Business processes for Web Services: Principles and applications

R. Khalaf
A. Keller
F. Leymann

The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) is an XML-based language for defining business processes that provides an interoperable, portable language for both abstract and executable processes and that was designed from the beginning to operate in the heterogeneity and dynamism that is commonplace in information technology today. BPEL builds on the layers of flexibility provided by the Web Services stack, and especially by XML. In this paper, we provide a brief introduction to BPEL with emphasis on architectural drivers and basic concepts. Then we survey ongoing BPEL work in several application areas: adding quality of service to BPEL, extending BPEL to activities involving humans, BPEL for grid computing, and BPEL for autonomic computing.

INTRODUCTION

Workflow computing aims to automate business processes by encoding them in a format that can be processed in a workflow management system (WFMS).[1,2] A workflow consists of *activities* that perform actions and a *flow of control* that governs the ordering of these activities. Present day WFMSes[3] usually run workflows that are defined in proprietary formats and thus are difficult to share.

Service-oriented architecture (SOA),[4] which has recently emerged on the computing scene, is based on the idea of providing application functions as services offered on the Internet (or an intranet), in an intrinsically distributed, heterogeneous, and very dynamic environment, in which boundaries of both systems and organizations are crossed. The most common instantiation of SOA is based on the Web Services framework.[4] The Web Services framework consists of a set of XML standards and specifications for describing these services, for exchanging and managing their endpoints, and for enforcing the associated quality of service (QoS) requirements. Service descriptions, around which a large part of the framework revolves, are defined by using the Web Services Description Language (WSDL). A WSDL definition contains the description of the service function and the mechanism for interacting with that service. The definition consists of an interface (`portType`), the binding of that interface to

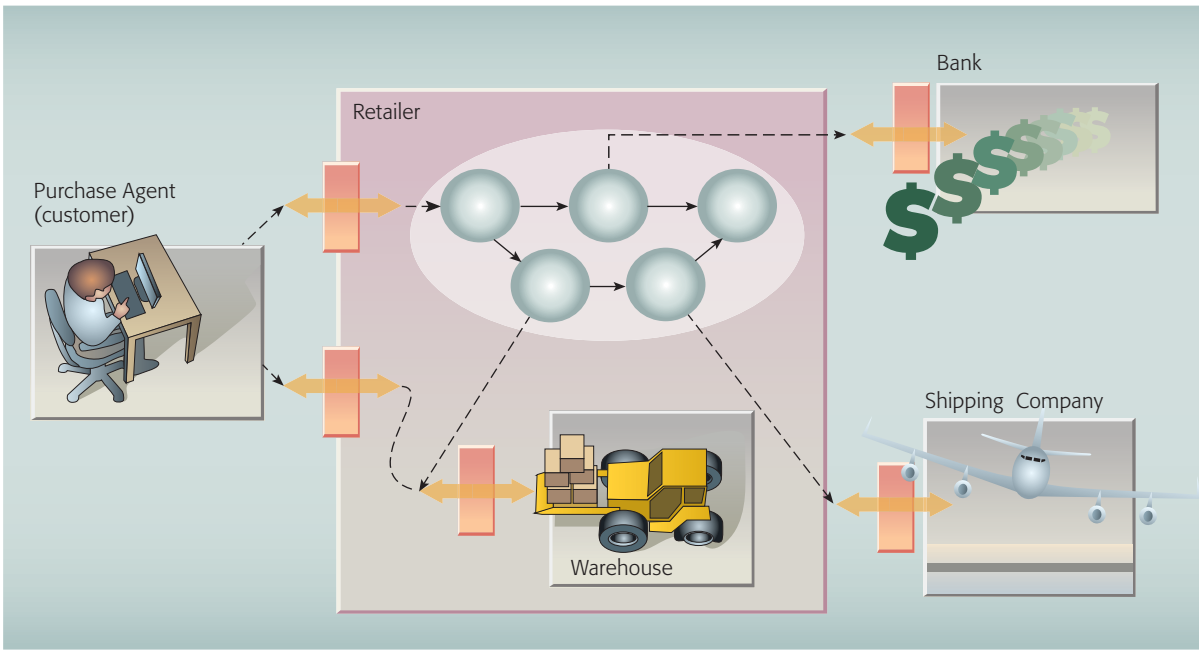

Figure 1
A business process implemented as a composition of services

a particular protocol (binding), and a service element that specifies an endpoint (port) for a particular `portType` over a specific binding.

Once an organization's core functions are modeled as services, the challenge of application integration becomes one of service composition. A composition of services can be modeled as a workflow that interacts with offered services. These workflows may be owned by the vendor of services, by consumers of services, or even by third-party vendors specializing in process modeling. ***Figure 1*** illustrates a business process implemented as a composition of services. The dotted lines represent message exchanges, whereas the directed graph at the center of the figure represents the workflow of the business process. In this example a purchasing service offered by a retailer is implemented as a business process that involves both in-house services, such as checking out an item from the company-owned warehouse, and vendor services, such as bank and shipping services.

Several initial efforts for defining a composition language for Web Services have evolved into the Business Process Execution Language for Web Services (BPEL4WS or BPEL for short).[5] Much has been written about BPEL, from the full specification to details about its constructs, mathematical modeling, and architectural concepts. In this paper, we provide only a brief overview of the language in order to acquaint the reader with it. Our focus is on its architectural drivers and its usage that goes beyond the traditional applications of workflow.

The area of workflow in general and BPEL in particular is extremely active at this time. A Web search in July 2005 produced links to 18 available BPEL engines, ranging from open-source implementations to feature-rich fully supported products. The volume of published material is such that covering every aspect of BPEL would result in a large collection of sources. In this paper, we focus on topics closest to our own direct experience and expertise, at the same time providing pointers to and high-level descriptions of other work for the interested reader.

The rest of the paper is organized as follows. In the next section, "Business processes in BPEL," we present the motivation behind the creation of BPEL, describe its main constructs, and introduce not only the executable variant of the language, but also the abstract variant, about which much less has been published. In the section "Abstract processes in BPEL," usage patterns of abstract

processes are laid out, followed by examples of applying abstract processes to the retail electronics domain and to process compatibility and search. In "Adding quality of service using WS-policy," we present the addition of QoS capabilities to a process by using Web-Services policy attachments. In the following two sections we discuss two application domains for executable BPEL in which activities interact with more than just Web services, namely, workflows involving direct human interaction and workflows involving grid computing. In "BPEL for autonomic computing," we present a case study involving the use of BPEL in IBM products—in the dynamic provisioning aspect of autonomic computing. The "Conclusion" section contains our final comments.

BUSINESS PROCESSES IN BPEL

BPEL was first released on July 2, 2002, as BPEL4WS V1.0, jointly by BEA Systems, Inc., IBM Corporation, and Microsoft Corporation. BPEL4WS V1.0 merges the flat-graph process definition approach of IBM's Web Services Flow Language (WSFL) with the structural constructs approach of Microsoft's XLANG. In 2003, BPEL4WS V1.1 was released with a set of revisions and an expanded list of authors; it is the version of the specification that was submitted to the Organization for the Advancement of Structured Information Standards (OASIS) standardization committee where committee members are working on producing the standardized version of the language, known as WS-BPEL 2.0. In this section we discuss the architectural drivers behind the BPEL language and the overall structure of a BPEL process and its key capabilities. Unless otherwise specified, our examples use Version 1.1.

BPEL's architectural drivers

Two major concerns in standardizing a business process language are: *portability* and *interoperability*. Portability enables one to standardize certain business processes for particular functions that can be published and used in the same manner by multiple organizations; it ensures that one can define a business process once and run it on any compliant system without rewriting. Some people view BPEL as an export format, referring to it as "the PDF of business processes."

Interoperability, on the other hand, enables two executable business processes, running on different engines at possibly different organizations, to interact with each other. This is ensured by BPEL's layering on top of the Web Services stack. A BPEL process is itself made visible as one or more Web Services (with WSDL `portType` entities) that it offers to its partners. Its interactions with any other components also occur as Web-Services invocations based on the `portType` of that component. This *recursive composition*, where a process is implemented as one or more Web Services and in whose implementation other Web Services are used, enables a BPEL process to leverage the interoperability provided by the lower levels of the Web Services stack, such as WSDL, SOAP, and WS-Addressing. Web Services interoperability and the combined usage of these lower levels is the *raison d'etre* of the Web Services Interoperability Organization (WS-I).

Other characteristics (Reference 4, Chapter 6 and Reference 6) built into the language that support the service-oriented paradigm include the following:

- Flexible integration
- Support for simultaneous stateful conversations with multiple partners
- Life-cycle management
- Recoverability

Flexible integration of BPEL processes is mainly achieved by keeping bindings to physical-partner endpoints and other such deployment-specific information out of the definition of the business process. This allows the binding of a BPEL process to partners to occur at design time, at deployment time, or at runtime (through the *assign* activity or by using underlying layers such as WS-Addressing). Other binding schemes are left to the implementation of a BPEL system and are not restricted by the specification.[6] Instead, interactions with Web Services are based only on their interfaces (WSDL `portTypes`) and thus the same activity in different running instances of the same process can interact with different endpoints over different transport mechanisms.

Multiple instances of a particular BPEL process may run simultaneously in the same engine, each interacting with multiple parties. To support this capability, the language provides a conversation channel for each partner called a `partnerLink`.

BPEL processes have an implied life-cycle model: instances are created and destroyed based on the process model and not explicitly by an invoking party. An instance identification mechanism called *correlation* enables a BPEL engine to route a message coming across a `partnerLink` to the correct instance of a process model.

Recoverability is vital, especially to long-running processes. BPEL's recoverability support consists of advanced rollback capabilities for undoing completed actions and fault handling for correcting failures.

By placing these characteristics at the core of its design, BPEL fits powerfully with the SOA paradigm. One can even think of BPEL's approach to partner interactions as a programming model for SOA applications regardless of the language or mechanisms used for service implementation.

Language concepts and structure

BPEL provides two main process usage patterns: *executable processes*, whose business logic can be run by a WFMS, and *abstract processes* that describe behavior and may omit certain information of no concern to the process recipient (e.g., sources of data, values of variables used in conditional expressions).

