Controlling Petri Net Process Models

Jörg Desel

Angewandte Informatik
Katholische Universität Eichstätt-Ingolstadt, Germany
joerg.desel@ku-eichstaett.de
http://www.informatik.ku-eichstaett.de

Abstract. We present and compare existing formalisms that consider the control of Petri net process models in the area of business processes and web services. Control has the aim to force a process to behave in a desirable way. Process models that behave properly without any control are often called "sound". For process models that behave properly when being controlled, i.e., for controllable processes, there are various related notions, such as "relaxed soundness" and "weak soundness". We argue that both, the usual notion of sound behavior and the usual notion of control by message passing can be generalized. This way, control synthesis results obtained in the field of automation can be reformulated and reused for business process models and in the area of web services.

1 Introduction

In the last decades, research on Petri net analysis concentrated on the question whether a given Petri net model enjoys a desirable property or not. More recently, people study the question whether a given Petri net model *can* behave properly, if its environment behaves accordingly. This question only makes sense if there are notions of environment and of interface to the considered Petri net model. The environment might be formulated as a Petri net as well. So the problem depends on notions of Petri net modules that can be composed, together with interfaces between Petri nets that express at what parts of the model, and in which way, the interaction between models can take place, and on *proper* versus *not proper* behavior of a Petri net. In particular, proper behavior can be considered for the controlled Petri net in separation or for the Petri net within its environment.

Given a Petri net model N with an interface that allows composition with other net models N' via some operator $\oplus$, one therefore can ask:

1. Does $N \oplus N'$ behave properly for some net N'?
2. Does N behave properly when composed with some net N'?
3. Does $N \oplus N'$ behave properly for any net N'?
4. Does N behave properly when composed with any net N'?
5. Can N' be automatically generated (synthesized) from N such that $N \oplus N'$ behaves properly?
6. Can N' be automatically generated (synthesized) from N such that N behaves properly when composed with N'?

All these questions are tackled in various papers, assuming various composition operators $\oplus$ and various definitions of proper behavior. In this paper, we will frequently come back to these questions.

In this paper we concentrate on Petri net *process* models, where the term "process" refers to business processes. Distinguishing process models from arbitrary system models, characteristic properties of processes and their models include (see [3, 10]):

- Process models have distinguished start and end states. Beginning with a start state, each run should eventually end with an end state. However, it is possible that the behavior of a process has loops, i.e., repetition of states.
- Whereas liveness (every activity can occur from every reachable state) is a desirable property for system models, process models should not be live, because in the end state no activity should be allowed to occur.
- Process models are considered to be embedded in information system models [8]. In contrast to process models, information system models should be live. The situation is comparable to operating systems and single user program executions. Each user program should eventually terminate, but the operating system – which is also an executable program – should not.
- Whereas a deadlock is a state without successor for general systems, end states are not considered deadlocks in process models. For information system and process models, deadlock freedom is desirable.
- Process models are based on single cases where each case corresponds to a single run of a process.
- As in reality, several cases can run concurrently. To reflect this situation, process models might also represent the concurrent run of several process instances, for example to investigate the usage of shared resources.

A process model may interact with its environment and, consequently, may have an interface to some other Petri net. This other Petri net influences the behavior of the process model in such a way that the process behaves properly. In this sense, the process model is *controlled* by the environment. This control happens by means of different kinds of stimulation, depending on the respective approach. Very often, the control also reacts on the behavior of the process. Therefore the control must have the possibility to observe the behavior of the process or at least some aspects of this behavior. Since thus information is flowing in both directions the process model also controls its environment.

Usually, not all elements of a process can be controlled and not all elements can be observed. In other words, it is useful to specify a process model together with its controllable and with its observable elements such that any composition of this net with a net representing the environment restricts to interaction via controllable and observable elements. This constitutes the interface definition of a Petri net process model.

The kind of interaction between a Petri net process model and a model of its environment varies in different approaches. Moreover, the elements of process models that can be controlled and those that can be observed are specified in

various ways. Finally, there are different suggestions for desirable behavior and its specification.

