Life After BPEL?

W.M.P. van der Aalst[1,2], M. Dumas[2], A.H.M. ter Hofstede[2], N. Russell[2], H.M.W. Verbeek[1], and P. Wohed[3]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl
[2] Queensland University of Technology, Brisbane, Australia.
{m.dumas,a.terhofstede,n.russell}@qut.edu.au
[3] Université Henri Poincaré, Nancy, France. petia.wohed@cran.uhp-nancy.fr

Abstract. The *Business Process Execution Language for Web Services* (BPEL) has emerged as a standard for specifying and executing processes. It is supported by vendors such as IBM and Microsoft and positioned as the "process language of the Internet". This paper provides a critical analysis of BPEL based on the so-called *workflow patterns*. It also discusses the need for languages like BPEL. Finally, the paper addresses several challenges not directly addressed by BPEL but highly relevant to the support of web services.

1 Introduction

Web services, an emerging paradigm for architecting and implementing business collaborations within and across organizational boundaries, are currently of interest to both software vendors and scientists. In this paradigm, the functionality provided by business applications is encapsulated within web services: software components described at a semantic level, which can be invoked by application programs or by other services through a stack of Internet standards including HTTP, XML, SOAP, WSDL and UDDI [3, 12]. Once deployed, web services provided by various organizations can be inter-connected in order to implement business collaborations, leading to *composite web services*.

The *Business Process Execution Language for Web Services* (BPEL4WS, or BPEL for short) has emerged as the de-facto standard for implementing processes based on web services [9]. Systems such as Oracle BPEL Process Manager, IBM WebSphere Application Server Enterprise, IBM WebSphere Studio Application Developer Integration Edition, and Microsoft BizTalk Server 2004 support BPEL, thus illustrating the practical relevance of this language. Although intended as a language for connecting web services, its application is not limited to cross-organizational processes. It is expected that in the near future a wide variety of process-aware information systems [13] will be realized using BPEL. Whilst being a powerful language, BPEL is difficult to use. Its XML representation is very verbose and only readable to the trained eye. It offers many constructs and typically things can be implemented in many ways, e.g., using links and the flow construct or using sequences and switches. As a result

only experienced users are able to select the right construct. Several vendors offer a graphical interface that generates BPEL code. However, the graphical representations are a direct reflection of the BPEL code and are not intuitive to end-users. Therefore, BPEL is closer to classical programming languages than e.g. the more user-friendly workflow management systems available today.

It is interesting to put BPEL in a historical perspective. In the seventies, people like Skip Ellis [15], Anatol Holt [27], and Michael Zisman [48] were already working on so-called office information systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. In the seventies, organizations were not connected and only few people inside one organization were linked through some kind of network. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the eighties. In the nineties, once again there was huge interest in these systems. The number of workflow products developed in the past decade and the many papers on workflow technology illustrate the revival of office information systems. Today workflow management systems are readily available [4, 33, 37] and workflow technology is hidden in many applications, e.g., ERP, CRM, and PDM systems. However, their application is still limited to specific industries such as banking and insurance. Since 2000 there has been a growing interest in web services. This resulted in a stack of Internet standards (HTTP, XML, SOAP, WSDL, and UDDI) which needed to be complemented by a process layer. Several vendors proposed competing languages, e.g., IBM proposed WSFL (Web Services Flow Language) [32] building on FlowMark/MQSeries and Microsoft proposed XLANG (Web Services for Business Process Design) [45] building on Biztalk. BPEL [9] emerged as a compromise between both languages.

The goal of this paper is to critically analyze BPEL. We analyze the language itself using a patterns-based approach [5]. In addition, we discuss the focus of BPEL. In our view organizations do *not* need to agree on a common execution language. We will argue that there are more important issues to be addressed, e.g., having a higher-level language to describe both processes and interactions and being able to monitor running composite web-services/choreographies.

The remainder of this paper is organized as follows. Section 2 briefly introduces the BPEL language and its focus. In Section 3 we discuss existing work on workflow patterns and relate this to BPEL. In Section 4 we question the need for a language like BPEL. Section 5 proposes the real challenges we should focus on in the context of BPEL: (1) generating and analyzing BPEL code (Section 5.1), (2) "real" choreography (Section 5.2), and (3) process mining, conformance testing and mediation (Section 5.3). Section 6 concludes the paper by providing some pointers to the "Petri and Pi" initiative which aims at combining efforts on theory, languages, tools, and applications in the web services domain.

