Goals of the BPEL4WS Specification

Frank Leymann, Dieter Roller, and Satish Thatte

This note aims to set forward the goals and principals that formed the basis for the work of the original authors of the BPEL4WS specification. The note is set in context to reflect the considerations that went into the work, rather than being presented as a set of axioms. Much of this material is abstracted from comments and explanations embedded in the text of the specification itself.

This is intended to be informative and a starting point for a consensus in the WSBPEL TC for the work of the TC. The goals set out here are also reflected in the charter of the WSBPEL TC.

Web Services as the Base

BPEL4WS is firmly set in the Web services world as the name implies. In particular, all external interactions occur through Web service interfaces described using WSDL. This has two aspects: (1) the process interacts with Web services through interfaces described using WSDL and (2) the process manifests itself as Web services described using WSDL. We concluded that although the binding level aspects of WSDL sometimes impose constraints on the usage of the abstract operations, in the interests of simplicity and reusability we should confine the exposure of process behavior to the "abstract" portType (i.e. "interface") level and leave binding and deployment issues out of the scope of the process models described by BPEL4WS. The dependence is concretely on WSDL 1.1, and should remain so, given the timeline for the WSBPEL TC, and the likelihood that WSDL 1.1 will remain the dominant Web service description model for some time to come. At the same time we should be sensitive to developments in WSDL 1.2 and attempt to stay compatible with them. This leads to

Overall Goal 1: BPEL4WS should define business processes that interact with external entities through Web service operations defined using WSDL 1.1 and that manifest themselves as Web services defined using WSDL 1.1. The interactions are "abstract" in the sense that the dependence is on portType definitions, not on port definitions. In future versions BPEL4WS should be sensitive to developments in WSDL 1.2 and attempt to stay compatible with them.

XML as the Form

In a world with many different business modeling methodologies and graphical notations, we thought that a precise description of the syntax and operational semantics of core process concepts is the key. In line with all the other standards and standards proposals in the Web services space, we used XML as the underlying representation using XML Schema as the schema language. Business process modeling tools can provide transformation routines for converting to and from BPEL4WS. This leads to

Overall Goal 2: BPEL4WS defines business processes using an XML based language. BPEL4WS is not concerned with the graphical representation of processes nor defines any particular design methodology for processes.

Common set of Core Concepts

Starting with the broad notion that we wanted to specify "orchestration" or "choreography" of Web service interactions, we recognized that there were divergent viewpoints on what that meant, even among the authors and certainly in the Web services community as a whole.

At the most basic level, there is the duality between external and internal views of the process. Standardization provides value for both usage patterns. For external views, the value is in enhancing Web service descriptions with description of behavior, enhancing predictability and hence interoperability. We like to refer to coupled (multi-role) interoperable external process views as "business protocols". For internal views the value is in unifying the concepts for process modeling, at least to the point where process models as templates embodying domain-specific best practices can be encoded in a platform-neutral and portable way by both enterprises and vertical standards organizations ("process models as tradable artifacts").

Going beyond this high level, clearly there are many ways to approach a solution for both usage patterns. We were of the opinion that

- Business protocols invariably include data-dependent behavior. For example, a supply-chain protocol depends on data such as the number of line items in an order, the total value of an order, or a deliver-by deadline. Defining business intent in these cases requires the use of conditional and time-out constructs.
- The ability to specify exceptional conditions and their consequences, including recovery sequences, is at least as important for business protocols as the ability to define the behavior in the "all goes well" case.
- The partners in a business protocol are autonomous entities that act independently and concurrently relative to other partners using full-duplex communication channels in which any entity may send messages at any point in the overall interaction.

These assumptions led us to two basic conclusions

- Data dependent behavior and full-duplex communication are difficult to specify unless the external behavior of each partner is specified independently. The desirability of independent specification for each partner is reinforced by the common experience in EDI and RosettaNet that partners in business protocol tend to need some "tweaking" of their external behavior specification for business or technical reasons.
- Data dependent behavior, error recovery, and other requirements imply that in order to provide precise predictable descriptions of service behavior for cross-

enterprise business protocols, we need a rich process description notation with many features reminiscent of an executable language.

