Architectures for Service-oriented Processes

Martin Henkel[1], Jelena Zdravkovic[2]

[1] Stockholm University and Royal Institute of Technology,
Department of Computer and System Sciences, Forum 100,
164 40 Kista, Sweden
martinh@dsv.su.se
http://www.dsv.su.se

[2] University of Gävle and Royal Institute of Technology,
Department of Computer and System Sciences
801 76 Gävle, Sweden
jzc@dsv.su.se
http://www.hig.se

Abstract. By the use of Web Service technologies and the Internet it is possible to lay the foundation for virtual value chains that cross enterprise boundaries. As the number of services and their interaction grow, it is evident that the flow of message exchange between services needs to be coordinated in a structured way. Executable process languages such as BPEL are proposed as an instrument for the coordination of services. Executable processes must be designed such that they solve technical coordination problems as well as provide a fundament for organizations to manage and monitor the progress of the business. In this paper we examine how the design of executable processes is affected by both technical and business issues. Furthermore, we examine a set of architectures that enable the use of executable processes to cater to both business and technical needs. We provide fundamental guidelines on how to apply the architectures.

1 Introduction

Software services are the building block of future systems that integrate existing IT assets and thereby provide the basis for building complex cross-enterprise systems. Even though the Web Service technologies SOAP and WSDL solve many interoperability issues, they do not provide support for the complex task of coordinating the execution of software services. Coordination of services involves executing and monitoring complex message exchanges, as well as to provide massive parallelism and robust error handling. The core of the coordination problem lies in controlling the dynamic aspect of service execution. Thus, when examining the coordination problem, the focus is on dynamic aspects such as the order of message exchange and the timing of messages, rather than static aspects of services such as interfaces. Since process descriptions focus on the dynamic aspects, it is natural to use processes as a tool for describing and solving coordination problems. Future systems

will increasingly rely on composing and coordinating individual services to create new, complex business interactions in the form of processes [1]. Executable process languages such as the Business Process Execution Language for Web Services (BPEL4WS,[2]) are specifically targeted towards solving coordination problems.

A successful process implementation relies on that the executable process can capture business aspects as well as handle technical details. Processes modelled to close resemble a business will depict the activities, message exchanges and rules of an organisation. When this *business process* is to be realised in a system, additional technical issues such as system interoperability and message synchronization must be solved. Thus, we identify two perspectives of process design, the business perspective and the technical perspective. Seen from the business perspective, executable processes offer the ability to monitor and alter the processes according to business rules. Workflow systems [3] are an example of processes used from the business perspective. Taking the technical perspective, executable processes offer the ability to perform complex system integration and synchronization. An example of the technical perspective is the numerous messaging patterns [4] that exist to handle system coordination.

In order to cater to both business and technical needs, the business and technical perspectives must coexist in future system architectures.

In this paper we examine the properties of processes designed from the technical and business perspectives. We base this examination on a conceptual framework for process specifications. Furthermore, we examine three architectures that can help merge the two perspectives of processes. We also provide basic guidelines on how to apply the architectures. The aim of the paper is to provide an overview of different process perspectives, and provide basic guidelines on how to merge them. The work presented in this paper is a continuation of previous work presented in [5].

This work specifically focuses on technical and business perspectives of executable processes. On a much more general level, this problem has been studied by other authors as a miss-fit between business and information system functionality ([6], [7], [8]). Our use of two perspectives on processes also bears some resemblance with the model layers proposed by the Object Management Groups (OMG) Model-Driven Architecture (MDA) approach. However, MDA uses model layers to separate models from platform technologies [9]. Compared to the work presented in this paper, MDA is thus to be considered as a general framework for model transformations.

This paper is structured as follows. In the following section we define and give a concrete example of business and technical processes. In Section 3, we discuss design differences between the two process types. Section 4 introduces three architectures that combine technical and business issues. Guidelines on how to apply the architectures and conclusions are presented in section 5 and 6.

2 Business and Technical Processes

Analysis of a business might result in the design of one ore more executable processes. These designs might later be implemented by using an executable process language, such as BPEL4WS. Based on the discussion presented in the Introduction section, we distinguish between two types of executable process designs:

Business processes directly corresponds to the processes that where documented during analysis. This means that the executable process reflects the pure business perspective, containing the same activities, roles etc as the business process.

