Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned

Olaf Zimmermann
IBM Software Group
Gottlieb-Daimler-Strasse 12
68165 Mannheim, Germany
+49 171 970 6341
ozimmer@de.ibm.com

Vadim Doubrovski
IBM Global Services
2/90 Collins Street
Melbourne, Australia
+61 412 485 909
vdoubrov@au1.ibm.com

Jonas Grundler
IBM Software Group
Schönaicher Strasse 220
71032 Böblingen, Germany
+49 7031 164 914
lt97grun@de.ibm.com

Kerard Hogg
IBM Global Services
2/90 Collins Street
Melbourne, Australia
+61 413 016 719
kerahogg@au1.ibm.com

ABSTRACT

Effective and affordable business-to-business process integration is a key success factor in the telecommunications industry. A large telecommunication wholesaler, supplying its services to more than 150 different service retailers, enhanced the process integration capabilities of its core order management system through wide-spread use of SOA, business process choreography and Web services concepts. This core order management system processes 120 different complex order types.

On this project, challenging requirements such as complexity of business process models and multi-channel accessibility turned out to be true proof points for the applied SOA concepts, tools, and run-time environments. To implement an automated and secured business-to-business Web services channel and to introduce a process choreography layer into a large existing application were two of the key requirements that had to be addressed. The solution complies with the Web Services Interoperability Basic Profile 1.0 and makes use of executable business process models defined in the Business Process Execution Language (BPEL).

This paper discusses the rationale behind the decision for SOA, process choreography, and Web services, and gives an overview of the BPEL-centric process choreography architecture. Furthermore, it features lessons learned and best practices identified during design, implementation, and rollout of the solution.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures.*

General Terms

Design, Standardization, Performance, Security.

Keywords

Application Server, B2B, Best Practices, BPEL, Business Process, HTTP, J2EE, Order Management, Process Choreography, Process Integration, Service-Oriented Architecture, SOAP, Telecommunications, Web Application, Web Services, WSDL, Workflow, XML.

1. INTRODUCTION

Our client is the wholesale subsidiary of one of the largest telecommunications companies in the world. Naturally, a wholesaler has to provide its services to other companies – in this case to telecommunications service providers (retailers), which in turn offer these services to their own customers.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

In the telecommunications industry, in many countries deregulated over the last decade, the virtual service provider landscape is rather heterogeneous and subject to change. On the other hand, physical networks are often still owned and operated by a few large companies. The service procurement and order management processes in this industry are inherently complex and long running; many IT systems have to be interfaced and integrated with. This combination of business dynamics and domain-specific functional characteristics creates many challenges for IT solutions in the telecommunications industry. There is a constantly increasing pressure to reassemble existing solutions quickly on demand, and to reduce systems development and operational costs.

Service-Oriented Architecture (SOA) [5] concepts such as *Business Process Choreography (BPC)* [13], enterprise service bus [12] and Web services technologies [25] are a recent response from the IT industry to the challenges faced by the telecommunications and other industries. Our client is an early adopter of these concepts for enterprise-scale, mission-critical applications, leveraging them to improve its core order management system. This paper features the resulting SOA- and BPC-centric solution.

The remainder of the paper is structured in the following way:

- Section 2 introduces the business problems to be solved, functional and nonfunctional requirements, as well as technical constraints for the order management SOA.
- Section 3 features the architecture elements comprising the SOA developed in response to these requirements, along with their roles and responsibilities. Significant aspects of the BPEL-based process layer and the B2B Web services channel are detailed along the way.
- Subsequently, Section 4 takes a rear view mirror to the SOA implementation project and discusses project results, lessons learned and best practices identified.
- Finally, Section 5 concludes by summarizing thoughts and providing an outlook to future work.

2. BUSINESS CONTEXT AND REQUIREMENTS

This section outlines our client's order management scenario and related challenges to be addressed by the SOA- and BPC-centric solution. The section goes on to list the most significant functional and nonfunctional requirements to be addressed by the related solution implementation project and to discuss why SOA and BPC were seen as key solution elements.

2.1 The Telecommunications Wholesaler

The featured SOA project concerns a large telecommunications wholesaler with more than 150 customers, ranging from many small to a few very large customers. The wholesaler sells a full range of telecommunication products to these customers.

The business model is based on the wholesaler owning the physical network, cable and telephone exchanges. Customers include other telecommunications companies supplementing their own network infrastructure, and companies whose core business does not include telecommunications, but who have a desire to bundle telecommunication services with other products. The wholesaler is responsible for provisioning and configuring telephone services right up to the end user's premises. Customers of the wholesaler are expected to use the order management processes of the wholesaler to connect, configure, or disconnect telephone services for end users.

The wholesaler has a strategic imperative to drive down the cost of operations by improving its ability to interact and collaborate efficiently with its customers, and reducing manual processing and rework. A range of interaction styles has to be offered to suit both small and large customers, with the objective to deepen organizational and process integration. Having achieved these goals, the telecommunications wholesaler can use them as a differentiator in an increasingly competitive market.

2.2 Order Management Challenges

The project described in this paper targeted the order management for traditional Public Switched Telephone Network (PSTN) products. Specifically, project scope included provisioning and activation of new telephone services and catering for the moving of telephone services to new addresses anywhere in the country.

The steps required to complete these processes are inherently complex and involve interaction with eight existing legacy applications providing core telecommunication functionality. The same functionality has to be offered through two channels: an interactive, *browser-based application* and a set of *Web services* for automated business-to-business processing of requests.

The project enhanced an existing, complex application which includes about 3,000 Java classes. The application has been operational for four years, servicing in excess of 5,000,000 provisioning orders in that time, which were entered by up to 1,500 concurrent users. The current application provides for over 250 products incorporated in over 120 types of provisioning requests.

2.3 Key Functional Requirements

The enhanced version of the order management application has to support *two fundamental business processes* that have to be serviced through both the browser and the Web services channel:

1. Provide a new PSTN telephone service.
2. Move a PSTN telephone service to a new address.

Each process consists of many activities involving rather complex validation rules to be supported by functions implemented in multiple legacy applications. There is a strong focus on ensuring *validation completeness* to minimize manual processing and rework. Several validations, for example during address identification, require a progressive filtering approach which requires an interactive dialog.

Some activities involve the reservation of telecommunication resources. These resources are valuable; the order management solution therefore has to ensure that reservation of resources is only attempted as part of a genuine process. Any *reserved resources must be released* when the process fails to complete successfully in the permitted time window.

