by the isomorphism relation, and transitions in conflict either all send or receive. However, as Figure 24(b) shows, this is not sufficient. Also the direction of the communication matters: any loop should contain at least one sending and one receiving transition, otherwise the composition can become unbounded.

These observations show that compositions of open net with an inner S-WFN structure should "agree" on the isomorphism: each transition should only communicate with its isomorphic counter part, all transitions in conflict



(a) Decisions should be "in sync"



(b) Direction of communication is important

Fig. 24. unsound compositions of state machine components

either all send a message or all receive a message, and each loop contains at least one sending and one receiving transition. We call this set of communication protocols IS-nets.

Markings of such a composition have a special property: as each of the components is an S-WFN, any marking in the composition can be split into a marked place of the first component, a marked place of the second component and some interface places are marked. As each loop contains both a sending and a receiving transition, the composition is safe: no interface place can contain more than one token. Therefore, if the composition agrees on the isomorphism, it is always possible to reach a marking in which the interface places are empty, and, if this is the case, the only marked places are their isomorphic counter parts. As a result, if the composition of two isomorphic open nets with an inner S-WFN structure agrees on the isomorphism, it is both safe and sound. A direct consequence of this property is that each pair of a place and its isomorphic counterpart is also synchronizable.

Based on the class of AT-nets and IS-nets, we recursively define a larger class of sound and safe communication protocols called ATIS-nets. Using the refinement rule over components, we can refine any two synchronizable places by a sound and safe communication protocol. Hence, if we refine two synchronizable places in an ATIS net with an ATIS net, the result is sound and safe again.

5 Conclusions

We have shown how relevant aspects of business processes can be modeled with Petri nets and explained why soundness, or weak termination, is an important sanity check for process modeling. We have presented a number of model construction rules allowing to develop sound by construction business processes. Some of these rules are refinement rules, where a node of a Petri net gets refined by another Petri net, or composition rules, where the original nets become subnets of the composed net. These construction rules can be combined with workflow patterns, which in fact define best practices.

Besides single business processes we also considered the modeling of cooperating, communicating, business processes. This topic is especially important since in practice, business processes seldom operate in isolation. We extended the construction rules for single business processes to construction rules for sets of communicating business processes. Since communicating business processes can be transformed into one big business process, the construction methods for communicating business processes can be used for modeling of single business processes as well.

Business processes do not only occur in physical organizations but also in software systems. If software systems are developed according to the paradigm of service oriented computing, then components that deliver services to each other form a direct analogy with cooperating organizations. Each component has an internal orchestration process, which is in fact a business processes, while sets of communicating components are in fact communicating business processes.

The whole system can be considered as one organization and all the components as business units. So the theory developed in this paper can be applied to component-based software systems as well.

Acknowledgements. The authors would like to thank Christian Stahl for the useful discussions and his valuable comments on cooperating business processes.

A Basic Notations

Let S be a set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. We use $|S|$ for the number of elements in S. Two sets U and V are *disjoint* if $U \cap V = \emptyset$. We denote the cartesian product of two sets S and T by $S \times T$. On a cartesian product we define two projection functions $\pi_1 : S \times T \to S$ and $\pi_2 : S \times T \to T$ such that $\pi_1((s,t)) = s$ and $\pi_2((s,t)) = t$ for all $(s,t) \in S \times T$. We lift the projection function to sets in the standard way.

A *bag* m over S is a function $m : S \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, \ldots\}$ denotes the set of natural numbers. We denote e.g. the bag m with an element a occurring once, b occurring three times and c occurring twice by $m = [a, b^3, c^2]$. The set of all bags over S is denoted by $\mathbb{N}^S$. Sets can be seen as a special kind of bag where all elements occur only once; we interpret sets in this way whenever we use them in operations on bags. We use $+$ and $-$ for the sum and difference of two bags, and $=, <, >, \leq, \geq$ for the comparison of two bags, which are defined in a standard way. The projection of a bag $m \in \mathbb{N}^S$ on elements of a set $U \subseteq S$, is denoted by $m_{|U}$, and is defined by $m_{|U}(u) = m(u)$ for all $u \in U$ and $m_{|U}(u) = 0$ for all $u \in S \setminus U$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \ldots, n\}$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. Let $\nu, \gamma \in S^*$ be two sequences. *Concatenation*, denoted by $\sigma = \nu; \gamma$ is defined as $\sigma : \{1, \ldots, |\nu| + |\gamma|\} \to S$, such that for $1 \leq i \leq |\nu|$: $\sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$: $\sigma(i) = \gamma(i - |\nu|)$. *Projection* of a sequence $\sigma \in S^*$ on elements of a set $U \subseteq S$, denoted by $\sigma_{|U}$, is inductively defined by $\epsilon_{|U} = \epsilon$ and $(\langle a \rangle; \sigma)_{|U} = \langle a \rangle; \sigma_{|U}$ if $a \in U$ and $(\langle a \rangle; \sigma)_{|U} = \sigma_{|U}$ otherwise. The *Parikh vector* of a sequence σ, denoted by $\overrightarrow{\sigma}$ is inductively defined by $\overrightarrow{\epsilon} = \emptyset$ and $\overrightarrow{\langle a \rangle; \sigma} = [a] + \overrightarrow{\sigma}$ for all $a \in S$.