Technical processes are designs that do not correspond directly to the documented business processes. This kind of processes is designed to adhere to business as well as technical requirements. The motivation for creating a technical process that does not resemble the business process is:

1. The business process cannot be implemented as-is, because of technology limitations, for example in programming languages and communication protocols.
2. The existing software services, when composed in a process, do not match the documented business process.

Since an executable business process (as defined above) is defined in business terms, it has the advantage of being easy to understand and modify for people within the business. However, technical processes cannot be overlooked since limitations in existing system are inevitable.

An example of a business process and its realization as a technical process is shown in Figure 1 (this example is also presented in [5]). The example, in the form of a technical process, is based on a case provided by Sandvik, a global industrial materials engineering company with offices in 130 countries. A major concern of Sandvik is integration and coordination of their existing ERP systems in the form of software services. As one of the pioneers in the use of Web services, Sandvik has recognized the need to use executable business processes to handle the coordination of its services.

The business process in Figure 1(a) depicts a basic order process where the process is triggered by an incoming order request. Since we are using the Business Process Modelling Notation (BPMN) [10] we depict the customer with a swimlane/pool symbol, message events with encircled envelopes and message flows with dotted arrows. After an order confirmation is sent to the customer, the process is forked into two parallel flows. A shipment plan is constructed while the order is being processed. Later on, the flow is synchronized using an AND-join. Before the end of the process a notification that the product has been shipped is sent to the customer.

Figure 1(b) depicts the realized technical process. This process is an excerpt of the process supplied by Sandvik. The technical process is based on existing services, in this case the ERP system (depicted by the ERP service pool in Figure 1) and a service interface to the customers' information systems (the customer service pool in Figure 1).

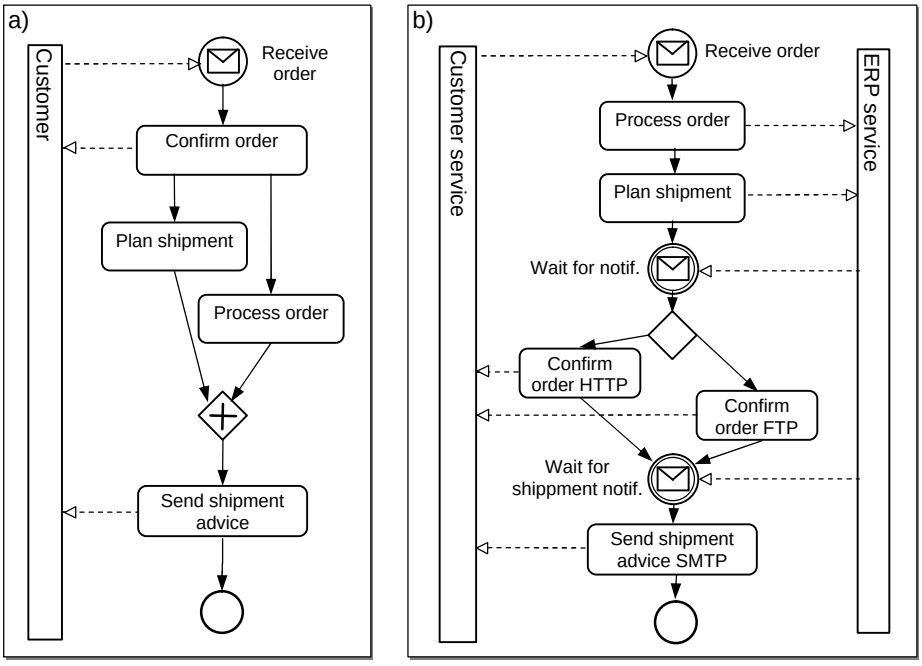

Fig. 1. Example business process (a) and realized technical process (b)

Compared to the business process, the technical process must adhere to a set of system constraints:

- S1 – The existing ERP service perform logistics planning and order processing in an integrated activity, a notification can be received when this process is completed.
- S2 – The validation of order information is integrated into the ERP service, order confirmation can be sent when the order is received by the ERP service.
- S3 – Based on the customers (software) service ability order confirmation should be sent as a HTTP message or a FTP file.