The following stepwise description of the 'Move a PSTN telephone service to a new address' process depicts these characteristics (note: the 'Provide a new PSTN telephone service' process is a subset of the 'Move a PSTN telephone service to a new address' process, not requiring the first step):

1. Identify the service to be moved and its current location or site address.
2. Identify the new address for the service. This has to be the address as recognized by the systems that record telecommunications plant and service information. Hence the validation is complex, and search aids are required.
3. Assuming a recognized address is identified, the next step is to search for transmission cable plant which exists at the target address and could be reused for provisioning this service. This is important for two reasons. The wholesaler wants to maximize the revenue being generated from cable plant. For the customer, it will cost less if existing plant is reused. In some cases the plant may still be in use, but its use may be scheduled for termination. In such cases, the timeframe will have to be assessed.
4. Having identified a particular copper transmission path, this intermediate result has to be recorded.
5. Determine the features of the service at the new address (e.g., transmission of calling line identifier). This step depends on a complex set of factors. Some features may already exist from a service which previously used the transmission path. Some features will be transferred from the service at the old address. Some new features may be requested in conjunction with the move.
6. Next, determine a phone number for the service at the new address and reserve it. There are a number of possibilities. If the new address is not serviced by the same telephone exchange switch as the old one, the customer has to choose a new telephone number. Alternatively, where the address permits, the customer may keep his/her existing telephone number. When choosing a new telephone number, a list of numbers available at the exchange must be supplied.
7. At this point in the process, enough information has been obtained to determine whether the service can be provided with or without a visit to the customer's new premises. If a visit is required, then a time must be negotiated which suits both the customer and the field staff to be assigned to the task. The chosen transmission path and its state, service features required and geographic location are all taken into account in determining the required tasks, how much effort they will take, the field staff's skills required, and when the visit can be scheduled. When a visit is not required, it may still be necessary to schedule tasks, but generally these are subject to less constraints and do not require negotiation.
8. In addition, it is common for business customers to have a number of services and to request to relocate these to a

new address in a coordinated manner. In circumstances where a visit to the customer's premises is required, it is advantageous to the wholesaler if only a single visit is undertaken. This requires an appointment that satisfies both the customer's constraints for being present during a site visit, and the constraints on the field staff required.

9. Lastly, the request to move the services from one address to another and the reservation of the resources required to complete the task is confirmed, allowing the commercial transaction to proceed.

At any point, the process may be abandoned, for example if certain timeout conditions occur. Business-level *compensation* is then required to undo the reservation of resources.

This high-level description shows that, inherent to the problem domain, the underlying process flow models can be rather complex. As discussed in later sections of this paper, these functional characteristics make the business process model challenging to implement – no matter whether a SOA-/BPC-centric or a different solution approach is chosen.

2.4 Key Nonfunctional Requirements

The order management solution supporting the two business processes described in the previous section must be accessible both over a private industry-sponsored network and the Internet.

Smaller customers require very low-cost entry points; hence a browser-based application interface is required. On the other hand, larger customers have a desire to leverage deeper process integration options as made available by business-to-business Web services. A third group of customers might want to employ 'batch' style integration (as already noted, the customers of the wholesaler are in turn telecommunication service providers serving end users).

Business volumes are approximately 20,000 'Provide a new PSTN telephone service' requests and 15,000 'Move a PSTN telephone service to a new address' requests per month. Given up to 20 steps per process, and a peak hour load of 30% above average, this equates to a peak load of about 4,550 steps executed per hour (based on core business hours of ten hours per day, 20 days per month). Initially, a process must be able to persist from first step to last for three hours; however, this time will be extended to up to 24 hours in the future.

The wholesaler's customers are spread across a number of time zones, operating 23 hours per day and seven days per week. The solution design has to be able to handle 2,000 concurrent users shared across the two channels (browser and Web services). Depending on the software architecture on the customer side, the number of concurrent users may not be a particularly relevant measure; for example, a customer system serving many concurrent end users may only use a single connection on the Web services channel.

Each instance of a business process must operate independently of others. Each process must be capable of managing the complex interdependencies between the steps. Both successful and unsuccessful processes must complete cleanly. Processes servicing the Web services channel have to be able to recover their state following an application server failure.

All messages on the public Internet must be encrypted. Each user must prove its authenticity by presenting a digital certificate. Users must be registered and authorized for each transaction type to be used. Response times and transaction volumes must be monitored in real time per transaction type and customer. Average response time targets vary by transaction type, typically 3-5 seconds; 95% of the transactions need to execute in 5-8 seconds.

2.5 Value of SOA and Business Process Choreography in this Scenario

An analysis of the functional and nonfunctional requirements outlined in Sections 2.3 and 2.4 led to identification of the following business value of introducing SOA, BPC and Web services technology:

- Increased *automation* and deeper *process integration*, as made possible by the introduction of a choreography component, lead to reduced manual effort and rework both in the wholesaler and the customer processes.
- Process and business rule *agility* leads to product and process changes being implemented faster and cheaper. Legitimate use of telecommunication resources can be ensured more easily, operational efficiency can be improved, and wastage reduced.
- A *variety of interaction styles* can be provided to customers of all sizes through the two channels, while still ensuring consistent application of business rules.

On the technical level, the following benefits can be identified:

- *Compensation* is an attractive BPC feature. A two-phase commit protocol cannot reasonably be applied to the services provided by the legacy applications due to the lifetime of the processes, timeout issues, and inherent lack of support for transactional integrity control in most of the backend applications.
- *Reuse* of shared application components is simplified. For example, address checks are required for both business processes, but implemented separately so far.
- *Standardization* allows replacing unsupported components with commercial-off-the-shelf software, which promises to simplify deployment and maintenance of the order management application. A custom workflow engine is an example, as well as the open source Web services protocol engine and XML processor in use.

3. SOLUTION OUTLINE

This section outlines the solution to the business problems presented in the previous section. First it describes the most fundamental architectural requirements that stem from the business context and requirements. It then lays out the solution and elaborates on the purpose and characteristics of the solution components. The section concludes with a more detailed look at some implementation decisions taken.

3.1 Architectural Requirements

The functional and nonfunctional requirements outlined above lead to the following major architectural requirements that the solution is based on:

- The use of atomic and conversational Web services.
- Managing enterprise business resources
- Implementing a channel-agnostic architectural pattern.

The following sections outline each of these requirements in greater detail.

3.1.1 *Atomic and Conversational Web Services*

The solution facilitates two types of Web services, *atomic* and *conversational,* that are exposed as external interfaces.

An interface can be defined as atomic if a single invocation of its operation triggers execution of a business process that runs until full completion of a business transaction. The input message contains the complete set of parameters necessary for the business process to run end-to-end (Figure 1).

