BUSINESS PROCESS DEVELOPMENT LIFECYLE METHODOLOGY:

Bringing together the world of business processes and Web services

Web services are rapidly becoming the de-facto distributed enterprise computing technology that is widely used for enabling collaborative business processes [1]. By now, most enterprises have gained some initial experience in deploying predominantly internal business applications by consuming Web services developed either in-house or offered by third parties. Most enterprises achieve this by Web service enabling legacy applications and enterprise information systems. However, currently there is little concern about development principles underlying the deployed Web services, their granularity or the development of components that implement them.

Implementing a thin SOAP/WSDL/UDDI layer on top of existing applications or software components that implement the Web services is by now widely practiced by the software industry. Yet, this is not sufficient to construct commercial strength enterprise applications. While relatively simple Web services may be effectively built that way, a development methodology is of crucial importance to specify, construct, refine and customize highly volatile business processes from Web services that are internal or external to an organization.

This article describes the insights of a methodology and roadmap that assists service providers and service aggregators in orchestrating multi-party business processes, such as invoicing and inventory control.

METHODOLOGY ROADMAP

The lifecycle of the Web services development methodology comprises five distinct phases: planning, analysis and design (A&D), realization, execution and deployment that may be traversed iteratively. This lifecyle is depicted in Figure-1.

The planning phase is a preparatory step (Phase-1) that serves to streamline and organize consequent phases in the methodology. The analysis and design phase is the next phase (Phase-2) that specifies Web services and business processes in a stepwise manner. Service realization (Phase-3) transforms specifications from the analysis and design phase into implementations. The development phase (Phase-4) aims at deploying the service and process realizations and publishing interfaces in a repository. The final phase entails execution (Phase-5), which supports the actual binding and run-time invocation of the deployed services. The transition through these phases need not be sequential and one-pass. It tends to be stepwise incremental and iterative in nature and should accommodate revisions in situations where the scope cannot be completely defined a priori.

The overall functioning of the methodology is exemplified by means of an order management process that involves a client, a seller and a trusted third-party. This process is organized as follows. On receiving a purchase order from a client, five tasks are executed concurrently at the seller's site: checking the credit worthiness of the customer, determining whether or not an ordered part is available in the product inventory, calculating the final price for the order and billing the customer, selecting a shipper, and scheduling the production and shipment for the order. While some of the processing can proceed concurrently, there are control and data dependencies between these tasks. For instance, the customer's creditworthiness must be ascertained before accepting the order, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule.