In this paper, we compare some approaches and introduce some relations between them. In the first section, we provide a rough introduction to Petri net models of processes and we repeat the definition of soundness. The second section is devoted to behavioral properties that are related to the soundness property but require a suitable controlling net ensuring sound behavior. In particular, the control makes sure that a process does not run into a deadlock. We sketch a different approach in the third section where control of a process makes sure that places behave in a bounded way, i.e., that the number of tokens on a place does not grow arbitrarily. Finally, the fourth section establishes a relation to known results of controller synthesis in the field of discrete event systems. It is argued that the composition operation used there is more general than usual message passing, whence the results in this area can be transferred to business processes.

2 Petri Net Process Models

Unfortunately, the term "process" was and is used in the Petri net research community in an ambiguous way. Since more than 30 years, a process net is known to be an occurrence net representing a concurrent run of a net representing a system. This naming does not nicely match processes in the sense of business processes [8] and will be avoided in this paper.

One of the first approaches, starting in the late eighties, to model information systems and business processes with Petri nets was the INCOME project by the group of Wolffried Stucky in Karlsruhe, Germany. The INCOME tool developed in this project was successfully used in industrial practice by the spin-off PROMATIS. Relevant publications from this project include [14, 18, 19].

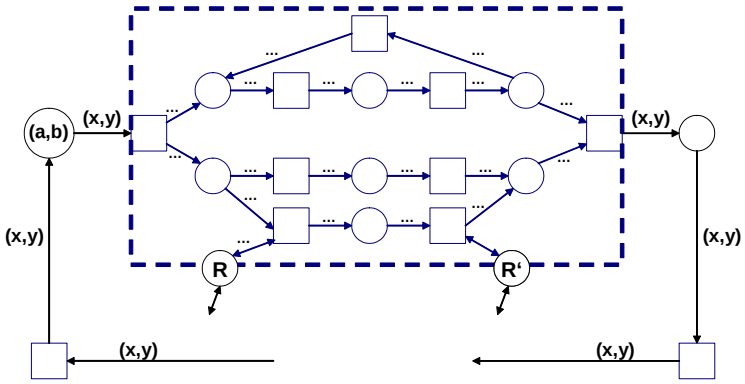

Fig. 1. A high-level Petri net representing a process

The process models used in INCOME are predicate/transition Petri nets (see Figure 1). These nets have distinguished input and output *transitions*, representing the start and the end action of a process. Using high-level tokens, data flow is thus thoroughly represented. In particular, processes of information systems including data bases are represented.

Therefore, (some) places represent data base relations (and tokens corresponding tuples), as the places R and R' in the example indicate. The interface of the process model to its environment is given by its input- and output transitions. In this way a process can be viewed as a refinement of a transition. Additionally, shared places model shared access to data bases.

A model of an information system preferably enjoys nice properties such as liveness and boundedness (an upper limit for the number of tokens). Within an information system model, processes can be identified [8], as in Figure 1. The model of the process, however, is not bounded because the initial transition can occur arbitrarily.

The INCOME approach concentrated on modelling and simulation of processes and information systems in early design phases. Analysis was performed on the level of the entire information system, i.e., the process net together with its environment was studied instead of the process net in separation. Therefore, according to the list of properties in the introduction, Question 3 was considered, because only the behavior of the composed model was of interest.

In the mid nineties, Wil van der Aalst came up with a different concept of a Petri net representation of a process [1–3]. His nets – called *workflow nets* – are place/transition nets, i.e., data aspects and control aspects are separated. A run of a workflow net represents a single case, no matter whether in reality several cases can run concurrently. These runs are assumed to behave without interference so that they all run properly provided a single case runs properly.

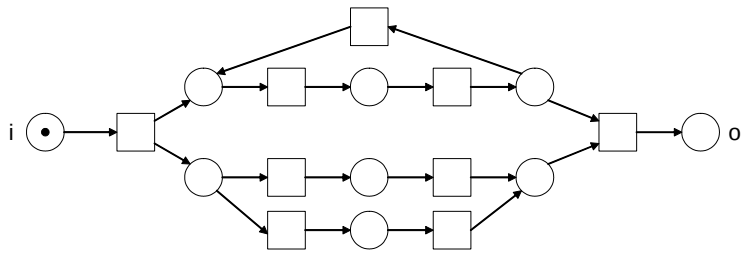

Fig. 2. A sound workflow net

Figure 2 shows a workflow net. As can be seen in the figure, workflow nets are assumed to have a distinguished input place i (representing that the event "case started" has happened) and a distinguished output place o (representing "case completed"). Formally, it is required that every net element of a workflow net is on a directed path from the input place to the output place. There is only

one initial token, marking the input place. This initial marking is called **i**. The intended behavior ends with a marking where only the output place carries one token, called **o**.

