Composing Services in SOA: Workflow Design, Usage and Patterns

Matti Koskimies

Abstract

The paradigm shift into Service-Oriented Architectures has intensified efforts in Enterprise Application Integration (EAI). Web services has been widely established as the common denominator within and across enterprise boundaries, paving the way for the SOA goals of loose coupling combined with high interoperability between services. This harmonisation in the form of WSDL definitions for both existing and new services has enabled a more seamless and thorough overall view of the business processes involved.

This seminar paper examines the objectives and process of composing applications exposed as Web services into workflows. As a prerequisite to understanding the philosophical and practical differences in approaches to workflow modelling, it collects and summarizes the standardization efforts for composing Web services. The capabilities of different standards are explored through the use of workflow patterns, a subcategory of design patterns which formalise and abstract recurring scenarios in the orchestration of business processes. Also discussed is the dilemma of several overlapping standards, notoriously referred to as the Web Services Acronym Hell (WSAH).

The remainder of the paper focuses on WS-BPEL (or BPEL for short), a fairly recent standard previously known as BPEL4WS which has gained widespread acceptance in the industry. The approach BPEL takes on modelling aspects such as transactions and exceptions is covered to an extent. Due to its popularity, it is also used in the paper for examples on crafting workflows and workflow design tools and engines discussed rely on BPEL for their process model. Furthermore, the advantages and challenges of deploying BPEL into existing environments are evaluated in terms of business agility (such as the impact on time-to-market and flexibility) and re-engineering efforts. Finally, shortcomings in data manipulation and portability as well as generic problem areas such as model synchronization are examined.
1 Introduction

The concept of workflows or *business processes* can be defined as ‘systems that help organizations to specify, execute, monitor, and coordinate the flow of work cases within a distributed office environment’ [Bul92]. Business process management systems originate in the *office automation* [MCC95] efforts of the 1970s and the groupware research of the 1980s. Part of the process-based systems such as the ill-fated paperless office concept, office automation aimed “to reduce the complexity of the user’s interface
to the [office information] system, control the flow of information, and enhance the overall efficiency of the office”[EIN80] and resulted in Office Information Systems such as Xerox OfficeTalk which claimed it allowed “the flexible manipulation of electronic forms on the display screen of users and helps to coordinate and control the flow of forms between user workstations”[EIB82]. In practice, this meant the system (and its contemporaries) provided electronical means of interoffice correspondence, mimicking what was already being done on paper. However, in reality OISs were far too rigid and not really designed with support for the inherent dynamic processes of the typical business environment in mind. Anecdotal evidence actually suggests that many of the companies which adopted such systems actually experienced a diminished efficiency. This was attributed to the fact that much of the productivity was due to creative breaches of protocol and procedures, which would no longer be possible with an Office Information System in place.

With the 1980s and 1990s, the focus of research shifted to more generic improvements in cooperative work, leading to groupware applications which would e.g. facilitate shared information spaces or shared editing [MCC95]. In the early 1990s, there was a surge of interest in business process reengineering (BPR), the mission of which was to improve on specific aspects of intra-enterprise processes through analysis and subsequently optimisation and automation. Restricting techniques to processes internal to a single organisation was for long a necessity due to the industry’s failure to deliver “pervasive standards in process-flow infrastructure”, contrary to other areas such as message queues and object transaction management.” [DHL01]

Modern workflow systems are often referred to more broadly as Business Process Management (BPM) systems and incorporate a broader scope than OISs or BPR methodologies, often being an integral part of OSS/BSS\(^1\) or Order Management systems. Contemporary business processes are used to orchestrate complex business service interactions and thus facilitate Enterprise Application Integration (EAI), complementing Service Oriented Architectures (SOA) by utilising the loose coupling and high interoperability of SOA-compliant components. Typically, more recent workflow languages and standards tend to be based on XML and often rely heavily on Web Services as the unifying technology for interacting with business services. Indeed, it can be argued that it is the widespread adoption of Web Services as the common platform for information interchange and service invocation which has enabled EAI at such a large scale. The composition of services can be internal to an organization, automating the routine flow between various independent, native components required to achieve composite tasks, or inter-organisational, utilising services provided by external parties in addition to local services; although as far as the workflow design and execution is concerned, whether the service is local or not may not account for any relevant differences. Table 1 shows a summary of the development of workflow management systems in terms of generations. Another way of dividing the evolution of workflows is given in section 2.1.

Although there is an undeniable trend towards generic, domain-agnostic business process languages, there are also many domains and scenarios where the cost and effort of re-engineering existing components towards e.g. Web Services compliance is excessive, at least when contrasted with short-term benefits. Consequently, many domain-specific workflow solutions still exist (Figure 2 shows an example workflow of a business process tool used in the telecommunications industry); although having

