Optimization of WS-BPEL Workflows through Business Process Re-Engineering Patterns

Jonas Buys, University of Antwerp, Belgium
Vincenzo De Florio, University of Antwerp, Belgium
Chris Blondia, University of Antwerp, Belgium

ABSTRACT

With the advent of XML-based SOA, WS-BPEL swiftly became a widely accepted standard for modeling business processes. Although SOA is said to embrace the principle of business agility, BPEL process definitions are still manually crafted into their final executable version. While SOA has proven to be a giant leap forward in building flexible IT systems, this static BPEL workflow model should be enhanced to better sustain continual process evolution. In this paper, the authors discuss the potential for adding business intelligence with respect to business process re-engineering patterns to the system to allow for automatic business process optimization. Furthermore, the paper examines how these re-engineering patterns may be implemented, leveraging techniques that were applied successfully in computer science. Several practical examples illustrate the benefit of such adaptive process models. These preliminary findings indicate that techniques like the re-sequencing and parallelization of instructions, further optimized by introspection, as well as techniques for achieving software fault tolerance, are particularly valuable for optimizing business processes. Finally, the authors elaborate on the design of people-oriented business processes using common human-centric re-engineering patterns.

Keywords: Business Process Re-Engineering (BPR), Business Processes, Service-Oriented Architecture (SOA), Workflows, WS-BPEL

1. INTRODUCTION

A cutthroat competition is currently raging between enterprises in which companies are compelled to constantly evolve in order to realize a competitive advantage. This goal of attaining market leadership is pursued by iteratively altering business processes\(^1\) and strategies aimed at improving operational efficiency (Reldin & Sundling, 2007). Business processes are thus continuously refined, mainly to resolve recurrent issues and as such rectify process performance. This concept is commonly referred to as business process re-engineering (BPR)\(^2\).

Large enterprises have extensively deployed information technology (IT) systems, and have recently started to automate their business processes. Regrettably enough, most

DOI: 10.4018/jaras.2010070102

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
of these volatile business processes are encased into rigid IT systems and this imposes limitations with respect to the speed with which changes are possible. In the beginning of this decade, this issue led to the concept of service-oriented architectures (SOA) in which IT is flexibly structured to better alleviate the re-engineering of processes by splitting up so-called business logic into a number of software components that are exposed as services (Erl, 2005). With service (operations) as an implementation for individual process activities, a business process can be automated by appropriately orchestrating and coordinating a set of services. Actually this service-oriented computing paradigm has adopted the best practices in distributed computing of - roughly estimated - the past twenty years, and commercially backed by major industry concerns, SOA continues to gain adherence (Stal, 2006).

As one possible SOA implementation technology, web services have managed to become the de facto standard for enterprise software in which various distributed, heterogeneous software systems are integrated in support of corporate e-business and e-commerce activities (Erl, 2005). A web service is typically exposed through a well-defined open XML interface described in the Web Services Description Language (WSDL) document that formally describes the syntax of application-specific messages in XSD Schema format (Microsoft & IBM, 2001) (Erl, 2009). Clients communicate with a web service through an endpoint reference that represents the address and context path where the service is deployed (Microsoft, Sun Microsystems & Computer Associates, 2006). The Web Services Business Process Execution Language (WS-BPEL) XML language is one of the standards that resulted from intensive standardization initiatives by industrial consortia, and shortly became a widely accepted standard for workflow modeling (Abode et al., 2007). The benefit of the central BPEL orchestration component is that the process definition is no longer interwoven inside the implementation code of the business logic. Because of this separation, SOA is said to alleviate the transformation and restructuring of business processes using highly reusable services that can easily be orchestrated into BPEL workflows (Erl, 2005).

The service-oriented paradigm turned out to be a giant leap forward in the construction of flexible IT systems indeed. XML-based SOA with BPEL further added to business agility, allowing for the quick development of new business processes leveraging service-wrapped legacy IT assets (i.e., business process redesign). But in spite of the popularity of BPEL and its clear separation of process and business logic, there remain some shortcomings (Buys, 2009; Modafferi, Musi, & Pernici, 2006). One of these issues is that a BPEL process definition is extremely static: it is designed manually using some software tools and is then loaded into the BPEL engine. Since service orchestration and business processes are at the core of SOA, it is imperative to continuously optimize BPEL process definitions to achieve an increase in system performance, besides having economic implications in realizing a competitive advantage required by the actual continual process evolution.

Although the BPR methodology originated in the early Nineties, until recently, businesses were still generally managed using an approach based on experience and intuition. As BPR is gaining adherence, we are on the verge of unifying the IT-driven service-oriented paradigm and the BPR managerial methodology: automatically applying prevailing BPR principles to BPEL process definitions can help in the further optimization of these process models, thereby help sustain process evolution.