Instead of analyzing the entire system model embedding the workflow net one can analyze the workflow net net in separation. So Question 4 from the introduction is considered. Proper behavior is formulated in terms of soundness [3]:

Definition 1. *A workflow net is sound if*
i) from every marking reachable from **i**, *the marking* **o** *is reachable, and*
ii) there are no dead transitions.

It can be shown that, as a consequence, **o** is the only reachable marking assigning a token to the output place (which was part of the original definition). Moreover, each sound workflow net is bounded.

A nice observation is that soundness is strongly related to the well-known notions of liveness and boundedness of general Petri nets [2]. Instead of representing the entire environment of an information system it suffices to add an additional transition moving the token from the output place to the input place, see Figure 3. This transition represents the behavior of the environment in a satisfactory way, as will be explained next.

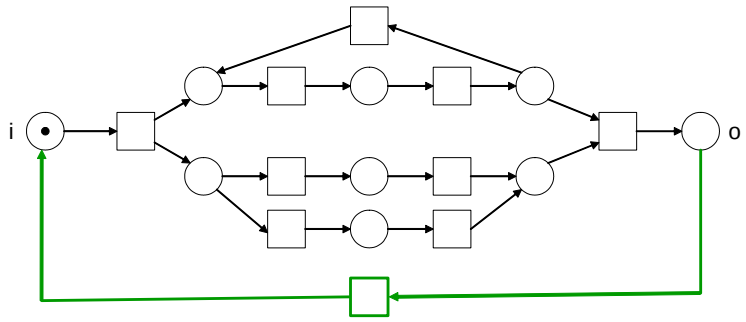

Fig. 3. The workflow net with an additional transition

If the workflow net is sound then the additional transition can always become enabled again (by the marking **o**) because of property i) of the definition of soundness. By property ii), every transition can always become enabled. Hence the extended net is live. It is bounded because its set of reachable markings coincides with the set of reachable markings of the workflow net. Conversely, liveness of the extended net implies that from each reachable marking a marking assigning a token to the output place is reachable. Boundedness implies that this marking must be **o** because otherwise tokens can be added arbitrarily to the net. Liveness also implies that the workflow net has no dead transitions. So soundness of the workflow net coincides with liveness and boundedness of the extended net. Therefore, the large amount of Petri net analysis techniques for

liveness and boundedness can be applied for analyzing soundness of workflow nets. Moreover, if the workflow net happens to be free-choice [7] (which is often the case), the property soundness is decidable in polynomial time.

There are various more suggestions to model processes with Petri nets. For example, in [10] the reader can find examples of process nets with a behavior considering their past. If two transitions of a process strictly occur alternatingly, one in the first case, the other one in the second etc., different initial states are necessary. The initial marking of a process net has to have additional tokens that represent the necessary "memory" of the process.

3 Relaxed and Weak Soundness

In this section, we consider workflow nets that are not sound but behave in a proper (to be defined) way when being connected to a controlling environment. First let us consider an example:

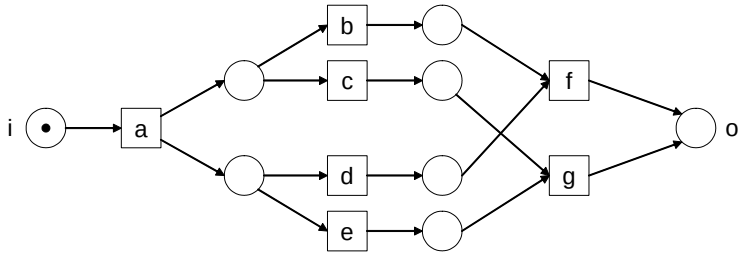

Fig. 4. A workflow net which is not sound

Figure 4 shows a workflow net which is not sound. By firing transitions a, b and e a marking is reached that enables no transition, i.e. a deadlock. In particular, the marking **o** cannot be reached from this marking. However, the net still has a positive property, formulated in the following definition [5]:

Definition 2. *A workflow net is relaxed sound if every transition occurs in some occurrence sequence leading from* **i** *to* **o**.