---

\(^1\)Operations and Business Support System
<table>
<thead>
<tr>
<th>Generation</th>
<th>Major characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>First (1970s–1980s)</td>
<td>application specific</td>
</tr>
<tr>
<td></td>
<td>• workflow capabilities expressed in particular applications (e.g. image, document management)</td>
</tr>
<tr>
<td></td>
<td>• hardcoded process definitions</td>
</tr>
<tr>
<td></td>
<td>• closed and proprietary</td>
</tr>
<tr>
<td>Second (1980s–1990s)</td>
<td>factored application</td>
</tr>
<tr>
<td></td>
<td>• workflow capabilities factored out from application domain</td>
</tr>
<tr>
<td></td>
<td>• workflow as a separate application</td>
</tr>
<tr>
<td></td>
<td>• limited selection of 3rd-party tools</td>
</tr>
<tr>
<td></td>
<td>• process definitions tailorable through script language</td>
</tr>
<tr>
<td>Third (early to mid-1990s)</td>
<td>tailorable service</td>
</tr>
<tr>
<td></td>
<td>• generic workflow services accessible to other applications through APIs</td>
</tr>
<tr>
<td></td>
<td>• open, standards-based architecture</td>
</tr>
<tr>
<td></td>
<td>• full integration of 3rd-party tools</td>
</tr>
<tr>
<td></td>
<td>• tailorable through GUIs</td>
</tr>
<tr>
<td></td>
<td>• proprietary workflow interfaces and interchange formats</td>
</tr>
<tr>
<td>Fourth (late 1990s until now)</td>
<td>embedded enabler</td>
</tr>
<tr>
<td></td>
<td>• workflow services fully integrated with other middleware services (email, desktop management, directory)</td>
</tr>
<tr>
<td></td>
<td>• standardized interfaces and interchange formats</td>
</tr>
<tr>
<td></td>
<td>• “workflow-enabled” applications</td>
</tr>
<tr>
<td></td>
<td>• ubiquitous but invisible</td>
</tr>
</tbody>
</table>

Table 1: Generations of Workflow Technology [AbS94]. The year ranges are estimates of this author and not included in the source material.
been designed with the constraints and special scenarios of the particular domain in mind, they are arguably also better-equipped to handle them. However, this comes at the expense of interoperability with external components.

Figure 2: Example workflow from a commercial business process system. The tool is used to combine provisioning services of different vendors.

2 Standardization efforts

2.1 The evolution of workflow standards

The integration of business applications through widely accepted standards has been a long way coming. Starting in the 1980s with Electronic Data Interchange (EDI), early efforts concentrated on the integration and exchange of data. At this stage, the consolidation of applications was still the result of "homegrown workflow [...] where the systems were monolithic in nature with all information and control flow hard-coded in the applications." [HsK96] The next stage was the object-routing workflow of the late 1980s and early 1990s, offering flexible but application-bound scripting of business processes. No widely recognised standards were in place at this point in time.

Figure 3: WfMC Workflow Reference Model. [Hol95]

According to [Sch99], “the basis for defining workflow standards lies in abstracting a common architecture for workflow applications and formalizing the interfaces of this architecture's components.”
This is what WFMC\textsuperscript{2} set out to do with The Workflow Reference Model [Hol95] (Figure 3), originally published in 1994 and recognised as the groundwork for defining a standard for workflow interoperability [HPS00], thus facilitating the shift towards open architected process managers, the final evolutionary step in [HsK96] comprising “an infrastructure for an enterprise to integrate applications, data, and procedures from disparate systems and organizations.” In spite of its age, the model has proven a highly useful framework even in recent standardization processes as well as in evaluating the conformance and expressiveness of proposed workflow specification languages. It acts as a guideline, identifying both high-level, abstract architectural patterns such as the component structure and interfaces to execution environments, as well as concrete bindings to specific environments such as business object servers or message brokers.

The following areas are standardized by the reference model:

- **Process definition interchange**: enabling the exchange of process definitions between workflow modeling tools and enactment services.

- **Client-application interaction**: allowing workflow participants to interact with and control the process.

- **Application-component interaction**: providing a framework for creating workflow-enabled components as well as adapters for existing implementations.

- **Workflow-application interoperability**: supporting the interaction between multiple independently developed and managed BPM systems.

- **Administration and monitoring**: enabling different business process applications to be administered and their states to be followed and compared consistently.

\textsuperscript{2}Workflow Management Coalition http://www.wfmc.org/
<table>
<thead>
<tr>
<th>Standard</th>
<th>Organization</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow XML (WfXML)</td>
<td>WfMC</td>
<td>Choreography (or similar to it)</td>
</tr>
<tr>
<td>Business Process Execution Language (BPEL)</td>
<td>OASIS</td>
<td>Execution language</td>
</tr>
<tr>
<td>Business Process Modeling Notation (BPMN)</td>
<td>Business Process Management Initiative (BPMI)</td>
<td>Notation Language</td>
</tr>
<tr>
<td>Business Process Modeling Language (BPML)</td>
<td>BPMI</td>
<td>Execution Language</td>
</tr>
<tr>
<td>Business Process Query language (BPQL)</td>
<td>BPMI</td>
<td>Administration and monitoring interface</td>
</tr>
<tr>
<td>Business Process Extension Layer (BPXL)</td>
<td>BPMI</td>
<td>BPEL extension for transactions, human workflow, business rules</td>
</tr>
<tr>
<td>UML Activity Diagrams</td>
<td>OMG</td>
<td>Notation language</td>
</tr>
<tr>
<td>Workflow Reference Model</td>
<td>Workflow Management Coalition (WfMC)</td>
<td>Architecture</td>
</tr>
<tr>
<td>XML Process Definition Language (XPDL)</td>
<td>WfMC</td>
<td>Execution language</td>
</tr>
<tr>
<td>Workflow API (WAPI)</td>
<td>WfMC</td>
<td>Administration and monitoring, human interaction, system interaction</td>
</tr>
<tr>
<td>Business Process Definition Metamodel (BPDM)</td>
<td>OMG</td>
<td>Execution language and/or notation language, as MDA metamodel</td>
</tr>
<tr>
<td>Business Process Runtime Interface (BPRI)</td>
<td>OMG</td>
<td>Administration and monitoring, human interaction, system interaction, as MDA metamodel</td>
</tr>
<tr>
<td>Web Services Choreography Interface (WSCl)</td>
<td>World Wide Web Consortium (W3C)</td>
<td>Choreography</td>
</tr>
<tr>
<td>Web Services Choreography Description Language (WS-CDL)</td>
<td>W3C</td>
<td>Choreography</td>
</tr>
<tr>
<td>Web Services Conversation Language (WSCL)</td>
<td>W3C</td>
<td>Choreography</td>
</tr>
<tr>
<td>XLANG</td>
<td>Microsoft</td>
<td>Execution language</td>
</tr>
<tr>
<td>Web Services Flow Language (WSFL)</td>
<td>IBM 7</td>
<td>Execution language</td>
</tr>
<tr>
<td>Business Process Schema Specification (BPSS)</td>
<td>OASIS</td>
<td>Choreography (and collaboration)</td>
</tr>
</tbody>
</table>