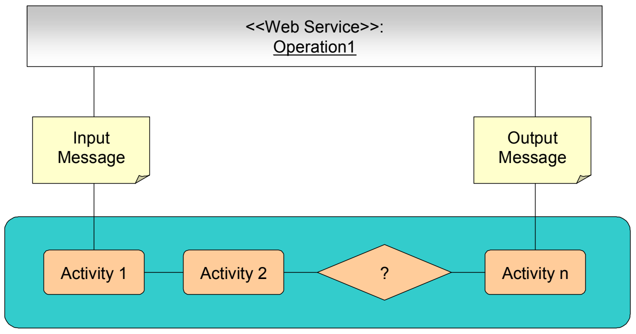

Figure 1: Business process triggered by atomic Web service

The solution exposes two styles of atomic interfaces as external Web services, both supporting the SOAP/HTTP protocol – *synchronous* and *asynchronous*. The synchronous style assumes that upon receipt of an HTTP request, the process in question completes in its entirety and then returns a full result from the business process to the client via an HTTP response. In contrast, Web service operations that use the asynchronous style return just a result identifier in the immediate HTTP response. It then becomes the client's responsibility to invoke another Web service operation after a while, providing the previously supplied identifier to retrieve the complete result.

An interface is required to be conversational when it is impossible to provide the complete set of the business process input parameters upfront. In such a case, the input information has to be determined incrementally, or gradually refined as part of the business process execution. This type of communication is shown in Figure 2.

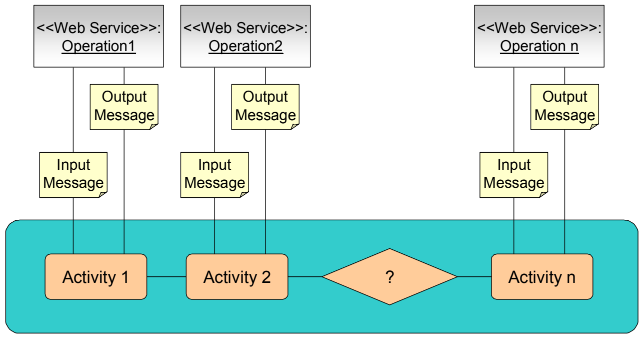

Figure 2: Business process and conversational Web services

It must be noted that regardless of the conversational nature of the communication with such Web services (Figure 2), the Web services consumers on the client side always initiate the conversation, and the server running the business processes always remains in listening mode.

Web service operations participating in the conversational pattern can also be synchronous and asynchronous. The implementation of the order management scenario, however, makes use only of synchronous conversational Web service operations.

Support of conversational Web services allows a wide variety of architectural patterns to be implemented on the client side. It can equally support both interactive human-driven clients as well as fully automated system clients that do not communicate with human users.

In the order management scenario for the telecommunication wholesaler this means that for smaller retailers it is possible to have a simple, perhaps Web-based, client that uses conversational Web services' responses to present the information to a human user who makes necessary decisions. On the other hand, larger retailers, striving to achieve higher level of automation can use more sophisticated, automated solutions that use the same set of conversational Web service operations. This provides for a wide range of interaction styles as motivated in Section 2.1.

The use of atomic Web services represents the common invocation pattern in traditional e-business architectures. It is well understood and widely implemented. Sometimes, however, the nature of more complex business processes dictates that the full set of process parameters can only be refined during the process execution. This calls for the use of the conversational Web service invocation pattern, which is, so far, not widely researched, implemented and described in the literature. Such a requirement exists for the order management scenario (see high-level workflow description in Section 2.3) and the rest of the paper is dedicated to the specifics of application of this pattern.

3.1.2 *Managing Enterprise Business Resources and Use of Compensatory Actions*

As motivated in Section 2.3, business processes may require the reservation of certain business resources during their execution. Business processes acquire such business resources from various parts of the enterprise and mark the resources as reserved for their exclusive use. If for some reason a business process does not complete successfully, it must make sure that the resources previously acquired are returned back to the pool of available resources. This is achieved by using *compensatory actions* that are defined for the relevant activities within the business process.

A single implementation of a compensatory action may have a number of different triggering mechanisms. Some triggers may be explicit and may relate to events for which explicit business rules exist within the business process. Other triggers may be time-related (i.e. temporal triggers) or may be related to certain exceptions thrown within the process in question.

3.1.3 *Channel-Agnostic Architectural Pattern*

Requirements for many business applications specify that the system must support a variety of communication channels. The solution described in this paper is not an exception; it is required to support both the traditional browser-based and a Web services access channel. While the nature of the information that is communicated through both channels is similar, the two channels have quite different modalities and have distinctly different ways of handling information.

According to architectural best practices, it is desirable to have a *single implementation* of business processes below the presentation layer, facilitating the work of both access channels. This ensures that the business process logic is only implemented and tested once, and maintained as a single code base supporting the two access channels.

This can be achieved by applying the *channel-agnostic architecture pattern* [6], in which the business process implementation is fully shielded from the presentation and modality details of each of the used channels.

3.2 SOA and Process Choreography Usage

The fundamental architectural requirements outlined above can be satisfied by a number of implementation options. Naturally, a custom-coded solution always remains an option. However, if the requirement for business agility and the ability to quickly adapt business processes and create new ones from available components is also taken into account, then the value proposition of adopting an SOA-based approach becomes highly attractive.

If it is possible to identify reusable parts of the business process, clearly define their interfaces and then use process choreography to assemble the reusable activities into a process implementation, the task of satisfying the channel-agnostic and conversational Web services requirements becomes achievable. The existing order management application has a history of using XML-based workflow engines that aimed to provide process choreography capabilities (see for example [7]). However, in the past it was difficult to define component interfaces using self-describing, openly standardized interface specifications which are now available, for instance Web Services Description Language (WSDL) [18] and Business Process Execution Language (BPEL) [4], [14] (to describe the workflow itself).

A decision to use the SOA model augmented with an open standard-based approach to business process specifications formed the foundation of the architectural solution presented in the following sections.

3.3 Component Model Overview

The solution presented in this section is based on the fundamental requirements described in the sections before. It leverages the Business Process Choreographer capabilities of the IBM WebSphere Business Integration Server Foundation (WBISF), but can be applicable to other Java 2 Enterprise Edition (J2EE) application servers with business process execution capabilities.

The solution's component model is shown in Figure 3. *Business Process Engine (BPE)* is a logical representative of the BPC functionality in WBISF, residing at the core of the solution's component model. BPE implements the Business Process Layer of the model, facilitating the execution of business processes.

Such business processes are defined by *business process specifications* comprising process flows, activities and compensations. Process specifications are expressed in BPEL. They are created for the Business Process Choreographer component of WBISF, using the WebSphere Application Developer Integration Edition (WSADIE) tooling environment including a BPEL editor. Once saved and published, BPEL specifications can be directly instantiated and executed within the BPE container available in WBISF.

Above the business process layer reside the *Presentation Layer* and the *Channel Controller Layer*, jointly accommodating channel-specific components. This approach allows for retaining all the channel-specific elements of the architecture at the Presentation Layer leaving the Business Process Layer totally channel-independent. This addresses the requirement for channel-agnostic implementation of business processes, which means that all communications between the two layers are implemented via exactly the same interfaces regardless of the channel. In other words, the business process layer does not use any channel-specific parameters as part of its logic.