All of this led to the basic direction of designing a common set of core concepts that would be available for both external (abstract) and internal (executable) views of a process. There is a bonus that flows from this decision. Whichever way internal and external behavior is defined, there is clearly a requirement to verify conformance between them. Given a common set of concepts on which both kinds of views are built, such conformance checking becomes more tractable. Moreover, at a tools level, import of an abstract process to form a skeleton executable process, and exporting an abstract process as a subset of the activities of an executable process become easier to provide.

This set of considerations result in

Overall Goal 3: BPEL4WS 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. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

Control Behavior

This is probably the hardest area to explain since the set of possibilities is so large. In this section we only deal with the high level issue of hierarchical and graph models and leave the discussion of the concrete feature set for another document.

To start with, BPEL4WS had two immediate predecessors, namely XLANG and WSFL. These predecessors happened to follow two different control regimes. XLANG represented hierarchical structure with specialized control constructs and WSFL represented graph structure with control patterns based on transition and join conditions. This dichotomy was not simply a contingent fact of history. Each of the predecessors had plenty of company and each represented a trend in the process modeling and specification space. Resolving the dichotomy in favor of one or the other regime was not an acceptable outcome for the simple reason that the entrenched constituencies for each would never accept the other as a conceptual *coup d' etat*. And we believed that it was important that the specification set forward concepts that were understandable and acceptable to as large a constituency as possible.

Does that mean we expected everyone to author BPEL4WS processes by hand? Not really. But we did expect the core concepts to be exposed and taught to a large audience. We believed the value of BPEL4WS to be at least partly that it would provide a common, understandable and sufficiently complete set of core process concepts for a broad constituency, thereby reducing the fragmentation of the process modeling space dramatically.

We thus believe the much discussed dichotomy between "execution language" and "authoring language" to be fallacious since it is neither possible nor desirable to dictate one or the other viewpoint. Developers invariably tinker with "execution level" process models, especially given that process models are supposed to be "higher level" than mere code, and are meant to support "agility" in change.

The result of all these considerations was that we spent a lot of time and effort to come up with a way to bridge the two control regimes seamlessly in a single model of control behavior. We believe we have succeeded in doing so. The blend provides a range of possibilities for combining the two styles according to the taste and convenience of the user. There are clearly some usage patterns that are easier to express in one regime or the other. In BPEL one does not have to make a choice—one can use whatever happens to be the most convenient usage at any level of nesting. We did have to place a number of constraints on the usage of links to avoid situations that are problematic for the operational semantics. This is self-explanatory for the most part but could use more explicit justification in some cases.

All this leads to

Overall Goal 4: BPEL4WS should provide both hierarchical and graph-like control regimes, and allow their usage to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.

Data Handling

As we discussed in the core concepts section, the ability to define data dependent behavior is a core requirement. Data occurs in messages sent and received in Web service interactions, and in conditions that guide process behavior. The data associated with a process is not intended to be used as a general purpose store nor are the data manipulation functions offered by BPEL4WS intended as a general purpose data manipulation language. Real work with data is performed by the Web services invoked by the process. The data handling features in BPEL4WS are meant to be just powerful enough to perform the simple manipulations necessary in defining process models including very simple message construction, and data expressions associated with control flow. This leads to

Overall Goal 5 : BPEL4WS provides limited data manipulation functions that are sufficient for the simple manipulation of data that is needed to define process relevant data and control flow.

Properties and Correlation

A running business process is an instance of a particular process model. Thus, each running business process requires at least one unique identifier to be able to pass an incoming message to the correct instance of a process model. Since running business processes are first class business artifacts, they are in practice identified in business terms. Different participants of a particular process may want to identify the process in their own terms and they may even want to change the identifier used in the course of their interaction with the process. We therefore came to the conclusion that defining artificial instance identifiers for processes is not the best solution. Instead, we added a flexible mechanism to BPEL4WS that supports multiple user-defined identifiers for a process instance: Correlation via message properties. In particular, these message properties are embedded in the application messages (i.e., WSDL abstract messages) exchanged by the process and are not defined as header fields of the communication protocol. This results in a binding independent correlation scheme.