Table 3: Overview of contemporary workflow-related standards
### 2.2 Recent efforts

There is a large amount of recent, more or less actively advocated, maintained and utilised standards and proposals related to workflows, most of which (with the notable exception of notation languages such as those UML) are XML-based; a non-exhaustive list is provided in Table 3. Add to these the number of various prerequisite Web Services or XML standards, and the landscape of SOA becomes increasingly confusing, a scenario commonly identified with the derogative term Web Services Acronym Hell (WSAH) [Aal03]. Competing and conflicting standards are corroding the intended goal of integrating applications and services and providing a common platform for defining, constructing and deploying them. This issue intensifies when it is propagated to the service composition layer of the Web Services architecture. On the other hand, amongst others [Wee05] points out that the situation is not unusual nor necessarily unwanted:

"Competition between standards, specifications, and ideas is not uncommon. To some extent, this is healthy for the industry, allowing competing ideas and approaches to emerge and prove themselves. This has happened in many other areas in the IT industry, such as communication protocols (TCP/IP, ISO OSI, SNA), database models (SQL/Relational, Object/ODMG), and languages (Java, C++, Smalltalk). However, the industry is seeing incremental movement toward consolidation and integration of the Web service standards. Expect a dominant platform to emerge, supported by all of the major vendors. This platform will complement the human-centric HTML/HTTP Internet functions with support for application to application interactions and integration."

In this section, some of the more noteworthy of these standardization activities are presented in order to present the aforementioned competing approaches as a reference for future discussion. Their strengths and drawbacks are also touched on; for more specific details on features, see the discussion on workflow patterns in the following section.

It is worth noting that many standards address a specific area of the workflow management process, while some attempt to provide an all-encompassing technique. In particular, the trend towards Web Services has emphasized the separation between orchestration and choreography of services. The former involves one central process which is at the control of all involved services and coordinates the related operations. This model can be seen in Figure 4 (a). The latter, shown in Figure 4 (b), is a collaborative model featuring several “equal” services which do not rely on a central coordinator. Each service participating in the process is aware of its role, when to interact and with which partners.

![Figure 4: Models of coordination within a Web Services workflow (excerpts from [Jur06])](image-url)
**XPDL** The XPDL format was conceived by the WfMC. It describes a model for the exchange of process definitions. In other words, it is essentially a metalanguage that is often used as a basis for higher level workflow languages. Therefore, XPDL-compliant applications have often been extended with proprietary definitions and aren’t necessarily able to exchange their workflow structures: in other words, they can export to XPDL but are unable to produce anything useful out of XPDL definitions exported using other applications.

**Wf-XML** Another release from the WfMC, Wf-XML’s purpose is to integrate business process engines using a proposed extension to SOAP, ASAP (Asynchronous Service Access Protocol), which allows for the basic remote control (creating, starting and stopping) of asynchronous web services. Wf-XML extends ASAP with specialised, more fine-grained control and monitoring in the event that the synchronous service is invoked on a process engine. Provided extensions include retrieval and introspection of the process diagram, relaying of decomposed (activity) states and modification of the process definitions.[SPG04]

**JDF** Job Definition Format (JDF) is a workflow language defined by Adobe and specifically designed for the print communication industry. While its use in other domains may be questionable, it has achieved a strong foothold in its own area and is widely applied in print workflows. Adobe characterise its three key benefits as follows [JDF05]:

1. The ability to unify the prepress, press and postpress aspects of any printing job, unlike any previous format;
2. The means to bridge the communication gap between production services and Management Information Systems (MIS); and
3. The ability to carry out both of these functions no matter what system architecture is already in place and no matter what tools are being used to complete the job. In short, JDF is extremely versatile and comprehensive.

**BPML** Created by the Business Process Management Initiative (BPMI), the Business Process Modeling Language overlaps substantially with WSCI (Web Service Choreography Interface) [Ark02] and is very similar to BPEL, discussed below, in its Web Services roots as well as making use of many of the same XML standards such as XPath and XSDL. It claims to offer more advanced features, though, such as nested processes and complex compensated transactions. With the merger of BPMI and OMG, BPML seems to have been quietly discontinued as many of its original proponents, including SAP, now support BPEL.

**BPMN** The Business Process Modeling Notation (BPMN) is BPMI’s companion standard to BPML. It focuses on the presentation aspects of business processes, striving to provide a graphical notation which is comprehensible to all participants of the business process modeling activity. It divides elements into four basic categories: flow objects, connecting objects, swimlanes and artifacts. The elements are shown in Figure 5. Unlike BPML, BPMN still seems to be actively advocated by OMG.
The analysis of existing workflow languages and BPM implementations as well as the deriving of workflow patterns from them (see section 2.3) led Wil van der Aalst, Arthur ter Hofstede and Lachlan Aldred to design Yet Another Workflow Language (YAWL) as an attempt to address the shortcomings they had discovered. YAWL is based on and extends on Petri nets, and provides support for, among other things, build-time workflow analysis, persistence, automated form generation, and workflow administration.