This article starts with an introduction on how BPR patterns can be applied to WS-BPEL process definitions using established techniques from computer science (section 2). Next, in section 3, we illustrate the applicability of BPR patterns to BPEL workflows, and show how this can result in performance improvements, such as a reduction in execution time. Sections 3.1 to 3.3 will then elaborate on the resequencing and parallelization of process activities, further optimized by introspection, after which the relationship is examined between techniques
for achieving software fault tolerance and critical process activities. Lastly, section 3.5 will highlight the key role we envisage for human-centric re-engineering patterns in the design of people-oriented business processes.

2. BUSINESS PROCESS RE-ENGINEERING AND BPEL

Numerous BPR principles (best practices, design patterns, heuristics) have been proposed in the literature (Reijers & Liman Mansar, 2004), yet there has not been any thorough inquiry into combining IT and BPR so far. In order to support a higher level of process agility, we propose to design an intelligent system able to optimize the BPEL processes in accordance with these conceptual BPR principles. An overview of some potentially useful patterns for BPEL process improvement is shown in Table 1.

The WS-BPEL XML language defines a set of primitives with which business processes can be modeled: basic activities (receive, assign, invoke, reply, etc.) can be set in order using control and data flow supported by structured activities (e.g., sequence, loop, pick) (Abode et al., 2007; Erl, 2005). These rudimentary structural activities turn out to be limited to the common control and decision structures available in most imperative programming languages.

It is no surprise, then, that we can spot similarities between the techniques for program optimization in computer science, operating on atomic units of instruction and the BPR patterns for business processes, acting on coarser units of instruction blocks with a variable size: service operations. Consequently, it is plausible to try and automate economic BPR patterns leveraging existing techniques from computer science.

One possibility to combine both disciplines is to add BPR intelligence into workflow design tools that will preprocess and transform the process model prior to its execution. Alternatively, attributing business intelligence to SOA could allow for the dynamic application of BPR principles to the original static BPEL process definitions, aiming at the optimization of the process at runtime depending on the system’s current state and resource availability. This runtime information may be used to adjust

Table 1. Some business process re-engineering patterns

<table>
<thead>
<tr>
<th>BPR directives</th>
<th>Basic techniques</th>
<th>Human-centric</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>resequencing</td>
<td>data and control flow analysis (Klopp &amp; Khalaf, 2007)</td>
<td>No</td>
<td>3.1</td>
</tr>
<tr>
<td>parallelization</td>
<td>Tomaszulo, scoreboarding (Hennessy &amp; Patterson, 1990)</td>
<td>No</td>
<td>3.1</td>
</tr>
<tr>
<td>exception</td>
<td>control flow and flow variable speculation (Hennessy &amp; Patterson, 1990)</td>
<td>No</td>
<td>3.2</td>
</tr>
<tr>
<td>knock-out</td>
<td>control flow and flow variable speculation (Hennessy &amp; Patterson, 1990)</td>
<td>No</td>
<td>3.3</td>
</tr>
<tr>
<td>(minimize</td>
<td>transactional support (Charfi et al., 2007)</td>
<td>No</td>
<td>3.4.2</td>
</tr>
<tr>
<td>process</td>
<td>redoing, design diversity (Randell &amp; Xu, 1995; Avizienis, 1995)</td>
<td>No</td>
<td>3.4.3</td>
</tr>
<tr>
<td>cost</td>
<td>chain of execution (IBM, SAP et al., 2007a)</td>
<td>Yes</td>
<td>3.5.1</td>
</tr>
<tr>
<td>reliability</td>
<td>nomination (IBM, SAP et al., 2007a, 2007b)</td>
<td>Yes</td>
<td>3.5.2</td>
</tr>
<tr>
<td>dependability</td>
<td>chain of execution (IBM, SAP et al., 2007a)</td>
<td>Yes</td>
<td>3.5.3</td>
</tr>
<tr>
<td>order assignment</td>
<td>Delegation, escalation, process administrator role (IBM, SAP et al., 2007a)</td>
<td>Yes</td>
<td>3.5.4</td>
</tr>
<tr>
<td>distribution</td>
<td>Delegation, escalation, process administrator role (IBM, SAP et al., 2007a)</td>
<td>Yes</td>
<td>3.5.4</td>
</tr>
</tbody>
</table>
either the overall process model or individual process instances. Obviously, this second approach is more powerful than the former which is operating exclusively at design-time, as it enables the system to tune a process taking into account the system’s running internal state as well as environmental conditions.

3. BUSINESS PROCESS RE-ENGINEERING PATTERNS

In this section, some examples will illustrate that applying BPR patterns to WS-BPEL processes can have a beneficial influence on the overall performance of the process model. We suggest some techniques from computer science techniques with the potential to implement these patterns.