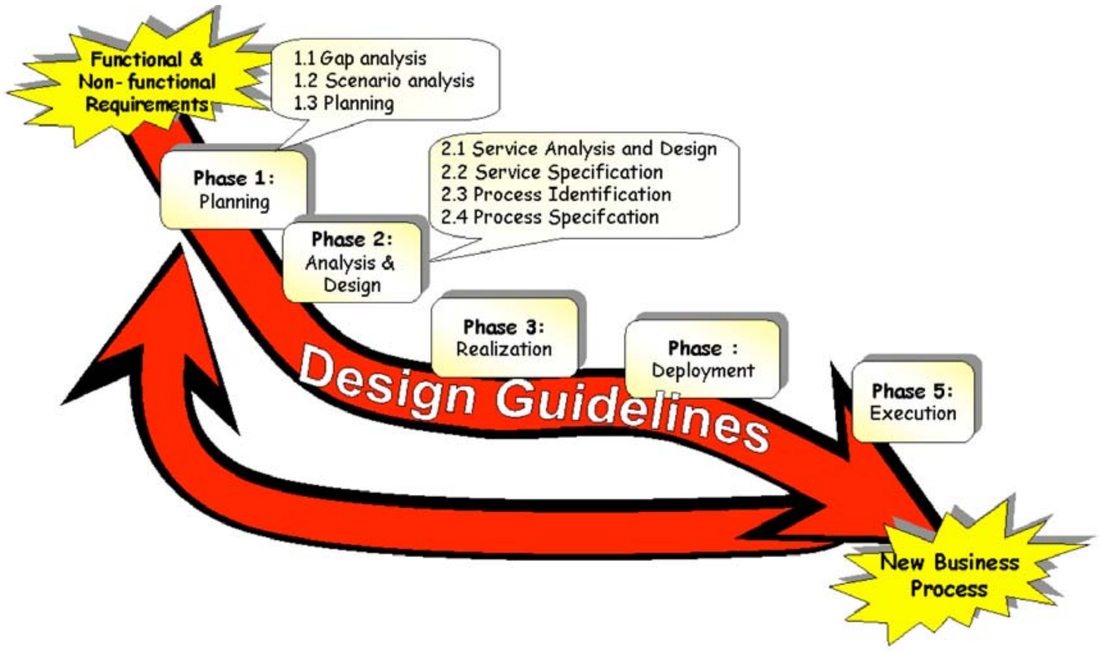

Figure 1 Methodology roadmap.

DESIGN GUIDELINES: SERVICE COUPLING AND COHESION

Business processes that are developed on the basis of existing or newly coded services must rely on sound service design principles to ensure autonomous, self-contained and modular business processes with clearly defined boundaries and service end-points. Two key principles serve as the foundation for service- and business process design: service coupling and cohesion.

Service coupling

Service coupling refers to the degree of interdependence between business processes or Web services. The design objective is to minimize coupling, that is, to make business processes as self-contained as possible by ensuring they need virtually no knowledge or service of any other business processes, thus increasing their maintainability and reuse potential. Similarly, services that are orchestrated inside a process should be loosely coupled to each other, avoiding interdependencies at the level of data or control logic. It is useful to distinguish between three ways to achieve service and process coupling:

1. **Representational coupling**: Business processes should not depend on specific representational or implementation details and assumptions underlying business processes, e.g., business processes do not need to know the scripting language that was used to compose their underlying services. Representational coupling is useful for supporting:

- Interchangeable/replaceable services: existing services may be swapped with new service implementations -- or ones supplied by a different provider offering better performance or pricing-- without disrupting the overall business process functionality;
- Multiple service versions: different versions of a service may work best in parts of a business processes depending on the application's needs. For example, a purchase order service may provide different levels of detail, e.g., internal or external requisition details such as internal or external accounts, depending on the ordering options.

2.**Identity coupling**: Connection channels between services should be unaware of who is providing the service. It is not desirable to keep track of the targets (recipients) of service messages, especially when they are likely to change or when discovering the best service provider is not a trivial matter.

3.**Communication protocol coupling**: The number of messages exchanged between a sender and addressee in order to accomplish a certain goal should be minimal, given the communication model, e.g., one-way, request/response, and solicit/response. For example, one-way style of communication where a service end point receives a message without having to send an acknowledgement places the lowest possible demands on the service performing the operation. The service that performs the operation does not assume anything about the consequences of sending the message without requiring that a notification about request completion be sent back.

Service Cohesion

Service cohesion defines the degree of the strength of functional relatedness between operations in a service or business process. The service aggregator should strive to offer cohesive (autonomous) processes whose services and service operations are strongly and genuinely related to one another. The guidelines by which to increase cohesion are as follows:

1.*Functional service cohesion:* A functionally cohesive business process should perform one and only one problem-related task and contain only services necessary for that purpose. Consider services such as get product price, check product availability, check creditworthiness, in an order management business process.

2.*Communicational service cohesion:* A communicationally cohesive business process is one whose activities and services use the same set of input and output messages. Communicationally cohesive business processes are cleanly decoupled from other processes as their activities are hardly related to activities in other processes.

3.*Logical service cohesion:* A logically cohesive business process is a process that performs a set of independent but logically similar functions (alternatives) that are tied together by means of control flows, e.g., mode of payment.

PHASE 1: THE PLANNING PHASE

Planning sets the scene for all ensuing phases by analyzing the business case of all viable mixtures of development approaches and realization strategies (see Figure-2). Fundamentally, business case analysis embraces two activities:

Step 1.1: Gap Analysis

Gap analysis commences with comparing planned services with available software service implementations that may be assembled within the enclosures of a newly developed business process. Four categories of existing service resources are discerned: legacy systems, COTS components, ERP packages, and finally, existing Web services. We collectively refer to these categories as the Web services landscape. Gap analysis matches high-level descriptions of new services that collectively make up a business process against the available resources in the Web service landscape.