The use of UML style diagrams for the purpose of business process modeling has been proposed and examined by several parties [Dum01, Hru98, BDR02, Whi04]. Activity diagrams in particular were originally designed to convey workflow structures. However, they are mostly of interest to the scientific community and there does not appear to be significant interest to incorporate UML diagrams in commercial products. According to [Dum01] they provide certain benefits compared to alternative languages, such as support for signal sending and receiving, waiting states, processing states and the seamless decomposition of an activity specification into subactivities. On the other hand, some constructs lack a precise syntax and semantics, and some important synchronizations such as the Discriminator and a N-out-of-M join (see section 2.3) are not captured at a sufficient granularity.

The Web Services Flow Language (WSFL) [Ley01] was designed by IBM to complement the Web Services framework with Web Services compositions. It depends on existing WS standards such as SOAP, WSDL, XMLP and UDDI, and separates compositions into two categories – the “flow model” usage pattern for defining an executable business process and the “global model” interaction pattern for specifying a business collaboration.

The XLANG standard was created by Microsoft as an extension to WSDL to create a “notation for the specification of message exchange behavior among participating web services” [Tha01]. It augments WSDL by providing a way to define service behaviour. On the other hand, the behaviour may rely on simple WSDL services to provide the necessary functionality for the business process. XLANG’s features include sequential and parallel control flow constructs; long running transactions

Figure 5: BPMN core elements as described on http://www.bpmn.org/.
with compensation; custom correlation of messages; flexible handling of internal and external exceptions; Modular Behavior Description; dynamic service referral; and multi-role contracts.

**BPEL** A joint development of BEA Systems, SAP, Siebel Systems, Microsoft and IBM, the Business Process Execution Language for Web Services [ACD03], originally named as BPEL4WS but now more widely known as WS-BPEL or simply BPEL. has gained a lot of traction since its first publication in 2002 and version 1.1 published in 2003. It builds on the XLANG and WSFL standards described above, incorporating the block structure of the former with the graph-basedness of the latter. BPEL is more thoroughly discussed in section 3.

**BPELJ** IBM and BEA Systems combined BPEL with Java to produce BPELJ [BGK04], an extension to BPEL which allows for the integration of snippets of Java code into the process flow. Many tasks are far simpler to do in Java than BPEL, such as determining variable values through calculation or string manipulation, or constructing and parsing documents.

As mentioned in the beginning of this section, and as this limited overview further emphasizes, there is an abundance of approaches and standards related to every aspect of business process management. In [Hav05], an approach is proposed which uses what the author considers as the best combination of standards available for the execution, design and choreography of workflows, shown in Figure 6. BPEL is chosen for the runtime engine because it is considered to be the best-of-breed and most widely adopted of BPM execution languages. For designing the processes graphically, the author recommends option for BPMN together with an exporter capable of mapping the output to BPEL. WS-CDL is added to the mixture because of the ability of choreography toolkits to generate initial business process definitions according to the communication requirements of participants, defined in WS-CDL. However the author does acknowledge that while the criteria for selection may be valid, there are no current tools that support all three chosen standards, especially WS-CDL.

![Figure 6: Recommended combination of standards. [Hav05]](image-url)
2.3 Workflow patterns

A comprehensive methodology for assessing the expressiveness and features of workflow languages, workflow patterns are largely the result of the gradual research done since 2000 by Wil van der Aalst, Arthur ter Hofstede and others, culminating in the 2003 paper [AHK03]. Also of note is the web site\(^3\) set up by the same group of people as a repository for workflow modeling patterns as well as a general resource site.

The following is a summary of the patterns exposed by [AHK03]. The first five patterns are categorized as basic control flow patterns, the subsequent four are considered advanced branching and synchronization patterns. Patterns 10 and 11 are designated structural patterns, whereas 12–15 involve multiple instances and patterns 16–18 are state-based. The rest of the patterns involve cancellation.

The comparison chart in Table 4 is the result of research comparing workflow language implementations with workflow patterns done by van der Aalst et al. It shows which patterns are supported by the languages (indicated with a + sign), which are not supported (indicated by a - sign) and which are not directly supported (indicated by a +/- sign).

**Pattern 1: Sequence**  A sequence is the most basic pattern, used to model consecutive steps. Each activity in a sequence is not initiated before the previous activity in the same process has completed.

**Pattern 2: Parallel Split**  A parallel split occurs when at a certain point in time, a single thread of control is divided into several threads running in parallel. It can be thought of as the workflow equivalent of the Unix `fork()` command.

**Pattern 3: Synchronization**  The convergence of several parallel activities into a single thread of control, thus synchronizing the threads, constitutes a synchronization pattern.

**Pattern 4: Exclusive Choice**  In the exclusive choice pattern, a condition determines which branch is eligible to be executed. Thus it is similar to the `switch` programming language construct.

**Pattern 5: Simple Merge**  An unsynchronized joining of branches of execution which do not run in parallel.

**Pattern 6: Multi-choice**  Similar to Exclusive Choice, Multi-choice allows for the choosing of eligible branches based on a decision, however with the difference that several branches can be opted to be executed as a result of the condition.

**Pattern 7: Synchronizing Merge**  Synchronizing Merge is a convergence of several paths into one where if several paths are taken, the active threads are synchronized, but if a single path is chosen, the other branches reconverge without synchronization.

---

\(^3\)Workflow Patterns [http://www.workflowpatterns.com/](http://www.workflowpatterns.com/)
Pattern 8: Multi-merge A Multi-merge is required in the unsynchronized convergence of two or more branches. If several branches are activated as a result of the merge, they each execute the same activities.

Pattern 9: Discriminator A point in the path of execution where several incoming branches converge, and execution only continues when the first of them completes is called a Discriminator. At that point the subsequent activity is launched and the remaining branches are allowed to finish, but ignored. Such an approach is often used for optimising distributed database searches where the search is launched simultaneously against several databases and only the first response is utilised.

Pattern 10: Arbitrary Cycles Also known as loops, Arbitrary Cycles enable the (conditioned) repetition of one or more activities.