For the purpose of clarity, the examples are presented in business process modeling notation (BPMN), a common graphical representation of the actual XML BPEL definition. This does not limit our contribution, as BPEL can easily be mapped to BPM and vice versa (Stephen, 2005). For detailed information about the WS-BPEL 2.0 specification, please refer to (Abode et al., 2007).

3.1. Re-Sequencing and Parallelization BPR Patterns

The execution of a WS-BPEL process is essentially sequential, though the WS-BPEL specification also contains syntactical facilities for executing activities in parallel (Abode et al., 2007). As a primary BPR pattern, the execution order of process activities, i.e., service invocations, can be optimized by considering data flow dependencies so as to execute mutually independent activities in parallel (Reijers & Liman Mansar, 2004). The underlying idea of simultaneously executing activities and advancing activity initialization is that some time can be gained by avoiding performance-degrading stalls caused by dependencies. Throughout this paper, we assume an SOA-based environment aggregating a set of distributed IT systems allowing for optimization by parallel execution, but the amount of parallelism that can actually be achieved is also limited by the number of resource (web services or employees) replicas and their processing capacity in the system.

Data dependencies can arise between different activities and relate to variables defined in the WS-BPEL process definition. For instance, a service invocation activity has a read-dependency on whatever variable is used to hold the input message for the service to be invoked, and also a write dependency to the variable that will ultimately store the service’s reply. Assignment statements normally construct and write to a variable after reading values from one or more other variables. Structural activities may also read certain variables during the evaluation of control flow variables.

Techniques for the dynamic scheduling of instructions, such as the Tomasulo approach and scoreboarding, have been successfully used in numerous domains of computer science and allow for an optimized, out-of-order execution of sequential streams of program instructions, which could be used as the basis for individual process instances (Hennessy & Patterson, 1990). These techniques could be extended and applied to WS-BPEL activities to avoid pointless waiting as the result of data dependencies. However, these approaches are limited to basic blocks of non-branching sequences of instructions, cf., one sequential scope in BPEL, and can benefit from techniques such as speculation to work around control flow statements and as such artificially increasing the number of instructions in these basic blocks (cf., Section 3.3). The Tomasulo approach exploits the knowledge on dependencies unraveled at runtime; thus it clearly outperforms all strategies that statically analyze the data and control flow of the BPEL process; nevertheless, it is considerably easier to analyze and restructure the overall BPEL process model at design-time (Klopp & Khalaf, 2007).

The process model in Figure 1 merely represents successive service invocation activities. It is furthermore assumed that an unaltered output message from a particular service invocation is stored in a BPEL variable, which is used as
input for invoking another service. Note that the dashed arrows representing these data dependencies are not part of the official BPMN notation and have been added for improved readability. The start event corresponds to the reception of a message that triggers execution of a new WS-BPEL process instance and the end event represents the process replying to its requester. The solid arrows in the diagrams indicate sequential flow. Supposing the respective execution times for activities A, B, C and D (invocation of an operation on a service) are 9, 4, 12 and 6 seconds, the execution time in the original process would simply be 31 seconds, whereas the optimized version would result in an execution time of 27 seconds. Obviously, the time required for the new scope containing the parallel flows to complete is determined by the branch that takes most time to complete (A→B).

3.2. Exception BPR Pattern

In re-engineering business processes, it is common to isolate the exceptional part from the normal process flow. Techniques like speculation are already applied in compiler optimizations to improve control flow, branches in particular (Hennessy & Patterson, 1990). This can be accomplished by conditionally executing the branch with highest probability, and compensating in case of misprediction. Moreover, the amount of parallelism that one can exploit is also limited by control dependencies. Speculation is a technique that can be used to overcome the penalty of control dependencies in some cases by shifting highly probable activities to eliminate control dependencies so as to match the parallelism offered by the execution environment. To achieve speculation techniques in WS-BPEL process definitions, scopes of activities may be shifted, provided that an estimation on the probability of each branch is available. This information can either be a constant value chosen at design time, or alternatively be gathered at runtime, during the execution of the program, through a monitoring component (as it is the case e.g., in feedback loops and autonomic computing systems) (Ganek & Corbi, 2003).

Consider for instance the example in Figure 2. Imagine the left branch in the original process model has a 30% chance of being executed. For the service invocations A, B, C and X, we suppose execution times being 15 seconds each. Then the execution of the original process model would take about 45 seconds for sequentially executing activities X, A and B, or 30 seconds for the other execution path comprising activities X and C. The time required to evaluate control flow structures, e.g., branch variables, is considered negligible.

Applying speculation will restructure the initial model to always execute C (because of the 70% chance the containing branch is chosen). The branching condition remains identical as long as the result of invoking service C is not written to a variable that is used for evaluating the branching condition. In case the process instance proceeds accordingly, the execution time equals 30 seconds. However, speculation may deteriorate and delay process execution is the execution does not fit: because of the undoing, the best-case execution time of the speculated WS-BPEL process is now 60 seconds.