2 BPEL

BPEL [9] supports the modeling of two types of processes: executable and abstract processes. An *abstract*, (not executable) *process* is a business protocol, specifying the message exchange behavior between different parties without revealing the internal behavior for any one of them. This abstract process views the outside world from the perspective of a single organization or (composite) service. An *executable process* views the world in a similar manner, however, things are specified in more detail such that the process becomes executable, i.e., an executable BPEL process specifies the execution order of a number of *activities* constituting the process, the *partners* involved in the process, the *messages* exchanged between these partners, and the *fault* and *exception handling* required in cases of errors and exceptions.

A BPEL process itself is a kind of flow-chart, where each element in the process is called an *activity*. An activity is either a primitive or a structured activity. The set of *primitive activities* contains: `invoke`, invoking an operation on a web service; `receive`, waiting for a message from an external source; `reply`, replying to an external source; `wait`, pausing for a specified time; `assign`, copying data from one place to another; `throw`, indicating errors in the execution; `terminate`, terminating the entire service instance; and `empty`, doing nothing.

To enable the presentation of complex structures the following *structured activities* are defined: `sequence`, for defining an execution order; `switch`, for conditional routing; `while`, for looping; `pick`, for race conditions based on timing or external triggers; `flow`, for parallel routing; and `scope`, for grouping activities to be treated by the same fault-handler. Structured activities can be nested and combined in arbitrary ways. Within activities executed in parallel the execution order can further be controlled by the usage of `links` (sometimes also called control links, or guarded links), which allows the definition of directed graphs. The graphs too can be nested but must be acyclic.

As indicated in the introduction, BPEL builds on IBM's WSFL (Web Services Flow Language) [32] and Microsoft's XLANG (Web Services for Business Process Design) [45] and combines the features of a block structured language inherited from XLANG with those for directed graphs originating from WSFL. As a result simple things can be implemented in two ways. For example a sequence can be realized using the `sequence` or `flow` elements (in the latter case links are used to enforce a particular order on the parallel elements), a choice based on certain data values can be realized using the `switch` or `flow` elements, etc. However, for certain constructs one is forced to use the block structured part of the language, e.g., a *deferred choice* (see next section and [5]) can only be modeled using the `pick` construct. For other constructs one is forced to use the links, i.e., the more graph-oriented part of the language, e.g., two parallel processes with a one-way synchronization require a `link` inside a `flow`. In addition, there are very subtle restrictions on the use of links: "A link MUST NOT cross the boundary of a while activity, a serializable scope, an event handler or a compensation handler... In addition, a link that crosses a fault-handler boundary MUST be outbound, that is, it MUST have its source activity within the fault handler and its target

activity within a scope that encloses the scope associated with the fault handler. Finally, a link MUST NOT create a control cycle, that is, the source activity must not have the target activity as a logically preceding activity, where an activity A logically precedes an activity B if the initiation of B semantically requires the completion of A. Therefore, directed graphs created by links are always acyclic." (see page 64 in [9]). All of this makes the language complex for end-users. A detailed or complete description of BPEL is beyond the scope of this paper. For more details, the reader is referred to [9] and various web sites such as the web site of the OASIS technical committee on WS-BPEL: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

3 Patterns-based Analysis of BPEL

Based on earlier experiences in the workflow domain, we have evaluated BPEL using the so-called workflow patterns [5]. The initial set of 20 workflow patterns focused exclusively on control-flow aspects. We will briefly discuss our experiences with BPEL based on these patterns. However, before doing so, we would like to emphasize that the patterns initiative (`www.workflowpatterns.com`) is not limited to control-flow. We have developed a comprehensive set of *data patterns* [43]. These patterns describe the different ways of dealing with data in the context of some process-aware information systems [13] (e.g., a workflow management system like Staffware, an ERP system like SAP R/3, or integration middleware like WebSphere). In the context of workflow management, we distinguish four classes of data patterns: *data visibility* (relating to the extent and manner in which data elements can be viewed by various components of a workflow process), *data interaction* (focusing on the manner in which data is communicated between active elements within a workflow), *data transfer* (considering the means by which the actual transfer of data elements occurs between workflow components and describe the various mechanisms by which data elements can be passed across the interface of a workflow component) and *data-based routing* (characterizing the manner in which data elements can influence the operation of other aspects of the workflow, particularly the control flow perspective). We have also developed a comprehensive set of *resource patterns* [42] that capture the various ways in which resources are represented and utilized in workflows. These patterns are also grouped into a number of categories: *creation patterns*, *push patterns*, *pull patterns*, *detour patterns*, *auto-start patterns*, *visibility patterns*, and *multiple resource patterns*. Since they are less relevant in the context of BPEL, we do not elaborate on them in any detail. Other related work includes Colored Petri Net (CPN) patterns [38], Enterprise Application Integration (EAI) patterns [26], and Service Interaction (SI) patterns [10].