Pattern 11: Implicit Termination The discontinuation of subprocesses when there is nothing more to be done is called Implicit Termination.

Pattern 12: Multiple Instances Without Synchronization The initiation of Multiple Instances Without Synchronization implies that several threads of execution are created out of a single activity definition, without the need to synchronize them. For example, when ordering multiple books from an online book store, several instances need to be created within the order to account for the impact of each book in the same places, such as stock levels, shipping etc.

Pattern 13: Multiple Instances With a Priori Design Time Knowledge A situation where a known number of activity instances require to be completed before entering the subsequent activity can be handled with the Multiple Instances With a Priori Design Time Knowledge pattern. For example, a decision of a stock broker to purchase stock may require the acceptance of a predetermined number of individuals, meaning several instances of the acceptance activity will be launched.

Pattern 14: Multiple Instances With a Priori Runtime Knowledge Similar to Multiple Instances With a Priori Design Time Knowledge, the Runtime Knowledge version of the pattern acquires the amount of instances at runtime. For example, booking flight tickets to a certain location may involve several flights, but the amount depends on the chosen destination.

Pattern 15: Multiple Instances Without a Priori Runtime Knowledge It is not always possible to determine the amount of activity instances needed even at runtime. For example, a number of deliverables may be expected before the following activity can be instantiated, but the amount of actual deliveries might vary. In this case the process must initiate new instances of the activity until the amount of deliverables has added up. In practice, this means that unlike the situation where the amount of instances are known, new instances can be created even when a part are already being executed or have completed.
**Pattern 16: Deferred Choice**  When there are several candidate branches of which only one is chosen, however unlike Exclusive Choice the actual decision is passed on to the environment, the Deferred Choice pattern is used.

**Pattern 17: Interleaved Parallel Routing**  In the case that activities which do not have a pre-defined order are not allowed to run in parallel, Interleaved Parallel Routing can be used to create a non-overlapping sequence out of the activities at runtime.

**Pattern 18: Milestone**  The instantiation of a particular activity may be possible only under certain circumstances. For instance, the customer of a web service may change an order while it has not yet been dispatched. For this purpose, the Milestone pattern can be used.

**Pattern 19: Cancel Activity**  Sometimes an activity may need to be discarded despite it being enabled e.g. due to time constraints. The Cancel Activity pattern addresses this need.

**Pattern 20: Cancel Case**  In a scenario where it becomes necessary to remove a complete case which is being executed the Cancel Case pattern can be applied. Such an incident could be for example the withdrawal of an application for a job position.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>XPDL</th>
<th>UML</th>
<th>BPEL</th>
<th>XLANG</th>
<th>WSFL</th>
<th>BPML</th>
<th>WSCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Parallel Split</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Synchronization</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Exclusive Choice</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Simple Merge</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Multi Choice</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Synchronizing Merge</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multi Merge</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Discriminator</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Arbitrary Cycles</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Implicit Termination</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MI without Synchronization</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MI with a Prior Design Time Knowledge</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MI with a Prior Runtime Knowledge</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MI without a Prior Runtime Knowledge</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Deferred Choice</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Interleaved Parallel Routing</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Milestone</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cancel Activity</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Cancel Case</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 4: Overview of the workflow pattern compliance of a selection of standards
3 WS-BPEL

3.1 WS-BPEL design and principles

As mentioned in section 2.2, WS-BPEL (from here on referred to as BPEL) constitutes a convergence of the features of the previously competing XML-based XLANG and WSFL specifications from Microsoft and IBM, which both approached the subject from a different perspective, each addressing a particular problem domain. In short, BPEL combines the graph-oriented style of WSFL with the algebraic style of XLANG.

BPEL is also heavily influenced by WSDL – to the degree that a workflow designed with BPEL is itself recursively exposed as a web service. Additionally, BPEL utilises XPath for its variable definitions.

The main authors of BPEL define their design goals in [LRT03] as follows:

1. Web Services should form the fundamental base of BPEL. In particular, any interactions with external services should occur through the use of WSDL. This implies that the business process should interact with Web services described with WSDL, but also that the process itself should be described as a Web service through WSDL.

2. XML should be the only standard used in formatting BPEL. In particular BPEL is not concerned with graphical representations of processes nor does it define any particular design methodology for processes.

3. “BPEL should define a set of Web Service orchestration concepts that are meant to be used in common by both the external (abstract) and internal (executable) views of a business process.” Possible extensions required by these views should be kept to a minimum.

4. Support should be provided for the seamless use of both hierarchical and graph-like control regimes.

5. Limited data manipulation support should be provided; however BPEL is not intended to be a general purpose data manipulation language. More complex operations should be performed by the Web services invoked by the business process.
6. Process instances should be able to identify themselves at the application message level using partner defined identifiers.

7. Basic lifecycle control such as the implicit creation and termination of process instances should be supported. More fine-grained control through e.g. suspend and resume may be added later.

8. Long-running processes should be supported through the use of proven techniques such as compensation actions and scoping to support failure recovery.

9. Web services should be used as the model for process decomposition and assembly. The use of WS-Policy statements would further enrich the modularization of business processes.

10. BPEL should utilise existing or proposed Web services standards whenever available, and new specifications should only be developed if no applicable standard or proposal exists.

3.2 Building a BPEL workflow

In order to gain understanding of the exact structure of BPEL and a more concrete perspective to the process of creating BPEL workflows, this section summarizes the core constructs defined by BPEL and works through a simple example.

There are four sections in the BPEL process description. The three first are:

- The `<variables>` section, where the data variables used by the process are defined
- The `<partnerLinks>` section, where the different parties that interact with the business process are defined
- The `<faultHandlers>` section, where the fault handlers defining the activities that must be performed in response to faults are contained.