The implementation of this principle is achieved by specifying interface contracts between the Presentation Layer and the business processes as Web services exposed only internally to the solution. These internal Web services are formally defined using WSDL specifications and can be invoked using the Web Services Invocation Framework (WSIF) [20] that supports bindings other than SOAP/HTTP. These well-defined, self-describing WSDL interfaces decouple the Process Layer from the two channels comprising the Presentation Layer, therefore providing a channel-neutral mechanism for invoking business processes.

Business process specifications are composed of sequences of activities that together achieve the desired outcome of the business process in question. Each activity is usually specified as a "black box" at the business process level. Therefore such activities are referred to as *Activity Stubs*, while the components that the stubs call are referred to as *Activity Implementations*. Each Activity Implementation also has a well-defined interface that is exposed internally using the *Business Service Façade (BSF)* pattern [1]. Similarly, each BSF exposes its interface as an internal Web service described in WSDL that can be invoked via WSIF.

Each activity represents a step in a business process and has a well-defined interface specified by its WSDL operation contract. A group of such Web service operations that can be invoked in a sequence represents the conversational Web service pattern (see Figure 2).

The use of BPEL for describing business process specifications allows for clear understanding as to what activities are involved in

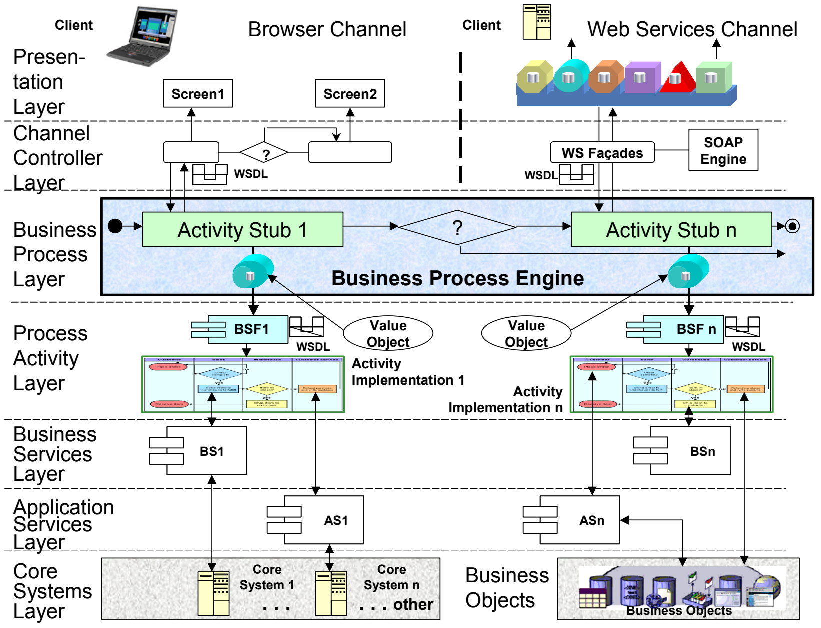

Figure 3: Component model that supports business process execution

each process and what transitions of control are possible between activities. Activities that are involved in reserving enterprise business resources can be marked and made visible from the business process specification. It is possible then to create compensatory actions that can be invoked from various stages of the business process if the process is to be terminated prematurely and the acquired business resources must be released.

Activity implementations that can be invoked through their BSFs can make use of other services, either *Business Services,* if a function understood by the business is implemented, or *Application Services,* for internal application functionality.

The fundamental principles of SOA assume that not only pieces of reusable functionality can be exposed as services (Web services described by WSDLs in this case), but also, in turn, such services can invoke other services as part of their business function implementation.

3.4 Business Process Layer

The implementation of the Business Process Layer is based on the BPE that instantiates and executes business processes. Within this layer, business processes are defined as long-running processes (or macro processes). This means that the full process state, consisting of BPEL *variables*, is maintained by the BPE in a persistent manner – all parts of the business process context are stored in a database (usually once per process activity execution) and can be restored at a later stage. A direct implication of this is that business processes can "live" in the system for hours and days (or even weeks and months) which is part of the application's business requirements as stated in Section 2.4.

In conversational Web service scenarios, there may be substantial time between Web services operation invocations, which requires the use of *long-running business processes*. Long-running business processes also allow business process instances to survive server reboot.

3.5 Presentation Layer and Channel Controller Layer

The Presentation Layer and the Channel Controllers support the different modalities of the two channels and mediate invocations of the standardized interfaces at the Business Process Layer.

Such mediation is relatively simple for the Web Services Channel, as the externally exposed WSDL interfaces of the conversational Web services are defined in close alignment with the internal WSDLs defined for each activity stub. There are only very few simple transformations that are implemented at the SOAP Engine and Web Service Façade Level.

Mediation is more complex for the controller in the Browser Channel, particularly if the user interface follows a page flow-oriented approach. Not only is the controller required to invoke the Business Process Layer's interface during request processing, it also has to maintain dialog state (for instance when more than one screen corresponds to a single process activity invocation to provide a rich user experience), support moving forward and backward through screens as required, handle validation errors, report them to the user and allow for repeatable data input.

3.6 Process Activity Layer

The Process Activity Layer provides Activity Implementations for each Activity Stub at the Business Process Layer. There is a fundamental architectural assumption that each activity implementation is short-lived. This differentiates activity implementations (that can also be viewed as business sub-processes) from the long-running processes at the Business Process layer.

The short-lived characteristic of activity implementations is dictated by the very nature of an external Web service operation implemented as an inherently synchronous SOAP over HTTP call. If the *in* message of a Web service operation is delivered as a HTTP request, the *out* (or *fault*) message must be delivered as the HTTP response to this request, otherwise the Web service operation in question will result in a time out. Therefore the execution of an activity implementation, triggered by the Process Layer, and occurring between the arrival of the HTTP request and formation of the HTTP response, must be short-lived.

Each Activity Implementation is also exposed (i.e. to the Process Layer) via a WSDL interface based on the signature of its BSF. BSFs represent Activity Implementations designed as stateless session beans. This defines another architectural principle of the solution that requires Activity Implementations not to keep state. The only layer within the architecture that keeps process state is the Business Process Layer. All data elements that are required for Activity Implementations to successfully perform their tasks are passed in (and out) via *Value Objects* (see Figure 3). Naturally, the structure of each Value Object is aligned with the corresponding BSF's signature reflected in the corresponding WSDL and XML Schema Definition (XSD).

3.7 Business and Application Services

Not only are parts of the system's architecture at the Business Process Layer and the Web Services Channel exposed as services, but a set of reusable components was identified as part of the design stage for the lower layer activity implementations. Such reusable components are defined as shared services within the system's architecture. They can belong to one of two distinct groups – *Business* and *Application Services*.