For a detailed description and discussion of the patterns and more pointers we refer the reader to `www.workflowpatterns.com` and [5, 43]. As an illustration, we describe control-flow pattern WCFP16 (Deferred Choice).

WCFP16 Deferred Choice A point in a process where one among several al-

ternative branches is chosen based on information which is not necessarily available when this point is reached. This differs from the normal exclusive choice, in that the choice is not made immediately when the point is reached, but instead several alternatives are offered, and the choice between them is delayed until the occurrence of some event.
Example: When a contract is finalized, it has to be reviewed and signed either by the director or by the operations manager, whoever is available first. Both the director and the operations manager would be notified that the contract is to be reviewed: the first one who is available will proceed with the review.

Note that WCFP16 is different from the WCFP 4 (i.e., Exclusive Choice): The choice is not based on a decision or data but on a choice resolved by the environment. BPEL clearly supports this pattern. The `pick` (for race conditions based on timing or external triggers) directly offers the desired functionality.

Table 1. An analysis of BPEL based on the workflow control-flow patterns [47].

pattern	pattern name	BPEL
WCFP1	sequence	+
WCFP2	parallel split	+
WCFP3	synchronization	+
WCFP4	exclusive choice	+
WCFP5	simple merge	+
WCFP6	multi choice	+
WCFP7	synchronizing merge	+
WCFP8	multi merge	–
WCFP9	discriminator	–
WCFP10	arbitrary cycles	–
WCFP11	implicit termination	+
WCFP12	multiple instances no synchronization	+
WCFP13	multiple instances design time knowledge	+
WCFP14	multiple instances runtime knowledge	–
WCFP15	multiple instances without a priori knowledge	–
WCFP16	deferred choice	+
WCFP17	interleave parallel routing	+/–
WCFP18	milestone	–
WCFP19	cancel activity	+
WCFP20	cancel case	+

Tables 1 and 2 summarize the results of our pattern-based evaluation of BPEL. For each control-flow and data pattern, we checked whether it is possible to realize the pattern with BPEL. If BPEL directly supports the pattern through one of its constructs, it is rated +. If the pattern is not *directly* supported, it is rated +/-. Any solution that results in "spaghetti-like constructs" or is not possible at all, is considered as giving no direct support and is rated –. These ratings should be interpreted with care as indicated in [5, 43].

We cannot give a detailed explanation of each pattern or of the evaluation of BPEL based on this material (for this we refer to [5, 43, 47]). However, a general observation that we would like to make is that BPEL is more powerful than

Table 2. An analysis of BPEL based on the workflow data patterns [43].

pattern	pattern name	BPEL
WDP1	task data	+/−
WDP2	block data	−
WDP3	scope data	+
WDP4	folder data	−
WDP5	multiple instance data	−
WDP6	case data	+
WDP7	workflow data	−
WDP8	environment data	+
WDP9	data interaction between tasks	+
WDP10	data interaction – block task to decomposition	−
WDP11	data interaction – decomposition to block task	−
WDP12	data interaction – to multiple instance task	−
WDP13	data interaction – from multiple instance task	−
WDP14	data interaction – case to case	+/−
WDP15	data interaction – task to environment – push-oriented	+
WDP16	data interaction – environment to task – pull-oriented	+
WDP17	data interaction – environment to task – push-oriented	+/−
WDP18	data interaction – task to environment – pull-oriented	+/−
WDP19	data interaction – case to environment – push-oriented	−
WDP20	data interaction – environment to case – pull-oriented	−
WDP21	data interaction – environment to case – push-oriented	−
WDP22	data interaction – case to environment – pull-oriented	−
WDP23	data interaction – workflow to environment – push-oriented	−
WDP24	data interaction – environment to workflow – pull-oriented	−
WDP25	data interaction – environment to workflow – push-oriented	−
WDP26	data interaction – workflow to environment – pull-oriented	−
WDP27	data passing by value – incoming	+
WDP28	data passing by value – outgoing	+
WDP29	data passing – copy in/copy out	−
WDP30	data passing by reference – unlocked	+
WDP31	data passing by reference – locked	+/−
WDP32	data transformation – input	−
WDP33	data transformation – output	−
WDP34	task precondition – data existence	+/−
WDP35	task precondition – data value	+
WDP36	task postcondition – data existence	−
WDP37	task postcondition – data value	−
WDP38	event-based task trigger	+
WDP39	data-based task trigger	+/−
WDP40	data-based routing	+

