Proc. EMISA 2004 - Informationssysteme im E-Business und E-Government, Luxemburg, Oktober 2004, Lecture Notes in Informatics (LNI), P-56, 2004, pp. 117-128

On the Modeling of Correct Service Flows with BPEL4WS

Manfred Reichert, Stefanie Rinderle, Peter Dadam

Department Databases and Information Systems
University of Ulm, D-89069 Ulm, GERMANY
{reichert, rinderle, dadam}@informatik.uni-ulm.de

Abstract: Frameworks for composing Web Services offer a promising approach for realizing enterprise-wide and cross-organizational business applications. With BPEL4WS a powerful composition language exists. BPEL implementations allow orchestrating complex, stateful interactions among Web Services in a process-oriented way. One important task in this context is to ensure that respective flow specifications can be correctly processed, i.e., there will be no bad surprises (e.g., deadlocks, invocation of service operations with missing input data) at runtime. In this paper we subdivide BPEL schemes into different classes and discuss to which extent instances of these classes can be analyzed for the absence of control flow errors and inconsistencies. Altogether our work shall contribute to a more systematic evolution of the BPEL standard instead of overloading it with too many features.

1. Introduction

Today there is a high need for active coordination of the various, distributed tasks necessary to perform enterprise-wide or even cross-organizational business processes. Service-oriented architectures offer a promising approach in this context. Usually, they provide a framework for specifying, implementing, and registering services as well as for composing them in a reliable and process-oriented manner [Al04]. The latter enables stateful interactions among services and provides the basis for process orchestration.

So far different languages for Web Service composition and Web Service orchestration have been proposed [An03, Ch03, KMW03, HB03, NM02]. Among them are WSFL (Web Service Flow Language) and XLANG [Ch03]. While WSFL has followed a graph-based approach for flow modeling, XLANG has been based on an algebraic language with block-based description concepts. BPEL4WS (Business Process Execution Language for Web Services – BPEL for short) represents a convergence of the ideas followed by these two languages. It combines the graph oriented flow representation of WSFL with the block-based flow description of XLANG [An03].

Main emphasis of this paper is put on selected modeling and verification issues related to BPEL control flow specifications. So far only few approaches exist which systematically deal with respective problems [BK03, Ma04]. As known from software development, errors which are detected in late phases of the development cycle are most expensive. For process-aware applications this means that design and implementation errors must

be detected as early as possible. For example, service flows must not run into deadlocks at runtime or must not invoke service operations with missing input data. In principle, this can be achieved by using BPEL as flow description language since it allows to model the flow logic of business processes separately and independently from the implementation of the used Web Services [KMW03, Wo02]. Thus flow changes can be accomplished at a high level and without affecting service implementations. In this paper we subdivide BPEL control flow schemes into different classes and discuss to which extent instances of these classes can be analyzed for the absence of the mentioned errors. This discussion, in turn, shall contribute to the further evolution of BPEL.

Section 2 summarizes basic concepts of the BPEL specification. In Section 3 we provide a classification of BPEL schemes, which is helpful for systematically dealing with issues related to control flow definition and verification. For each of the identified classes, Section 4 discusses to which degree correctness properties of corresponding BPEL schemes can be guaranteed. The paper concludes with a short summary in Section 5.

2. Basic Concepts of BPEL4WS

In this section we summarize selected features of BPEL4WS – background information which is needed for the further understanding of this paper. BPEL comprises a powerful process meta model for describing business processes based on the interactions between the process and its partners [An03]. The BPEL specification consists of an XML grammar and makes use of several well-known XML specifications (including WSDL 1.1). For the sake of readability, however, in the following we mainly abstain from XML-based flow representations and use graphical illustrations instead.

In a BPEL flow the interaction with each partner occurs through Web Service interfaces. The structure of respective relationships is encapsulated in *partner links*. Process activities may invoke service operations of partners synchronously or they may receive messages from service invocations of partners and reply to them asynchronously at a later point in time. A BPEL flow specifies how service interactions are coordinated. More precisely, BPEL allows modeling the service flow explicitly and independently from the implementation of the Web Services activity execution is based on. Finally, BPEL comprises modeling elements for dealing with activity failures. For example, designers may specify how activities are to be compensated at the occurrence of semantic failures.