A BPEL process consists of a top-level process element that can contain: variables, event handlers, fault handlers, compensation handlers, partner links, and a single (complex structured) activity. Variables are typed containers that hold data. They may be typed by using WSDL messages, XML Schema types, or XML Schema elements. The handlers provide advanced capabilities for event handling and error recovery that we will discuss briefly later. The `partnerLinks` define the connections of the process to the outside world: they are named instances of typed connectors that define the `portType` that the process offers to a partner or the `portType` that it requires from that partner. In the former case, the process offers certain functionality to the partner that the partner can call as regular Web Services operations (i.e., the process is a service). In the latter case, the process invokes functionality offered by a partner again as regular Web Services operations (i.e., the process is a client). A two-sided `partnerLink` therefore represents a channel over which a two-way, peer-to-peer conversation occurs between the process and the partner. The logic of the business process itself is mainly contained inside the top-level activity. Here is a skeleton of a BPEL process:

■ BPEL is extensible in that it allows domain-specific extensions ■

```
<process>
  <variables>
    <variable name="x"
        messageType="def:purchasedItem"/>
  ...
  </variables>
  <partnerLinks>
     <partnerLink name="inventoryService"
      partnerLinkType="def:inventorySerPLT"
      partnerRole="inventoryProvider"/>
      ...
  </partnerLinks>
  <flow>
  ...
    <invoke partnerLink="inventoryService"
      portType="inventoryPT"
      operation="removeItem"
      inputVariable="x"
      outputVariable=";itemRemovalStatus"/>
  </flow>
</process>
```

BPEL activities can be either structured or simple. The language also provides conditional (directed) control links. If `transitionCondition` is `true` for a link from activity A to activity B, then A must be completed before B can be started. An activity that is the target of multiple links has a `joinCondition` (default is `or`) that determines when it can run, based on the status of its incoming links. Control constructs include sequential order using the `sequence` activity, parallelism using the `flow` activity—which is the only activity in which links are allowed, nondeterministic choice using the `pick` activity, the familiar if-then-else using the `switch` activity, a looping activity using `while`, and a `scope` activity for scoping variables and handlers. ***Figure 2*** shows an example of BPEL control constructs: a `while` activity containing a `flow` activity. Inside the `flow` activity are three activities, one of which is a

sequence. The sequence has two activities, the lower one also depending on the top-left activity through a control link.

The simple activities have predefined functions: invoking Web Services (invoke), receiving a reply to Web Services invocations (receive and reply), throwing faults (throw), ending a process (terminate), waiting (wait), and so on. All activities involved in messaging must refer to the relevant partnerLink, operation and portType. A process with a receive and a reply activity referring to the same operation exposes that operation on its WSDL. The invoke activity, however, refers to the operation offered by the WSDL of the partner. Finally, data can be manipulated using the assign activity, which allows copying parts of one variable into another, as well as copying between endpoint references and copying of parts of variables. Below is an example of some simple BPEL activities.

```
<!-- invoke a partner's operation -->
<invoke partner="..." portType="..." operation="..."
        inputVariable="..." [outputVariable="..."]/>
<!-- copying data between variables -->
<assign>
    <copy>
       <from variable="..."/> <to variable="..."/>
    </copy> ...
</assign>
```

Data is handled in BPEL by using expression languages. XPath 1.0 is the default and the only one the specification addresses. BPEL supports its own XPath functions for obtaining the value of a variable or the status of a link.

The life cycle of a process starts with the creation of an instance when the system receives a message that can be consumed by a receive activity and whose createInstance attribute has the value yes. The process starts by activating its top-level activity. The process terminates when the top-level activity is completed, when the process throws a fault for which a handler is not found, or when a terminate activity runs.

BPEL correlation is used to maintain conversations with a particular instance. Correlation enables one to refer to specific parts of different messages aliased to named properties. The interaction activities of a process can set its correlation values when a message is received or is about to be sent. A partner sending a message to a running instance should use values that match the values of the correlation sets in the targeted instance. Although this is not the most natural mechanism in middleware-managed routing, it is the common case in business applications where messages are routed based on application information (e.g., social security numbers, confirmation numbers). BPEL does not preempt middleware-managed routing because correlation is optional. In the non-correlation case, a middleware-created token would need to be included in the headers of all messages exchanged with an instance; nonetheless, using correlation enables cases to be supported that would not be possible with middleware-based routing (e.g., a process where a partner needs to send a message to a running instance before any exchange of messages has occurred).

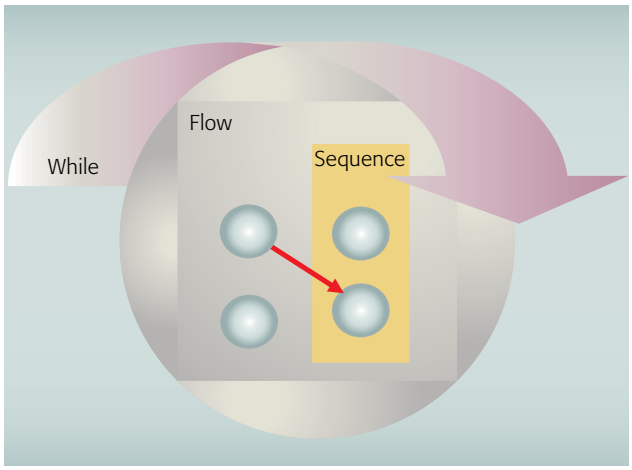

Figure 2
Example of BPEL control constructs

Finally, BPEL provides advanced fault handling, event handling (e.g., messages and alarms), and compensation capabilities on activities grouped in a scope construct. A fault handler attempts to remedy work that failed while running. A compensation handler enables work that was completed successfully to be undone in the case when a fault is thrown elsewhere.

Consider a process that charges the customer and simultaneously removes a purchased item from its own inventory. If the payment is completed but the inventory removal fails (item not available), the

compensation handler of the payment activity is invoked, and it credits the customer's account.

BPEL supports long-running transactions by using compensation. Each invocation activity can be defined as a pair consisting of the actual activity and an associated compensation activity that can undo the former's work when necessary. If a fault occurs, the compensation activities of already completed work within the scope of a failing activity are used to undo the work. Thus, a scope can be seen as a long-running unit of work.

Short-running units of work can be supported by using *atomic scopes*.[1] Activities that are implemented as transactional programs can again be grouped into a scope that has ACID (atomicity, consistency, isolation, and durability) semantics: either all completed transactions within a scope are committed, or all are rolled back. In Reference 7, this concept is described as a BPEL extension.

ABSTRACT PROCESSES IN BPEL

A BPEL abstract process provides a description of a related range of behaviors; one can think of it as representing a set of executable processes. Abstract processes have access to the same range of syntax and semantics as executable BPEL processes. Opaque tokens enable explicit hiding of information, and in some cases, may themselves be omitted.

In BPEL4 V1.0, the only opacity allowed involves variable references on activities that exchange messages and opaque data assignments. In the next version, there will be three types of opaque tokens: activities, expressions, and attributes. It was soon realized that the meaning of these opaque tokens and the restrictions on them vary greatly based on the use case one has in mind. The new approach in BPEL V2.0 is about the *base* and *profiles*. A base defines basic requirements of all abstract BPEL processes. An abstract process profile defines the allowed subset of the syntax of the base, a URL to identify processes belonging to that profile, and the allowable *executable completions*. The executable completions consist of the set of executable processes whose behavior the profile represents. Two profiles are being defined: one for creating process templates that can later be expanded for specific scenarios, and the other for observable message exchanges. The rest of this section focuses on the newer, more flexible approach to abstract processes.

The base consists of the full syntax of executable BPEL, but allows all expressions, activities, and attributes to be opaque. If a syntactic token is mandatory by the XML Schema, then its omission means that it is opaque. The base also mandates that every abstract process must be flagged as such, be schema verifiable, and have at least one valid *basic executable completion*. This is an executable BPEL file created by replacing every opaque token in the abstract process with a corresponding nonopaque token (including those tokens made opaque through the omission shortcut).

A profile addresses a particular usage area of BPEL abstract processes. For example, a templating profile could only allow replacing opaque tokens, whereas a message-exchange behavior profile could allow addition of new activities in arbitrary places as long as they do not interact with the existing partners of the abstract process.

Patterns for using abstract processes

People often relate abstract processes to other processes that are either more general or more detailed. In this section, we present several patterns[8] that help clarify how abstract processes may be used. These patterns will not be included in the specification.

- *Export pattern*—In the export pattern, an abstract process is created from one or more executable or other abstract processes by abstracting (through making opaque or simply deleting) parts that are not relevant to the behavior one wishes to expose. For example, one may use an abstract process to represent common behavior in a set of executables and drop any nonshared behavior. An executable process of a more general business model may need parts tagged as points of variability, and those are made explicitly opaque. In another case, one may teach a business partner the interactions that the partner must follow, in which case the interactions with all other partners are ignored.
- *Import pattern*—In the import pattern, an abstract process is used to create either one or more executables or one or more abstract processes that are refinements of the original process. For example, a user needs to create an implementation of an abstract process provided as a behavioral prescription for complying with a known, domain-specific business function. The abstract process may have been purchased from a consulting firm

as a model of an optimized approach to a problem. The implementation can be created in a series of iterative refinements to the abstract process.

- *Protocol matching pattern I (mirror-image)*—In this case, one constructs the process of a particular partner from a given abstract process that the partner needs to interact with. Then, the import pattern can be used to create an executable artifact from that abstract.
- *Protocol matching pattern II (search)*—This pattern uses one abstract process to search for another process (in a repository of processes, for example) that can perform the same steps as it can. It is easiest to find an exact replica, but that is extremely unlikely as the processes have probably been created by different people.
- *Protocol matching pattern III (compatibility)*—This pattern checks whether several abstract processes can work together.

Using abstract BPEL processes

Abstract BPEL processes seem to be more difficult to understand than executable BPEL processes. This section covers some research work and case studies involving abstract BPEL processes in order to provide a more concrete understanding of their use.

Abstract processes for compatibility and search

The usage patterns for the abstract processes above relate processes to each other. Notions of equivalence and simulation in process models and software artifacts are not new: different ones[9] have been proposed over the years based on the class of problems being addressed. For Web Services and BPEL in particular, there is ongoing work in checking properties between processes.