It is easy to verify that the occurrence sequences $a\ b\ d\ f$ and $a\ c\ e\ g$ of the example net both lead to the marking **o** and that every transition occurs in one of these sequences. Hence this net is relaxed sound.

The example net has two forward branching places representing a choice between b and c and a choice between d and e. Whenever one of the places is marked, both output transitions are enabled but firing one of the transitions disables the other one.

Generally, Petri net choices can represent quite different concepts:

- The choice is done within this process but the respective part of the process is not modelled. For example, two transition could represent two users that both could take care of a work item. Any solution is as good as the other one. This view relates to Question 4 of the introduction: Does N behave properly for any partner net N'? In a sound workflow net, we expect that any other component N' which decides which of the conflicting transition fires would not destroy the desired property.
- The choice depends on data of the case, which is not modelled. There are different suggestions how to handle data dependent choices. In the INCOME approach all relevant data is captured in the high-level tokens. This information can be reduced to *routing information* if the only purpose of this data is to decide choices. If one choice depends on data, another choice can depend on the same data as well, and this way deadlocks could be avoided. In our example, it might be the case that either transitions b and d or transitions c and e are chosen, whence the net can behave in a sound way. This view was originally taken in [5].
- Similarly, choices can depend on additional pre-conditions of the conflicting transitions which are not modelled first (see Figure 5). In other words, an embedding of the process net in a larger net is considered. With this view, we can ask whether there is an appropriate environment controlling the process net such that this net behaves soundly (Question 2 of the introduction). It is easy to see that putting tokens to the other new places instead yields the other sound run.

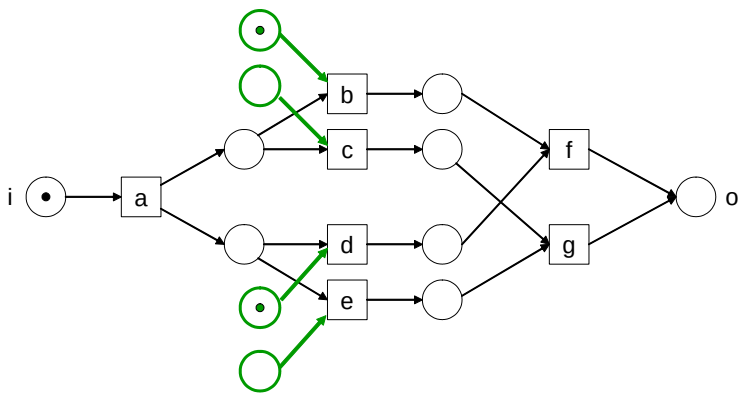

Fig. 5. Enforcing sound behavior by additional places

Now let us consider the next example, shown in Figure 6. This net is not even relaxed sound because there is no run leading from **i** to **o** which includes an occurrence of transition h. In this example net we have three conflicts. The

addition of respective pre-conditions (Figure 7) shows that it is still possible to reach **o** from **i**.

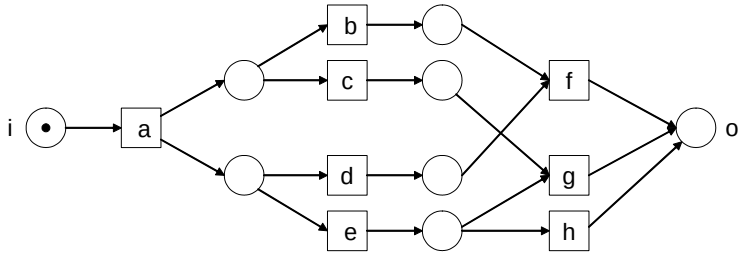

Fig. 6. A process net which is not relaxed sound...