Consequently, speculation in itself does not necessarily result in an enhanced process model, but combined with parallelization, a significant gain in execution time can be harvested. In the absence of data dependencies between activities X and C, the intermediate model that was transformed using speculation can now be restructured to execute C in parallel to X. In case of the normal process flow (the branch comprising C in the original model), there is a considerable improvement: 15 seconds instead of 30 seconds (a speedup by factor 2). In the worst case, should the time required to execute C not exceed the time lapse for the execution of the other parallel flow of activities, the performance degradation is given by the overhead for undoing, and the alternative flow will perform no worse than in the original model.

In conclusion, the exception re-engineering pattern should only be applied if there are no data dependencies between the activities in the branch that is most likely to be chosen
(the normal flow) and the activities before the branching condition. Furthermore, we suggest this pattern to be applied only if the difference in probabilities of the alternative branches exceeds a minimal threshold. In that case, the overall performance might benefit from speculation, and the negative impact of non-fitting BPEL instances might be subdued.

3.3. Knock-Out BPR Pattern

Business processes generally contain a number of knock-outs, conditional checks that may cause the complete process instance to cease, skipping all subsequent process activities. Upon occurrence, the BPEL process instance should be abandoned, possibly compensating in order to reverse the service invocations that were required for evaluating the knockout conditions. The knock-out BPR pattern is a special version of the resequencing pattern aiming to manipulate the process yielding on average the least costly execution by arranging knock-outs in decreasing order of effort and increasing order of termination probability (Reijers & Lijman Mansar, 2004). The rationale behind this pattern is that knock-outs should be inserted in the process flow as early as possible to avoid the allocation of resources during the execution of other process activities for process instances that halt. We illustrate this principle in Figure 3:

Suppose knock-out condition $K_i$ has a 40% probability of evaluating negatively and it takes 2 seconds to invoke service A and compute this branching condition. Likewise, for knock-out $K_j$, these values respectively equal 65% and 4 seconds including the invocation of service B. Then the ratio $0.40/2$ for $K_i$ is higher than $0.65/4$ for $K_j$. Hence, assuming the absence of data dependencies, application of the knock-out pattern should restructure the arrangement of knock-outs in the upper diagram shown in Figure 3 into the lower process model, i.e., $K_i$ should be checked before $K_j$.

As time goes by and more process instances have been executed, for both the speculation and knock-out re-engineering patterns, the estimated probabilities of the branching conditions might change, which in turn may trigger new changes
to the process model at runtime, resulting in adaptive business processes.

3.4. Dependability Aspects

During its execution, failures may occur that impede a WS-BPEL process instance from proceeding correctly or even worse, simply terminate it. Moreover, a process may contain a number of business-critical scopes that require a higher degree of reliability or some form of transactional support. In this section, we present a survey on how processes can be designed for increased reliability using proven techniques for application-level fault tolerance.

3.4.1. WS-BPEL and Fault Tolerance

Fault tolerant mechanisms can only be expressed in a syntactically adequate linguistic structure (De Florio, 2000). In spite of numerous WS-* specifications related to reliable messaging and security, little emphasis has been placed upon the fault tolerance aspect of SOA (Modafferi, Mussi, & Pernici, 2006; Charfi et al., 2007). In this section, we first introduce the syntactical constructs for error recovery as well as their semantics in the WS-BPEL specification. We will then point out that these features prove to be inadequate for complex scenarios aiming at increasing dependability by means of transactions and fault-tolerance.

A BPEL process definition consists of a number of scopes that typically represent a particular piece of functionality, most likely a complex activity. Similar to high-level programming languages, scopes can be nested within the overall process definition as the outer scope. Three types of handlers can be defined upon a scope (Abode et al., 2007):

- The purpose of BPEL fault handlers is similar to that of catch blocks and exceptions in the Java programming language: to undo the partial and unsuccessful work of a scope and performing forward error

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
recovery with the aim of re-attaining a state where the execution of the BPEL instance can resume. A web service may explicitly throw a SOAP fault message when it is invoked from a BPEL process. BPEL also defines a number of standard faults that will be thrown by the BPEL runtime as a consequence of erroneous conditions during process execution, for instance, a join failure. Furthermore, the process designer may include application-specific knockouts that throw BPEL faults when deviations from normal behavior are detected (self-checking pattern, (Strigini et al., 2005)). Faults are identified by an XML qualified name. Two types of fault handlers can be attached to a scope for intercepting faults thrown during the execution of the activities contained inside the scope: a fault handler can either handle one specific type of fault, or a “catch-all” fault handler can handle all faults for which no specific handler was defined. When a fault is raised, all remaining activities in the current scope are terminated, and an appropriate handler that is capable of handling the fault will be selected and activated. If a fault cannot be treated by the handlers pertaining to the current scope, it is recursively forwarded to the enclosing scope. If the fault is not caught by any fault handler, the process instance will exit, triggering the default termination handler. Note that fault handlers are only enabled when the execution of a scope is in progress.