The rest of the process consists of steps called activities, which can be either primitive or structure activities. Primitive activities are basic constructs used to perform common tasks. These include:

- `<invoke>` Used to invoke other web services.
- `<receive>` Wait for an incoming message from the client.
- `<reply>` Send a response in a synchronous operation.
- `<assign>` Assign values to variables.
- `<throw>` Throw an exception to indicate a fault in the process.
- `<wait>` Wait for a given amount of time.
- `<terminate>` Terminate the process.

Structured activites combine primitive activities into composite actions to allow more complex algorithms. These include:
Define a set of activities to be executed sequentially.

Define a set of activities to be executed in parallel.

Construct for implementing branches (like the C/C++/Java switch).

Define a loop.

Select a single path from several alternatives.

The rest of this section is dedicated to an example of how a BPEL process description is crafted. The chosen example is the Loan approval process from [ACD03]. An overview of the process can be seen in Figure 8; Algorithm 1 shows the same process written in pseudo-code.

It should be noted that BPEL was not intended to be created manually; instead, it is advisable to use a graphical design tool to create the code, such as ActiveBPEL Designer or Oracle BPEL Process Manager. Section rounds up a few known BPEL supporting applications, most of which offer a graphical modeling environment.

![Loan approval process diagram](http://www.activ ebpel.org/)

Figure 8: Loan approval process diagram (picture courtesy of http://www.activ ebpel.org/)

```
Algorithm 1 Pseudo-code version of Loan approval process

String processLoanRequest(message) {
    if (message.amount < 10000 && "low".equals(assessor.risk(message)))
        return "approved";
    else return approver.approve(message);
}
```

For the sake of clarity, the example BPEL document is built in layers, beginning from the outermost <process> definition and adding logical sections to the description one by one.
The skeleton of the BPEL description is shown in Algorithm 2. The `<process>` section may only contain a single activity, which is usually a container. Namespaces added to the process element are not of relevance to the example. The `suppressJoinFailure` attribute, however, indicates that no exception will be generated in case a merge of two threads of execution fails.

**Algorithm 2 Loan approval BPEL process skeleton**

```xml
<process name="loanApprovalProcess" suppressJoinFailure="yes"

targetNamespace="http://acme.com/loanprocessing"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
 xmlns:lns="http://loans.org/wsdl/loan-approval"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<flow>
</flow>
</process>
```

The following step involves defining what kind of messages the process should receive and what response it should send. The attribute definition `createInstance="yes"` has the effect that when an incoming message is received, a new business process instance is created. As mentioned before, the `<partnerLink>` attribute references the web service which the process is interacting with and the `<portType>` attribute specifies the operation to be executed. Algorithm 3 shows the inserted section in bold.

**Algorithm 3 Adding `<receive>` and `<reply>` definitions to the process.**

```xml
<receive createInstance="yes" name="receive1" operation="approve"
 partnerLink="customer" portType="apns:loanApprovalPT"
 variable="request">
</receive>
<reply name="reply" operation="approve" partnerLink="customer"
 portType="apns:loanApprovalPT" variable="approval">
</reply>
</flow>
</process>
```

Next, required variables are defined as shown in Algorithm 4. They include the original message received from the client, the message required for risk assessment (discussed later), the approval message sent back to the client and a message for relaying errors. In the following excerpts, the entire BPEL message is not reproduced; instead, only the added section is shown.
Algorithm 4 Adding variables.

```xml
<variables>
  <variable messageType="lns:creditInformationMessage" name="request"/>
  <variable messageType="lns:riskAssessmentMessage" name="risk"/>
  <variable messageType="lns:approvalMessage" name="approval"/>
  <variable messageType="lns:errorMessage" name="error"/>
</variables>
```

The other partner services include the assessor which makes the risk calculation for the loan, and the approver, whose decision is required for loans which have a risk level too high to be approved instantly. The interaction with partners is defined using the `<invoke>` element, as described in Algorithm 5.

Algorithm 5 The operations required to the external Assessor and Approver services

```xml
<invoke inputVariable="request" name="InvokeAssessor" operation="check" outputVariable="risk" partnerLink="assessor" portType="asns:riskAssessmentPT">
</invoke>
<invoke inputVariable="request" name="InvokeApprover" operation="approve" outputVariable="approval" partnerLink="approver" portType="apns:loanApprovalPT">
</invoke>
```

Dependencies between activities are imposed using the `<links>` definition, as detailed in Algorithm 6. The links are identifiers for connectors used to link two activities together.

Algorithm 6 Defining links.

```xml
<link name="receive-to-assess"/>
<link name="receive-to-approval"/>
<link name="assess-to-setMessage"/>
<link name="assess-to-approval"/>
<link name="setMessage-to-reply"/>
<link name="approval-to-reply"/>
```

Variable values are assigned in the manner described in Algorithm 7. In this particular instance, the approval variable is given the value ‘yes’ if the assessor evaluates the loan as low-risk.

Algorithm 7 Assignment of values.

```xml
<assign>
  <target linkName="assess-to-setMessage"/>
  <source linkName="setMessage-to-reply"/>
  <copy>
    <from expression=""yes""/>
    <to part="accept" variable="approval"/>
  </copy>
</assign>
```

The receive-to-assess and receive-to-approval links are used in the `<receive>` activity. A link will only be followed if allowed by the transition conditions. In Algorithm 8, the added `<source>` definition is shown in the previously inserted `<receive>` section in bold. The transition conditions specify that
if the request's amount is less than $10000, the receive-to-assess link is followed; otherwise, the receive-to-approval link is chosen. There are equivalent <target> sections for these two links, forming the (in this case conditional) bindings between activities necessary to build a flow. One of the targets, receive-to-assess, is shown in Algorithm 9 in bold.