For illustration purposes we consider the process for handling a purchase order (as described in [An03]). The aim is to introduce the graphical notation on which we base our illustrations. Fig. 1 represents the basic logic of this process: Solid arrows represent sequencing. Free grouping of sequences represents concurrent activities and dotted arrows represent links. The latter can be used, for example, to synchronize activities arranged in parallel so far. On receiving the purchase order from a customer the flow depicted in Fig. 1 initiates three tasks in parallel: calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order. While some activities related to these tasks can be processed concurrently, there are others with dependencies between them. Particularly, shipping information is needed to send the shipping price, and the shipping schedule is required for completing the production sche-

duling. When completing the three tasks, the invoice is processed and sent to the customer. As can be seen from Fig. 1, different kinds of activities can be used to describe the interaction and communication patterns between process and partners. Activity *Receive Purchase Order*, for example, corresponds to the service invocation of a customer. From the process viewpoint this activity represents a *receive activity* leading to the creation of a new flow instance. The flow itself invokes several service operations synchronously at partner sites (*invoke activities*). Finally, the process replies to the received purchase order by sending a corresponding message to the customer (*reply activity*).

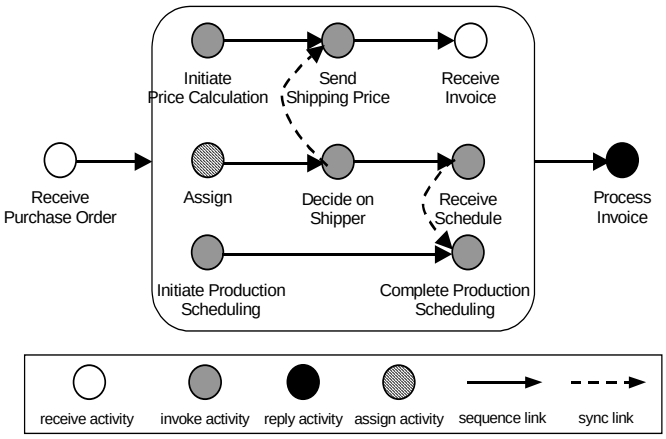


Fig. 1: Purchase order example (control flow perspective)

Data flow between activities (not depicted in Fig. 1) can be specified by mapping input/ output messages of activities to process variables. Depending on the kind of activity, read or write access to these process variables can be allowed. A receive activity may only write process variables whereas a reply activity may only read them. Invoke activities may have both read and write access to variables (for details see [An03]).

BPEL provides a multitude of possibilities to describe the desired flow logic. For example, Fig. 2 shows a WSFL-like modeling of the purchase order example, which is based on a graph of activity nodes and links. Generally, a wide variety of modeling facilities exist, including the assignment of transition conditions (i.e., predicates on flow variables) to links and the definition of activity join conditions. As opposed to this, Fig. 3 mainly reflects the XLANG-like modeling style which uses structured activities (`sequence`, `flow`) and two additional links to synchronize concurrent steps.

3. Classification of BPEL Schemes

As illustrated in Section 2, BPEL allows specifying the same control flow pattern in different ways [Wo02]. Basically, two modeling styles are supported: One of them follows a block-structured approach whereby control flow is modeled through the nested use of structured activities (`sequence`, `flow`, `while`, `switch`, `pick`). The other uses a graph-oriented approach based on nodes, links and related join / transition conditions.