The difference between the two is important. Application Services can be defined as reusable components or pieces of reusable business logic that can be used and, more importantly, only make sense within the given application.

On the contrary, Business Services can be defined as reusable components or reusable pieces of business functionality that have initial affinity with the application, but largely can be used and make sense within the wider enterprise. An example of a Business Service is a *ListAvailableNumbers()* service briefly introduced in Step 6 within Section 2.3. It provides a list of telephone numbers available for allocation at a given telephone exchange.

An elegant way to define services is via WSDL interfaces. The immediate implication of this principle for Business Services is that their WSDL definitions must include only primitive fundamental data types or types that form part of the global enterprise's XSD library. In the future, an enterprise-wide service registry (e.g. UDDI [17]) possibly could assist in adoption and proliferation of the important Business Services.

Each such service with a well-defined interface represents a potentially reusable component. Ideally, such components could be deployed separately as separate deployment units (especially in the case of Business Services) and could be invoked via WSIF. It is worth noting that due to the availability of multiple protocol bindings in WSIF, the existence of a WSDL contract does not mandate the usage of SOAP/HTTP as a transport protocol at runtime.

3.8 Conclusions

The previous sections have shown that in order to satisfy the major architectural requirements and achieve greater business process agility, an SOA-based approach with process choreography is perfectly valid for the order entry management scenario introduced in Section 2.

This approach is based on a set of reusable process components using well-defined internal Web services, with self-describing WSDL interfaces, connected together by business process specifications defined in BPEL. Defining clear, consistent internal Web service interfaces to a single business process implementation achieves complete channel independence for both the Web browser and the Web services channel.

4. PROJECT RESULTS AND LESSONS LEARNED

Following the description why and how SOA and BPC concepts were leveraged in a telecommunications order management scenario, this section reflects on the results of the project delivery phase and good design practices harvested from it, starting with the overall results and then investigating technical details, project approach, and lessons learned.

4.1 Project Results

In response to the advanced workflow requirements described in Section 2, a BPEL- and Web services-centric SOA was designed, which can be accessed via a browser application as well as a Web services channel. The solution successfully went into production in April 2005. In the first week of operation in excess of 10,000 BPEL process instances were executed and the throughput continues to grow.

Throughout the first eight weeks of production operation, the solution was stable and exceeded both response time and reliability targets. Performance has improved in comparison to previous, non-SOA releases.

With regards to the maturity of the core Web services stack consisting of XML, SOAP, and WSDL, we can confirm the same positive experience as reported in an OOPLSA 2004 practitioner report on a project in the finance industry [24].

4.1.1 Process Modeling Aspects

We successfully implemented the two business processes in the scenario, 'Move a PSTN telephone service to a new address' and 'Provide a new PSTN telephone service'. These two processes are implemented as *subprocesses* of a generic 'Request' process.

All three processes approximately have the same size and complexity. There are more than 300 BPEL activities such as invoke, receive, reply, and assign, as well as Java snippets within each process model. Over 70 variables and ten while loops are defined. These while loops and the use of sequence and flow constructs required the introduction of 15 nested scopes. More than 30 Web service calls (invoke activities) appear in each of the three process models; more than 35 value objects and XSDs are used.

Due to significant upfront design work on BPEL process model patterns, XML namespace conventions, approach to exception handling, and other key design issues, the development of these processes progressed rather smoothly. The scenario to be implemented is complex and challenging (see Section 2); therefore, thorough preparation was required to contain the related risks and keep the problems encountered to a minimum.

As already mentioned in Section 3, we used a commercial J2EE application server and related tooling as our BPEL runtime and development environment. WSADIE, our Eclipse-based Integrated Development Environment (IDE), let us connect Web services to the BPEL layer rapidly, and allowed to generate helper classes for various XML-based artifacts, as well as Web services client proxies and server stubs. All required tools are integrated in the WSADIE platform; Web services and BPEL development therefore was a one-stop shopping experience.

With this approach, the vision of being able to rapidly develop and change business process rules and deploy changes at low cost is tantalizingly close. However, without additional tooling, just-in-time process deployment initiated on the business analyst level cannot easily be accomplished yet (at least not in an automated fashion). Furthermore, round-tripping remains a manual task.

In contrast, the quest for flexibility and business-level agility is an important element of the SOA value proposition (as outlined in Section 2). An apparent conclusion from this project is that an IDE and J2EE container hosting a BPEL engine alone cannot enable business analysts, let alone business users, to modify business process execution on the fly in response to an external market force, opportunity or threat. Additional tools and further development life cycle optimizations are therefore required to accomplish this vision, for example BPEL export and import capabilities in the business modeling tools used by the business analysts and domain experts.

4.1.2 Project Approach

Key to success was to schedule a development-level Proof-of-Concept (PoC) early in the project along with the high-level solution outline work. The PoC included a fit-gap analysis with regard to functional and nonfunctional requirements, current IT environment and SOA concepts, and was highly valuable in terms of training the project team on BPEL and Web services.

During the development phases of the project, we employed an iterative and incremental development style adopting many principles from the Manifesto for Agile Software Development [3], for example continuous delivery and collaboration. Before we initiated project activities such as BPEL process flow modeling, definition of WSDL contracts, and coding of key Java artifacts, we invested in an analysis phase involving several fact-to-face workshops within the architecture, development and system administration teams. Topics included basic requirements analysis, assessment of existing Java code and Web services interfaces, existing deployment cycles, maintenance and operations. It turned out that the decision to spend time on these analysis and coordination activities paid dividends, speeding up the ensuing BPEL and Java development significantly.

In summary, two of the most important general lessons learned on, and reinforced by, this project are to identify possible areas of concerns early and to define appropriate risk mitigation strategies before kicking off any premature implementation work.

4.2 Lessons Learned

Due to the size and complexity of this project, many lessons learned about large-scale usage of SOA and BPEL originate from it. This section reflects both on general development challenges and on BPEL technology-related issues.

Backward navigation and event-driven processes are cumbersome to model. BPEL is a Web services composition language, and vendor implementations typically add workflow capabilities, for example interactions with human users.

According to the experience gained on this project, the current BPEL specification has a fundamental limitation: activities are executed once in a defined sequence – there is no inherent modeling construct for representing an event-driven process in BPEL. Moreover, there is no notion of repeating a failed activity. This became apparent when the following process requirements had to be modelled (compare with general workflow description in Section 2):

1. The processes to be modeled in the order management scenario have to be tightly integrated with the Browser Channel – hence, a *step back* functionality had to be implemented in BPEL (resembling the browser's back button).
2. A typical scenario is that the business process *validates* a user's input. If the validation fails, the user must repeat this step.
3. While the user is navigating through the business process, he/she can *re-enter* the business process at a certain, predetermined entry point (and drop current inputs).