If we give a tuple a name, we subscript the elements with the name of the tuple, e.g. for $N = (A, B, C)$ we refer to its elements by A_N, B_N, and C_N. If the context is clear, we omit the subscript.

Labeled Transition Systems. A *labeled transition system* (LTS) is a 5-tuple $(S, \mathcal{A}, \longrightarrow, s_i, \Omega)$ where

- S is a set of *states*;
- $\mathcal{A}$ is a set of *actions*;
- $\longrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times S$ is a *transition relation*, where $\tau \notin \mathcal{A}$ is the silent action [17];
- $s_i \in S$ is the *initial state*; and
- $\Omega \subseteq S$ is the set of *final states*, also called *accepting states*.

Let $L = (S, \mathcal{A}, \longrightarrow, s_i, \Omega)$ be an LTS. For $s, s' \in S$ and $a \in \mathcal{A} \cup \{\tau\}$, we write $(L : s \xrightarrow{a} s')$ iff $(s, a, s') \in \longrightarrow$. If $(L : s \xrightarrow{a} s')$, we say that state s' is *reachable* from s by an action labeled a. A state $s \in S$ is called a *deadlock* if no action $a \in \mathcal{A} \cup \{\tau\}$ and state $s' \in S$ exist such that $(L : s \xrightarrow{a} s')$. We define $\Longrightarrow$ as the smallest relation such that $(L : s \Longrightarrow s')$ if $s = s'$ or $\exists s'' \in S : (L : s \Longrightarrow s'' \xrightarrow{\tau} s')$. As a notational convention, we may write $\xRightarrow{\tau}$ for $\Longrightarrow$. For $a \in \mathcal{A}$ we define $\xRightarrow{a}$ as the smallest relation such that $(L : s \xRightarrow{a} s')$ if $\exists s_1, s_2 \in S : (L : s \Longrightarrow s_1 \xrightarrow{a} s_2 \Longrightarrow s')$. We lift the notations of actions to sequences. For the empty sequence ϵ, we have $(L : s \xrightarrow{\epsilon} s')$ iff $(L : s \Longrightarrow s')$. A sequence $\sigma \in \mathcal{A}^*$ of length $n > 0$ is a firing sequence from $s_0, s_n \in S$, denoted by $(L : s_0 \xrightarrow{\sigma} s_n)$ if states $s_{j-1}, s_j \in S$ exist such that $(L : s_{j-1} \xRightarrow{\sigma(j)} s_j)$ for all $1 \leq j \leq n$. If a firing sequence σ exists such that $(L : s \xrightarrow{\sigma} s')$ we say that s' is *reachable* from s. The set of all reachable states from s are the states from the set $\mathcal{R}(L, s) = \{s' \mid \exists \sigma \in \mathcal{A}^* : (L : s \xrightarrow{\sigma} s')\}$.

An LTS $L = (S, \mathcal{A}, \longrightarrow, s_i, \Omega)$ is *weakly terminating* if $\Omega \cap \mathcal{R}(L, s) \neq \emptyset$ for all states $s \in \mathcal{R}(L, s_i)$, i.e. from every state reachable from the initial state some final marking can be reached.