Using a Petri-net mapping of (relevant parts of) BPEL processes, Martens proves such properties as consistency between processes[10] and uses one process to search for required behavior in a repository.[11] In Reference 10, a syntactic approach for consistency is discarded in preference to one that is based on the behavior of the processes at hand. The main concern here is externally visible behavior from exchanged messages. After converting the BPEL process to a Petri net, a communication graph (c-graph) is created and then used in a consistency-checking algorithm. Intuitively, two processes are consistent for Martens if one can be replaced by the other without requiring changes to the environment in which they interact. Here, the executable process must receive at least the messages that the abstract process can receive, but may accept more because of additional functionality not used by the environment. On the other hand, it cannot send out more messages than the abstract process does because the environment is unable to consume them. This work focuses on a subset of BPEL and makes certain assumptions about queuing of messages that are not generally accepted in the industry. In Reference 12, new notions of observable equivalence for workflows are presented (not focusing on BPEL) with and without different classes of silent actions.

Abstract processes have been used to search for desired functional behavior in a repository. Most Web Services repositories provide WSDL-based, not behavioral, search. Reference 13 presents a search-by-example approach in which the searching party provides a query containing a service's desired behavior. Another approach better suited to searching for concurrent, multipartner communication (Reference 14) publishes operating guidelines for *each partner* of the process instead of a single behavioral definition (abstract process) of the process itself.

■ BPEL can be viewed as an export format, and some refer to it as "the PDF of business processes" ■

Although creating tools and algorithms for checking relations between different BPEL process definitions is a very active research area, the results do not converge yet to any single unified approach.[12, 15–17]

Standard interfaces for electronic commerce

RosettaNet is a consortium dedicated to standardizing interfaces for electronic commerce between supply chain partners. To encode the business interactions required to perform a particular business function, such as processing a purchase order or inquiring about a price, RosettaNet defines Partner Interface Processes (PIPs).[18] A PIP definition consists of a textual description, message Document Type Definitions (DTDs), and QoS requirements (time-outs, security, etc.). PIP messages are packaged, routed, and transported by a RosettaNet Implementation Framework-compliant system.

Work is under way in RosettaNet to determine whether the framework can be extended to existing messaging protocols and infrastructure, and in particular to Web Services. An approach for porting RosettaNet to Web Services is described in Reference 19.

We have used BPEL to represent the business logic in a PIP.[20] A BPEL abstract process encodes exactly the business behavior of one party, as defined in the PIP documents; message exchanges and their ordering are represented by using flow, links, and the messaging activities in BPEL. RosettaNet time-outs and the associated faults are represented using alarm, fault, and event handlers. QoS issues such as reliable message delivery and security are pushed down to the appropriate layers of the Web Services stack. From the abstract process, an executable process may be derived by using simple rules to be executed by the partner.

PIP processes for various behaviors follow certain known patterns, such as the asynchronous two-action PIP or the asynchronous one-action PIP, with clearly defined points of variability; therefore, in Reference 20, we propose the template→specialize→ implement approach to PIP definitions. The first step is the most abstract—"template" processes are defined for each of the patterns with clearly defined points of variability (e.g., a template for all two-action PIPs). The template can be specialized by specifying additional details, resulting in abstract processes for a particular pattern (e.g., a purchase order two-action PIP). Finally, simple completion rules are provided for creating executable processes.

ADDING QUALITY OF SERVICE USING WS-POLICY

The specifications and standards of the Web Services stack are designed to be modular. One can use just the subset necessary for the task; additional functionality can be modularly and noninvasively added at will.

The Web Services Policy Framework (WS-Policy)[4] provides a pluggable mechanism for attaching non-functional requirements to different parts of other Web Services specifications in a declarative manner. Most commonly, such policies are attached to a WSDL definition. In addition to providing clients with functional service requirements, one can now attach QoS requirements either to a `portType`, or just to a particular operation. Domain-specific policy languages currently exist for reliable messaging and security. Although a syntax for distributed transaction policies is presented in Reference 21, it has not been released yet as a formal proposal. A policy attachment can contain references to several different policies (reliability, security, etc.); these may be combined by using Boolean operators and may be tagged as either *required* or *optional*.

Some examples of work that specifically uses WS-Policy attached at the WSDL level are presented in References 22, 23, and 24. In Reference 24 the issues involved in using Web Services policies are discussed and illustrated through the architecture of a prototype that configures policies on a per-interaction basis. Policy-based support is integrated in the Colombo prototype as described in Reference 23. In Reference 24, the authors define the GlueQoS policy language and present a middleware system to support it. GlueQoS is an extension to WS-Policy and is geared especially to cases where policies change during the lifetime of a service. In the standards arena, Web Services Metadata Exchange is being proposed as a specification for handling dynamically changing policies. For example, the policy below indicates the use of reliable messaging with a retransmission interval of 4 seconds and a time-out interval of one hour:

```
<wsp:Policy wsu:Name="tns:RMPolicy" >
    <wsrm:RMAssertion>
        <wsrm:InactivityTimeout
          Milliseconds="3600000" />
        <wsrm:BaseRetransmissionInterval
          Milliseconds="4000" />
    </wsrm:RMAssertion>
       ...
</wsp:Policy>
```

The binding of the service can then refer to the above policy:

```
<wsdl:binding name="ServiceSOAPBinding"
              type="wsdlns:servicePT"
              wsp:PolicyRefs URI="tns:RMPolicy" />
      ...
</wsdl:binding>
```

Once the invocation request is received by the underlying (policy-supporting) messaging middleware, the WSDL of the partner is looked up to determine how and where the message should be

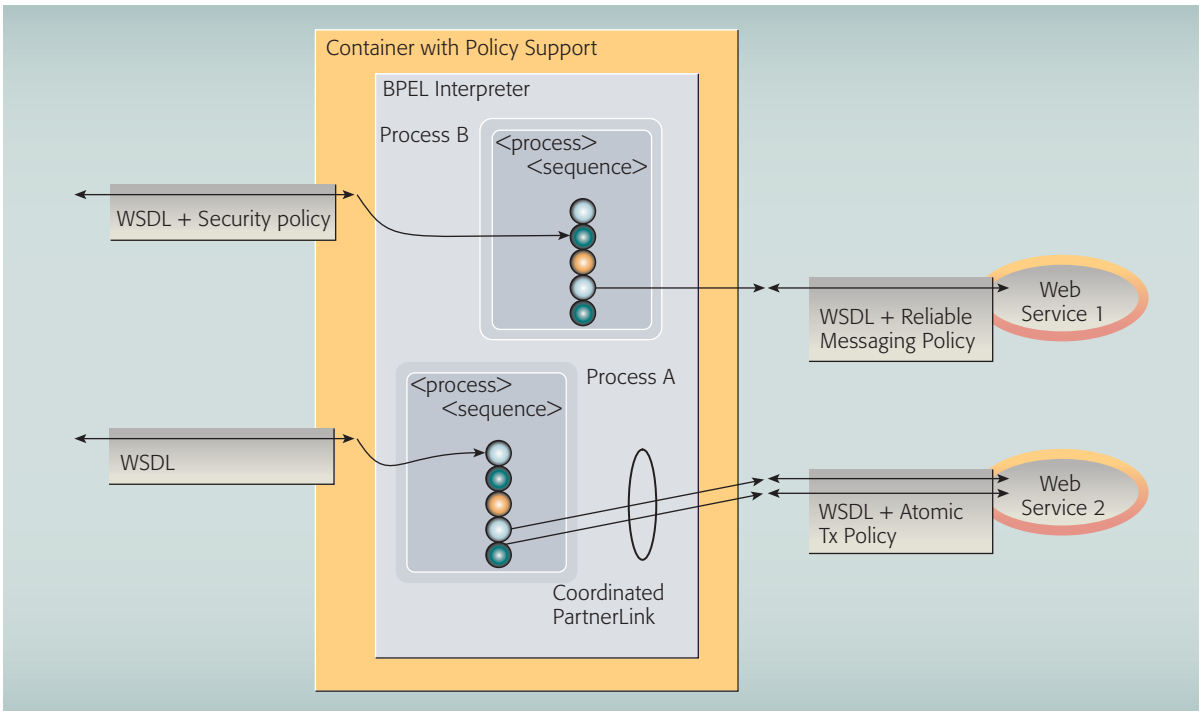

Figure 3
Attaching policies to WSDL definitions and BPEL processes

sent and whether there are any applicable policies. In fact, Colombo uses the policy in WSDL as the default for the particular partner—but the system is set up to support later updates to this policy on a per-interaction basis.

For BPEL services, one can attach policies pertaining to a particular operation, message, or `portType` to the WSDLs that represents the `portTypes` the process offers. The policies that a process requires of its partners can also be attached on the partners' WSDLs. Both can then be handled by policy-supporting Web Services middleware,[23] just as non-BPEL services are handled.

It is, however, often necessary to attach policies directly to BPEL constructs, especially when one needs a finer grained attachment than WSDL provides, that is, a policy to be applied only to a subset of the `invoke` activities of a particular operation. This also applies when one requires a different grouping of attachments than WSDL can provide, such as a policy on all interactions in a scope (possibly with several partners). Attaching policies to BPEL constructs (scopes and `partnerLinks`) is presented in Reference 25, with a focus on combining the functionality of the WS-Transactions specification with BPEL. The model prescribed in Reference 25 is described next.

Transactional policy assertions, using the syntax in Reference 21, can be attached to either scopes or `partnerLinks`. The coordination protocol of a policy on a `partnerLink` is used in all interactions with that `partnerLink` which belong to the same scope. ***Figure 3*** illustrates attaching policies to WSDL definitions and BPEL processes. The figure shows a system with two BPEL processes, A and B, and Web Services that are invoked by these processes. Also shown are examples of attaching policies to WSDL (security, reliable messaging, and atomic-transaction policies) and to BPEL (a WS-AtomicTransactions policy is attached on the `partnerLink` of process A, shown by the coordinated partnerLink pair of arrows).

The partner itself has to be able to support this policy, and its WSDL should confirm that. A scope delineates the transaction, either if it has a coordination policy attached to it (coordinated scope) or if it has to invoke activities to any `partnerLinks` with such policies. The start of the scope begins the

transaction, and the end of the scope signals the end of the transaction; any `invoke` activities in a coordinated scope and any `invoke` activities to a coordinated `partnerLink` in a regular scope occur as part of that transaction. To enforce this, the BPEL engine should be integrated with policy support and with a subsystem handling the specific policy types defined on the process. Work is ongoing to consider advanced uses, such as coordinated `partnerLinks` and scopes in the case of scope nesting, and enabling the process itself to participate in a transaction instead of just initiating it.