BPEL does not have a built-in mechanism to model these three requirements. Therefore, complex while loops, flows and sequences had to be used – more than 15 nested layers of sequences, flows and while loops were required.

A modeling approach based on event-driven finite state machines would be worth considering for this and many other scenarios.

Conversational process invocation models must be exposed on the client interface. BPEL processes interact with other services through *partner links*; either the BPEL processes or their clients can initiate such interactions (*invoke* versus *receive activities*). In conversational processes like those two implemented in this project, several receive activities are defined. Consequently, a conversational BPEL process exposes multiple interdependent Web service operations; related pre- and post-conditions for successful invocation of the receive operations exist. It is a challenge to communicate to partners as to which operations on the BPEL process are available at a given point in time. This challenge can be addressed partially by modeling the BPEL in such a way that the order of events is defined intuitively, and by annotating the WSDL definition of the process interface appropriately.

With regards to the human user interface for conversational processes, the time the BPEL engine needs to navigate from one step to the next has to be considered. Moreover, synchronization of the human user interface with the BPEL engine is a considerable challenge, as a conversational process requires a number of user interactions to be completed over its lifecycle. Two styles of human user interfaces for conversational processes are commonly used, *work lists* and *page flows*.

In the work list approach to dealing with multiple user interactions, work items representing interactive steps in a conversational BPEL process are created for each ongoing process activity step. The user is presented with a single, typically domain-independent work list, from which he/she selects a work item to work on.

The page flow approach often is perceived to be more user-friendly. An interface is provided that navigates to a specific input page for a particular process step following a completed one. This way a user can navigate through a business process step-by-step, without having to choose items from a generic work list.

Page flow-based user interfaces are more complex to synchronize with the process layer than work list-based ones. If the page flow-based approach is chosen, the synchronization of the user interface and the BPEL engine is a major challenge, introducing mutual dependencies between the user interface and the BPEL process. These dependencies must be designed explicitly.

Many BPEL-specific design decisions have to be taken. BPEL offers a variety of possible implementation alternatives for common design patterns. To ensure architectural and implementation-level consistency, a lot of fine-grained design decisions have to be taken in addition to the overall architectural decisions, for example regarding layering. Examples include interoperability and other protocol issues, BPEL and WSDL modeling and mapping details (e.g. structure of partner links), guidelines for usage of BPEL variables and correlation sets, and error handling strategy (e.g. SOAP fault elements versus Java exceptions).

A pragmatic approach leveraging existing experience and assets quickly led to satisfying solutions in all mentioned areas. For instance, we could have implemented all BPEL variable mappings in *pure* BPEL, that is, with XPath [22] or XSLT [23]. As strong skills in this area had not been built previously, we decided to use basic BPEL assign activities for simple mappings, and Java Snippets, a nonstandard implementation-level extension to the BPEL specification, for complicated mappings.

Multiple technology stacks are involved. Elements of risk likely to be perceived on SOA/BPC projects are a rather steep learning curve and initially a negative impact on developer productivity, originated by the fact that it is quite difficult for a single practitioner to master all involved technologies such as BPEL, Web services (WSDL and SOAP at a minimum), XML, and J2EE in parallel.

On this project, we had invested in building deep skills on development level right from the beginning. We ensured that selected members of the team were familiar with the technical implementation of all application layers. This investment in broad and deep education turned out to be very valuable, particularly during integration testing and defect fixing.

There is an impact on deployment and build cycle. During the early test phases of the project, the team was confronted with a deployment and build cycle that consumed more time than that on previous project stages. Enabling BPEL support increased the footprint of the involved J2EE tools and runtimes (in terms of memory requirements, class path management, etc.), as many additional artifacts had to be created, deployed, and maintained. The resulting overhead slowed down the testing and defect fixing process initially.

Analysis and refactoring of the deployment and build process decreased the consumed time to 30% of the original time needed. An additional pre-build cycle on a dedicated workstation helped to identify build process issues and ensured a basic level of quality before deployment to the actual test system. We also parallelized steps like tagging in the version control system and the early build stages, and assigned build and deployment activities to a specifically trained team member. Further advice regarding deployment process improvements is as follows:

- Avoid storing design artifacts redundantly; create modules and dependencies such that an artifact only exists once (e.g. XSD for a business object).
- Sort artifacts by type and by meaning (facilitated by naming conventions for packages and namespaces), and share application server configurations within teams.
- Define role-specific Enterprise Archive (EAR) assemblies for the different types of developers, e.g. BPEL developers, JSP developers, and Enterprise Java Bean (EJB) developers.

Following these general considerations, project-specific build path analysis, and insight into product tuning opportunities made it possible to bring the speed of the build and deployment up to an acceptable level.

Unit, integration and load test drivers have to be defined. At the very beginning of the project, we built a test application that exposed all key process characteristics. The test application was developed using the same data structures, input/output volumes, protocols, and communication patterns as they were to be found in the target application.

This test application was used to assess system performance at a rather early stage. Furthermore, it allowed us to identify potential problem areas in the application and the BPEL engine before we actually started testing our target processes.

At all stages, we tested our business processes using JUnitEE [11]. WSADIE generates Java proxies for interaction with the business processes. These SOAP client proxies made it simple to create test cases for the communication with the BPEL processes. The generated proxies were used in JUnitEE classes to drive the BPEL processes without requiring a Web client or other external systems. This approach worked very well.

During the implementation of the business processes, the test cases were created in parallel by different developers – as a side effect, these test cases evolved with the evolution of the business processes. We saved a significant amount of time with this approach (compared to a manual approach to testing).

As soon as the real business process implementations were close to being ready for production deployment, we started load testing with them rather than the test application, stubbing out the Web services invocations. The JUnitEE test cases we had developed during the time of process development could be easily re-organized into test servlets driven by load testing tools.

4.3 BPEL Best Practices

Due to its scope, this project was ideal to mine best practices in almost all stages of the project lifecycle. This section presents only the most relevant and valuable ones.

SOA and BPEL must provide real value to the business. A best practice that cannot be overstated is that before any detailed technology decisions can be made, it must be proven that an innovative approach (here: SOA, Web services and BPEL usage) is applicable and provides real business value from a project stakeholder perspective. There simply is no way to justify the use of technology for its own sake. Section 2.5 of this paper justified the rationale of the usage of SOA and BPC in the order management scenario.

Process choreography support has to be positioned carefully in the overall architecture. Once a decision has been made for BPEL (and Web services, if not existing already), one has to carefully decide where and how BPEL processes are introduced.