Step 1.2: Scenario Analysis

Gap analysis is intertwined with a scenario analysis, which considers costs, risks, benefits and return of investment of the following options to develop new business processes:

1. Green-field development: follows a classical development model covering specification down to development. With this option we assume that the business processes have to be developed from scratch without any reuse of existing process chunks or service specifications.
2. Top-down development: usually a blueprint of business use cases provides the specification for business services. The top-down process decomposes the business domain into its functional areas and subsystems, including its flow or process decomposition into processes, sub-processes and high-level business use cases.
3. Bottom-up development: starts from existing enterprise applications and repositories and transforms them to new services by wrapping.

4. Out-of-the-middle development: combines top-down and bottom-up development by allowing service aggregators to select those fragments of enterprise resources that satisfy best new business process requirements. Fragments are then retrofitted using services.

If gap analysis results in high functional fit this points towards a bottom-down approach for the remainder of the development process whereas green-field or top-down development approach suit best in case of low functional fit. Out-of-the-middle development may be preferred in those cases where legacy assets have considerable overlap with new service requirements and lock substantial business value.

One of the issues with the top-down and bottom-up development is that they are rather ambiguous regarding which business processes an enterprise should start from and how these business processes can be combined to form business scenarios. To address this problem, service development solutions need to target specific focal points and common practises within the enterprise such as those that are specified by its corresponding sector reference models. Reference models - a typical example of which is RosetaNet - address a common large proportion of the basic 'plumbing' for a specific sector, from a process, operational function, integration, and data point of view. Such a *verticalized development model* presents the ideal architecture for supporting service development and will be used throughout this paper to explicate elements of the services development methodology.

Each of the above development options comes equipped with one or more of the four realization strategies:

(a) Reuse existing (internal) Web services or business process logic,
(b) Develop new Web services or business processes logic from scratch,
(b) Purchase/outsource web services or (parts of) business processes,
(c) Revamp existing (COTS) components or existing (ERP/legacy) systems in Web services or business processes.

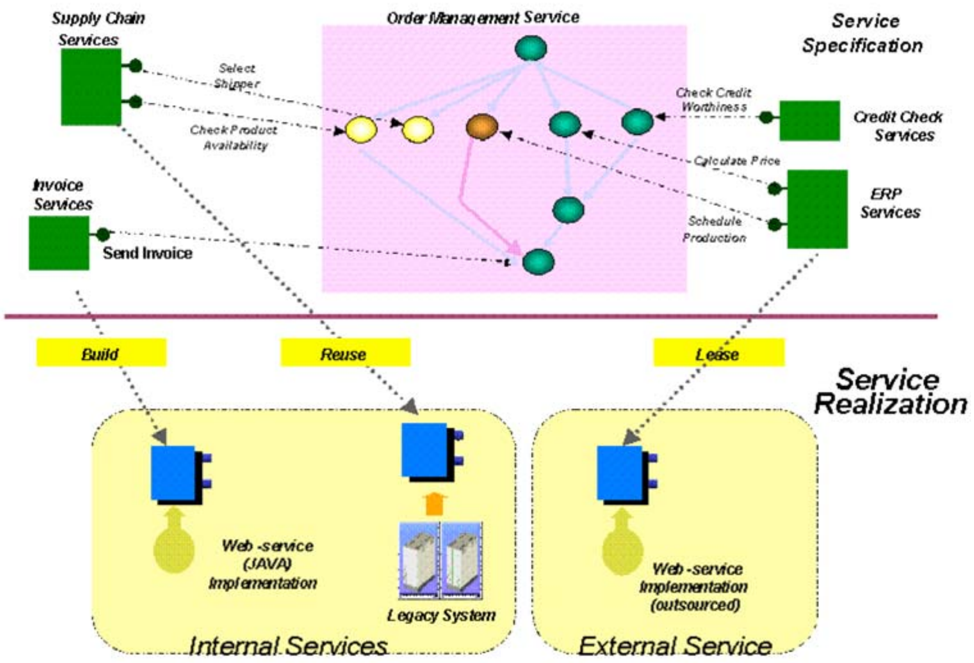

Figure 2 Scenario Analysis

During scenario analysis estimates for existing and expected operational costs, integration costs, service and process customisation costs, service and process provisioning costs and architecture costs are assessed. Architecture costs are associated with acquiring artifacts for realizing the target architecture including hardware, software (OS), training, and network bandwidth. Service acquisition costs are based on a one-off or annuity-based payment for implementation and usage of the software and are very different from service provisioning costs. For complex business-related Web services, service providers may operate different charging alternatives, incurring various types of costs for the service aggregator, e.g., usage costs, subscription costs, leasing costs and free services.