Although the attachment of policies for different QoS requirements is neatly modular from an XML point of view, its implications for the underlying middleware are not as clear-cut. For example, a possible clash between the ordering and sequencing constructs of WS-ReliableMessaging and the control constraints of the corresponding BPEL activities would require close coordination between the reliable-messaging subsystem and the BPEL interpreter.[25]

Approaches that do not use WS-Policy attachments for adding QoS to BPEL are found in References 26, 27, and 28. For example, References 26 and 27 use aspect-oriented programming with BPEL, the latter focusing on aspects for QoS. In Reference 28, partner-service endpoints are selected while a process is running so that it can meet global QoS constraints.

WORKING WITH PEOPLE: EXTENDING BPEL TO ACTIVITIES INVOLVING HUMANS

BPEL is an extensible language in which domain-specific extensions may be created, thus avoiding the necessity to incorporate the needed functionality into the base language. The result is the creation of towers on top of the base syntax and semantics, thus maintaining the separation of concerns. The addition of tasks involving humans to a BPEL process can be done with such extensions.

Consider the example of a business process that combines human activities with application calls. The process manages maintenance requests of a multicity maintenance company with several mobile workers making service calls in the area of coverage. The process interacts with the accounting department for updating client billing records and for managing maintenance requests. Upon reaching the target location of the next maintenance request, the maintenance person logs into the system and retrieves the list of remaining service calls for the day, perhaps optimized according to the current location. The list contains, say, three service calls. The maintenance worker selects two of these, logs off, and proceeds to perform these two tasks. Later, the worker logs in again from a location that offers Internet connectivity, updates the database with the status and costs of the tasks performed, and retrieves the list of remaining service calls. By that time the third task is no longer in the to-do list because it has been picked up by another worker.

In BPEL, direct support for human-facing capability has been dropped: everything is a Web service invocation. Of course, one could implement a Web service that forwards the request to middleware that handles the human-facing functionality, including staff resolution (finding the persons who can perform a particular task) and management of a worker's task list, and finally responds to the BPEL process. Even though that works technically, the information about the people and skills needed to perform a task is part of the *business logic*. Using a pure Web Services approach hides from a designer or a user that some activities are to be performed by humans. Additionally, staff resolution and task-list management can often be done by the workflow management system directly.

IBM workflow products already support human-facing activities[30] with the Flow Definition Markup Language (FDML). It was natural to continue that support when the products evolved to BPEL. Therefore, an extension to the `invoke` activity has been included in the IBM WebSphere* Application Server Process Choreographer,[31] an example of which is shown below:

```
<bpel:invoke partnerLink="null"
    portType="abc:maintenancePT"
    operation="performMaintenance"
    inputVariable="ClientInformation"
    outputVariable="statusAndCost">
  <wpc:staff>
  <wpc:potentialOwner><staff:membersOfRole
        role="fieldWorker"/></wpc:potentialOwner>
  <wpc:reader><staff:membersOfRole
        role="manager"/></wpc:reader>
  </wpc:staff>
</bpel:invoke>
```

The staff extension is a subelement of `invoke`. Each staff activity includes the operation and `portType` identifying the work to be done, and instead of a `partnerLink` includes the staff element containing the information about the people that can edit, manage, read, or actually execute this activity. Each gets resolved into one or more userids, relating the activity to actual people. The lookup can extend to entire organizational directories and their hierarchies, taking roles, memberships, and skills into account. When such an activity is triggered, a person must be found to take over the role of `potentialOwner` and perform the requested work.

Upon logging into the system, the user can pull up, in a provided user interface, a list of all the activities (work items) that require attention. By selecting a work item the user becomes its owner (the item is then removed from the lists of all potential owners). The work item owner can obtain the activity's input data, perform its work, and send any resulting messages when done (possibly a fault notification) to the system. Because human-facing tasks are usually far slower than automated tasks, possibly taking several days or longer, the response is sent asynchronously.

In addition to enabling human-facing activities, it is sometimes necessary to allow a human administrator to fix a problem involving a running process, especially a long-running one.[31] By avoiding a complete rollback and restart of the entire process, valuable time and resources are saved. A process-administrator extension in Process Choreographer has been implemented for this purpose. In the standards arena, an extension of BPEL is proposed in Reference 29 toward creating a standard for such human-facing activities.

BPEL FOR GRID COMPUTING

Grid computing[32] enables one to treat a large number of networked computers and other resources (disks, services, etc.) as a single virtual (and much bigger) computer. The application that brought the grid to the masses was the SETI@home project,[33] allowing anyone to aid in the search for aliens by signing up their computer to perform, in its spare time, part of the massive data analysis on radio telescope signals. Placing open standards at its core,[34] the Grid community created an architecture for the Grid that builds on and extends the XML specifications and standards created for Web Services, which is known as the Open Grid Services Architecture (OGSA).

The Grid community, a largely scientific community, has considered a number of workflow systems and languages. Since the OGSA treats Grid components as Grid services with WSDL interfaces, there is momentum towards adopting BPEL as the language of choice. BPEL meets interoperability and openness requirements and is designed to operate in a service-oriented world. Additionally, it enables the scientific community to use any of the many industrial-strength workflow engines, or any of the open-source or free implementations available on the Web. In Reference 35 the authors justify the community's need for a workflow representation

■ One can think of BPEL's approach to partner interactions as a programming model for SOA applications ■

and present a case study, finding BPEL a suitable language for scientific flows using Grid services. To ease process design and make it more accessible to scientists (in this case, computational chemists), they provide domain-specific extensions and shortcuts, such as indexed flows that are expanded into standard BPEL by a preprocessor or style sheet.

The Grid treats computing resources as a special kind of services described by the Web Services Resource Framework (WSRF) specifications.[11] A WS-Resource is a data context required by a Web service in order to operate properly on a particular request. A resource has properties that can be queried at any time and explicit life-cycle management (creation using a factory pattern, destruction, timed destruction). The Implied Resource Pattern (WS-Resource Access Pattern) provides a mechanism to identify that context, usually by using reference properties as part of the endpoint reference (EPR) identifying the resource that the request should manipulate.

In References 36 and 37, a number of best practices are presented for using BPEL in Grid environments, aimed at streamlining BPEL with WSRF. We highlight these below. Overcoming challenges using Grid-specific extensions to BPEL is preferred over

the option of changing BPEL's existing capabilities. Taking advantage of XML extensibility enables the user to remain compliant with the specification, reuse much of the available infrastructure, and run pure BPEL processes in the same engine as the one that runs the extended processes.

First, one must be able to treat a process as a WS-Resource. Recall that BPEL uses correlation and not reference properties for identifying instances. One must be able to use EPRs with reference properties to address a BPEL process instance and for its interactions with its partner (WSRF-compliant) services. This is possible because using WS-Addressing is not disallowed by BPEL. The endpoints that a process offers must also support the implied resource pattern.

The resource life cycle can be explicitly managed, whereas BPEL processes have an implicit life cycle. However, explicit life-cycle capabilities can be easily offered by a BPEL process. The factory pattern (explicit creation) can be implemented indirectly by using a dispatcher or directly by the process through an `assign` that copies its own EPR and uses it to reply to the sender of the creating message. A BPEL event handler can be included in the process to allow explicit destruction of a process instance; an alarm handler can be placed to support timed destruction.

A WS-Resource has resource properties that can be accessed and modified. A BPEL variable can be created whose type is that of the resource property document, and event handlers on the process can handle calls to get, set, and query the resource property. Advanced queries can be handled by the process through delegation to another service.[36]

In Reference 37, Slominski proposes providing services commonly used by grid workflows locally as pseudo-partners, such as the transfer of large amounts of data using GridFTP or Reliable File Transfer. The difference between a pseudo-partner and a regular partner is that invocations to pseudo-partners can be optimized by allowing the system to shortcut to them directly. Handling the large amounts of data used in scientific workflows must be done efficiently. Although pseudo-partners may help in the short term, Slominski proposes looking into using explicit data streams or data links. Other possibilities include passing endpoint references in messages with a pointer to where one can retrieve relevant pieces of the data.[36]

Grid QoS requirements are similar to those for Web services at large and can be provided by using a combination of local BPEL fault and compensation capabilities and complementary specifications for security, reliability, and coordination. Nonfunctional requirements can be attached either to WSDL or to parts of a process as described in the section "Adding quality of service using WS-policy," and executed by middleware supporting the BPEL engine.[23]

The Grid community requires open, standardized process-monitoring capability. However, BPEL views this capability as a system service to be provided optionally by a workflow management system in which a BPEL process runs, and it is not addressed explicitly in the BPEL specification. One option for adding nonproprietary monitoring support[37] is to use WS-Notification, an eventing specification. Another is to provide a BPEL extension for monitoring.[36] However, that would require an open and standardized state model for BPEL processes and their activities, which might lead to conflict because different vendors' state models reflect the different capabilities (and extensions) of their engines, and agreeing on a single model that contains the necessary information for advanced monitoring could be a major obstacle.

BPEL FOR AUTONOMIC COMPUTING

Autonomic computing[38,39] is an initiative spearheaded by IBM that aims at reducing the complexity of managing computer systems by making them self-managing and adaptable to changing conditions. A study shows that operator errors account for the largest fraction of failures of Internet services;[40] hence, properly managing changes is critical to the availability of information technology (IT) services.

An autonomic manager automates a set of management functions and externalizes these functions according to the behavior defined by management interfaces. Examples of autonomic managers are systems management platforms, mid-level managers, service provisioning systems, and management logic embedded in a managed resource, such as the administration console of a Web application server or the performance advisor of a database management system (DBMS). Managed resources expose

management data—such as counters and gauges—to autonomic managers by means of *sensors*. *Effectors*, on the other hand, allow an autonomic manager to modify the behavior of a managed resource by setting its configuration parameters or tuning knobs to specific values. Managed resources can be perceived as WS-Resources with sensors and effectors provided as operations of corresponding services.[11]

In the autonomic computing architecture,[41] the decision process carried out by autonomic managers is represented by a control loop, which consists of four functions: monitor, analyze, plan, and execute. ***Figure 4*** depicts the conceptual architecture of an autonomic manager.

The *monitor* function collects sensor data and organizes them into *symptoms* that need to be analyzed. An example for this function is computing the average response time for a transaction spanning multiple systems by consolidating performance data collected at various points of a distributed system.