- Compensation handlers represent the application-specific undo process for rolling back the effects of scoped activities that were already executed (backward error recovery). If compensation is triggered, for instance from within fault handlers, all nested scope will have their compensation handlers activated recursively.
- Lastly, termination handlers can be used kick in a series of activities when a BPEL process exits unexpectedly. The default compensation handler will trigger compensation.

Apart from its compensation and fault handling, the WS-BPEL syntax is limited to describe the functional part of workflows. Moreover, these standard BPEL recovery mechanisms prove to be inadequate to define sophisticated recovery patterns/procedures, for example rollback or the execution of alternative web services (Buys, 2009; Charfi et al., 2007).

Finally, a hidden assumption in WS-BPEL is that designers have complete knowledge of the fault and system model of the partner services, so that they are able to define a process flow that contains all the required strategies for recovering from faulty situations (Modafferi, Mussi, & Pernici, 2006). We believe this assumption not to be a realistic one.
3.4.2. Transactional Support

The WS-AtomicTransaction specification, part of the WS-Transaction family of specifications, enables the coordination of distributed transactions using the two phase-commit protocol with ACID-compliant transaction features (atomicity, consistency, isolation, durability) (Erl, 2005). Even though backward error recovery techniques such as this commit and rollback approach are generally not suitable to apply to long-running WS-BPEL workflows, there are situations where consistency should be guaranteed during short-lived subprocesses (service invocations in a particular scope).

BPEL’s compensation mechanism, which allows undoing the effects of completed activities, cannot cope with atomic transactions, as the coordination model of WS-BPEL is local to the process definition. As there is no external coordination, a partner service in the transaction may not be notified by the process, which may leave the system in an inconsistent state. The BPEL enhancements published in (Charfi et al., 2007) enable the use of atomic transactions and business activities in the context of WS-BPEL processes by using aspects to inject BPEL code to use the external coordination mechanism defined in WS-AtomicTransaction.

3.4.3. Redoing and Design Diversity

This section will demonstrate the feasibility of using WS-BPEL to apply proven techniques for application-level fault tolerance.

A. Redoing and Recovery Blocks

Redoing Blocks is a technique that addresses residual software design faults. It is similar to the hardware fault-tolerance approach known as “stand-by sparing”. The approach works as follows: on entry to a recovery block, the current state of the system is checkpointed. A primary alternate is executed. When it ends, an acceptance test checks whether the primary alternate successfully accomplished its objectives. If not, a backward recovery step brings the system state back to its original value and a secondary alternate takes over the task of the primary alternate. When the secondary alternate ends, the acceptance test is executed again. The strategy goes on until either an alternate fulfills its tasks or all alternates are executed without success. In such a case, an error routine is executed.

The effectiveness of recovery blocks rests to a great extent on the acceptance test. A failure of the acceptance test is a failure of the whole recovery blocks strategy. For this reason, the acceptance test must be simple, must not introduce huge run-time overheads, and it must not retain data locally. Recovery blocks have been successfully adopted throughout 30 years in many different application fields. It has been successfully validated by a number of statistical experiments and through mathematical modeling (Randell & Xu, 1995).

Retrying is not directly supported by WS-BPEL, as BPEL does not offer a checkpointing service. Assuming stateless web services as alternates, consider a scope with a structured sequence activity. First it will invoke an operation on service A. Then, a conditional branch will evaluate the acceptance test on the service response message. If the acceptance test is successful, the process execution proceeds. However, in case the acceptance test fails, a fault is thrown. This fault is caught by a fault handler attached that in turn triggers compensation. The compensation handler contains another scope with precisely the same activities, only this time an alternate service B will be invoked. By nesting scopes inside compensation handlers recursively, WS-BPEL allows to redo a web service invocation (Dobson, 2006).

B. N-Version Programming

N-Version Programming (NVP) systems are built from generic architectures based on redundancy and consensus. NVP is defined by its author (Avizienis, 1985) as “the independent generation of n > 1 functionally equivalent programs from the same initial specification.” These n programs, called versions, are developed for being executed in parallel. This system constitutes a fault-tolerant software unit that depends on a generic decision algorithm to determine a consensus or majority result from
the individual outputs of two or more versions of the unit (Figure 4).

Such a strategy has been developed under the fundamental conjecture that independent designs translate into random component failures. Such a result would guarantee that correlated failures do not translate into immediate exhaustion of the available redundancy, as it would happen, e.g., by using n copies of the same version. Replicating software would also mean replicating any dormant software fault in the source version.