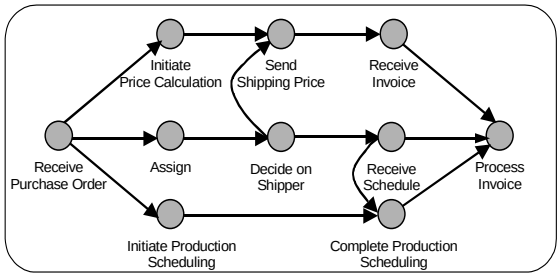


Fig. 2: WSFL-like modeling based on a network of links

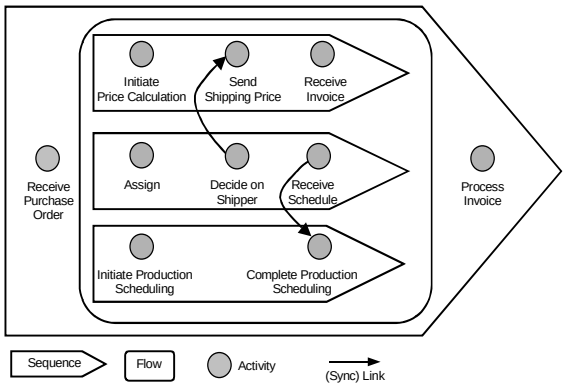


Fig. 3: XLANG-like modeling with minimal use of links

Generally, it is possible to mix both modeling styles by having links crossing the boundaries of structured activities (cf. Fig. 3). On the one hand BPEL enables high expressiveness as well as upward compatibility of WSFL and XLANG specifications; on the other hand this aggravates flow modeling, analysis and verification significantly.

In order to better structure our discussions we classify BPEL schemes according to the control flow elements used for their definition. Doing so we distinguish between flow schemes with strict block structuring (Class-0), flow schemes with block structuring and controlled use of links (Class-1), and flow schemes with block structuring and arbitrary use of links (Class-2). Due to lack of space we omit a formal definition of the semantics of the BPEL language and the properties of the different schema classes. Instead we illustrate main issues by means of expressive examples. Class-0 is the most specific class which is subsumed by the two other ones. Its instances are defined by mixing basic and structured activities, but do not contain links. Class-1 relaxes this strict block structuring. However instances of this class must follow certain guidelines (i.e., restrictions) regarding the use of links (when compared to the current BPEL specification). Finally, Class-2 comprises all control flow schemes that can be currently described with BPEL. In particular, it contains schemes which can be solely modeled as network of nodes and links (and a surrounding `flow` activity). Due to lack of space, in this paper we exclude issues related to exception handling and compensation, and therefore do not further consider the BPEL constructs *throw*, *scope*, and *compensation* [An03].

3.1 Flows With Strict Block Structuring and Without Use of Links (Class-0)

Class-0 contains all schemes that can be described by means of both basic and structured activities. Doing so, sequences, branchings, and loops can be modeled in a block-oriented fashion with well-defined start/end nodes. Such control blocks may be nested but are not allowed to overlap. Note that Class-0 already supports `flow` activities for enabling concurrent activity executions. However, links are excluded for instances of this class. An abstract example is depicted in Fig. 4. As shown in [KHB00], pure block structuring does not provide same expressiveness as it can be obtained by the use of graph-based languages with arbitary use of links. However, the block concept is sufficient for many practical cases. In particular, it is advantageous with respect to structuring of flows and verification of static as well as dynamic flow properties.

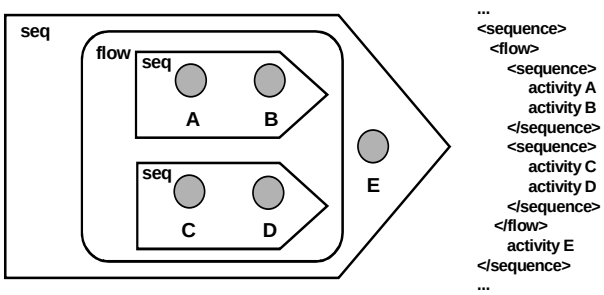


Fig. 4: Example of a BPEL schema corresponding to Class-0

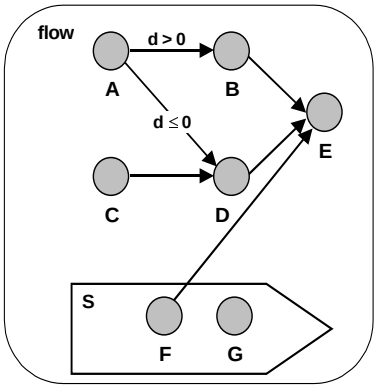
3.2 Flows with Block Structuring and Arbitrary Use of Links (Class-2)

Generally, BPEL `flow` activities enable the expression of complex synchronization dependencies between (concurrent) activities nested directly or indirectly within them. For this the BPEL *link* construct can be used. First, we summarize basic properties of links as defined in the BPEL specification [An03]. Then we discuss issues related to BPEL schemes based on both structured activities and links with (arbitrary) transition conditions. Respective schemes constitute the most generic BPEL schema class (Class-2), but may aggravate flow analyses and verification by orders of magnitudes.