most traditional process languages. The control-flow part of BPEL inherits almost all constructs of the block structured language XLANG and the directed graphs of WSFL. Therefore, it is no surprise that BPEL indeed supports the union of patterns supported by XLANG and WSFL. BPEL offers direct support for the Multi Choice (WCFP6) and Synchronizing Merge (WCFP7), but not for Arbitrary Cycles (WCFP10). This is a consequence of the "dead-path elimina-

tion" principle inherited from WSFL. BPEL, through the concept of serializable scopes, is one of the few languages to support the Interleaved Parallel Routing pattern (WCFP17), although with some restrictions. BPEL is also one of the few languages that fully supports the notion of scope data elements (WDP3). It provides support for a scope construct which allows related activities, variables and exception handlers to be logically grouped together. The default binding for data elements in BPEL is at case level and they are visible to all of the components in a process. However, variables can be bound to scopes within a process definition which may encompass a number of tasks and there is also the ability for messages to be passed between tasks when control passes from one task to another.

So the overall observation is that BPEL is an expressive language with some limitations. However, BPEL is also a very complicated language with many concepts. This complexity is reflected in the large number of issues that have been raised within the OASIS WS-BPEL standardization committee (217 as of June 2005), and which have delayed the release of the WS-BPEL 2.0 standard specification.

4 Do we need BPEL?

In the previous section, we concluded that BPEL may be too complex but, compared to other languages, it is also very powerful. In this section, we do not focus on the specific qualities of BPEL. Instead we focus on the question: "Do we need a language like BPEL?".

Although BPEL can be used as a classical workflow language, its development was triggered by the web service paradigm. Therefore, BPEL was intended initially for pure cross-organizational processes in a web services context: "BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions." (see page 1 in [9]). However, it can also be used to support intra-organizational processes. The authors of BPEL [9] envision two possible uses of the language: "Business processes can be described in two ways. Executable business processes model actual behavior of a participant in a business interaction. Business protocols, in contrast, use process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. The process descriptions for business protocols are called abstract processes. BPEL is meant to be used to model the behavior of *both executable and abstract processes*." In our view, executable and abstract processes should not be supported by a single language. Most attention has been devoted to BPEL as an execution language. In our opinion BPEL failed as a language for modeling abstract processes. Moreover, a BPEL specification is always given from the viewpoint of *one* of the interacting partners. Web Services provided by partners can be used to perform work in a BPEL business process. Invoking an operation on such a service is a basic activity that can be specified using BPEL.

Figure 1 shows the two possible uses of BPEL. The figure clearly illustrates that in both cases the work is *seen from the perspective of one of the partners*!

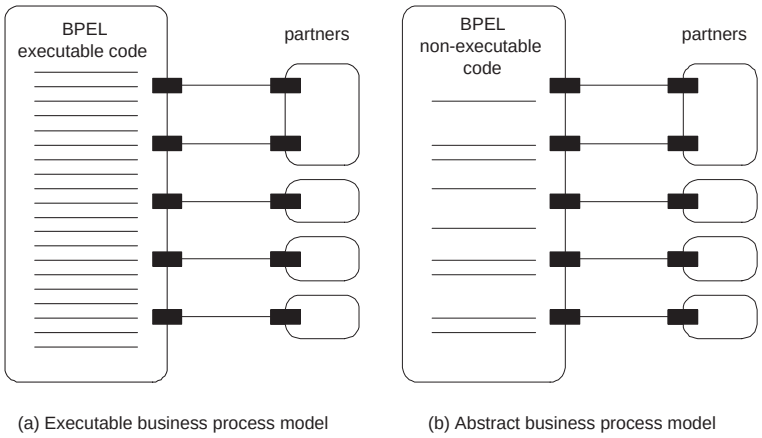

Fig. 1. The two ways in which BPEL can be used.