Risks weigh benefits and costs in the provisioning scenarios and verify feasibility before a particular option is chosen. During risk analysis, both the impact and probability of an event that may occur is estimated. This is expressed in terms of Return on Investment (ROI) that is calculated for each development option as the ratio of net benefits over costs. After calculating the ROI for each scenario and assessing the technical quality of available service resources, the most appropriate design option may be selected. We concentrate on vertical service development.

Step 1.3: Planning
The planning plots a strategy for realizing new business processes, given the risk that is involved, adopting a suitable process model: high-risk projects may require a “Chicken-Little” approach advocating gradual migration whereas low-risk projects allow a “Big-Bang” strategy.

PHASE 2: SERVICE AND PROCESS ANALYSIS AND DESIGN
Service analysis aims at identifying, conceptualizing and rationalizing business processes as a set of interacting Web services. This step is logically succeeded by service design, during which conceptual processes and services are transformed into a set of related, platform-agnostic interfaces.

Step 2.1 Service Analysis and Design
Service interfaces are identified in the problem domain by carving out and grouping service capabilities based on service coupling and cohesion criteria. For example, consider the shipping service. By applying the dominant cohesion criterion, namely functional cohesion, this service is decomposed into service operations like “Select Shipper” and “Check Product Availability”.
In case reference models are available (e.g., the XML Common Business Library, xCBL, or RosettaNet), business functions can be derived on their basis. For example, the Order Management business process in Figure 2 could be derived on the basis of RosettaNet’s Partner Interface Process, “Manage Purchase Order” (PIP3A4).

Step 2.2 Service Specification
In the following we will briefly examine the process of writing interface specification for a Web service. We adopt the standard, XML-based Web Service Definition Language (WSDL) [2] for this purpose. This process basically comprises four steps: describing the service interface, specifying operation parameters, designating the messaging and transport protocol, and finally fusing port types, bindings and actual location (a URI) of the Web services. These steps themselves are rather trivial and already described in-depth in literature, e.g., in [4] a detailed approach for specifying services is outlined.

Step 2.3 Identifying Processes
The objective of this step is to identify services that may be aggregated into a business process whose interface has a high viscosity. Again, we can apply the design principles of coupling and cohesion to achieve this. For example the order management process has low communication protocol coupling

with the material requirements process. In case of top-down development, reference models are given that standardize processes and correlated services and serve as a point of reference for new services.

Process identification could, for instance, start with comparing the abstract business process layout with RosettaNet's "Order Management" PIP. This PIP encompasses four complementary process segments supporting the entire chain of activities from purchase order creation to tracking-and-tracing, each of which is further decomposed into multiple individual processes. A quick-scan of this PIP reveals that Segment 3A "Quote and Order Entry" and Segment 3C "Returns and Finance" may be combined into the process Order Management.

Step 2.4 Specifying Processes

Once business processes are extracted and their boundaries are clearly demarcated, they need to be described in the abstract in four steps.

A. Determine Style of Service Composition

We discern two basic types of orchestration from which the service designer may choose: orchestration and choreography.

Orchestration describes how web services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint. This is for instance the case for the process flow described in Figure 3 where the business process flow is designed from the vantage point of a supplier. Orchestration refers to an executable business process that may result in a long-lived, transactional, multi-step process model. With orchestration, the business process interactions are always controlled from the (private) perspective of one of the business parties involved in the process.

Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints, rather than a specific business process that is executed by a single party. Choreography is more collaborative in nature than orchestration. It is described from the perspective of all parties (common view), and defines the complementary observable behaviour between participants in business process collaboration.

Orchestration is the preferred composition style for internal business processes with tightly coupled internal actors, while choreography is typically chosen for developing cross-organizational business processes between loosely coupled partners. Orchestration and choreography may also be applied simultaneously.