BPEL Link Concept: Each link has one activity nested directly or indirectly within a `flow` activity as its source and one such activity as its target. Additionally, links (or their source activity respectively) may be associated with a *transition condition*, which represents a Boolean expression on process variables. When completing an activity the transition conditions of its outgoing links are evaluated. Depending on the results, links are either signaled as *True* or *False*.[1] In turn, an activity `X` with incoming links has a *join condition*; i.e., a Boolean expression on the status of these links. As necessary prerequisite for executing `X` all incoming links must have been signaled and its join condition been evaluated to *True* (default). If an explicit join condition is missing, it is implicitly required that at least one incoming link is signaled as *True*. When an activity's join condi-

[1] If a link has no related transition condition attribute, it is deemed to be present with value *True*.

tion evaluates to *True*, its execution can be started on condition that all other preconditions are met. However, if the join condition evaluates to *False*, activity execution will be skipped and outgoing links be signaled as *False*. This, in turn, may lead to a *dead path elimination.*[2]




```
...
<flow>
  <links>
    <linkname = "L_AB"/>
    <linkname = "L_AD"/>
    <linkname = "L_FE"/>
    …
  </links>
  activity A
    <source linkName = "L_AB" transitionCondition="d>0"/>
    <source linkName = "L_AD" transitionCondition="d≤0"/>
  activity B
    <target linkName = "L_AB"/>
    <source linkName = "L_BE"/>
  …
  activity E
    joinCondition = "(L_BE OR L_DE) AND L_FE"
    <target linkName = "L_BE"/>
    <target linkName = "L_DE"/>
    <target linkName = "L_FE"/>
  <sequence name = "S">
     activity F
       <source linkName = "L_FE"/>
     activity G
  </sequence>
</flow>
...
```

Fig. 5: Example of a BPEL schema corresponding to Class-2

Mixed Use of Structured Activities and Links: Class-2 represents the most generic schema class and contains all control flow patterns that can be described by means of BPEL. A simple example is depicted in Fig. 5. In particular, flow schemes from Class-2 can be defined by the mixed use of structured activities and links. Doing so, activity join conditions as well as link transition conditions can be based on arbitrary Boolean expressions (on link states and process variables respectively). Generally, a link may cross the boundary of structured activities; i.e., its source activity may be nested within a structured activity, but its target activity may be not, and vice versa. An example for this is shown in Fig. 5: The depicted flow contains a boundary-crossing link named `L_FE` that starts at activity `F` (directly nested within a sequence) and ends at activity `E`.

On the one hand modeling flows as network consisting of basic / structured activities and links (with arbitrary join / transition conditions respectively) provides higher expressiveness when compared to pure block structuring. On the other hand, it aggravates flow analyses which is disadvantageous with respect to schema maintenance and evolution. Another severe problem of BPEL arising in connection with the use of structured activities and links concerns link semantics. As an example take activities `A`, `B`, and `C`, and assume that they are to be executed sequentially. To express this either one may use a sequence activity and embed `A`, `B`, and `C` within it in desired order (cf. Fig. 6 a) or one may nest `A`, `B`, and `C` within a flow activity and enforce the desired order by inserting links $\texttt{A} \rightarrow \texttt{B}$ and $\texttt{B} \rightarrow \texttt{C}$ (cf. Fig. 6 b). Obviously, in the given case the resulting flow schemes S_{seq} and S_{link} are trace equivalent; i.e., each execution log producible on S_{seq} can be created on S_{link} as well, and vice versa. However, things become more difficult if activities nested within a structured activity `S` have to be synchronized with activities outside `S`.