Figure 1 raises the question why every partner should standardize on BPEL as a process language. A partner providing a service may implement its underlying processes in any language without the other partner knowing, i.e., *interacting partners do not need to agree on a language like BPEL.* Therefore, the answer to "Do we need a language like BPEL?" is No! Nevertheless, BPEL has become the de-facto standard and may in the future facilitate organizations migrating from one system to another. In addition, BPEL incorporates a number of specialized features for web services development including direct support for XML data definition and manipulation, a dynamic binding mechanism based on explicit manipulation of endpoint references, a declarative mechanism for correlating incoming messages to process instances (which is essential for asynchronous communication), etc. As such, BPEL may be seen as an attractive alternative to conventional (object-oriented) programming languages when it comes to developing web services.

5 Let us focus on the real challenges!

Although a language like BPEL is not essential for parties to cooperate, its dominance raises the question of how to facilitate the use of BPEL and to identify the missing functionality. In other words: we want to address the "real" challenges in the context of BPEL. This is the reason the title of this paper is "Life after BPEL?". In this section we briefly discuss three challenges: "generating and analyzing BPEL code", "real choreography", and "process mining, conformance testing, and mediation".

5.1 Generating and Analyzing BPEL Code

Since BPEL is increasingly supported by various engines it becomes interesting to link it to other types of models. This is useful for two reasons: (1) BPEL more closely resembles a programming language than a modeling language and (2) BPEL itself does not allow for any form of analysis other than being executable (e.g., no verification, performance analysis, etc.). Therefore, there are two interesting translations: (1) a translation from a "higher-level" notation to BPEL and (2) a translation from BPEL to a model that allows for analysis.

Until now, attention has focused on the second translation. Several attempts have been made to capture the behavior of BPEL in a formal way. Some advocate the use of finite state machines [19–21], others process algebras [18, 31], and yet others use abstract state machines [16, 17] or Petri nets [39, 35, 44]. A comparative summary of mappings from BPEL to formal languages is given in Table 3. The columns of the table correspond to the following criteria:

- *Tech* indicates the formalization technique used: FSM for finite state machines, PA for Process Algebra, ASM for Abstract State Machines and PN for Petri Nets.
- *SA* indicates whether the mapping covers structured activities fully (+), partially (+/–) or not at all (–). It can be seen that this feature is covered by all proposed mappings.
- *CL* indicates whether the formalization covers control links. Here a +/- rating is given for partial mappings of control links (e.g. not covering join conditions which is a feature associated to control links).
- *EH* indicates whether the formalization covers event and exception handling. Some references cover fault handling, but do not cover compensation and/or event handling, in which case, a +/– rating is assigned.
- *Comm* indicates whether the mapping can be applied to systems of interconnected BPEL processes (+) or if they are restricted to individual processes (–). In the former case, it is possible to use the mapping to detect potential mismatches between two or more BPEL processes which are expected to communicate with each other.
- *TAV* indicates whether a tool for automatic verification is provided. Some authors [19, 31] have developed and/or used tools for BPEL verification but only to perform simple syntactic checks such as detecting cyclic dependencies generated by control links, or unnecessary checks such as deadlock-freeness of individual BPEL processes.[1] In these cases a +/– rating is given. This latter rating is also given to proposals where formal analysis is possible but requires significant manual steps. Finally, some authors refer to the possibility of performing formal verification [18, 17], but do not develop any automated means of doing so. In this case, a – rating is given.

In industry, various tools and mappings are being developed to generate BPEL code from a graphical representation. Tools such as the IBM WebSphere

[1] Individual BPEL processes are deadlock-free by construction [35].

Table 3. A comparative summary of some of the related work on BPEL formalization and analysis.

	Tech	SA	CL	EH	Comm	TAV
[21]	FSM	+	–	–	+	+
[20]	FSM	+	–	–	+	+/–
[19]	FSM	+	–	+/–	–	+/–
[18]	PA	+	–	+	–	–
[31]	PA	+	+	–	–	+/–
[17]	ASM	+	+/–	–	–	–
[35, 44]	PN	+	+/–	+	+	+/–
[39]	PN	+	+	+	–	+

Choreographer and the Oracle BPEL Process Manager offer a graphical notation for BPEL. However, this notation directly reflects the code and there is no intelligent mapping. This implies that users have to think in terms of BPEL constructs (e.g., blocks, syntactical restrictions on links, etc.). More interesting is the work of Stephen White that discusses the mapping of BPMN to BPEL [46] and the work by Jana Koehler and Rainer Hauser on removing loops in the context of BPEL [30]. However, none of these publications provides a mapping of some (graphical) process modeling language onto BPEL: [46] merely presents the problem and discusses some issues using examples and [30] focuses on only one piece of the puzzle. This motivated us to develop a mapping from Colored Petri Nets (CPNs) to BPEL [6]. Clearly, both types of mappings are highly relevant. However, the quality of these mappings needs to be improved and there should be more agreement on the precise semantics of BPEL.