For example, in Figure 3 the client and supplier are shown to integrate their business processes. This figure assumes that the respective business analysts at both companies have agreed upon the processes and SLAs involved for the process collaboration. Using a GUI and a tool that can serve as a basis for the collaboration, the client and supplier agree upon their interactions and generate a choreography, e.g., a WS-CDL (Choreography Description Language) representation [6]. This representation can then be used to generate a orchestration, e.g., Business Process Execution Language for Web Services (BPEL for short) workflow template for both the manufacturer and supplier. The two BPEL workflow templates reflect the business agreement.

In the following section we shall first concentrate on BPEL [3], which is the standard industry specification that is designed specifically for web services based orchestration.

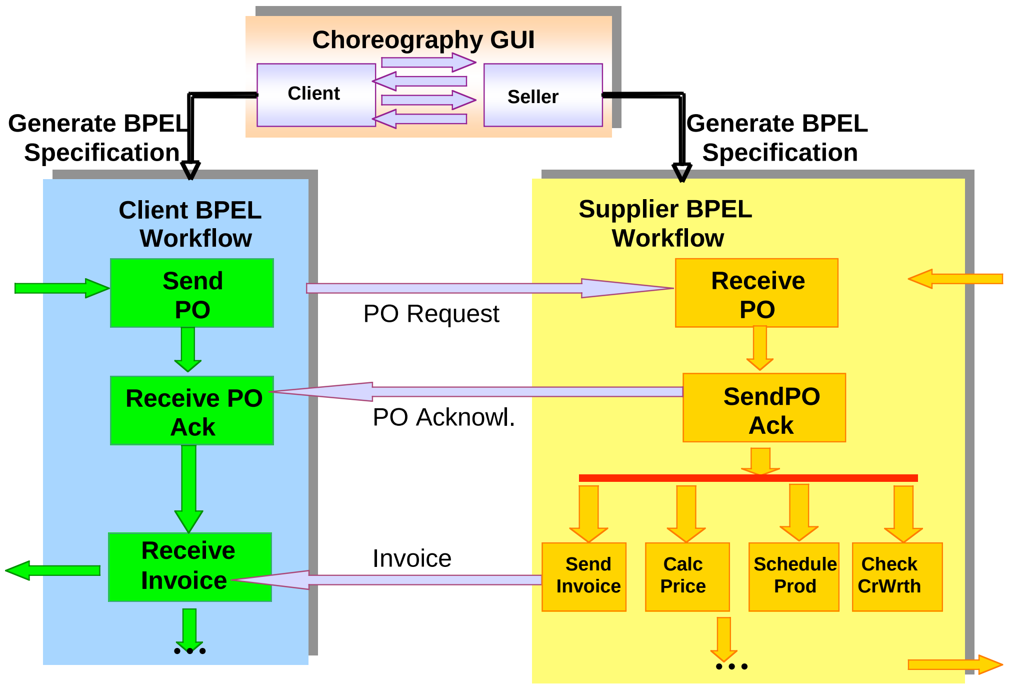

Figure 3 Combining choreography and orchestration.

B Determine objectives and derive the business process structure:
The private services of the client and supplier can then be scripted using BPEL. This language enforces the representational coupling requirement. In particular, BPEL is a platform-agnostic, block-structured workflow-like language to define business processes that may orchestrate one or more web services.
Key part of the BPEL document is the definition of the sequence of business activities required to handle an incoming purchase order request. The internal process flow of the supplier (see right hand side of Figure 3) includes an initial request from a client, followed by invocations of a scheduling service, credit service and billing service in parallel, and ultimately a response to the client from the seller sending a invoice.

C. Describe business activity responsibilities (roles).
The third step during business process design is to identify responsibilities associated with business process activities and the roles that are responsible for performing them. Roles may invoke, receive and reply to business activities. The result of this phase actually constitutes the foundation for implementing business policies, notably role-based access control and security policies. For example, the supplier may play the role of billing, credit checking and scheduling agency.

E. Identify Non-Functional Business Process Concerns
A service development methodology must go far beyond ensuring “simple” functional correctness, and deal with non-functional process design concerns including performance, payment model, security model, and, transactional behavior. In the following we will briefly mention typical business related, non-functional concerns that are captured in SLAs.
Service Level Agreements (SLAs) provide a proven vehicle for not only capturing non-functional requirements but also monitoring and enforcing them. SLAs are special legal agreements that

encapsulate multiple concerns, and symmetrically fuse the perspective of service supplier and customer.