---

[2] The BPEL attribute *suppressJoinFailure* must be set to "yes" in order to initiate a deadpath elimination in the given context. Otherwise, fault *joinFailure* will be thrown which, in turn, leads to interruption of the flow.

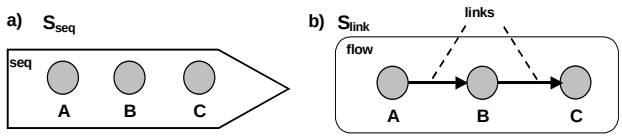


Fig. 6: Modeling a sequence of activities A, B, and C

Take the above example. Assume that B has a conditional dependency on activity X outside the sequence; i.e., B is target of a link with X as its source and P(X, B) as related transition condition. Considering this we obtain schemes S_{seq}^{*} and S_{link}^{*} (cf. Fig. 7). At first glance both seem to be trace equivalent again. However, this is not the case if the default link semantics of BPEL is used. Then the join condition of B corresponds to the disjunction of the status of all incoming links targeted to this node. Suppose that both A and X are completed and P(X,B) evaluates to *False*. Taking schema S_{seq}^{*} the (default) join condition of B evaluates to *False*. In this case, either the standard fault *joinFailure* will be thrown or a deadpath elimination be initiated. In any case activity B will not be performed. By contrast, when regarding S_{link}^{*} the join condition of B will always evaluate to *True* even if link X → B is signaled as *False*. The reason for this different execution behavior is that the desired execution order between A and B is described by means of a link with B as its target. Since this link has no transition condition it is always signaled as *True* when completing activity A. Thus the (default) join condition of activity B will always evaluate to *True* independently from whether X → B is signaled as *True* or *False*.

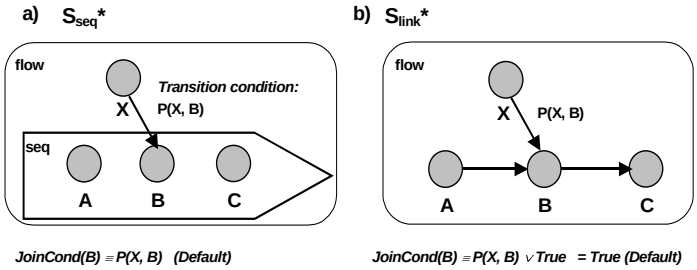


Fig. 7: Link semantics and related problems

This simple example demonstrates that the mixed use of structured activities and links with arbitrary setting of activity join and link transition conditions may cause undesired side-effects. In particular, a BPEL schema that meets the execution behavior as intended by the designer may not do this anymore when changing this schema. Currently, there is no systematic and sufficient support for adequately dealing with such side-effects.

As indicated the block concept is advantageous with respect to the structuring of flows and the verification of static as well as dynamic flow properties. As opposed to this, the modeling of flows by a network of activities and links with (arbitrary) transition conditions is more expressive in several respects, but aggravates flow verification and flow changes significantly. For example, data flow analyses and validation may suffer from the "uncontrolled" use of both transition and join conditions.

3.3 Flows with Restricted Use of Links (Class-1)

We now sketch useful restrictions and suggestions regarding the application of links. Basic to this is the observation that in most cases we do not require full expressiveness of BPEL in order to adequately capture the business processes deployed in today's organizations. Very often control flow can be already expressed by the nesting of structured and basic activities. If this is not sufficient, in addition, one can use links to synchronize activities from parallel branches. However, this should be done in a controlled way in order to avoid the described problems. In particular

1. links should not be used to express an activity sequence if this sequence can be defined by the use of a structured activity (`sequence`) as well. This design principle, for example, is not met by the flow schemes from Fig. 5 and Fig. 6 b).
2. one should avoid the use of attributed links (i.e., links with *transition conditions*) at all. This would contribute to avoid undesired side-effects when deleting or inserting links (cf. Fig. 7). Note that BPEL already allows to specifiy conditional branchings of different semantics by means of structured activities (`switch`, `pick`).