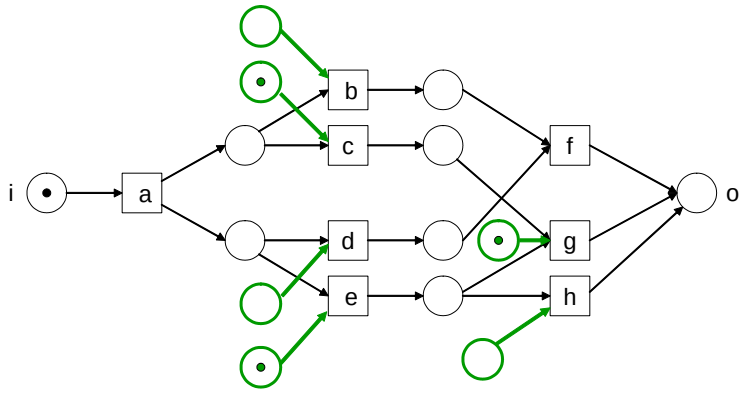

Fig. 7. ... together with pre-conditions

Definition 3. *A workflow net, extended by input places to some of the conflicting transitions, is weakly sound if from every marking reachable from* **i**, *the marking* **o** *is reachable.*

Notice that this definition is very similar to i) in the definition of soundness. Actually, the original definition of weak soundness employs so-called open workflow nets which contain the workflow net together with additional input places of some of the conflicting transitions [15, 17]. Therefore the definitions of relaxed soundness and of weak soundness cannot be compared immediately. However, at least for nets in which no transition can occur more than once, every relaxed workflow net can be extended by accordingly marked places such that the resulting open workflow net is weakly sound.

Whereas relaxed soundness clearly corresponds to Question 2 of the introduction, one might argue that weak soundness refers to Question 1. There are other approaches, e.g. [13], where local and global soundness is explicitly distinguished. Therefore the work described in [13] definitely answers Question 1.

Whereas weak soundness does not explicitly refer to a controller, the closely related property controllability as used in [21], does. In the application context of web services, the property is called usability [16]. The term controllability is used in [6] to characterize relaxed soundness.

4 Weak Boundedness

Although soundness refers to liveness and boundedness, its derivates relax the liveness condition by assuming that the net remains live if, in case of conflicts, only the right transitions are chosen. In this section we introduce a related, but different approach, where liveness is guaranteed but boundedness needs additional control. This approach stems from the area of schedulability of concurrent programs on a chip [4], but can similarly be formulated for processes in our sense.

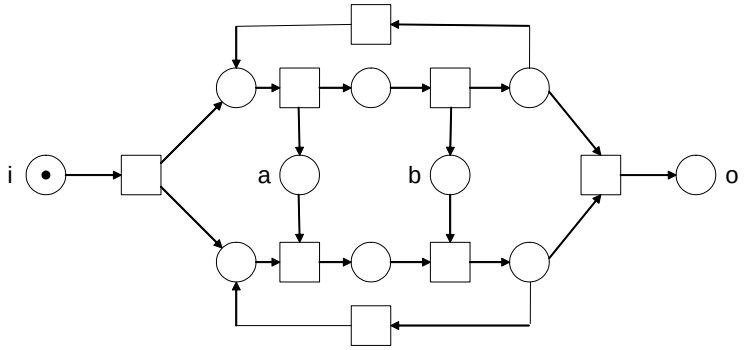

Fig. 8. A weakly bounded process net

The process net shown in Figure 8 can always reach the marking **o**. But, if the upper cycle occurs more often than the lower cycle, then there will be an arbitrary number of tokens in the places a and b, whence the net is not bounded. However, this effect can be avoided by firing transitions in the lower cycle at least as fast as those in the upper cycle.

Definition 4. *A workflow net is weakly bounded if there is a bound b such that, for each occurrence sequence from* **i** *to* **o**, *there is a permutation of this sequence (leading from* **i** *to* **o** *as well) such that the token count on any place does not exceed b at any intermediate marking.*

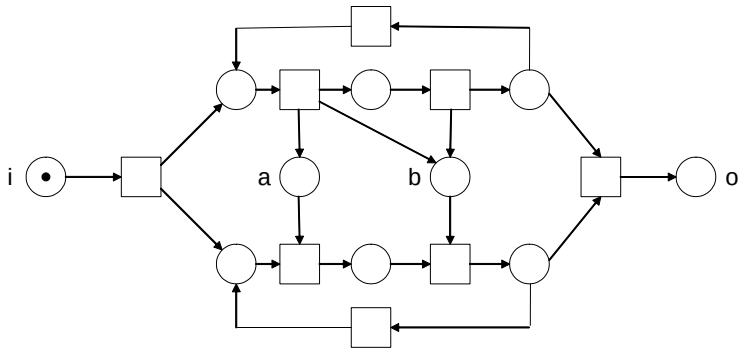

Fig. 9. A process which is not weakly bounded

Figure 9 shows a workflow net which is not weakly bounded. Due to place a the lower cycle cannot run faster than the upper cycle but to keep place b bounded it would have to run twice as fast. [4] contains sufficient and necessary conditions for weak boundedness of a net defined in the other application domain. Since this definition and the definition of workflow nets is not too different, these results should be transferable to the domain of business processes and web services.