Algorithm 8 Adding <source> definitions.

```xml
<receive createInstance="yes" name="receive1" operation="approve"
  partnerLink="customer" portType="apns:loanServicePT"
  variable="request">
  <source linkName="receive-to-assess"
    transitionCondition= "bpws:getVariableData('request','amount') < 10000"/>
  <source linkName="receive-to-approval"
    transitionCondition= "bpws:getVariableData('request','amount')
    &gt;=10000"/>
</receive>
```

Algorithm 9 Target of link within an <invoke> statement.

```xml
<invoke inputVariable="request" name="invokeAssessor"
  operation="check" outputVariable="riskAssessment"
  partnerLink="assessor" portType="asns:riskAssessmentPT">
  <target linkName="receive-to-assess"/>
</invoke>
```

3.3 Applications supporting BPEL

A large amount of business applications have added BPEL support in recent times. Several web sites provide a comprehensive listing of BPEL compliant implementations\(^4\), however for the sake of providing an overview a selection of tools is presented in this section.

**Agila BPEL**  Part of the open source Apache product family, Agila is still at a very early stage. It does not include a graphical editor for creating BPEL XML documents.

**ActiveBPEL**  Developed and maintained by Active Endpoints, Inc., ActiveBPEL(TM)is among the most stable and mature open source BPEL engines. It purports to provide complete support for the BPEL4WS 1.1 specification, industrial strength features such as deployment packaging, process persistence and event notifications, and active development both by Active Endpoints, Inc. as well as the open source community. The engine is complemented by a free graphical modeling tool named ActiveBPEL(TM) Designer (shown in Figure 9 (b)), only available for the Windows® operating system.

**IBM WebSphere Process Server**  Part of WebSphere Application Server, the flagship product in IBM's WebSphere range, WebSphere Process Server extends WebSphere Enterprise Service Bus and provides a runtime engine for the wide variety of business-driven service components. The companion IDE, WebSphere Integration Developer, can be used to create BPEL models, or they can be imported from a business model that has been created in WebSphere Business Modeler.

**JBoss jBPM**  An open source tool which supports both its own language, jBPL, as well as BPEL, JBoss’s jBPM also offers a graphical design tool called jBPM Process Designer (as seen in Figure 9 (a)).

**Microsoft BizTalk Server**  Formerly based on XLANG, BizTalk has supported BPEL importing and exporting since its 2004 revision. Microsoft provides a graphical workflow editor called Orchestration Designer (shown in Figure 9 (c)).

**Oracle BPEL Process Manager**  Part of the large Oracle Fusion Middleware family of products, Oracle BPEL Process Manager provides an end-to-end solution for designing, deploying and managing BPEL workflows. The JDeveloper BPEL Designer tool is provided for graphical modeling.

![Screenshots of graphical tools](image)

(a) JBoss jBPM Process Designer  
(b) Active Endpoints ActiveBPEL Designer  
(c) Microsoft Orchestration Designer  
(d) Oracle JDeveloper BPEL Designer

**Figure 9:** Screenshots from graphical tools for designing workflows. The JBoss jBPM tool works within the Eclipse IDE suite. Microsoft’s Orchestration Designer is integrated into the Visual Studio environment.

### 3.4 Motivating the re-engineering of legacy systems into BPEL

BPEL’s Web Services centric approach to interfacing with services external to the business process means that there is a fundamental prerequisite in deploying a BPEL-based BPM system into an environment which is comprised of a large amount of existing application infrastructure, namely converting or encapsulating the legacy components which form that infrastructure into SOA compliant Web Ser-
vices. In turn, the driving force in converting or encapsulating legacy systems to SOA components are the business requirements which the IT industry is concerned about. These include [Käh06]:

- Reducing Operational Expenditures (OPEX)
- Optimizing Capital Expenditures (CAPEX)
- Increasing efficiency
- Gaining competitiveness
- Decreasing time-to-market
- Added business flexibility
- Increased integration requirements
- The need for multiple channels
- Continuous technology changes

It is argued in [Käh06] that a SOA-based model increases business agility, which is seen as a key element for business success in today's companies and therefore the most imperative motivation for assigning resources to the adoption of SOA.

3.5 Shortcomings of BPEL

In spite of BPEL's apparent success in becoming the de facto language for modelling business processes, some shortcomings have been identified both in regard to scientific criteria as well as business purposes.

In [Cha05], David Chappell suggests that BPEL is receiving undeserved attention in areas where its use may not be warranted. He argues that while BPEL may be useful for describing business protocols, it has a number of limitations if used for the actual execution of a business process:

- Commonly, BPEL processes require access to local objects written in Java, C# or some other modern programming language, because BPEL cannot (and is not intended to) provide all the functionality required by a typical business process. However, BPEL does not define any standard way of accessing such objects.

- BPEL does not present a standard approach for dealing with services which do not expose a Web Services interface.

- While BPEL does provide the means for processing XML documents, no such support exists for other common data sources, especially relational databases.

- No standard mechanisms are available in BPEL for defining activities which require human interaction.
While Chappell acknowledges that BPEL was never intended to specify such features, he argues that the implications of such an approach defy perhaps the most fundamental advantage of the standard, portability. If such commonly required features are not provided by the standard, they will be provided by product-specific extensions which by definition are not portable. And, [Cha05] concludes, “given this lack of portability, it’s reasonable to ask why a BPEL-based tool for defining business processes is better than one using a proprietary language. Does the language offer other benefits for creating executable processes?”