5.2 Real Choreography

As indicated in Section 4 interacting partners do not need to agree on a language like BPEL. However, they need to agree on an overall global process. Currently terms like *choreography* and *orchestration* are used to refer to the problem of agreeing on a common process. Some people distinguish between choreography and orchestration, e.g., "In orchestration, there's someone – the conductor – who tells everybody in the orchestra what to do and makes sure they all play in sync. In choreography, every dancer follows a pre-defined plan - everyone independently of the others." We will not make this distinction and simply assume that *choreographies define collaborations between interacting parties*, i.e., the coordination process of interconnected web services all partners need to agree on. Figure 2 illustrates the notion of a choreography.

Within the Web Services Choreography Working Group of the W3C, a working draft defining version 1.0 of the *Web Services Choreography Description Language* (WS-CDL) has been developed [29]. The scope of WS-CDL is defined as follows: "Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each party can then use the global definition to build and test

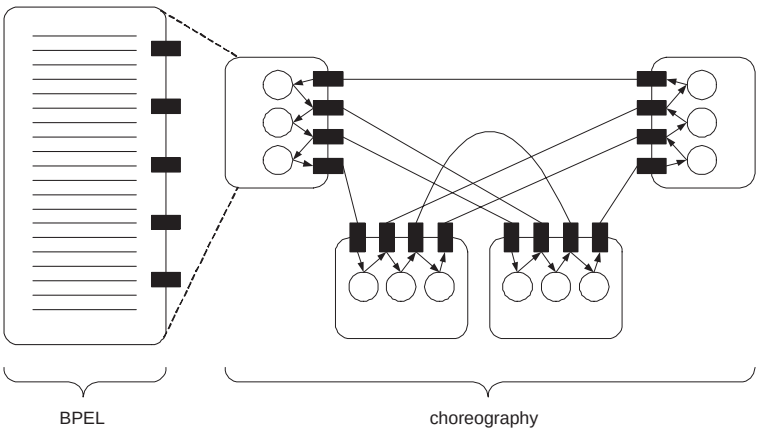

Fig. 2. A choreography defines collaborations between interacting parties.

solutions that conform to it. The global specification is in turn realized by a combination of the resulting local systems, on the basis of appropriate infrastructure support. The advantage of a contract based on a global viewpoint as opposed to any one endpoint is that it separates the overall global process being followed by an individual business or system within a domain of control (an endpoint) from the definition of the sequences in which each business or system exchanges information with others. This means that, as long as the observable sequences do not change, the rules and logic followed within a domain of control (endpoint) can change at will and interoperability is therefore guaranteed." [29]. This definition is consistent with the critique in Section 4 and Figure 2. Unfortunately, like most standards in the web services stack, the language is verbose and complex. Somehow the essence as shown in Figure 2 is lost. Moreover, the language again defines concepts such as "sequence", "choice", and "parallel" in some ad-hoc notation with unclear semantics. This suggests that some parts of the language are an alternative to BPEL while they are not. The main problem is that WS-CDL is not declarative. A choreography should allow for the specification of the "what" without having to state the "how". This is similar to the difference between a program and its specification. One can specify what an ordered sequence is without specifying an algorithm to do so!

In [1] we describe a more theoretical approach to the problem. The paper describes the P2P (Public-To-Private) approach which addresses one of the most notorious problems in this domain: How to design an inter-organizational workflow such that there is local autonomy without compromising the consistency of the overall process. The approach uses a notion of inheritance and consists of three steps: (1) create a common understanding of the inter-organizational workflow by specifying the shared public workflow, (2) partition the public workflow over the organizational entities involved, and (3) for each organizational entity: create a private workflow which is a subclass of the relevant part of the public workflow. In [1] it is shown that this approach avoids typical anomalies in business-to-business collaboration (e.g., deadlocks and livelocks) and yields an

inter-organizational workflow which is guaranteed to realize the behavior specified in the public workflow. The P2P approach relies heavily on the use of Petri nets and a formal notion of inheritance. Nevertheless, it would be interesting to adopt these ideas in the context of languages such as WS-CDL and BPEL. Another, more declarative, approach could be based on *temporal logic* [34, 40]. Languages such as *Linear Temporal Logic* (LTL) allow for the definition and verification of desirable behavior [23–25].