Overall Goal 6: BPEL4WS should support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be partner defined and may change over time.

Lifecycle

In BPEL4WS, processes interface with their partners via Web services. Due to the possibly long-running nature of a process the corresponding collection of Web services is "stateful". WSDL's paradigm of a Web service is "stateless". Having to distinguish between both kinds of Web services would complicate the usage of Web services. To avoid the need for a client to distinguish between stateful and stateless Web services the programming model for using both kinds of Web services must be the same. In particular, no specific lifecycle interface must be assumed for stateful Web services representing a process to its users. A client must be unaware whether or not it sends a message to a standalone Web service or to a Web service surfacing a process.

As a consequence, we decided in favor of implicit management of the lifecycle of processes: A new instance of a process model is automatically created when a message is targeted to an appropriately annotated receiving operation of a Web service externalized by the process model. An instance is deleted when the control flow reaches the terminal activities of the process, or if the control flow within the instance reaches an activity that terminates the instance explicitly as action of the process itself.

Overall Goal 7: BPEL4WS should support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations like suspend, resume may be added in future releases for enhanced lifecycle management.

Long-Running Transaction Model

Many business processes are long running. Because of this, an erroneous situation should not result in wasting all of the process' work performed in the past, i.e. the effects of a fault or error should be "as local as possible" rather than resulting in catastrophic failure. To make this possible, we decided to add a unit-of-work concept to BPEL4WS which had to support backward recovery at small granules. And we wanted to rely on fundamental mechanisms for undoing work that have been used in practice in contrast to relying on fancy extended transaction models that haven't been proven yet. Based on our experience we have chosen to build the long-running transaction model of BPEL4WS on compensation techniques and an adaptation and extension of the Saga and open nested transaction mechanism for processes.

Overall Goal 8: BPEL4WS should define a long-running transaction model that is based on practically proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.

Modularization

The decomposition and assembly of business processes using sub-processes is addressed in BPEL4WS by exploiting its native capabilities of making a business process available as a Web service and having the business process invoking Web services. Thus a business process is a Web service that can be used in another business process which is a Web service as well, using implicit lifecycle management. This very simple recursive aggregation model provides for flexibility in organizing business processes. Decorating this approach with WS-Policy statements for specifying the dependencies/coupling of the different Web services would provide an extremely rich concept for modularization of business processes.

Overall Goal 9: BPEL4WS should use Web services as the model for process decomposition and assembly. Combining this approach with WS-Policy would make this model even stronger.

Composition with other Web Services Functionality

A key objective of BPEL4WS is to fit seamlessly into the current and evolving landscape of Web services standards by focusing on its core mission of defining the Web services process modeling concepts and using existing Web services standards and standards proposals for all other required mechanisms wherever possible. We tried to be rigorous about this "separation of concerns" and invented new Web service mechanisms only in cases where there was an essential requirement in process modeling for which there was no appropriate standard or proposed standard available. When we had to invent such mechanisms we tried to specify them separately from BPEL in a modular and composable fashion. A typical example is the specification of the BA-Protocol of WS-Transaction to describe the behavior of compensation-based recovery within BPEL4WS. Another example is the replacement of service references with Endpoint References from the WS-Addressing standards proposal. Property and property alias definitions for correlation is an example of an essential requirement where there is no applicable standard available. This can be summarized as

Overall Goal 10: BPEL4WS should build on compatible Web services standards and standards proposals as much as possible in a composable and modular manner. Only if no appropriate standard or standards proposal is available for a particular requirement, an appropriate specification should developed within the BPEL4WS specification or as a separate Web services standards proposal.