NVP is different from recovery blocks in that the latter is a sequential strategy, whereas NVP allows concurrent execution. Moreover, recovery blocks require the user to provide a fault-free, application-specific acceptance test, while NVP adopts a generic consensus or majority voting algorithm that can be provided by the execution environment. Finally, recovery blocks allow different correct outputs from the alternates, while the general-purpose character of the consensus algorithm of NVP calls for a single correct output. The two models collapse when the acceptance test of recovery blocks is done as in NVP, i.e., when the acceptance test is a consensus on the basis of the outputs of the different alternates.

Figure 4 shows how NVP may be implemented in WS-BPEL. Assuming stateless services A, B and C have the same WSDL interface, we concurrently execute the same method on the three available services (n = 3). After joining, the voting procedure will compare the values returned. Finally, if the outcome of the voting procedure is negative, a BPRL fault will be thrown to signal failure. Otherwise, the voted value will be stored in a variable and the process will continue.

3.5. Human-Centric BPR in People-Oriented Business Processes

Human interactions frequently occur in business processes for the manual execution of tasks, e.g. an approval (IBM, SAP et al., 2007a). Expenses resulting from employment of people are still a major cost factor in enterprises. Therefore, an efficient allocation of the staff is imperative, and IT can also help to achieve this goal.

This brings us to another shortcoming of BPEL, which is rightly blamed of being too automation-centric since it lacks the recognition of employees in process workflows (Abode et al., 2007) (IBM, SAP et al., 2007a). The WS-BPEL4People and WS-HumanTask specification drafts, recently submitted to OASIS for ratification, allow for hybrid SOA in which human actors occur next to customary IT systems exposed as web services (IBM, SAP et al., 2007b). Consequently, we claim people-centric BPEL4People processes should be designed barred human-centric BPR patterns in mind, so that the system can automatically determine which employees should take care of the process activity, depending on the current availability of these human resources. To our knowledge, this idea has not been previously investigated. Table 1 shows a few of these patterns and points out relevant procedures defined in the BPEL4People specifications. Most of these patterns deal with the issue of assigning the task to the best suitable person available.

3.5.1. Introduction to BPEL4People and WS-HumanTask Specifications

The term BPEL4People actually covers two specifications that have been devised in a modular approach as an attempt to cover the complete spectrum of human-to-process interaction.

WS-HumanTask proposes an industry standard for defining and managing human-based activities in a BPEL4People process:

- It stipulates the syntax and semantics for defining human tasks in XML format, where a task is considered as an indivisible unit of work performed by a human process actor. Similar to a subprocess, the execution of a task is closely related to the context of the parent process. The interoperable WS-HumanTask coordination protocol has been conceived to attain a tight coupling with synchronization of state between the task
and the process, where state changes can be propagated in either direction.

- New tasks are usually offered to a task inbox, the central point of interaction for human actors. WS-HumanTask defines a comprehensive client API interface for the implementation of task boxes that can be used for manipulating tasks and controlling their life cycle in accordance to the WS-HumanTask coordination protocol. A task inbox is capable of rendering the user interface that is associated with a particular type of task so that all relevant information is displayed and the employee can successfully complete the work.

On a higher level, the WS-BPEL4People extension defines a number of features layered on top of WS-BPEL to seamlessly integrate WS-HumanTask tasks into WS-BPEL process definitions. Three concepts are at the core of BPEL4People:

- People activities were introduced as a new type of basic activity, to allow the integration of user interactions within BPEL processes. WS-HumanTask tasks can be declared either inside the activity declaration, or may be remotely deployed as a web service that supports the WS-HumanTask protocol (though the service is not implemented by a piece of software, but by a task box such that a user will eventually perform the work manually). A people activity declares the inputs and outputs required to invoke the task, just like its equivalent invoke activity used for calling a web service.

- Similar to partner links which are used to bind a web service to a WS-BPEL process, people links bind a group of people to a business process. People links are generally associated with the generic human roles defined in WS-BPEL4People and WS-HumanTask and represent a group of people who are associated with the execution of a particular people activity.
• In order to determine the actual group of individuals involved in dealing with a particular activity, the action of people resolution has to be performed by assigning people queries to people links. An example of a people query may be an XPath expression to be evaluated against a people directory, a database describing an organizational model to represent the employees of some company or department (WS-BPEL4People only describes the entity and the XSD schema type the query should return; the actual implementation is not covered). People resolution is actually a two-phase procedure: first the people query is evaluated to determine the set of people that have the potential owner generic human role. The task infrastructure will subsequently offer the task to all potential owners who are eligible to claim that activity. Eventually a single potential owner that claimed the activity will become the actual owner and will be responsible for completing the activity.

One may wonder whether these specifications cover most commonly used constellations of human-to-process interaction. Extensive studies with regard to this issue were already published in Russell and van der Aalst (2008) and Mendling, Ploessl, and Strembeck (2008). Having compared these specifications against the universal transaction pattern as the core of the communication theory formalized in the Design and Engineering Methodology for Organizations (DEMO) methodology (Dietz, 2006), we concluded that the specifications support a vast majority of the human-process interaction scenarios.