The upper and the lower cycle of our example process could be viewed as separate processes which are both started by the occurrence of the only enabled transition in the figure. These processes communicate via message passing. In this sense, tokens on the places a and b can be viewed as requests. In this setting, it is an important question whether the lower process is able to serve all requests. For the weakly bounded example, the answer is positive. It is negative for the other example. Notice that this is not a negative property of any of the two subprocesses; both are fine in separation. Only their combination is ill. In the weakly bounded case, a scheduler – which is nothing else but an additional net module controlling processes – can only be applied to the combined process. Two independent schedulers of the two single processes would not work. In this sense, weakly bounded process nets are controllable and process nets which are not weakly bounded are not controllable.

5 Controller Synthesis

Based on previous work in discrete event systems [20, 22, 23, 11], we give in [9] an overview on our work on controller synthesis. Processes (which are cyclic in our setting in [9]) are given in terms of Petri nets and communication is based on events, formalized by means of *event arcs*. The aim of this section is to show that the results can be translated to the area of business processes.

Figure 10 (taken from a presentation of Gabriel Juhás) shows how Petri nets representing web services communicate via event signals, formalized by event arcs. The meaning of an event arc is as follows: The occurrence rule for the source transition is the usual one. Assume it is enabled at a marking. If the target

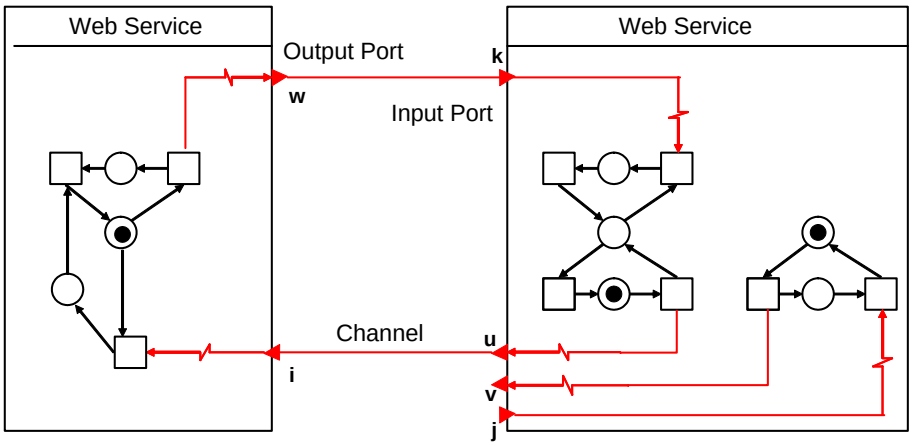

Fig. 10. Web service composition with event arcs

transition is enabled as well, both transitions occur coincidentally (in a step). Otherwise the source transition occurs alone, as usual. The target transition can only occur coincidentally with the source transition. See [12] for a translation of event arcs to nets using inhibitor arcs. This paper also provides a framework for controller synthesis on the basis of so-called open Petri nets.

The main result of [9] is an algorithm that provides for a given Petri net and a given specification another Petri net such that the behavior of the composed net matches the specification. So this approach provides a solution to Question 5 of the introduction. The specification is given in terms of a regular language (a regular expression, syntactically). The composition operator only uses event arcs. An event arc can only lead to a controllable event, and it can only start at an observable event. The composition of modules can be viewed as another module in the obvious way. The interface, i.e. the controllable and observable events, of the composed net is the set of transitions which are not controlled (observed, respectively) by one of the composed modules.

Instead of summarizing the result of [9] in more detail, we roughly explain why this result can be viewed as a generalization of the synthesis problem of workflow nets.