A first key decision is whether a process layer should be introduced explicitly, or whether BPEL support is viewed just as yet another logical component that provides higher-level Web services. It also has to be decided which service consumer components communicate with a business process, and which protocol is used. There are long- and short-running business processes, which expose fundamentally different transactional semantics and quality-of-service characteristics. Further architectural decisions are related to *granularity* of the Web services that are invoked from the business process, as well as the granularity of the Web service interface that is exposed by the business process itself.

Section 3 of this paper discussed a subset of these architectural decisions in the context of the development of the telecommunications order management SOA.

Not everything is a BPEL process. BPEL and Web services are highly attractive concepts at present receiving much attention in industry and academia. BPEL is an XML language with operational semantics; its control constructs are comparable to those found in programming languages such as Java, C++, or C# (loops, conditions, and fault handlers just being three examples).

These characteristics make it difficult to identify the best use of these technologies from a business and an architectural perspective. Certainly not each and every logical building block of a solution has to be implemented as a BPEL flow just because technology and tools are available and capable of doing so.

Some guidelines that help identifying scoping the usage of BPEL on a project are as follows (also see discussion in Section 3):

- BPEL usage should focus on turning discrete units of business function into a business process, rather than on micro-level algorithms and direct manipulations of persistent data structures. Such algorithms and database access functions should reside in lower level layers of the overall architecture and be exposed through internal WSDL interfaces for consumption in business process flows.
- Presentation layer dialogs are different from business processes (page vs. activity flows) – intermediate steps that simply capture information from a user are best reflected in dialog control frameworks such as Apache Struts [2] or Java Server Faces (JSF) [10] applications.
- Even if a solution exposes a process layer, atomic (non-conversational) Web services might co-exist with this process layer. Depending on the functional requirements to the solution, it is still valid to make direct calls to such atomic services. Facilitating reuse, such atomic Web services can also be invoked from the BPEL process layer, serving as activity implementations.

Consciously decide for or against subflow usage. Frequently, discussions about an actual BPEL design include the usage of *subflows*. From an SOA perspective, a subflow represents a component or module of a higher-level process service. We can also use subflows to further divide large processes – theoretically, a very modular BPEL design can be created with such an approach.

On the BPEL implementation level, subprocesses introduce yet another layer of abstraction. Placing certain functionality into a subflow might decrease performance due to increased component initialization needs and related communication overhead. On the other hand, subflows can be a method to parallelize process development tasks, decrease the complexity of a single process and allow employing a fine-grained unit testing strategy.

Before introducing multiple layers of subflows, one has to assess the tradeoffs and implications such as impact on performance, complexity of the overall process model, as well as deployment and maintenance issues (more artifacts have to be managed).

In our scenario, usage of subflows is limited to modeling the relationship between the generic Request process and the two specific instantiations 'Provide a new PSTN telephone service' and 'Move a PSTN telephone service to a new address'.

A "pure BPEL" principle should be followed – if feasible. Typically, a solution has to meet many different non-functional requirements, including (but not limited to) being compliant to standards, adhering to a strict layering scheme and writing understandable and maintainable code.

Many of these goals can be competing even for non-BPEL projects. From an application architect's perspective, a business process implementation in *pure* BPEL is best – no proprietary, but only standardized language features should be used. However, as the BPEL standardization is not finalized at the time of writing, it sometimes may be necessary to use vendor extensions such as staff resolution and staff activities for human interaction support.

Not using extensions for the sake of standards compliance can cause extra development efforts and have a negative impact on the nonfunctional characteristics of the solution (if a proprietary, but optimized solution exists). For instance, from a runtime performance perspective, using a Java snippet in a BPEL process to invoke an EJB is faster than wrapping the EJB in a Web service and then invoking it via an activity that calls the EJB Web service. On the other hand, it is desirable to implement processes in a fashion that is compliant with the BPEL specification.

These considerations lead to the general advice to prioritize conflicting requirements, particularly nonfunctional ones.

Business process data placement has to be managed. Often, an area of concern is the management of state variables that are bound to a business process. Issues similar to those arising when using HTTP session objects in a J2EE Web container can occur, for example related to scalability, fault tolerance, and performance. BPEL variables are global within a process, which can possibly lead to undesired side effects in case several activities read and update the same variable.

Good practice is to keep the amount of data that is passed to and kept within business processes as limited as possible, as BPEL engines hosting long-running business processes typically store this data in a process database. Therefore, we recommend limiting the data held within business processes to a minimum. However, it is not always worth introducing costly changes to an existing design just to restrict the size of such data. In our project, the overall size of data has been calculated to be up to 2.5 megabytes, spread over more than 70 variables per process. Compared with HTTP session objects, this is huge – but, as this data is just stored in the database, it does not always affect performance and memory allocations; data is uploaded to memory only when required.

General alternatives to state management in a process layer are pushing this responsibility out to the client, or using a dedicated database shared by process and service components. A conscious architectural decision for one (and only one) of these alternatives should be taken.

Artifact dependency and asset management is required. In a BPEL project, a number of artifacts have to be managed – for example those outlined in Figure 4.

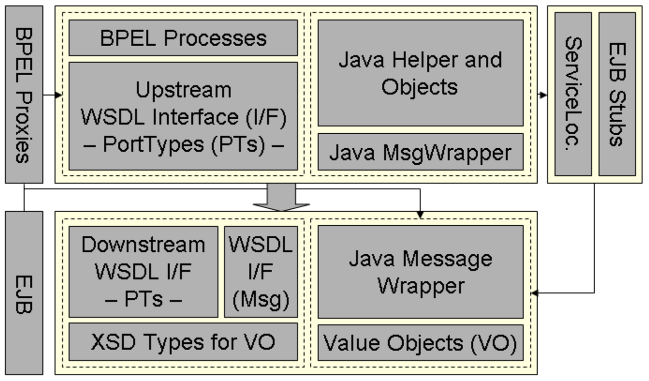

Figure 4: BPEL project artifacts and dependencies

Many of these artifacts depend on each other, and there are dependencies from and to the code generators in use (e.g. Value Object generators taking XSD documents as input). Versioning of process definitions (and their instances) is another key issue.

We recommend keeping the number of redundant artifacts (e.g. Java representations of Web services) to a minimum. Setting up a common repository keeping those artifacts is imperative. Naming conventions should be defined before development starts – this affects BPEL and WSDL element names (and package structures or positions within the file system) of generated components like client-side Web service invocation proxies.

5. CONCLUSIONS AND OUTLOOK

In this paper, we described how we leveraged SOA concepts, BPC/BPEL and Web services technologies to implement the next generation of a large multi-channel order management application for a telecommunication wholesaler. Deeper integration and flexibility of the process chains between the wholesaler and its customers, as well as cost savings originating from a higher level of automation, business rule validation, and service reuse were among the anticipated business benefits that suggested a BPEL-centric SOA for this scenario. Technical benefits included separation of concerns through strict layering, improved resource management via compensation and runtime reuse of shared application functionality made available as business and application services.