Adherence of these restrictions offers advantages with respect to flow modeling, analysis, and verification. In order to follow the second guideline, for instance, the use of links with *fixed join semantics* of their target nodes offers a promising perspective. We consider two "link patterns" in this context. Both have a pre-fixed semantics and can be used for synchronizing activities nested directly or indirectly within a `flow` activity:

- A **weak link** X → Y defines a *delay dependency* between X and Y (arranged in parallel so far). As necessary prerequisite for executing Y either activity X must have been completed or X cannot be performed anymore (except due to a loop back).
- A **strict link** X → Y defines a *causal dependency* between X and Y. As prerequisite for executing Y we require that X must have been successfully completed.

These link types allow designers to capture an intermediate synchronization point between parallel paths. Generally, such an inter-branch synchronization cannot be expressed using basic and structured activities only. Both weak and strict links can be simply realized. If we solely use these link patterns the join condition of an activity corresponds to the conjunction of the status of all strict links targeted to this activity. In particular, the use of weak links does not have any effect on the join condition of its target node but only delays the start of the respective activity. In any case, all BPEL restrictions concerning the use of links must be met for weak / strict links as well, e.g., respective links must not cause deadlocks or cross boundaries of a loop block. Weak links are useful for practical cases since they represent the most common application of the BPEL link concept. As an example, take the purchase order process and its representation depicted in Fig. 3 – both links may possess semantics "weak". By contrast, strict links are required in rather seldom cases. They are useful to synchronize an activity nested within a `switch` activity with an activity nested within a `pick` activity.

In our experience, most control flow patterns can be described by the combined use of (structured) activities and the above link types. In particular, when applying link-related

schema changes undesired side-effects can be avoided. We assign flow schemes based on such a mixed use of structured activities and weak/strict links to Class-1. An example of a flow schema from Class-1 is depicted in Fig. 8. Since activity F and C respectively are performed in any case, flow behavior will be the same independently from whether a weak or a strict link is used. Interestingly, the depicted schema shows same execution behavior as the one from Fig. 5. However, the use of links has been reduced to a minimum (according to the described guidelines). Obviously, when substituting $C \rightarrow D$ by inverse link $D \rightarrow C$ in Fig. 8, link semantics – weak or strict – has influence on the flow behavior. More precisely, when using weak link $D \rightarrow C$ activity C will be performed either when D is completed or disabled. The latter case occurs if the upper branch of the switch block ($d > 0$) is chosen. As opposed to this, using strict link $D \rightarrow C$, C will be only executed if D has been successfully completed.

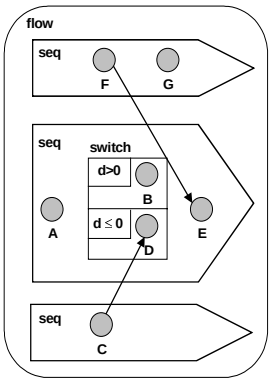


Fig. 8: Representation of the flow schema from Fig. 5 as schema instance of Class-1

Of course, one may define other link patterns as well. For example, the distinction between links crossing the boundary of structured activities and links for which this is not the case may be advantageous in several respects. However, in the following we restrict our considerations to weak and strict links since they cover most practical cases.

4. Verifying the Correctness of BPEL4WS Specifications

In this section we cope with correctness issues related to the definition and change of BPEL control flow schemes. We discuss important requirements arising in this context and sketch whether and – if yes – how they can be met for BEPL schemes of the different classes. In Section 4.1 we summarize characteristic problems related to the definition and change of BPEL control flow schemes. In Sections 4.2 - 4.4 we then sketch how the different model classes from Section 3 cope with these problems.

4.1 A Selection of Typical Control Flow Problems

Particularly, the following important requirements have to be met:

1. Flows must be *deadlock-free*, i.e., there is no situation where a flow instance has not

yet reached a correct final state but no activity can be finished anymore.

2. The flow must *terminate exactly once* for each initiation; i.e., it must terminate completely without any residual branch being still under execution.
3. Each activity X should be *reachable*; i.e., starting with a correct initial state there must be a valid sequence of activity executions and outputs that will lead to activation of X.
4. A receive and a reply activity, which belong to same synchronous invocation of a complex operation, both must be present in the flow schema. The reply activity is to be executed exactly once in case the corresponding receive activity is carried out.