First, for workflow nets the desired property is that from each reachable marking the marking $\mathbf{o}$ can be reached. In other words, each run should end with one of the final transitions which put a token to the place o. If we abstract from tokens that do not enable any transition, we moreover require that no other transition is enabled after the occurrence of a final transition. Clearly, this can be expressed by means of a regular expression.

Second, the usual communication primitives used for process models and web services is message passing, formalized by a place in the post-set of a sending transition and in the pre-set of a receiving transition. This communication is purely asynchronous. In contrast, event arcs provide means to formalize syn-

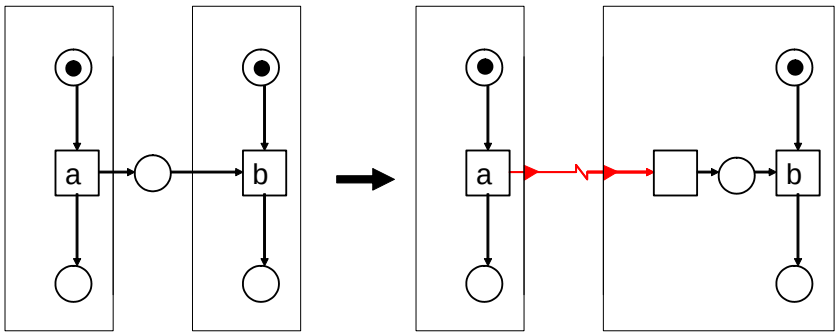

Fig. 11. Translating message passing in event arcs

chronous aspects as well, but in an asymmetric way. However, Figure 11 shows how message passing can be modelled by means of event arcs: Instead of sending a message, a signal is sent which forces the receiver to create the message itself. Messages carrying data, modelled by high-level Petri nets, can be translated in a similar way because the event arcs can have high-level annotations, just as regular arcs. Similarly, overwriting of messages etc. can also be modelled by event arcs, in a similar way as they can be modelled by high-level Petri nets, see Figure 12.

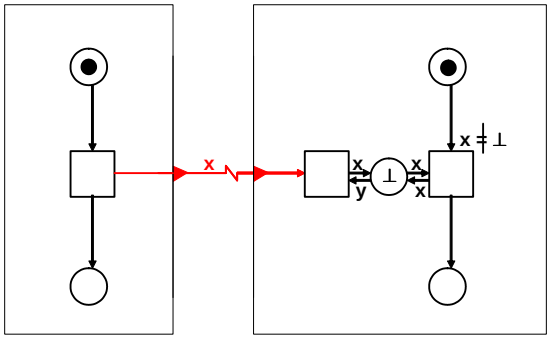

Fig. 12. Modelling overwriting of messages

6 Conclusion

Starting with a number of problems related to controllability of Petri nets in the introduction, we showed that, and how, different approaches to controlling Petri net process models are related but answer slightly different questions. The list of mentioned approaches is by far not complete. A first selection criterion

was the popularity of the concepts in process modelling (soundness and relaxed soundness) or in web services (weak soundness and controllability/usability of open workflow nets). I added approaches which were (co-)developed in my research group and which might be usable to solve additional problems raised for process models and web service models.

Since one of the main differences between the mentioned approaches is the way modules interact which each other, one might ask whether asynchronous communication is more natural than synchronous communication or whether asymmetric synchronous communication, as provided by event arcs, is more natural than real synchronicity, etc. As shown before, message passing can be translated to asymmetric synchronous communication. Event arcs can also model mutual dependency between two processes, ensuring that each process can only proceed after the other one performed a corresponding activity (by sending an acknowledge via an event arc), see Figure 13.

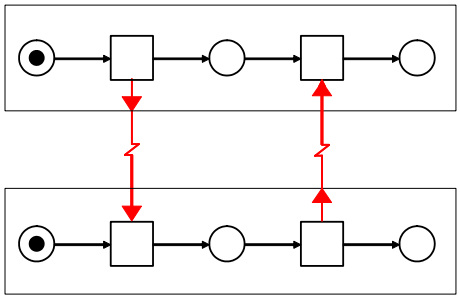

Fig. 13. Send and acknowledge of a message

So event arcs are quite general, and the quest for the right way of communication is not a matter of expressivity.