Mutual agreements can be partially realized by specifying transactional Quality of Service (QoS) akin to business interactions that contain business-related atomicity properties including: business conversation-, payment-, (certified) delivery- and goods atomicity [5]. For example, the purchase order business process adopts a multi-level atomicity protocol signifying that after the customer is billed payment follows (payment atomicity), that producing the order and shipment results in the goods being delivered (goods atomicity) and being of correct type (certified delivery atomicity). Moreover, the terms of any negotiation sequence resulting in a purchase needs to be transcribed and kept in storage for future reference (conversation atomicity).

Another important ingredient of an SLA is security policies to protect multi-party collaborations. Knowing that a new business process adopts a Web services security standard such as WS-Security is not enough information to enable successful composition. The client needs to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which ones they prefer. Moreover, the client must determine if the service should communicate using signed messages. If so, it must determine what token type must be used for the digital signatures. Finally, the client must decide on when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service. For example, the purchase order service in the order management process may indicate that it only accepts username tokens that are based singed messaged using X.509 certificate that is cryptographically endorsed by a third party.

PHASE 3: THE REALIZATION PHASE

Once the service- and process specifications have reached a steady state, they need to be transformed into service implementations. We portray business process realization always assuming a verticalized development model. This phase encompasses two broad steps:

1. *Code Web Services*

Web services that were specified in WSDL may be programmed with any preferred programming language as long as the language is capable of supporting WSDL's protocol bindings including XML-RPC and SOAP.

2. *Code Business Processes*

Once the Web services have been codified, the flow logic that is comprised in the BPEL specification is compiled and injected into a BPEL execution engine. As a preparatory step, the business process is revamped as a coarse-grained asynchronous operation, which is, like any other service, defined in WSDL.

PHASE 4: THE DEPLOYMENT PHASE

The tasks associated with the deployment phase of the Web service development lifecycle include the following three steps:

1.Publish the service interface.

The publish operation is an act of service registration or service advertisement, e.g., using UDDI. It acts just like a contract between the service registry and the service provider.

2.Deploy the Web service and business process.

The run-time code for the service and process and any deployment meta-data that is required to run it are deployed during this step.

3.Publish service implementation details

The service implementation definition contains the definition of the network-accessible endpoint(s) where the Web service can be invoked.

PHASE 5: THE EXECUTION PHASE

During the execution phase, the business processes and supporting Web services are fully deployed and made operational. During this stage of the methodology, a service requestor can find the service definition and invoke all defined service operations. Also, the progress of running business processes can be monitored. For the time being only reactive monitoring mechanisms are in place.

DISCUSSION

Currently, enterprise applications that are constructed with Web services tend to be developed in a rather ad-hoc fashion. This unavoidably leads to processes and enterprise systems with fragile architectures, if any at all, which are hard to maintain, reuse and extend. Even more importantly, these practices do not take into account important business and organizational considerations, e.g., leasing costs of Web services, and organizational issues, making them unsuitable for constructing industrial-strength business applications.

The methodology that was presented herein reflects an attempt in defining a foundation of development principles for Web services based on which real-world business processes can be assembled into business scenarios. Currently, we are enriching the methodology with patterns, (design and coding) templates that are derived from empirical material. Next, we aim at developing an integrated toolset supporting the phases of the methodology.

REFERENCES

1. M.P. Papazoglou and G. Georgakapoulos, *Introduction to the Special Issue about Service-Oriented Computing*, CACM, October 2003, 46(10): 24-29
2. R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer and S. Weerarana, Web Services Description Language (WSDL) Version 2.0, March 2004, w3c.org
3. T. Andrews et al., Business Process Execution Language for Web Services, Version 1.1
4. G. Alanso, F. Casati, H. Kuno and V. Machiraju, Web Services: Concepts, Architectures and Applications, Springer, Heidelberg, 2004
5. M.P. Papazoglou, "Transactions in a Web Services World", WWW Journal, vol. 6, 2003
6. Kavantas et al., Web Service Choreography Description Language, Version 1, W3C Working Draft, August 2004

M.P. Papazoglou (mikep@uvt.nl) is a professor at Tilburg University in the Netherlands.
W.J. van den Heuvel (wjheuvel@uvt.nl) is an assistant professor at Tilburg University in the Netherlands.