Definition 14 (Hiding). *Let $L = (S, \mathcal{A}, \longrightarrow, s_i, \Omega)$ be an LTS. Let $H \subseteq \mathcal{A}$. We define the operation τ_H on an LTS by $\tau_H(L) = (S, \mathcal{A} \setminus H, \longrightarrow', s_i, \Omega)$, where for $m, m' \in S$ and $a \in \mathcal{A}$ we have $(m, a, m') \in \longrightarrow'$ if and only if $(m, a, m') \in \longrightarrow$ and $a \notin H$ and $(m, \tau, m') \in \longrightarrow'$ if and only if $(m, \tau, m') \in \longrightarrow$ or $(m, a, m') \in \longrightarrow$ and $a \in H$.*

An LTS L' delay simulates an LTS L if in every two related states, each action L can do, LTS L' can perform as well, possibly after some silent steps. If both L' simulates L and L simulates L' with simulation relations R and R^{-1}, we say L and L' are delay bisimilar.

Definition 15 (Delay (bi)simulation). *Let $L = (S, \mathcal{A}, \rightarrow, s_i, \Omega)$ and $L' = (S', \mathcal{A}', \rightarrow', s_i', \Omega')$ be two LTSs. The relation $Q \subseteq S \times S'$ is a* delay simulation, *denoted by $L \preceq_Q L'$, if:*

1. $s_i \, Q \, s_i'$;
2. $\forall s_1, s_2 \in S, a \in \mathcal{A} \cup \{\tau\}, s_1' \in S' : ((L : s_1 \xrightarrow{a} s_2) \wedge s_1 \, Q \, s_1') \implies (\exists s_2' \in S' : (L' : s_1' \xRightarrow{a} s_2') \wedge s_2 \, Q \, s_2')$; *and*
3. $\forall s' \in S', s_f \in \Omega : s_f \, Q \, s' \implies s' \in \Omega$.

If both Q and Q^{-1} are delay simulations, Q is a delay bisimulation *denoted by $L \simeq_Q L'$.*

In the remainder, if we use the term (bi)simulation, we refer to delay (bi)simulation. For a more elaborate overview of simulation relations, we refer the reader to [29].

Petri Nets. A *Petri net* N is a 3-tuple (P, T, F) where (1) P and T are two disjoint sets of *places* and *transitions* respectively; (2) $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the *flow function*. The elements from the set $P \cup T$ are called the *nodes* of N. A pair $(n_1, n_2) \in (P \times T) \cup (T \times P)$ is called an *arc* if $F(n_1, n_2) > 0$. Places are depicted as circles, transitions as squares. For each pair $(n_1, n_2) \in (P \times T) \cup (T \times P)$ such that $F(n_1, n_2) > 0$, an arc is drawn from n_1 to n_2. Two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$ are *disjoint* if and only if $(P \cup T) \cap (P' \cup T') = \emptyset$. Let $N = (P, T, F)$ be a Petri net. Given a node $n \in (P \cup T)$, we define its *preset* ${}^\bullet_N n = \{n' \mid F(n', n) > 0\}$, and its *postset* $n^\bullet_N = \{n' \mid F(n, n') > 0\}$. We lift the notation of preset and postset to sets and sequences. Given a set $U \subseteq (P \cup T)$, ${}^\bullet_N U = \bigcup_{n \in U} {}^\bullet_N n$ and $U^\bullet_N = \bigcup_{n \in U} n^\bullet_N$. The preset of a sequence $\sigma \in T^*$ is the set of all places that occur in a preset of a transition in σ, i.e., ${}^\bullet_N \sigma = \{p \mid \exists 1 \leq i \leq |\sigma| : p \in {}^\bullet_N \sigma(i)\}$. Likewise, the postset of σ is the set of all places that occur in a postset of a transition in σ, i.e., $\sigma^\bullet_N = \{p \mid \exists 1 \leq i \leq |\sigma| : p \in \sigma(i)^\bullet_N\}$. If the context is clear, we omit the N in the subscript.

Let $N = (P, T, F)$ be a Petri net. A *marking* of N is a bag $m \in \mathbb{N}^P$, where $m(p)$ denotes the number of *tokens* in place $p \in P$. If $m(p) > 0$, place $p \in P$ is called *marked* in marking m. A Petri net N with corresponding marking m is written as (N, m) and is called a *marked Petri net*. A *system* $\mathcal{N}$ is a 3-tuple $((P, T, F), m_0, \Omega)$ where $((P, T, F), m_0)$ is a marked Petri net and $\Omega \subseteq \mathbb{N}^P$ is a set of *final markings*.