The *analyze* function analyzes the observed symptoms and determines the change needed to rectify the problem in conformance with policy rules. This involves, for example, comparing the obtained response time to its allowable range, as specified in a service level agreement (SLA). If the SLA is broken or at risk of being broken, a *change request* is issued for the deployment of additional servers.

The *plan* function provides the mechanisms that construct a *change plan*, a partially ordered set of tasks needed to achieve goals and objectives. The component in charge of creating a change plan for a given change request is the *change manager*. Its purpose is to assign tasks to available resources, according to a variety of cost-based and technical constraints, such as SLAs, administrator policies, and compatibility and collocation requirements for software. In addition to task-to-resource assignments, a change plan comprises deadlines and maximal durations for each activity as well as for the overall change plan. It can be represented as a (BPEL) workflow.

The *execute* function carries out the change by interpreting the change plan. This function is typically implemented by provisioning systems, such as IBM Tivoli Provisioning Manager.[42] An important part of this function is keeping track of the way changes are rolled out on the target systems and feeding this status information back to the change manager.

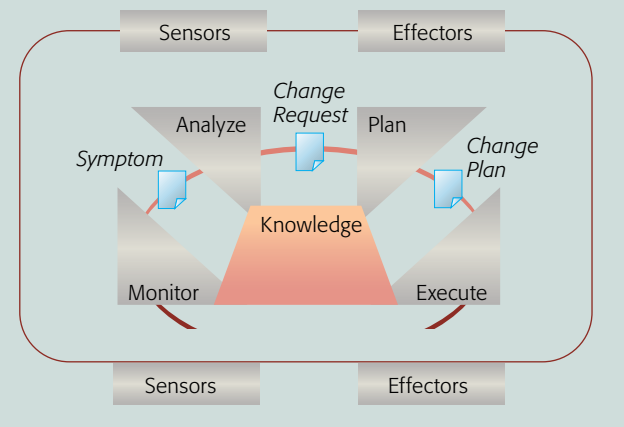

Figure 4
Architecture of an autonomic manager

Note that in addition to interacting with the sensors and effectors of managed resources, an autonomic manager may provide sensors and effectors that expose its functionality to other autonomic managers. Consequently, sensor and effector interfaces enable autonomic managers to be aggregated into hierarchies or peer-to-peer relationships in a manner that is transparent to the managed resources.

We use BPEL for representing change plans in order to seamlessly integrate the planning and execution functions of an autonomic manager and to automate the provisioning of distributed applications and services. Expressing the change plan in BPEL facilitates further manual modification and customization by an administrator, if needed. In the next section we introduce the provisioning of a complex enterprise application as our driving scenario. After that, we will analyze the features of BPEL that make it particularly suitable for representing change plans.

Provisioning the enterprise application

Our case study involves installing and configuring a J2EE**-based enterprise application and its supporting middleware on multiple machines. The middleware includes IBM's HTTP Server, WebSphere Application Server, WebSphere MQ* embedded messaging, DB2* Universal Database* (UDB)

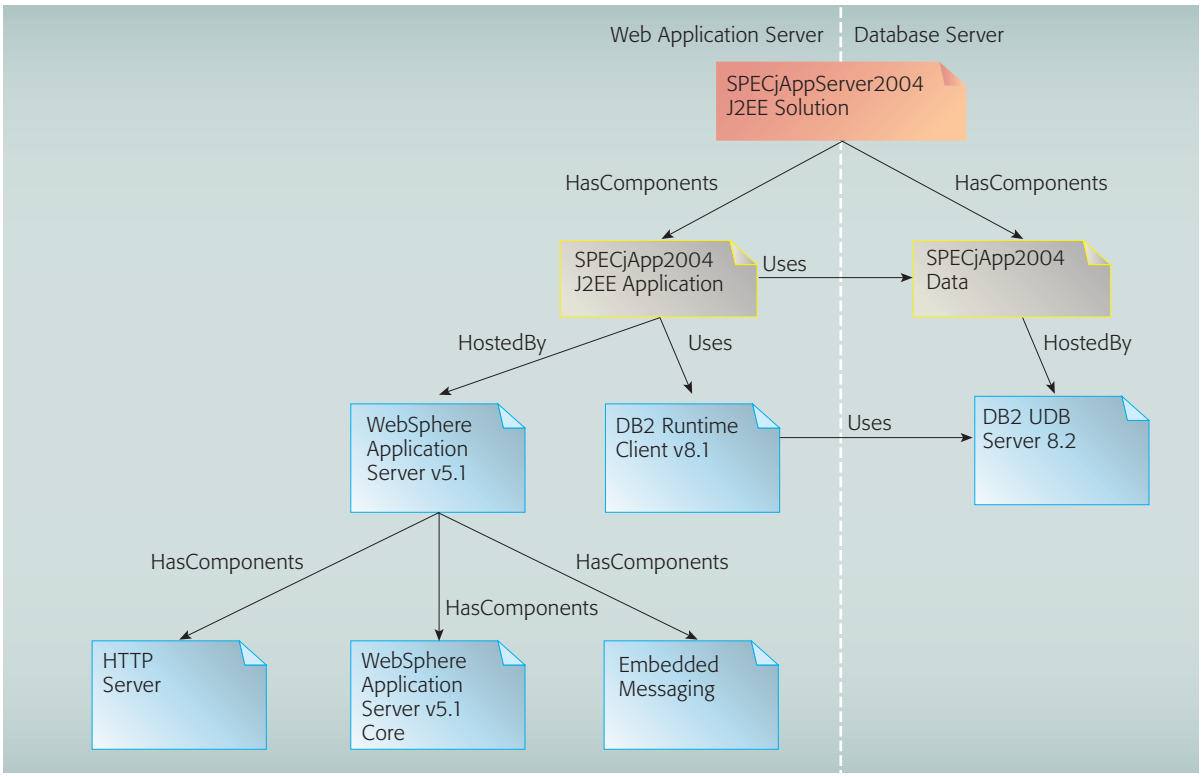

Figure 5
Dependencies between components of a SPECjAppServer solution

server, and DB2 runtime client. The application we use is the SPECjAppServer2004 (Sp04 for short) enterprise-application performance benchmark,[43] a complex multitiered online e-Commerce application that emulates applications found in an automobile manufacturing company and its associated dealerships. Sp04 comprises typical manufacturing, supply chain, and inventory applications that are implemented with Web, EJB**, messaging, and database tiers. We jointly refer to the Sp04 enterprise application, its data, and the underlying middleware as the Sp04 solution.

The Sp04 solution depicted in ***Figure 5*** spans an environment consisting of two systems (separated by the dashed line in the figure): one system hosts the application server along with the Sp04 J2EE application, whereas the second system runs the DBMS that hosts the various types of Sp04 data (catalog, orders, pricing, user data, etc.). One of the many challenges in provisioning such a solution consists in determining the proper order in which its components need to be deployed, installed, started, and configured. For example, the HostedBy dependencies between the components *SPECjAppServer2004 J2EE Application* and *WebSphere Application Server V5.1* state that the latter acts as a hosting environment for the former. This indicates that all the WebSphere Application Server components need to be operating before one can deploy the J2EE application.

Change plan requirements and their representation in BPEL

A change plan is a procedural description of activities, each of which maps to an operation that the provisioning system reveals by means of a set of WSDL interfaces. As these interfaces are known well in advance before a change plan is created, they can be referenced by a change plan in a straightforward manner as `portTypes` in BPEL `partnerLinks`. Every operation has a set of input parameters; an operation to install a particular software component on a target system, for example, requires references to the software and to the target system as input parameters. Some of the parameters may be supplied by a user when the change plan is executed by means of the `receive` activity (e.g., the host

name of the target system that will serve as the database server) and need to be forwarded to several activities in the change plan. The assignment of these parameters to activities is done by the BPEL *assign* and *copy* statements, which move parameters between messages by populating the various parts of a variable. We note that the parts of a variable typically correspond to configuration parameters of a system; however, their specific values for a given system are either obtained from the user (or another BPEL process) by means of the `receive` activity, or extracted from data sources whose interfaces are defined as `partnerLinks` in the BPEL workflow. Whereas the structure of a change plan is known up front, the values for its parameters are determined at runtime in order to address the dynamic reconfiguration requirements of autonomic systems.

Precedence constraints reflect the order in which provisioning activities need to be carried out. Some of these constraints are implicit (e.g., the containment relationship `HasComponent` between software components), whereas others (typically resulting from communication dependencies such as *uses* or *federates*) require an explicit representation (e.g., the DB2 runtime client needs to be installed on a system whenever a database located on a remote host needs to be accessed). In the former case, the (potentially nested) BPEL *flow* and *sequence* constructs are used to group activities accordingly; in the latter case, *links* represent a convenient means to specify precedence constraints between activities that can be nested arbitrarily deep in sequences or flows.

An important consideration for minimizing the overall provisioning times lies in the exploitation of concurrency, especially when fairly large software packages (the size of middleware installation images is often several hundreds of megabytes) need to be installed on different hosting environments. It is particularly advantageous to define a change plan as a flow-based business process that comprises several sequences, each corresponding to a hosting environment. Unless explicit precedence constraints are specified by means of BPEL links, a workflow engine processes them in parallel, thus avoiding unnecessary delays.

The specification and enforcement of durations and deadlines for provisioning activities in a change plan allows a change manager to keep track of whether the rollout of a distributed system proceeds as planned or whether a schedule overrun is likely. This is needed because a change manager needs to make sure that the maintenance intervals specified in an SLA are respected in order to avoid the payment of penalties due to the unavailability of a service. The BPEL *scope* and *onAlarm* constructs help address this requirement, as one can attach a scope containing a timer to every provisioning activity and to the overall change plan. If an activity runs behind schedule, a *schedule overrun* event is sent from the workflow engine to the change manager, which then decides if the execution of the change plan should be terminated prematurely or if it should continue despite the delay.