3.5.2. Order Assignment BPR Pattern

The order assignment pattern prefers the same employee to work on several successive process activities for a particular process instance. This is directly supported by the BPEL4People concept of chain of execution, where the actual owner that took care of the previous activity is selected as the sole potential owner for the task at hand (see Figure 5, activities A and B). In addition, an escalation action should be defined to offer the task to the regular set of potential owners in case the default scenario would fail (i.e., the owner of the previous activity does not claim the task before the expiration deadline). Assigning several consecutive process activities to one person should result in a reduction of execution time as this person has got acquainted with the case. The side effect, however, is that the employee’s workload will slightly increase compared to his or her colleagues.

3.5.3. Flexible Assignment and Specialist-Generalist BPR Pattern

Next, according to the flexible assignment BPR pattern, and supplemented by the specialist-generalist pattern, one should distinguish between highly specialized human resources and generalist employees that can be assigned to execute a diversity of tasks. The availability of generalists adds more flexibility to the business process and can lead to a better utilization of resources. Unfortunately, the generic human roles defined in the specifications are insufficient, and the people query facility and the organizational people directory that is searched by this query, both proposed in the above mentioned specifications, remain undefined. We should find a way to annotate people in this directory describing their skills, capabilities and permissions so that the system can reason about the degree of an individual’s specialization. We envision an important role for techniques such as semantic processing and semantic matching in particular. The mutual assistance community, as it was proposed in (Sun et al., 2007), aiming to provide elderly people with the services they require in a timely and cost-effective way, introduces a system where human resources are registered with a semantic description according to an OWL-S ontology model (Antoniou & van Harmelen, 2004). Further research is required on semantic WS-HumanTask annotation before this type of service-wrapped registry can be used to determine the potential owners of a task.
3.5.4. Split Responsibilities BPR Pattern

Assigning different tasks within a process to people from different functional units should be avoided (split responsibilities pattern). Again, enhancing the expressiveness of people queries and the structure of the people directory could allow the system to optimize the dispatching of human tasks to the appropriate available human resources at runtime. Related to this pattern is the concept of separation of duties, also referred to as the 4-eye principle in which mutually independent individuals each perform an instance of the same task for the purpose of combating fraud and avoiding disastrous mistakes (IBM, SAP et al., 2007b). An example is shown in Figure 6, where activity C may not be executed by whomever performed task A.

3.5.5. Case Manager BPR Pattern

The case manager BPR pattern originally introduced an additional process actor—the case manager— that is responsible for a business process. However, as the emphasis is on the management of the process rather than actually participating in its execution, the case manager is not necessarily the only resource that will work on process tasks. Providing a single point of contact from a client perspective, detour patterns, such as delegation and escalation which are directly supported in BPEL4People, can result in delegating process activities to other people (Russell & van der Aalst, 2008; IBM, SAP et al., 2007a). Apart from this single point of contact, the case manager is also the person accountable for correcting mistakes. Fortunately, the business administrator and process stakeholder generic human roles, defined in the WS-BPEL4People specification, can be used to represent the case manager respectively when managing the entire process or merely a single process case (IBM, SAP et al., 2007a).

In a situation where the execution of a process has gone astray, chances are that it will jam and require manual intervention of the case manager in order not to aggravate the situation. Therefore, one-way WS-HumanTask notifications can be used to notify the case manager of noteworthy events, or application-specific administration tasks for forward or backward recovery may be embedded inside WS-BPEL fault or termination handlers.

As a final consideration, human-computer interaction faults were rarely considered in fault-tolerance designs as, in the past, they were considered external to the system boundaries. The BPEL4People specifications finally allow extending these boundaries by seamlessly integrating human tasks and service-wrapped software components into hybrid WS-BPEL workflows that can consequently realize a higher degree of dependability. Also, because of WS-HumanTask, such interaction faults can
Figure 6. Sample process comprising human tasks and an excerpt from the BPEL4People process definition illustrating the language constructs for separation of duties for activities A and C.

be detected and dealt with by adding additional checks, possibly with the intervention of an external case manager.

Combining human-computer interaction by means of the BPEL4People and WS-HumanTask specifications with BPR patterns for automatically and intelligently dispatching workload to human system resources is an exciting research challenge with the potential for a substantial performance improvement in process execution, which may lead to increased productivity and competitiveness. At the same time this also endorses the significance of BPEL4People in SOA, and we plead for a speedy ratification of the specification drafts.

4. CONCLUSION

We started this paper by briefly introducing BPR as a relatively new managerial methodology and SOA as a way to sustain the volatility of business processes resulting from the fierce competition in the market. It was pointed out that the static nature of BPEL process definitions imposes limitations to quickly and easily re-engineer business processes in the quest for operational efficiency.