The semantics of a system $\mathcal{N} = ((P, T, F), m_0, \Omega)$ is defined by an LTS $\mathcal{S}(\mathcal{N}) = (\mathbb{N}^P, T, \to, m_0, \Omega)$ where $(m, t, m') \in \longrightarrow$ iff $F(p, t) \leq m(p)$ and $m'(p) = m(p) - F(p, t) + F(t, p)$ for all $p \in P$, $m, m' \in \mathbb{N}^P$ and $t \in T$. We write $(N : m \xrightarrow{t} m')$ as a shorthand notation for $(\mathcal{S}(\mathcal{N}) : m \xrightarrow{t} m')$ and $\mathcal{R}(\mathcal{N}, m)$ for $\mathcal{R}(\mathcal{S}(\mathcal{N}), m)$.

Let $\mathcal{N} = ((P, T, F), m_0, \Omega)$ be a system. Place p is *k-bounded* in $\mathcal{N}$ for some $k \in \mathbb{N}$, if $m(p) \leq k$ for any marking $m \in \mathcal{R}(\mathcal{N}, m_0)$. If all places are k-bounded, we say that the system is k-bounded. A system is bounded if there exists a $k \in \mathbb{N}$ such that the system is k-bounded. A transition $t \in T$ is *live* in $\mathcal{N}$ if for all markings $m \in \mathcal{R}(\mathcal{N}, m_0)$ a $\sigma \in T^*$ and $m' \in \mathbb{N}^P$ exist such that $(N : m \xrightarrow{\sigma} m')$ and $(N : m' \xrightarrow{t})$. If all transitions of a system are live, the system is called live. A transition $t \in T$ is *quasi-live* in $\mathcal{N}$ if there exists a reachable marking $m \in \mathcal{R}(\mathcal{N}, m_0)$ such that $(N : m \xrightarrow{t})$. If all transitions in the system are quasi-live, the system is called quasi-live.

Weak termination of a system corresponds to weak termination of the corresponding transition system.

A Petri net $N = (P, T, F)$ is a *marked graph* if $|{}^\bullet t| \leq 1$ and $|t^\bullet| \leq 1$ for all transitions $t \in T$. It is a *state machine* if $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for all places $p \in P$.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
2. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
3. van der Aalst, W.M.P., Beisiegel, M., van Hee, K.M., König, D., Stahl, C.: An SOA-Based Architecture Framework. International Journal of Business Process Integration and Management 2(2), 91–101 (2007)
4. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W(E.), Weijters, A.J.M.M.T.: ProM 4.0: Comprehensive Support for *Real* Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
5. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods and Systems. Academic Service, Schoonhoven (1997)
6. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. In: Formal Aspects of Computing, pp. 1–31 (2010)
7. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Verdonk, M.: Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. IEEE Computer 43(3), 102–105 (2010)
8. van der Aalst, W.M.P., van Hee, K.M., Massuthe, P., Sidorova, N., van der Werf, J.M.E.M.: Compositional Service Trees. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 283–302. Springer, Heidelberg (2009)
9. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Kumar, A., Verdonk, M.C.: Conceptual model for online auditing. Decision Support Systems 50(3), 636–647 (2011)
10. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Advanced workflow patterns. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 18–29. Springer, Heidelberg (2000)
11. van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views – Correctness-by-Design for Services. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 139–153. Springer, Heidelberg (2008)
12. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
13. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowlegde Engineering 53(2), 129–162 (2005)
14. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services – Concepts, Architectures and Applications. Springer, Heidelberg (2004)
15. Alves, A., Arkin, A., Askary, S., et al.: Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
16. Araki, T., Kasami, T.: Some decision problems related to the reachability problem for petri nets. Theor. Computer Science 3, 85–104 (1977)
17. Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)