Note that we decided not to perform error recovery or deal with schedule overruns (e.g., by means of BPEL compensation handlers) within the change plan itself. This is because in service provider environments, resources are often shared among different customers, and a change of a customer's hosted application may affect the quality of service another customer receives. Before rolling out a change for a given customer, a service provider needs to trade off the additional benefit it receives from this customer against a potential monetary loss because an SLA with another customer may be violated due to the change. The scheduling of change plans, a core change manager function, is the result of solving an optimization problem that carefully balances the benefits it receives from servicing one customer's change request against the losses it incurs from not being able to service other customers. In an on demand environment, the cost/profit situation may change very rapidly as many change plans are concurrently executed at any given instant. In some cases, it may be more advantageous to carry on with a change despite its delay, whereas in other cases, terminating a change prematurely and instead servicing another newly submitted request that is more profitable may lead to a better global optimum. This big picture, however, is only available to the change manager, which is why compensation handlers are not encoded within

■ A change plan should be represented as a flow-based business process in BPEL ■

individual change plans. Note that a change manager implements an autonomic control loop and can therefore be regarded as a special-purpose autonomic manager.

Finally, an important requirement for provisioning composed applications (obtained by composition of services) is the dynamic aggregation of already existing and tested change plans. In the case of Sp04 and its underlying middleware, change plans for provisioning some of its components (such as WebSphere Application Server or the DB2 DBMS) may already exist. By exposing a WSDL interface, BPEL workflows can be aggregated and composed to reflect the assembly of several software components into distributed applications. Over time, a library of best practices for changing and configuring software systems—codified as BPEL workflows—can evolve through componentization at a workflow level. Such best practices may then be further aggregated into higher-level change management processes to accomplish a process-based approach to IT service management.[44–46] While some activities in the IT service management process can be automated (e.g., creating change plans and executing them), other activities (such as approving a change) require human intervention. The introduction of extensions for human-facing activities, into BPEL (see the section "Working with people: Extending BPEL to activities involving humans") is an important step toward addressing this requirement.

Change plans can be created either manually from scratch or automatically generated from domain-specific knowledge. Our research prototype of a CHAnge Manager based on Planning and Scheduling (CHAMPS)[47] relies on software dependency models specified as Installable Unit Deployment Descriptors[48] to automatically generate change plans.

A change plan for SPECjAppServer2004 provisioning

Figure 6 illustrates selected activities of the BPEL workflow for provisioning and configuring the Sp04 solution, rendered in the BPEL Editor of WebSphere Process Choreographer. The workflow consists of two major sequences (Application Server Sequence and Database Server Sequence), which group the provisioning and configuration activities according to the systems on which the activities need to be carried out. We distinguish between the various change management operations as well as the hosting environments to which they apply. Whenever no dependencies exist between activities in a flow, a workflow engine will carry them out concurrently, which has the potential of significant time savings, especially if they are to be carried out on different systems. Additional sequences are nested in the major sequences. These additional sequences group the activities for installing and configuring the WebSphere Application Server, the Sp04 application, and the DB2 database system. Each of the sequences contains the individual deployment and configuration activities, which deal with deploying the various software products from a centralized software repository to the target systems, installing them, and configuring them. Before installing a component, its hosting environment needs to be started.

Configuration-related activities cover a wide variety of tasks, which go beyond the mere setting of parameters; rather, they involve the creation of logical structures, such as table spaces and tables in a database system, populating these logical structures with data, or creating data sources, connection pools, and factories in an application server. In order to keep the workflows at an acceptable level of granularity, we rely on a set of small configuration scripts[49] that break down each of the configuration activities in Figure 6 into their atomic tasks and execute them on the target systems.

Configuration activities sometimes require a set of input parameters that are produced by other activities. These parameters need to be passed between configuration activities; this—in turn—imposes constraints on their execution order. As an example, the *CREATE JDBC** Provider and SPECjHostAliases* activity in the *CONFIGURE Application Server (AS) WAS5.1* sequence requires various parameters (such as database name, host name of the DB server, port number of the database demon) that result from the *CREATE Database* activity, which is carried out on a different system. Whenever this is the case, one needs to insert a link between these activities to ensure that the workflow engine does not start the execution of an activity before the predecessor activity has finished. This is especially important when the flow of configuration parameters crosses system boundaries. In Figure 6, the two magenta horizontal arrows indicate such cross-system constraints; they are expressed by means of the BPEL *link* construct.

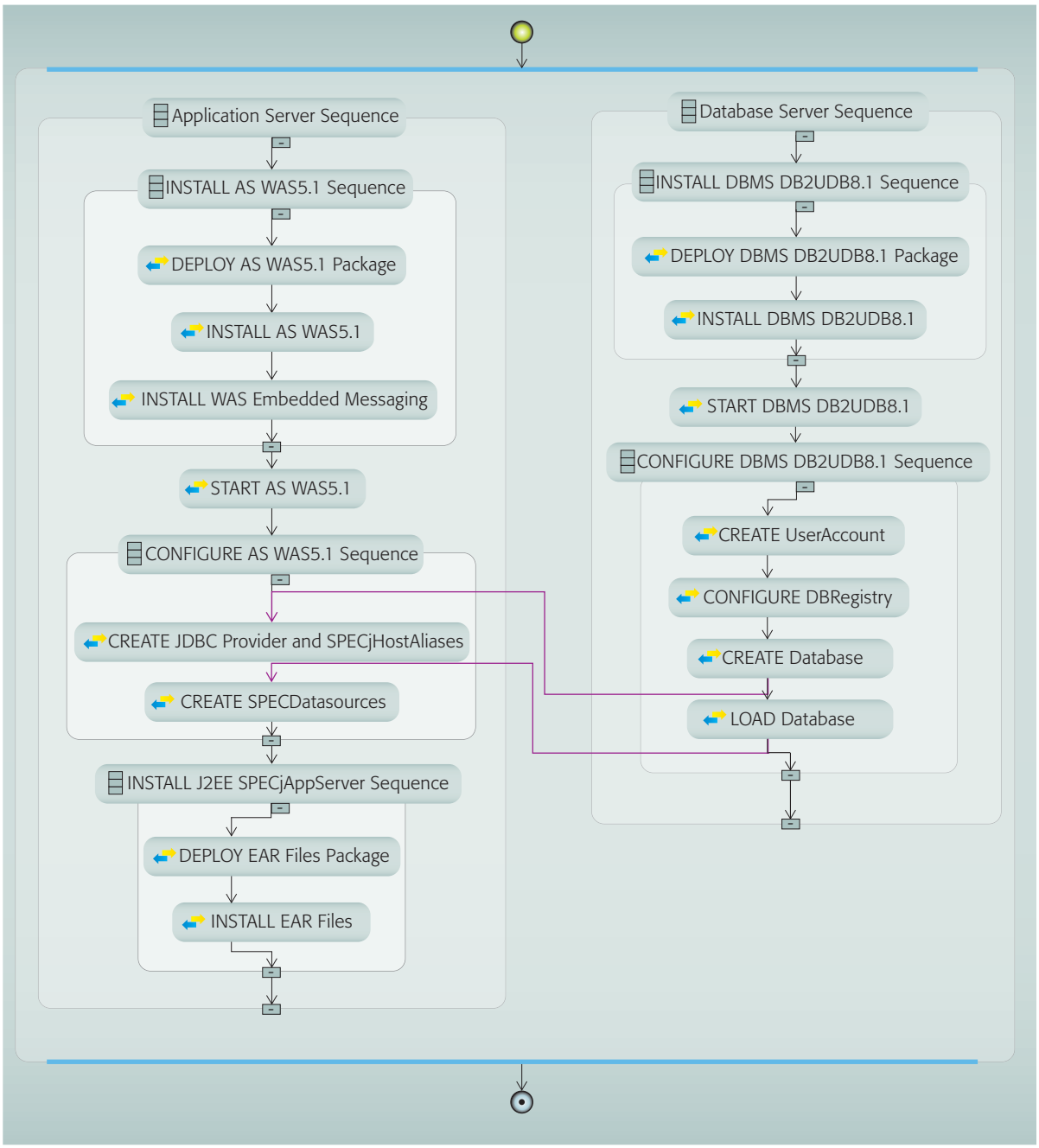

Figure 6
Selected activities in the BPEL workflow for provisioning the SPECjAppServer solution

Executing the change plan

Upon receiving a newly submitted change request, the change manager needs to determine on which resources and at what time the change will be carried out. As depicted in *Figure 7*, the change manager first inspects the resource pools of the provisioning system (Resource Availability arrow) to determine which target systems are best assigned to the change by taking into account the operating system they run, their system architecture, and the cost of assigning them to a change request. Based on this information, the change manager creates a change plan, which may be composed of already existing change plans that reside in a change plan