Overall, our experience with the foundational Web services technologies such as WSDL and SOAP was very positive and resembled that gained on previous projects. A fully automated, secure B2B Web services channel has been running in production successfully for about two years now; service consumer applications implemented in a variety of programming languages have access to the order management system. Seamless interoperability is ensured by the WS-I Basic Profile [19].

Our experience with still evolving SOA concepts such as BPEL is twofold. The value proposition of business process choreography with BPEL is promising and presents a perfect match for this rather complex inter-company process integration scenario. We successfully implemented two key business processes; the solution has been running smoothly in production since April 2005, meeting all functional and nonfunctional requirements.

However, according to our experience technology and implementations still have some way to go before all promises can be fulfilled. The first generation of BPEL tools exposes many technical details to the developer, and the step from business analysis to development is semi-manual or manual. Cross-discipline model exchange and code generation capabilities are needed to align business and IT more closely; second-generation tools are only beginning to address these requirements.

We outlined the following key findings regarding architectural positioning and development process:

- Event-driven process modeling and backward navigation between process activities is not well-supported; BPEL process modeling cannot easily be based on a finite state machine metaphor.
- Conversational process invocation models must be exposed on the client interface; human user interfaces providing page flows are more complex to synchronize with the process layer than work list-based ones.
- Many BPEL-specific design decisions have to be taken.
- Many technology stacks are involved, which leads to significant training efforts.
- There is an impact on the development and test process (in terms of turnaround times and resource requirements), which can be countered through refactoring.
- The test strategy has to account for BPEL usage; unit, integration, and load test drivers have to be defined with BPEL- and service-oriented design in mind.

Just like on any other nontrivial enterprise application development project, a structured architectural decision making process has to be employed; in addition to general decisions such as overall component model, platform selection, and capacity planning, many SOA- and BPEL-specific decisions have to be taken. These decisions include the strategy for state and transaction management, as well as exception handling in the process layer.

Many good practices can already be identified, including:

- Not every component qualifies to be implemented as a BPEL process (macro- vs. micro-level programming); BPEL should not be used for implementing something that is not a business process.
- Conscious use of the subflow concept is recommended.
- A pure BPEL philosophy should be followed, usage of vendor extensions reduced to areas not yet covered by the BPEL specification (e.g. human user activities).
- Business process data placement has to be managed.
- Artifact dependency and asset management is required.

Many of the issues we encountered and worked around in this project are inherent to the complexity of the problem domain; however, several of those described in this paper can be regarded as limitations of the BPEL technology presently available. BPEL therefore has to be assessed as not fully mature yet, even though it created a lot of value on this project already.

For the future, we consider pursuing several additional steps and directions. First, we intend to compare our project results and lessons learned with those from other large-scale BPEL projects, with the intention to harden them into true best practices.

Regarding the telecommunications wholesaler scenario, more order management processes can be supported by BPEL, and several other application domains can benefit from BPC technology in the future. It is intended to evaluate these opportunities after having performed a retrospective technology value assessment for this project.

As a separate project, work is underway to define a wholesaler–wide enterprise SOA model; we also investigate more formal service modeling and asset management approaches to facilitate successful reuse of business services.

6. ACKNOWLEDGMENTS

This paper is the result of a joint project and community effort. In particular, we would like to recognize the following individuals (in alphabetical order): Tony Altamura, Michael Bader, Adrian Berry, Peter Bittner, Neil Buckingham, Gary Chang, Peter Chilcott, Rohan Cluning, Steve Davies, Daren Deppeler, Mark Endrei, Marc Fiammante, Dagmar Fink, David McGorrery, Justin Graham, Meltem Haylaz, Jeffrey Heyward, Satish Kanchanna, Thomas Kasemir, Matthias Kloppmann, Neil Mulholland, Wendy Neave, Ying Ni, Spiro Paleologoudias, Bobby Papadopoulos, Gerhard Pfau, Mandeep Sahni, Tabet Tabet, and Kenneth Worland.

7. REFERENCES

[1] Alur, D., Crupi, J., Malks, D. *Core J2EE^{tm} Patterns – Best Practices and Design Strategies,* Prentice Hall, 2003.

[2] Apache Struts, http://struts.apache.org

[3] Beck K. et al, *Manifesto for Agile Software Development*, http://agilemanifesto.org

[4] Business Process Execution Language for Web Services Version 1.1, available from http://www.ibm.com/ developerworks/library/specification/ws-bpel

[5] Component Based Development and Integration (CBDI), Insight for Web Service & Software Component Practice, http://www.cbdiforum.com

[6] Doubrovski V., *Channel-Agnostic Architecture Pattern for e-Business Applications. Proc. of APITAC 2004, Kuala Lumpur*, IBM, 2004.

[7] Doubrovski V., *Towards Formal Specification of Client-Server Interactions for a Wide Range of Internet Applications. Proc. of WI2001, Japan, Oct 2001*, Springer-Verlag, 2001.

[8] Ferguson, D. F., Storey T., Lovering B., Shewchuk, J., *Secure, Reliable, Transacted Web Services*, IBM and Microsoft 2003, http://www.ibm.com/developerworks/ webservices/library/ws-securtrans

[9] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

[10] Java Server Faces, http://java.sun.com/j2ee/javaserverfaces/index.jsp

[11] JUnitEE, http://www.junitee.org

[12] Keen M. et al, *Patterns: Implementing an SOA using an ESB*, IBM Redbook 2004

[13] Leymann F., Roller D., Schmidt, M. T., *Web Services and Business Process Management*, IBM Systems Journal, Vol. 41, No 2, 2002

[14] OASIS Web Services Business Process Execution Language (WSBPEL), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[15] Service-Oriented Architecture and Web Services, IBM, http://www.ibm.com/services/us/imc/html/soa.html

[16] Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[17] Universal Description, Discovery and Integration (UDDI), http.//www.uddi.org

[18] Web Services Description Language (WSDL), W3C Note, http://www.w3.org/TR/wsdl

[19] Web Services Interoperability Initiative (WS-I), http://www.ws-i.org

[20] Web Services Invocation Framework (WSIF), http://ws.apache.org/wsif

[21] Workflow Patterns, http://www.workflowpatterns.com

[22] XPath, http://www.w3.org/TR/xpath

[23] XSLT, http://www.w3.org/TR/xslt

[24] Zimmermann O., Milinski M., Craes M., Oellermann F., *Second Generation Web Services-Oriented Architecture in Production in the Finance Industry*, OOPSLA Practitioner Report, 2004

[25] Zimmermann O., Tomlinson M., Peuser S., *Perspectives on Web Services – Applying SOAP, WSDL and UDDI to Real-World Projects*, Springer-Verlag, 2003.