5.3 Process Mining, Conformance Testing, and Mediation

Assuming that there is a running process (possibly implemented using BPEL) and a choreography specification (possibly specified in WS-CDL), it is interesting to check whether each partner/web-service is well behaved. Note that partners have no control over each other's services. Moreover, partners will not expose the internal structure and state of their services. The closed and uncontrollable nature of web-services may generate a variety of problems. Fortunately, *process mining* [7] and *conformance testing* [2] techniques may be of assistance. For both we need to assume the existence of an *event log* [7]. For example, one may log the messages exchanged between all parties involved in a choreography (either distributed or through some coordinator). Using this event log, we may use process mining techniques to reconstruct part of the process that actually took place. This way one can "discover" the actual choreography. However, in an ideal situation this choreography is given in terms of a predefined process model. The coexistence of event logs and process models raises the question of conformance. This question may be viewed from two angles. First of all, the model may be assumed to be "correct" because it represents the way partners should work, and the question is whether the events in the log are consistent with the process model. For example, the log may contain "incorrect" event sequences not possible according to the model. This may indicate violations of choreography all parties previously agreed upon. Second, the event log may be assumed to be "correct" because it is what really happened. In the latter case the question is whether the choreography that has been agreed upon is no longer valid and should be modified. To actually measure conformance, we have developed a tool called *Conformance Checker*. This tool has been developed in the context of the *ProM framework*[2]. The ProM framework offers a wide range of tools related to process mining, i.e., extracting information from event logs [7]. At this point in time we are investigating the addition of plug-ins specific for the mining of web services. Some preliminary investigations have been reported in [14, 22].

Another prominent issue, complementary to conformance, is that of *mediation*. When it is found (either a priori through model comparison or a posteriori through mining), that the conversation protocol that a given service provides does not match the conversation protocol that it is expected to provide, there are basically two options: (1) modify the service to suit the new expected conversation protocol; or (2) mediate between the conversation protocol of the service

[2] Both documentation and software can be downloaded from `www.processmining.org`.

as it is, and the conversation protocol as it should be. The former option is usually not suitable because the same service may interact with other services that rely on the conversation protocol that the service currently provides. In other words, the same service may participate in different collaborations such that in each of these collaborations a different conversation protocol is expected from it. Thus, mediation between the *provided conversation protocol* of a service, and the various conversation protocols that are expected from it (i.e., the *required conversation protocols*), is generally unavoidable. This issue has been widely studied in the area of software components where it is known as *adaptation*. However, most of the work on component adaptation focuses on structural mediation (i.e., mediating different structural interfaces and specifically, between different data types). Since services are expected to participate in collaborations driven by process models, behavioral mediation is a prominent requirement. Some work has been done in this area both in the components and services community [28, 11, 8], but there is still no overarching framework and supporting tools for behavioral service mediation are missing.

6 Petri and Pi

In discussions, Petri nets [41] and Pi calculus [36] are often mentioned as two possible formal languages that could serve as a basis for languages such as BPEL and WS-CDL. Some vendors claim that their systems are based on Petri nets or Pi calculus and other vendors suggest that they do not need a formal language to base their system on. In essence there are three "camps" in these discussions: the "Petri net camp", the "Pi calculus" (or process algebra) camp, and the "Practitioners camp" (also known as the "No formalism camp"). This was the reason for starting the "Petri nets and Pi calculus for business processes" working group (`http://www.smartgroups.com/groups/petri_and_pi`) in June 2004. Its goal is to have discussions and meetings on the formal foundations of BPM in general and languages like BPEL in particular. The working group was initiated by Robin Milner, Wil van der Aalst, Rob van Glabbeek, Roger Whitehead, and Keith Harrison-Broninski. The first meeting of this working group took place in June 2005 at Eindhoven University of Technology. Interesting elements of the first meeting were the identification of meaningful patterns and the sharing of solutions of common examples using languages such as BPEL, WS-CDL, colored Petri nets, Pi calculus, YAWL, statecharts, CCS, SOS, RAD, etc.

Most of the topics discussed in this paper are relevant to the Petri and Pi working group. In fact, this paper was inspired by the Eindhoven workshop of this group. Interested readers are invited to join this working group by sending an e-mail to `petri_and_pi-owner@smartgroups.com` or one of its members with the request to become a member.