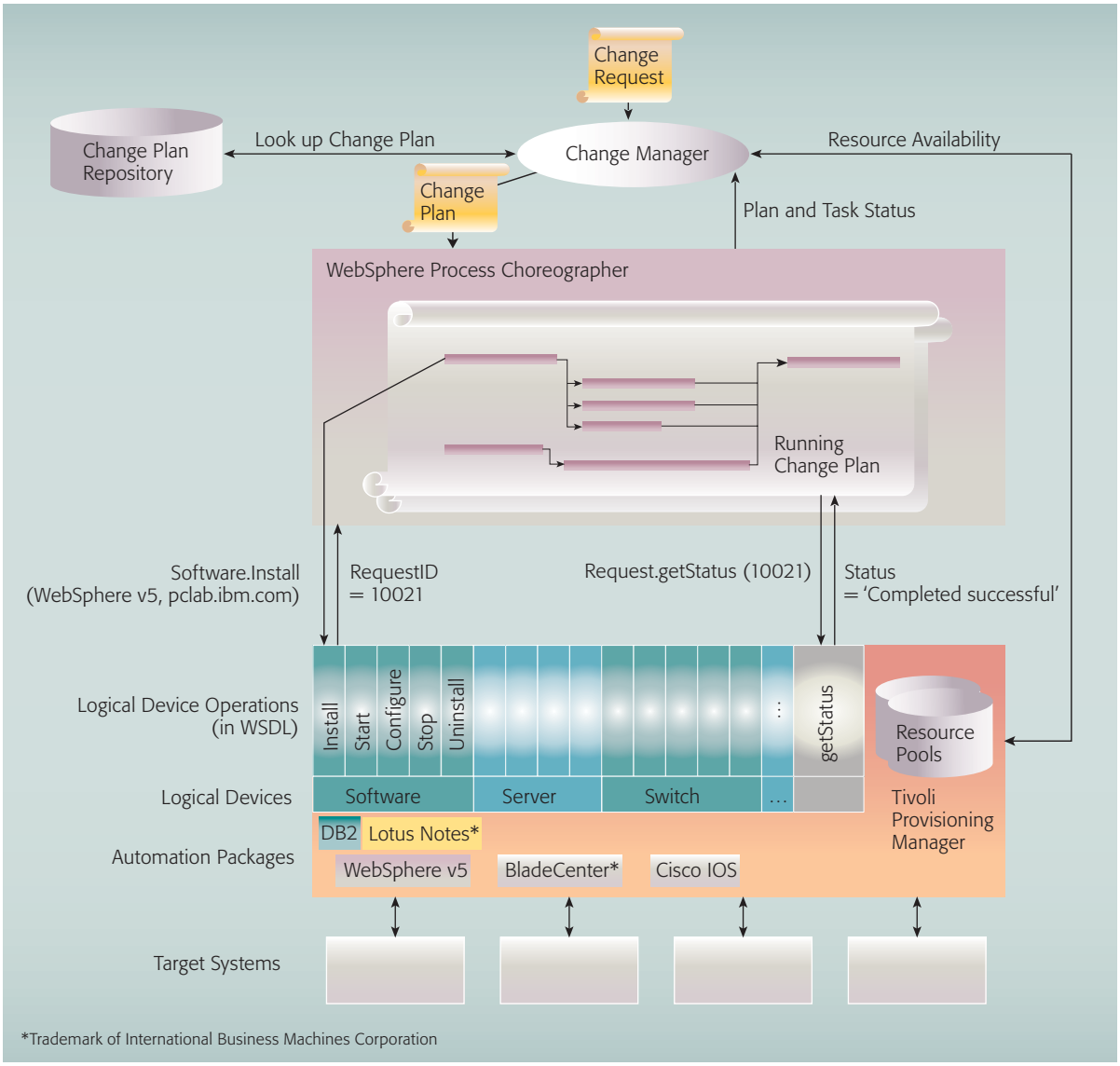

Figure 7
Architecture of a workflow-driven change manager prototype

repository (Look Up Change Plan arrow). Once the change plan is created, it is submitted to the workflow engine (Change Plan arrow). In our prototype system, we use IBM WebSphere Process Choreographer, a general-purpose workflow engine that is able to interpret and execute change plans defined in BPEL.

IBM Tivoli* Provisioning Manager, an autonomic manager that we use in our prototype, maps the actions defined in the change plan to operations that are understood by the target systems. Its object-oriented data model is a hierarchy of *Logical Devices* that correspond to the various types of managed resources (e.g., software, storage, servers, clusters, routers, and switches). The methods of these types correspond to *Logical Device Operations* (*LDOs*) that are exposed as WSDL interfaces, which allow their inclusion in the change plan as `partnerLinks`. *Automation Packages* are product-specific implementations of logical devices. For example, an automation package for the DB2 DBMS would provide scripts that implement the `software.install`, `software.start`, and `software.stop` LDOs. An automation package consists of a set of Jython scripts, each of which implements an LDO.

Every script can further embed a combination of PERL, Expect, and bash shell scripts that are executed on the remote target systems. We note that the composition pattern applies not only to BPEL workflows, but occurs in various places within the provisioning system itself.

The workflow engine inputs the change plan and starts each provisioning operation by directly invoking the LDOs of the provisioning system. These invocations are performed either in parallel or sequentially, according to the flows, sequences, and links defined in a change plan. A major advantage of using a workflow engine for our purposes is that it automatically performs state checking; that is, it determines whether all conditions are met for triggering the next activity in a workflow. Consequently, there is no need for developing additional program logic to perform such checks. This, however, is still required when interpreters for scripting languages are used, as is often the case in traditional systems management.

In a second step, the provisioning system is invoked by the workflow engine and performs the requested operations. It reports the status of each operation execution back to the workflow engine. This status information is used by the workflow engine to check if the workflow constraints defined in the plan (such as deadlines) are met and to inform the change manager whether the rollout of changes runs according to the schedule defined in the change plan. For further details on our implementation, see Reference 50.

Advantages of using BPEL for autonomic computing

Our prototype implementation demonstrates that BPEL is fully applicable to representing change plans for execution by an autonomic manager. BPEL offers a number of advantages over other workflow languages. First, its built-in support of Web Services and XML allows the invocation of provisioning operations in a platform-independent and portable way. Second, the use of a BPEL workflow engine makes delegation of activity-execution status checking and compliance monitoring possible, with specified deadlines. Based on the instructions in a change plan, the change manager is notified by the workflow engine whenever an activity is completed and when a deadline has passed. Third, the declaration of a change plan as a flow-based business process ensures a high degree of parallelism for a set of tasks. In the Sp04 example, the ability to carry out multiple activities in parallel allows us to achieve a total time of about 34 minutes on average (compared to about 50 minutes for strictly sequential activity execution) for provisioning the Sp04 application and its middleware stack. Fourth, by leveraging the Web service composition capabilities of BPEL, one can aggregate and reuse already existing change plans. Over time, a library of tested BPEL workflows for changing and configuring software systems can be built, which can serve as a basis for higher-level IT service management processes[44] whose activities can be either automated or carried out by humans, if needed. Finally, BPEL helps in integrating and composing autonomic managers out of existing autonomic managers, thereby facilitating the delegation and reuse of autonomic functions in distributed, heterogeneous environments.

■ A library of best practices for configuring software systems can be built from tested BPEL workflows ■

CONCLUSION

In this paper we have presented the concepts behind BPEL, focusing on language extensions and several application areas, which included human-facing activities, the use of abstract processes, Grid computing, and a case study in autonomic computing. We covered upcoming changes to the specification, especially with regard to BPEL abstract processes, which are beginning to see some activity but are still lagging behind the executable variant in uptake. We show the strengths of BPEL and point to its shortcomings and possible future enhancements.

From a strictly technical point of view, BPEL can be seen as one of many proposed workflow languages. Considering nontechnical aspects, however, BPEL is a significant step forward because it brings together two formerly separated workflow communities (graph-oriented and calculus-based), and—more important—is implemented by all major vendors. This is enormous progress for users because it provides something they have long required: a single workflow language portable across environments.

BPEL uses Web Services to implement core activities and also renders processes as Web Services. A criticism of BPEL has been that there are application domains where a pure Web Services abstraction does not suffice or is inappropriate. However, BPEL's extensibility can be used to adapt it to such areas. For example, extensions for supporting subprocesses in order to tie various processes into a whole are described in Reference 51, whereas extensions for direct support of Java within BPEL for applications that do not require Web Services and XML are described in Reference 7. Such extensions require standardization to benefit from portability.

The use of extensions to BPEL for supporting the entire spectrum of business process management can be viewed as a corollary to the modularity and composability "axiom" of the Web service platform.[4] This axiom is responsible for another criticism of BPEL, namely, its complexity, resulting from the number of standards needed for a full solution. For example, specifying deployment information is not BPEL-specific and therefore left out of the specification but is required at many places in the Web Services stack. As a consequence, specifying such information declaratively (e.g., through policies) results in highly adaptive processes in terms of partner bindings.

The momentum behind the BPEL specification and its support from industry and academia is largely due not only to its architecture and capabilities but also to its having been layered on top of the Web Services stack, the standardization effort, and the number of implementations that quickly became available, thereby decreasing learning and adoption curves. Some have seen BPEL as a programming language in XML, others have described it as the export format for business processes, while still others (amongst them the authors) see it as a powerful workflow language that presents the natural (conversational) model for programming in-the-large in a service-oriented world. Most important, if it were not for the XML base, we would still be talking about proprietary languages and platforms.

ACKNOWLEDGMENTS

We acknowledge Dieter Koenig for feedback on earlier drafts. We thank our colleagues at IBM, including Stefan Tai, Thomas Mikalsen, and the Component Systems Group for collaborations described in this paper.

CITED REFERENCES AND NOTES

1. F. Leymann and D. Roller, *Production Workflow*, Prentice Hall, Upper Saddle River, New Jersey (2000).
2. The Workflow Management Coalition, http://www.wfmc.org.
3. D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases* **3**, No. 2, 119–153 (1995).
4. *The Web Services Platform Architecture*, S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson, Editors, Addison Wesley, Reading, MA (2005).
5. *Business Process Execution Language for Web Services Version 1.1*, BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, and Siebel Systems (2002), developerWorks (updated February 1, 2005), http://www.ibm.com/developerworks/library/specification/ws-bpel.
6. R. Khalaf, N. Mukhi, and S. Weerawarana, "Service-Oriented Composition in BPEL4WS," *Proceedings of the Twelfth International World Wide Conference (WWW2003), Web Services Track,* Budapest, Hungary, May 20–24, 2003, Kluwer Academic Publishers, Norwell, MA (2003).
7. M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, and M. Rowley, *BPELJ: BPEL for Java Technology*, BEA Systems and IBM Corporation (2004), http://www.ibm.com/developerworks/library/specification/ws-bpelj/.
8. "BPEL Abstract Processes," S. Thatte and R. Khalaf, Editors, Input document to the OASIS Technical Committee's discussions on possible changes to BPEL abstract processes (2004), http://lists.oasis-open.org/archives/wsbpel/200409/doc00000.doc.
9. R. J. van Glabbeek, "The Linear Time—Branching Time Spectrum," http://theory.stanford.edu/~rvg/abstracts.html#19.
10. A. Martens, "Consistency between Executable and Abstract Processes" *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Services (EEE 2005)*, March 29–April 1, 2005, Hong Kong, China, IEEE, New York (2005), pp. 60–67.
11. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, "From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution," Fujitsu, Globus Alliance, IBM Corporation, 2004. http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf.
12. J. Hidders, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Verelst, "When Are Two Workflows the Same?" *Proceedings of Computing: The 11th Australasian Theory Symposium (CATS)*, Newcastle, Australia, February 2005. Australian Computer Society (2005), pp. 3–11.

13. A. Martens, "Process-Oriented Discovery of Business Partners," *Proceedings of the Seventh International Conference on Enterprise Information Systems (ICEIS'05)*, May 25–28, 2005, Miami, Florida (May 2005), pp. 57–64.

14. P. Massuthe, W. Reisig, and K. Schmidt, "An Operating Guideline Approach to SOA," *Proceedings of the 2nd South-East European Workshop on Formal Methods 2005 (SEEFM05),* Ohrid, Republic of Macedonia (2005).