In [ADH05], BPEL is criticized for its difficult usage: its XML constructs are verbose as well as diverse, making the resulting representation hard to read and the right constructs for the particular scenario difficult to select. Moreover, the graphical tools designed for BPEL are accused of following its model too directly, placing it closer to classical programming languages than more user-friendly workflow management systems. This point of view seems to be in contrast with the opinion raised above by Chappell that BPEL does not provide enough functionality and therefore requires extending it with traditional programming languages. Further conflict is provided by the assertion that “organizations do not need to agree on a common execution language” as “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”. In the same spirit, the need to harmonize process languages between interacting partners is questioned due to the fact that the implementation technique of the business process is not visible to the outside. However, [ADH05] note that the use of BPEL in a Web Services environment may still be lucrative due to its incorporation of many features which are especially designed for use in the domain.

In an earlier work by the same authors, [WAD03], BPEL is more specifically reviewed in terms of its compliance with workflow patterns, and the conclusion is made that BPEL does not support the invocation of sub-processes and therefore cannot be used to model the Multi-merge pattern, nor does it support arbitrary cycles. An apparent earlier version of the analysis, [WAD02], also points out that the semantics of BPEL are not always clear.

A specific challenge of workflows in general is presented in [Kon06]. Business applications undergo constant changes on two fronts: on one hand, business processes, tasks, activities and responsibilities are subject to evolution and customization by the business manager. On the other hand, developers add new features and migrate to new technologies and apply updates and fixes. As a result, the associations between the workflow process model describing the business perspective and the information system process model describing the technical perspective are lost. [Kon06] proposes a framework which assists on the synchronization of business flows and source code through the use of domain models, heuristics identification, and the establishing of dependencies between the business flow domain model and the source code domain model.

4 Summary

This paper has examined the emergence and significance of workflows as a paradigm for the integration and composition of business services. The concept and origins of workflows were introduced, followed by a description of the current state and trends in business process management. The significance of the
Workflow Reference Model to the standardization process of workflows was highlighted, and the large amount of current standardization efforts were explored, with the notion that while interoperability is hampered by the diversity of standards, on the other hand it facilitates progress through comparison and allows for the selection of an ideal combination of approaches. Furthermore, workflow patterns were introduced as a means of comparing and analysing the features and expressiveness of workflows in an agnostic manner.

In the remainder of the paper, the WS-BPEL OASIS standard was described and analyzed. Its design principles were discussed and an example BPEL process was constructed one layer at a time to demonstrate the building blocks of BPEL and to illustrate the modeling techniques involved. A selection of BPM systems supporting BPEL was also presented as an overview.

The business factors affecting the migration to BPEL based business process systems were assessed and it was suggested that the critical issue is that of reengineering existing applications towards Web Services, which in turn should be motivated by the increase in business agility. Finally, some of the criticism towards the attention BPEL is receiving in areas where it may not be the ideal choice was examined, along with the weaknesses perceived in its ease of use, flexibility and semantics. In conclusion, an ongoing research challenge in the form of model synchronization was mentioned in conjunction with proposed methods for dealing with the issue.

References

[Aal03] van der Aalst, W., Don’t go with the flow: Web services composition standards exposed. IEEE Intelligent Systems, 18, 1, 2003, pp. 72–76.


Appendix: BPEL Process Description for Loan Approval Process

```
-- </--
BPEL Process Definition
Edited using ActivitiBPEL Designer version 8.9.0 (http://www.active-endpoints.com)

--<process name="loanApprovalProcess" suppressJoinFailure="yes"

targetNamespace="http://acme.com/loanprocessing"/>

--<partnerLinks/>

--<partnerLink name="myRole" name="customer" partnerLinkType="ins:loanApprovalLinkType"></partnerLink>

--<partnerLink name="assessor" partnerLinkType="ins:riskAssessmentLinkType" partnerRole="assessor"></partnerLink>

--<variables/>

--<variable name="loaddef:creditInformationMessage" name="request"></variable>

--<variable name="loaddef:loanRequestErrorMessage" name="error"></variable>

--<catch faultName="apsn:loanProcessFault" faultVariable="error">

--<reply faultName="apsn:loanProcessFault" operation="approve" partnerLink="customer" portType="apsn:loanApprovalPT" variable="request"></reply>

</catch>

--<faultHandlers/>

--<flow/>

--<link name="receive-to-approval"></link>

--<link name="receive-to-assess"></link>

--<link name="approval-to-reply"></link>

--<link name="setMessage-to-reply"></link>

--receive createInstance="yes" name="receive1" operation="approve" partnerLink="customer" portType="apsn:loanApprovalPT" variable="request">

--source linkName="receive-to-approval" transitionCondition="bpws:getVariableData('request', 'amount') > 100000"/>

--source linkName="receive-to-assess" transitionCondition="bpws:getVariableData('request', 'amount') > 100000"/>

</receive>

--invoke inputVariable="request" name="invokeApprover" operation="approve" outputVariable="approvalInfo" partnerLink="approver" portType="apsn:loanApprovalPT">

--target linkName="receive-to-approval"/>

--target linkName="assess-to-setMessage"/>

</invoke>

--invoke inputVariable="request" name="invokeAssessor" operation="check" outputVariable="riskAssessment" partnerLink="assessor" portType="apsn:riskAssessmentPT">

--source linkName="receive-to-assess" transitionCondition="bpws:getVariableData('riskAssessment', 'risk') == 'low'"/>

--source linkName="assess-to-setMessage" transitionCondition="bpws:getVariableData('riskAssessment', 'risk') == 'low'"/>

</invoke>

--reply name="reply" operation="approve" partnerLink="customer" portType="apsn:loanApprovalPT" variable="approvalInfo">

--target linkName="approve-to-reply"/>

--target linkName="setMessage-to-reply"/>

</reply>

--assign name="assign"

--from expression="approved"/>

--to part="accept" variable="approvalInfo"/>

</assign>

</flow>

</process>

```