References

1. W.M.P. van der Aalst. Inheritance of Interorganizational Workflows: How to agree

to disagree without loosing control? *Information Technology and Management Journal*, 4(4):345–389, 2003.
2. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering Journal*, 2005 (to appear).
3. W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Web Service Composition Languages: Old Wine in New Bottles? In *Proceeding of the 29th EUROMICRO Conference: New Waves in System Architecture*, pages 298–305. IEEE Computer Society, Los Alamitos, CA, 2003.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
6. W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL4WS. BETA Working Paper Series, Eindhoven University of Technology, Eindhoven, 2005.
7. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
8. M. Altenhofen, E. Boerger, and J. Lemcke. An execution semantics for mediation patterns. In *Proceedings of the BPM2005 Workshops: Workshop on Choreography and Orchestration for Business Process Managament*, Nancy, France, September 2005.
9. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
10. A. Barros, M. Dumas, and A.H.M. ter Hofstede. Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. QUT Technical report, FIT-TR-2005-012, Queensland University of Technology, Brisbane, 2005. (To appear in BPM 2005.)
11. B. Benatallah, F. Casati, D. Grigori, H. Motahari-Nezhad, and F. Toumani. Developing Adapters for Web Services Integration. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, Porto, Portugal, June 2005. Springer Verlag.
12. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.
13. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems*. Wiley & Sons, 2005.
14. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Technical Report TUV-1841-2004-16, Information Systems Institute, Vienna University of Technology, Wien, Austria, 2004.
15. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
16. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, 2005.

17. R. Farahbod, U. Glässer, and M. Vajihollahi. Specification and validation of the business process execution language for web services. In *Abstract State Machines 2004*, volume 3052 of *Lecture Notes in Computer Science*, pages 79–94, Lutherstadt Wittenberg, Germany, May 2004. Springer-Verlag, Berlin.
18. A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
19. J.A. Fisteus, L.S. Fernández, and C.D. Kloos. Formal verification of BPEL4WS business collaborations. In *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, August 2004. Springer-Verlag, Berlin.
20. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Composition. In *Proceedings of 18th IEEE International Conference on Automated Software Engineering (ASE)*, pages 152–161, Montreal, Canada, October 2003.
21. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *International World Wide Web Conference: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM Press.
22. W. Gaaloul, S. Bhiri, and C. Godart. Discovering Workflow Transactional Behavior from Event-Based Log. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 3–18, 2004.
23. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 412–416. IEEE Computer Society Press, Providence, 2001.
24. K. Havelund and G. Rosu. Monitoring Programs Using Rewriting. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 135–143. IEEE Computer Society Press, Providence, 2001.
25. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, Berlin, 2002.
26. G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley Professional, Reading, MA, 2003.
27. A. W. Holt. Coordination Technology and Petri Nets. In *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, Berlin, 1985.
28. H.W. Schmidt and R.H. Reussner. Generating adapters for concurrent component protocol synchronisation. In *Proceedings of the Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, Enschede, The Netherlands, March 2002. Kluwer Academic Publishers.
29. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language, Version 1.0. W3C Working Draft 17-12-04, 2004.
30. J. Koehler and R. Hauser. Untangling Unstructured Cyclic Flows A Solution Based on Continuations. In *On the Move to Meaningful Internet Systems*

2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004, volume 3290 of *Lecture Notes in Computer Science*, pages 121–138, 2004.
31. M. Koshkina and F. van Breugel. Verification of Business Processes for Web Services. Technical report CS-2003-11, York University, October 2003. Available from: http://www.cs.yorku.ca/techreports/2003/.
32. F. Leymann. Web Services Flow Language, Version 1.0, 2001.
33. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
34. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
35. A. Martens. Analyzing Web Service Based Business Processes. In *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
36. R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.
37. M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems*. Logos, Berlin, 2004.
38. N.A. Mulyar and W.M.P. van der Aalst. Patterns in Colored Petri Nets. BETA Working Paper Series, WP 139, Eindhoven University of Technology, Eindhoven, 2005.
39. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-13, BPMcenter.org, 2005.
40. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Providence, 1977.
41. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
42. N. Russell, W.M.P.van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer-Verlag, Berlin, 2005.
43. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. Accepted for publication in *Proceedings of the 24th International Conference on Conceptual Modeling (ER'2005)*, Springer-Verlag, Berlin, 2005.
44. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
45. S. Thatte. XLANG Web Services for Business Process Design, 2001.
46. S. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, March 2005.
47. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.
48. M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.