18. Beisiegel, M., Khand, K., Karmarkar, A., Patil, S., Rowley, M.: Service Component Architecture Assembly Model Specification Version 1.1 (2010)
19. Berthelot, G.: Transformations and Decompositions of Nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 360–376. Springer, Heidelberg (1987)
20. Chan, D.Y., Vasarhelyi, M.A.: Innovation and practice of continuous auditing. International Journal of Accounting Information Systems 12(2), 152–160 (2011)
21. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001), http://www.w3.org/TR/wsdl
22. Clarke, E., Emerson, E.: Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
23. Decker, G., Overdick, H., Weske, M.: Oryx – An Open Modeling Platform for the BPM Community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008)
24. Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007)
25. Dehnert, J., Rittgen, P.: Relaxed soundness of business processes. In: Dittrich, K.R., Geppert, A., Norrie, M. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 157–170. Springer, Heidelberg (2001)
26. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press (1995)
27. Desel, J., Reisig, W., Rozenberg, G. (eds.): Lectures on Concurrency and Petri Nets. LNCS, vol. 3098. Springer, Heidelberg (2004)
28. Frutos-Escrig, D., Johnen, C.: Decidability of home space property. Technical Report 503, LRI (1989)
29. van Glabbeek, R.J.: The Linear Time - Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
30. Goud, R., van Hee, K.M., Post, R.D.J., van der Werf, J.M.E.M.: Petriweb: a Repository for Petri Nets. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 411–420. Springer, Heidelberg (2006)
31. Object Management Group. Unified Modeling Language: Superstructure, version 2.0 (August 2005)
32. Hammer, M.: Re-engineering Work: Don't automate, Obliterate. Harvard Business Review, 104–112 (July/August 1990)
33. Hammer, M., Champy, J.: Re-engineering the Corporation. Nicolas Brealy Publishing, London (1993)
34. Heckel, R.: Open Petri Nets as Semantic Model for Workflow Integration. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 281–294. Springer, Heidelberg (2003)
35. van Hee, K.M.: Information System Engineering - A formal approach. Cambridge University Press (1994)
36. van Hee, K., Hidders, J., Houben, G.-J., Paredaens, J., Thiran, P.: On-the-Fly Auditing of Business Processes. In: Jensen, K., Donatelli, S., Koutny, M. (eds.) ToPNoC IV. LNCS, vol. 6550, pp. 144–173. Springer, Heidelberg (2010)
37. van Hee, K.M., Keiren, J., Post, R., Sidorova, N., van der Werf, J.M.E.M.: Designing Case Handling Systems. In: Jensen, K., van der Aalst, W.M.P., Billington, J. (eds.) ToPNaC I. LNCS, vol. 5100, pp. 119–133. Springer, Heidelberg (2008)

38. van Hee, K.M., Mooij, A.J., Sidorova, N., van der Werf, J.M.E.M.: Soundness-Preserving Refinements of Service Compositions. In: Bravetti, M. (ed.) WS-FM 2010. LNCS, vol. 6551, pp. 131–145. Springer, Heidelberg (2011)
39. van Hee, K.M., Post, R.D.J., Somers, L.J.: Yet Another Smart Process Editor. In: European Simulation and Modelling Conference 2005, pp. 527–530 (2005)
40. van Hee, K.M., Serebrenik, A., Sidorova, N., Voorhoeve, M.: Soundness of Resource-Constrained Workflow Nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 250–267. Springer, Heidelberg (2005)
41. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 337–356. Springer, Heidelberg (2003)
42. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Generalised Soundness of Workflow Nets Is Decidable. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 197–215. Springer, Heidelberg (2004)
43. van Hee, K.M., Sidorova, N., Voorhoeve, M., van der Werf, J.M.E.M.: Generation of Database Transactions with Petri nets. Fundamenta Informatica 93(1-3), 171–184 (2009)
44. van Hee, K.M., Sidorova, N., van der Werf, J.M.E.M.: Construction of Asynchronous Communicating Systems: Weak Termination Guaranteed! In: Baudry, B., Wohlstadter, E. (eds.) SC 2010. LNCS, vol. 6144, pp. 106–121. Springer, Heidelberg (2010)
45. van Hee, K.M., Sidorova, N., van der Werf, J.M.E.M.: Refinement of Synchronizable Places with Multi-workflow Nets - Weak termination preserved! In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 149–168. Springer, Heidelberg (2011)
46. van Hee, K.M., Voorhoeve, M.: Soundness of free choice workflow nets. In: Formal Approaches to Business Processes and Web Services - International Workshop (2007)
47. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Treves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. Petri Net Newsletter 76, 9–28 (2009)
48. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
49. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer, Berlin (2010)
50. Holzmann, G.J.: SPIN Model Checker, The: Primer and Reference Manual. Addison-Wesley Professional (2004)
51. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, Berlin (2009)
52. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language Version 1.0 (November 2005), `http://www.w3.org/TR/ws-cdl-10/`
53. Keller, G., Nüttgens, N., Scheer, A.W.: Semantische Process- modellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, University of Saarland, Saarbrücken (1992) (in German)
54. Kindler, E., Petrucci, L.: Towards a Standard for Modular Petri Nets: A Formalisation. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 43–62. Springer, Heidelberg (2009)
55. Kindler, E.: A Compositional Partial Order Semantics for Petri Net Components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)