Considering a natural way to model communication, the level of abstraction plays a significant role. For example, web services are usually assumed to communicate strictly asynchronously. On a more technical level (i.e., on a lower layer) asymmetric communication turns out to be realized by means of synchronous communication primitives. I claim that the signal arc approach provides one of the most natural views of communication. As mentioned before, it can be restricted to mimic asynchronicity and it can be restricted to mimic synchronicity.

References

1. van der Aalst, W.M.P.: A class of Petri nets for modeling and analyzing business processes. Computing Science Report 95/26, Eindhoven Univ. of Technology, 1995
2. van der Aalst, W.M.P.: Verification of workflow nets. Application and Theory of Petri Nets 1997, LNCS 1248, Springer (1997) 407–426

3. van der Aalst, W.M.P., van Hee, K.: Workflow Management – Models, Methods and Systems. MIT Press (2002)
4. Cong Liu, Kondratyev, A., Watanabe, Y., Desel, J., Sangiovanni-Vincentelli, A.: Schedulability analysis of Petri nets based on structural properties. Applications of Concurrency to System Design (ACSD), IEEE (2006) 69–78
5. Dehnert, J., Rittgen, P.: Relaxed soundness of business processes. Conference on Advanced Information Systems (CAiSE), LNCS 2068, Springer (2001) 157–170
6. Dehnert, J.: Expressing the controllability of business processes. Petri Net Newsletter 61 (2001) 9–17
7. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press (1995)
8. Desel, J., Oberweis, A.: Petri-Netze in der Angewandten Informatik. Wirtschaftsinformatik 38 (4) (1996) 359–367
9. Desel, J., Hanisch, H.-M., Juhás, G., Lorenz, R., Neumair, C.: A guide to modelling and control with modules of signal nets. Integration of Software Specification Techniques for Applications in Engineering, LNCS 3147, Springer (2004) 270–300
10. Desel, J.: Process modelling using Petri nets. Process-Aware Information Systems - Bridging People and Software through Process Technology, Wiley (2005) 147–177
11. Hanisch, H.M., Rausch M.: Synthesis of supervisory controllers based on a novel representation of condition/event Systems. IEEE International Conference on Systems, Man and Cybernetics 4, 1995, 3069–3074
12. Heckel, R., Chouikha, M.: Control synthesis for discrete event systems – A semantic framework based on open Petri nets. Transactions of the SDPS 6 (4) (2003) 63–104
13. Kindler, E., Martens, A., Reisig, W.: Inter-operability of workflow applications: local criteria for global soundness. Business Process Management: Models, Techniques, and Empirical Studies, LNCS 1806, Springer (2000) 235–253
14. Lausen, G., Müller, H., Németh, T., Oberweis, A., Schönthaler, F., Stucky, W.: Integritätssicherung für die datenbankgestützte Software-Produktionsumgebung INCOME. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW) Informatik-Fachberichte 136, Springer (1987) 152–156
15. Martens, A.: On compatibility of web services. Petri Net Newsletter 65, Gesellschaft für Informatik (2003) 12–20
16. Martens, A.: On usability of web services. 1st Web Services Quality Workshop (WQW 2003), Rome, Italy, 2003.
17. Martens, A.: Analyzing web service based business processes. Fundamental Approaches to Software Engineering (FASE'05), LNCS 3442, Springer (2005) 19–33
18. Oberweis, A., Scherrer, G., Stucky, W.: INCOME/STAR: methodology and tools for the development of distributed information systems. Information Systems 19 (8) (1994) 643–660
19. Oberweis, A., Sander, P.: Information system behavior specification by high-level Petri nets. ACM Transactions on Information Systems 14 (4)(1996) 380–420
20. Ramadge, P.J., Wonham, W.M.: The Control of Discrete Event Systems. Proceedings of the IEEE 77 (1989) 1, 81–98
21. Schmidt, K.: Controllability of open workflow nets. Enterprise Modelling and Information Systems Architectures (EMISA), LNI 75, Gesellschaft für Informatik (2005) 236–249
22. Sreenivas, R.S., Krogh, B.H.: On condition/event systems with disrete state realizations. Discrete Event Dynamic Systems – Theory and Applications 2 (1991) 1, 209–236
23. Sreenivas, R.S., Krogh, B.H.: Petri net based models for condition/event systems. 1991 American Control Conference 3 (1991) 2899–2904