15. A. Martens, "Analyzing Web Service Based Business Processes," *Proceedings of the Eighth International Conference on Fundamental Approaches to Software Engineering (FASE'05),* Edinburgh, Scotland, April 4–8, 2005, in *Lecture Notes in Computer Science* **3442**, Springer, Berlin (2005), pp. 19–33.

16. WOMBAT4WS (in German), online from Humboldt University, Berlin, http://www.informatik.hu-berlin.de/top/wombat/.

17. BABEL tools, Queensland University of Technology, Brisbane, Australia, http://www.bpm.fit.qut.edu.au/projects/babel/tools/.

18. S. Damodaran, "B2B Integration over the Internet with XML: RosettaNet Successes and Challenges," *Proceedings of the 13th International Conference on World Wide Web (WWW 2004)—Alternate Track Papers and Posters,* New York, May 17–20, 2004, ACM, New York (2004), pp. 188–195.

19. P. Bunter, R. Hertlein, R. Khalaf, and A. Nadalin, *An Approach to Moving Industry Business Messaging Standards to Web Services*, developerWorks, IBM Corporation (2004), http://www.ibm.com/developerworks/webservices/library/ws-move2ws.html.

20. R. Khalaf, "From RosettaNet PIPs To BPEL Processes: A Three Level Approach for Business Protocols," *Third International Conference on Business Process Management (BPM 2005),* Nancy, France, September 5–8, 2005, in *Lecture Notes in Computer Science* **3649**, Springer, Berlin (2005), pp. 364–373.

21. S. Tai, T. Mikalsen, E. Wohlstadter, N. Desai, and I. Rouvellou, "Transaction Policies for Service-Oriented Computing," *Data and Knowledge Engineering Journal* **51**, No. 1, 59–79 (2004).

22. N. K. Mukhi, P. Plebani, I. Silva-Lepe, and T. Mikalsen, "Supporting Policy-Driven Behaviors in Web Services: Experiences and Issues," *Proceedings of the 2nd International Conference on Service-Oriented Computing (ICSOC '04),* ACM New York (2004), pp. 322–328.

23. F. Curbera, M. J. Duftler, R. Khalaf, W. A. Nagy, N. Mukhi, and S. Weerawarana, "Colombo: Lightweight Middleware for Service-Oriented Computing," *IBM Systems Journal* **44**, No. 4, 799–820 (2005).

24. E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, and P. Devanbu, "GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions," *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004),* May 23–28, 2004, Edinburgh, United Kingdom. IEEE, New York (2004), pp. 189–199.

25. S. Tai, R. Khalaf, and T. Mikalsen, "Composition of Coordinated Web Services," *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2004)* Toronto, Canada, October 18–20, 2004, in *Lecture Notes in Computer Science* **3231**, Springer, Berlin (2004), pp. 294–310.

26. C. Courbis and A. Finkelstein, "Weaving Aspects into Web Service Orchestrations," *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, July 11–15, 2005, Orlando, Florida, IEEE, New York (2005), pp. 69–77.

27. A. Charfi and M. Mezini, "Using Aspects for Security Engineering of Web Service Compositions," *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, July 11–15, 2005, Orlando, Florida, IEEE, New York (2005), pp. 59–66.

28. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality Driven Web Services Composition," *Proceedings of the Twelfth International World Wide Web Conference (WWW2003),* Budapest, Hungary, May 20–24, 2003, ACM, New York (2003), pp. 411–421.

29. M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic, *WS-BPEL Extension for People (BPEL4People)*, IBM Corporation and SAP AG (2005), http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.

30. K. Lind and E. Norman, *WebSphere Application Server Enterprise Process Choreographer: Staff Resolution Architecture*, developerWorks, IBM Corporation (2003), http://www.ibm.com/developerworks/websphere/library/techarticles/wasid/WPC_StaffArch/WPC_StaffArch.html.

31. M. Kloppmann, D. König, F. Leymann, G. Pfau, and D. Roller, "Business Process Choreography in WebSphere: Combining the Power of BPEL and J2EE," *IBM Systems Journal* **43**, No. 2, 270–296 (2004).

32. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman Publishers, San Francisco, CA (1999).

33. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM* **45**, No. 11, 56–61 (November 2002).

34. I. Foster, "What is the Grid: A Three Point Checklist," *Grid Today* (July 20, 2002).

35. W. Emmerich, B. Butchart, L. Chen, B. Wasserman, and S. L. Price, *Grid-Service Orchestration Using Business Process Execution Language (BPEL)*, University College London, CS Research Note RN/05/07 (June 2005).

36. F. Leymann, "Choreography for the Grid: Toward Fitting BPEL to the Resource Framework," *Concurrency and Computation: Practice and Experience* (2006, to appear).

37. A. Slominski, "On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows," *Concurrency and Computation: Practice and Experience* (2006, to appear).

38. A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era," *IBM Systems Journal* **42**, No. 1, 5–18, 2003.

39. R. Murch, *Autonomic Computing*, IBM Press/Prentice Hall (2004).

40. D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why Do Internet Services Fail, and What Can Be Done about It?" *Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems*, March 26–28, 2003, Seattle, WA, USENIX Association (2003).

41. *An Architectural Blueprint for Autonomic Computing*, Autonomic Computing White Paper, Third Edition, IBM Corporation (June 2005), http://www-128.ibm.com/developerworks/autonomic/library/ac-summary/ac-blue.html.

42. IBM Tivoli Provisioning Manager, http://www.ibm.com/software/tivoli/products/prov-mgr/.

43. *SPECjAppServer2004 Design Document, Version 1.01*, Standard Performance Evaluation Corporation (January

2005), http://www.spec.org/osg/jAppServer2004/docs/DesignDocument.html.

44. A. Brown and A. Keller, "A Best Practice Approach for Automating IT Management Processes," *Proceedings of 2006 IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, Vancouver, BC, Canada, IEEE, New York (April 2006, to appear).
45. IT Infrastructure Library, "ITIL Service Support, Version 2.3," Office of Government Commerce, United Kingdom (June 2000).
46. L. Simcox, K. Shah, T. Dunton, and D. Groves, "Introduction to IT Service Management, Part 1: Automate Your Key IT Processes," developerWorks, IBM Corporation (May 2005), http://www.ibm.com/developerworks/library/ac-prism1/.
47. A. Keller, J. L. Hellerstein, J. L. Wolf, K.-L. Wu, and V. Krishnan, "The CHAMPS System: Change Management with Planning and Scheduling," R. Boutaba and S.-B. Kim, Editors, *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS'2004)*, Seoul, Korea, April 2004. IEEE, New York (2004), pp. 395–408.
48. M. Vitaletti, (Ed.), "Installable Unit Deployment Descriptor Specification, Version 1.0," *W3C Member Submission*, IBM Corporation, ZeroG Software, InstallShield Software Corp., and Novell Inc. (July 2004), http://www.w3.org/Submission/2004/SUBM-InstallableUnit-DD-20040712.
49. T. Lau, "Set Up a SPECjAppServer2004 Application with DB2 Universal Database," developerWorks, IBM Corporation, July 2004. http://www-106.ibm.com/developerworks/db2/library/techarticle/dm-0407lau/.
50. A. Keller and R. Badonnel, "Automating the Provisioning of Application Services with the BPEL4WS Workflow Language," *Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2004)*, November 15–17, 2004, Davis, CA, in *Lecture Notes in Computer Science* **3278**, Springer, Berlin (2004), pp. 15–27.
51. M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic, *WS-BPEL Extension for Subprocesses (BPEL-SPE)* IBM Corporation and SAP AG (September 2005).

Accepted for publication December 7, 2005.
Published online May 10, 2006.

Rania Khalaf
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (rkhalaf@us.ibm.com). Ms. Khalaf is a software engineer in the Component Systems group at the Watson Research Center. She received her Bachelor's and Master's degrees in computer science and electrical engineering from MIT in 2000 and 2001. Her interests include component-based software engineering, workflow, and service-oriented computing, Web Services in particular. Ms. Khalaf is a co-developer and co-architect of the IBM BPEL4WS prototype implementation (BPWS4J) and the Java Record Object Model (JROM). She has published a number of papers on service-oriented computing and has served on the program committees of conferences and workshops in the field. Ms. Khalaf is pursuing her Ph.D. studies in service aggregation and composition under Prof. Dr. Frank Leymann at the University of Stuttgart while continuing to work at IBM.

Alexander Keller
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (alexk@us.ibm.com). Dr. Keller is a research staff member and manages the Service Delivery Technologies department at the Watson Research Center. He received his M.Sc. and Ph.D. degrees in computer science from Technische Universität München, Germany, in 1994 and 1998, and has published more than 40 refereed papers in the area of distributed systems management. He joined the IBM Research Division in 1999. Dr. Keller's research interests revolve around change management for applications and services, information modeling for e-business systems, and SLAs (service-level agreements). He serves on several technical program and organizing committees of related conferences and workshops and is a member of the USENIX Association, the IEEE, and the DMTF CIM Applications and Metric Extensions working groups. He was a main contributor to the IBM Web Service Level Agreement (WSLA) framework, which served as the basis for the upcoming GGF WS-Agreement standard.

Frank Leymann
University of Stuttgart, Universitätsstr.38, 70569 Stuttgart, Germany and IBM Software Group, Böblingen, Germany (Frank.Leymann@informatik.uni-stuttgart.de). Prof. Dr. Leymann is a full professor of computer science and director of the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. His research interests include service-oriented computing, workflow and business process management, transaction processing, and architecture patterns. Frank worked for two decades in the IBM Software Group, building database and middleware products. He was awarded the title of IBM Distinguished Engineer in 2000 and was elected to the IBM Academy of Technology in 1994. He has worked continuously on workflow technology since the late 1980s, becoming known as the father of IBM's workflow product set. He contributed heavily to the architecture and strategy of IBM's entire middleware stack and IBM's on demand computing strategy and is co-architect of the Web Services stack. He is co-author of many Web Services specifications, including WSFL, WS-Addressing, WS-MetadataExchange, WS-Business Agreement, the WS-Resource Framework, and, of course, BPEL4WS. Dr. Leymann has published many papers in journals and proceedings, co-authored three text books, and holds a multitude of patents especially in the area of workflow management and transaction processing. He served on program and organizing committees of many international conferences, and he is editor-in-chief or associated editor of several journals. ■