56. König, D., Lohmann, N., Moser, S., Stahl, C., Wolf, K.: Extending the Compatibility Notion for Abstract WS-BPEL Processes. In: 17th International Conference on World Wide Web (WWW 2008), pp. 785–794. ACM (April 2008)
57. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: An advanced process model repository. Expert Systems with Applications 38(6), 7029–7040 (2011)
58. Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
59. Massuthe, P.: Operating Guidelines for Services. PhD thesis, Technische Universiteit Eindhoven (2009)
60. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
61. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidability of partner existence for open nets. Information Processing Letters 108(6), 374–378 (2008)
62. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Berlin (1980)
63. Miltra, N., Lafon, Y.: Soap version 1.2 part 0: Primer, 2nd edn. (2007), http://www.w3.org/TR/2007/REC-soap12-part0-20070427/
64. Mooij, A.J., Parnjai, J., Stahl, C., Voorhoeve, M.: Constructing Replaceable Services Using Operating Guidelines and Maximal Controllers. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 116–130. Springer, Heidelberg (2011)
65. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)
66. Object Management Group. Business Process Modeling Notation, V1.1 (2008), http://www.omg.org/spec/BPMN/1.1/PDF/
67. Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 (2008), http://www.omg.org/spec/SBVR/1.0/PDF/
68. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson-Prentice Hall (2007)
69. Paton, N.W., Díaz, O.: Active database systems. ACM Comput. Surv. 31, 63–103 (1999)
70. Peterson, J.L.: Petri net theory and the modeling of systems. Prentice-Hall, Englewood Cliffs (1981)
71. Pnueli, A.: The Temporal Logic of Programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
72. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003)
73. Reisig, W.: Petri Nets: An Introduction. Monographs in Theoretical Computer Science: An EATCS Series, vol. 4. Springer, Berlin (1985)
74. Reisig, W.: Petri nets with individual tokens. Theoretical Computer Science 41, 185–213 (1985)
75. Scheer, A.W.: ARIS Business Process Modelling. Springer (1999)

76. Schmidt, K.: Distributed Verification with LoLA. Fundamenta Informatica 54(2-3), 253–262 (2003)
77. Sidorova, N., Stahl, C., Trčka, N.: Workflow Soundness Revisited: Checking Correctness in the Presence of Data While Staying Conceptual. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 530–544. Springer, Heidelberg (2010)
78. Software-Ley. COSA User Manual. Software-Ley GmbH, Pullheim, Germany (1998)
79. Stahl, C.: Service Substitution. PhD thesis, Technische Universiteit Eindhoven (2009)
80. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 172–191. Springer, Heidelberg (2009)
81. Stahl, C., Wolf, K.: Deciding service composition and substitutability using extended operating guidelines. Data & Knowledge Engineering 68(9), 819–833 (2009)
82. Stevens, W.P., Meyers, G.J., Constantine, L.L.: Structured Design. IBM Systems Journal 13(2), 115–139 (1974)
83. Suzuki, I., Kasami, T.: Three measures for synchronic dependence in petri nets. Acta Informatica 19, 325–338 (1983)
84. Szyperski, C.: Component Software – beyond Object-Oriented Programming. Addison-Wesley and ACM Press (1998)
85. van der Toorn, R.A.: Component-Based Software Design with Petri Nets - An Approach Based on Inheritance of Behavior. PhD thesis, Technische Universiteit Eindhoven (2004)
86. Valk, R., Girault, C.: Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications. Springer, Berlin (2003)
87. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnosing Workflow Processes using Woflan. The Computer Journal 44(4), 246–279 (2001)
88. van der Werf, J.M.E.M.: Compositional Design and Verification of Component-Based Information Systems. PhD thesis, Technische Universiteit Eindhoven (2011)
89. Wolf, K.: Generating Petri Net State Spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)
90. Wolf, K.: Does my service have partners? In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 152–171. Springer, Heidelberg (2009)
91. Workflow Management Coalition. Workflow management Coalition Terminology and Glossary. Technical Report Document Number WMFC-TC-1011 – issue 3.0, Workflow Management Coalition (October 2002)
92. Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language, Document Number WMFC-TC-1025 – 1.0 final draft (October 2002)