In order to exclude these problems the following basic issues must be addressed: The *T-Problem* (*Termination Problem*) is to determine whether flow execution will always terminate correctly (i.e., Req. 1 and 2 are met). The *I-Problem* (*Initiation Problem*) problem is to check whether there is a sequence of activities leading to activation of a particular activity (i.e., Req. 3 is met) – generally, respective flow checks have shown to be NP-complete [HOR98]. Finally, the *S-Problem* (*Synchronization Problem*) is to check whether receive / reply activities are correctly used within a flow schema (cf. Req. 4).

4.2 Ensuring Control Flow Correctness for Flow Schemes of Class-0

Flow schemes with strict block structuring guarantee the above control flow properties almost for free: On condition that some fairness assumptions are met (e.g., no infinite loops, each branch of a switch activity to be selectable) the flow will always properly terminate and each flow activity will be reachable; i.e., the *T-Problem* and the *I-Problem* are uncritical for schemes from Class-0. Furthermore the *S-Problem* can be easily decided when transforming the BPEL schema into a canonical graph-based representation and by applying simple analyses on the generated graph.

4.3 Ensuring Control Flow Correctness for Flow Schemes of Class-1

Regarding flow schemes with block structuring and restricted use of links we must deal with additional issues in order to avoid the mentioned control flow problems. To deal with the *T-problem*, for example, we must ensure that link usage does not lead to deadlock-causing cycles. Fig. 9 depicts two examples: In Fig. 9 a) orders are solely defined by the use of links whereas the schema from Fig. 9 b) uses links as well as a sequence activity for this purpose. In both cases a deadlock will occur during flow execution.

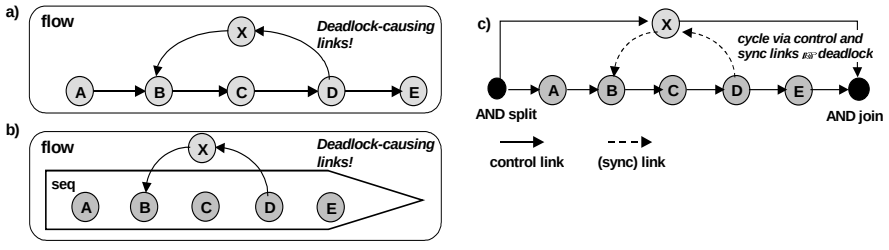


Fig. 9: Two models with deadlock-causing links and internal graph-based representation

It would certainly be no good idea to treat these similar cases in different ways. Thus a canonical graph-based representation of the flow contributes to detect respective problems quickly and by means of well-known procedures. Following this, the lower schema (of Class-1) has to be translated into a corresponding graph-based representation (cf. Fig. 9 c) in order to perform the necessary graph analyses (in the given example with respect to absence of deadlock-causing cycles).

Another example of a BEPL schema (of Class-1), which is not deadlock-free, is depicted in Fig. 10. Activity G will not be activated and a deadlock will occur if the upper branch (i.e., activity C) of the switch activity is selected for execution. Obviously, the use of a strict link does not make sense in this context and should therefore be prohibited. Generally, to detect respective deadlocks, we must perform more expensive reachability analysis (again based on a canonical graph-based representation of the flow schema). Such analyses have shown to be NP-complete, but for the given schema class we can additionally benefit from the (partial) block structuring of the flow schemes (and of related graph-based representation). If only weak links are used, deadlocks can be solely detected on basis of (less complex) cycle tests. This result is of high practical relevance since the most common use of the link concept corresponds to semantics of weak links.

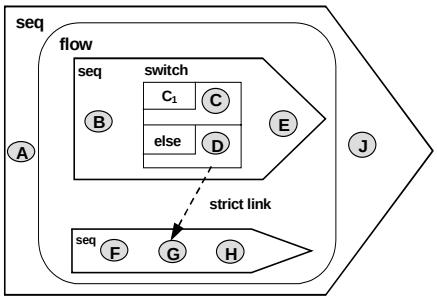


Fig. 10: Incorrect use of a strict link

Regarding the *I-problem* similar considerations can be made. If solely weak links are used, task reachability does not constitute a problem due to the special semantics of this link type. By contrast, in certain cases the use of strict links could lead to situations where an activity cannot be activated (e.g., activity G in Fig. Fig. 10).