SOA strongly embraces the principle of business agility. Hence, incorporating BPR system intelligence into the BPEL engine allows for the dynamic application of BPR principles to the original static BPEL process definitions, with the goal of optimizing the process at runtime with respect to the system’s current state and runtime environment.

We propose an innovatory approach in which BPR principles are explicitly applied to WS-BPEL processes by means of established techniques and practices from computer science so that the process semantics are preserved and whereby at the same time the process execution is being optimized. It is expected that this will allow for a reduction in execution time, e.g., as the result of parallelization. This conjecture has been corroborated by several small examples. We then addressed the issue of process reliability using a bottom-up approach starting from proven techniques for software fault tolerance. Furthermore, the WS-BPEL4People standard enables to design people-oriented business processes such that human-centric BPR patterns are applied at runtime to intelligently dispatch human tasks to suitable human process actors depending on availability.

Furthermore, BPR-aware SOA have the potential to turn static BPEL process definitions into adaptive workflows that match the current environmental and systemic conditions so as to make a more efficient use of these system resources thus achieving higher performance. The WS-BPEL specification need not be modified: this ensures a smooth transition in adopting these ideas. Complex BPR patterns can be implemented using runtime system information and possibly service metadata and annotations.
We are still in the early phase of elaborating on the ideas presented in this paper. As a proof of concept, we intend to develop a prototype illustrating the feasibility of the exemplified BPR patterns. Research on the introduced human-centric BPR patterns will depend on the ratification process of the BPEL4People specification draft, and the availability of compliant implementations.

We conclude that BPR-aware SOA environments, automatically applying re-engineering patterns to BPEL processes, result in adaptive business processes, which is a crucial requisite for achieving an enhanced form of business agility and as such better sustaining process evolution.

ACKNOWLEDGMENT

This paper extends (Buys, De Florio, & Blondia, 2009), which appeared in the Proceedings of the IT Revolutions 2008 conference. For their permission we express our gratitude to the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST).

REFERENCES


ENDNOTES

1 The notion of business process is defined as an orchestration of several process activities carried out by computer systems or people within an enterprise with the objective of supplying a product or service to the customer. Because of the vague definitions found in most text books, the BPR acronym is commonly used interchangeably for business process re-engineering as well as business process redesign. The former has an evolutionary character, while the latter is revolutionary. For more information, we refer to (Reidin & Sundling, 2007).

2 BPEL processes are usually complex long-running processes. Every time the process model is triggered, the BPEL engine will construct a new process instance that runs in isolation from the other instances.

3 As the service-oriented computing paradigm promotes the development of stateless web services, the overhead of the undo activity may be considered negligible.


---

**Jonas Buys** received his degree in computer science from the University of Antwerp in 2007, after which he joined the PATS research group. Ever since, he has been working as a Ph.D. student on adaptability and dependability issues in service-oriented architectures. His current research interests include XML-based web services and the syntactical adequacy of WS-* specifications, techniques for collecting dependability attributes at runtime and assessing the overall reliability of web services.
Vincenzo De Florio obtained his “Laurea in Scienze dell’Informazione” (MSc, computer science) from the University of Bari (Italy, 1987) and his PhD in engineering from the University of Leuven (Belgium, 2000). He was researcher for eight years and part-time professor for three years with the University of Leuven. He is currently a researcher with the Performance Analysis of Telecommunication Systems (PATS) research group at the University of Antwerp, where he is responsible for PATS’ branch on adaptive and dependable systems. Vincenzo De Florio is also a researcher of IBBT, the Flemish Interdisciplinary Institute for BroadBand Technology. He published about seventy reviewed research papers, fifteen of which in international research journals, and the book “Application-layer fault-tolerance protocols”, edited by IGI Global. He is co-chair of workshop ADAMUS (the third Workshop on Adaptive and Dependable Mobile and Ubiquitous Systems, http://www.adamus.ua.ac.be), and editor-in-chief of the International Journal of Adaptive, Resilient and Autonomic Systems.

Chris Blondia obtained his Master in Science and Ph.D. in Mathematics, both from the University of Ghent (Belgium) in 1977 and 1982 respectively. In 1983 he joined Philips Belgium, where he was a researcher between 1986 and 1991 in the Philips Research Laboratory Belgium. Between 1991 and 1994 he was an Associate Professor at the University of Nijmegen (NL). In 1995 he joined the Department of Mathematics and Computer Science of the University of Antwerp, where he is currently a Professor and head of the research group “Performance Analysis of Telecommunication Systems” (PATS). His main research interests are related to mathematical models for performance evaluation of computer and communication systems and the impact of the performance on the architecture of these systems. He has published a substantial number of papers in international journals and conferences and is currently involved in the National and European Research Programmes in numerous projects.