4.4 Flow Schemes of Class-2 (Block Structuring and Arbitrary Use of Links)

When regarding arbitrary BPEL schemes from Class-2 (i.e., flow schemes with arbitrary use of BPEL model elements and links) things become much more difficult. In particular, in BPEL there are many ways for expressing the same control flow pattern and for configuring the flow behavior (e.g., with respect to handling of join failures). However, this variety and flexibility makes it a very tough job to study and verify the flow behavior at buildtime (if possible at all). Regarding the use of transition conditions, for example, several problems exist. They range from variables not correctly initialized when a related predicate (e.g., a transition condition or a loop condition) is evaluated up to flow behaviors on which we cannot decide at buildtime. An activity with two outgoing links, for example, may now represent an AND split and then an XOR split

depending on the current values of the variables used for evaluating the respective transition conditions. However, this mixed semantics significantly aggravates flow verifications.

5. Summary

We have described different BPEL schema classes and sketched to which extent their instances can be analyzed for the absence of design errors. As illustrated the full expressiveness of BPEL4WS is usually not needed to cover today's business processes. In order to reduce complexity and to provide the basis for more advanced process functions (e.g., supporting dynamic flow changes [RD98, RRD03]) a good compromise would be to make use of the flow structuring and the presented link guidelines as far as possible. Furthermore we need a common formal graph model for representing (arbitrary) BPEL schemes. Such a graph-based representation is not only fundamental with respect to flow verification, but may also serve as basis for defining a precise formal and operational semantics for BPEL flows. In summary, we have made similar considerations regarding other BPEL concepts (e.g., flow of data). We believe that the further development of BPEL would benefit by a more systematic and critical treatment of the existing proposal. Unfortunately, at the moment, the discussion mainly focuses on how to introduce additional modeling concepts and thus additional complexity to the BPEL specification.

References

[Al04] G. Alonso, F. Casati et al: Web Services, Springer, 2004

[An03] T. Andrews et al: Business Process Execution Language for Web Services, V. 1.1, May 2003

[BK03] F. Breugel, M. Koshkina: Verification of business processes for web services. Technical Report CS-2003-11, York University, Ontario, CA

[Ch03] S. Chandrasekaran: Composition, performance analysis and simulation of web services. Master thesis, University of Georgia, USA, 2003

[HB03] R. Hamadi, B. Benatallah: A Petri net-based model for Web Service composition. Proc. 14th Australasian Database Conference, Adelaide, Australia, 2003.

[HOR98] A. ter Hofstede, M. Orlowska, J. Rajapakse: Verification problems in conceptual workflow specifications. Data & Knowledge Engineering, 24(3):239–256, 1998

[KHB00] B. Kiepuszewski, A. ter Hofstede, C. Bussler: On structured workflow modeling. Proc. CAiSE'00, Stockholm, June 2000, LNCS 1789, pp. 431-445.

[KMW03] R. Khalaf, N. Mukhi, S. Weerawarana: Service-oriented composition in BPEL4WS. Proc. WWW'03, Budapest, May 2003

[Ma04] A. Martens: Analysis and reengineering of Web Services. Proc. ICEIS'04, Porto, 2004

[NM02] S. Narayanan, S. McIlraith: Analysis and Simulation of Web Services. Proc. 11th Int'l Conf. World Wide Web, 2002

[RD98] M. Reichert, P. Dadam: $ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. JIIS, 10(2):93-129, 1998

[RRD03] M. Reichert, S. Rinderle, P. Dadam: On the Common Support of Workflow Type and Instance Changes Under Correctness Constraints. Proc. CoopIS'03, Nov. 2003

[Wo02] P. Wohed, W.M.P. van der Aalst, M. Dumas, A. ter Hofstede: Pattern Based Analysis of BPEL4WS. TR FIT-TR-2002-04, Queensland University of Technology, Australia