These system constraints affect the realization of the business process. The characteristics of the ERP system (S1) prompt us to put shipment planning and order processing in a sequence. Since the message event from the ERP system signals when the order confirmation can be sent (S2), the sending of the order confirmation is placed directly after (instead of before) shipment planning and order processing. Furthermore, the usage of different protocols for sending the order confirmation (S3) prompts us to use an XOR-split of the process flow.

Even though the example is small, it shows how a realization of a business process can be affected by system constraints. In this example we used a subset of the process supplied by Sandvik, for brevity reasons we excluded customer authentication and the handling of invoices. We also simplified the handling of protocols, the original

process handles more protocols than FTP and HTTP for all activities that notify the customer.

The example elucidates the point that realization does change the design of a process. In the next section we define process design aspects to provide methodical examination of how a business process might differ from a technical process.

3 Designing Business and Technical Processes

In this section we examine the design of business and technical processes to identify possible differences in respective process designs. To make a structured examination, we use a conceptual framework that identifies four aspects of process design. The framework is based on modelling aspects of workflows, as proposed in [11] and [12]. The first basic aspect of the framework is the *functional* aspect; it describes how a process is decomposed into activities, i.e. what activities are to be executed. The *behavioural* aspect depicts process control flow, i.e. when an activity is to be executed in relation to others. The *informational* aspect concerns content and structure of process data. The *transactional* aspect concerns consistent execution and recovery of a set of activities.

When designing a business as well as a technical process, each of the process aspects must be considered. In the rest of the section we describe how each of the four aspects may be affected when realizing a business process as a technical process. For a more detailed description of the design differences between technical and business processes we refer to [5].

Functional Aspect When comparing business and technical processes from the functional perspective we compare how the functionality is composed into activities. We also must consider the pre- and post-condition of the activities. Executable business processes cater directly to business needs, activities in a business process will thus be designed according to the activities in the organization. When designing technical processes the design issues will shift towards expressing how technology and existing system operations can be combined to solve business problems. The functionality of existing systems will be the base for selecting the activities for inclusion in a technical process. These design differences might result in the following differences that may appear in the functional aspect of a business and a technical process:

An activity in the business process corresponds to more than one activity in the technical process, where those activities jointly achieve the same goal as the activity in the business process.

Activities in the business and the technical process having similar goal differ in pre- and/or post-conditions.

Behavioural Aspect Comparing the behavioural aspects of technical and business processes involves examining the criteria's that governs the process control flow. In business processes, the rules that govern the flow coordination of the activities are stated by business rules, such that payment should be done before a product is

shipped. Technical processes must also adhere to the desired business behaviour. However, the order in which to perform activities might also be governed by the technical features of the back-end systems, for example by dependencies between existing services. Due to these design differences the processes might differ in sequencing (including parallelism) and in the use of conditional branching. We, therefore, identify the differences that may appear in the behavioural aspect of a business and a technical process as the following:

Sequentially ordered activities in one process (business or technical) may correspond to differently sequenced or parallel ordered activities in the other process.

A condition might have the same, fewer or more branches in a technical process compared to a business process.

Informational Aspect The design of the informational aspect concerns the information structures used by the process. The design of business processes focus on the information need of the business parties; the process information will closely resemble the concepts used in the business. The information content used when designing a technical process is based on the business concepts. A difference between the business and technical processes is however how the information is structured. In the technical process, the structure simply needs to be adjusted to fit to the existing services and technologies. In addition to the change of structure, the protocols used might require the addition of extra service or protocol specific concepts, such as system or transaction identifiers. Based on this, we identify the differences that may appear in the informational aspect of a business and a technical process:

A technical process may extend/exclude the concepts identified in the business process.

Concepts defined in the business process might correspond to one or more concepts in the technical process.

Transactional Aspect Comparing two processes from the transactional perspective involves examining similarity in transactional behaviour reflected by specified transaction models (atomic vs. long-running [13]) and transaction boundaries. When an error occurs the process must return to a valid state by withdrawing results of completed activities (by compensation for long-running transactions or by simple cancelling for atomic transactions). The focus when designing a business process is to keep the process aligned with business contract rules. When designing technical processes it must be ensured that no systems or services are put in an inconsistent state with respect to the overall process Thus, transactional aspect of a business process, with existence of supported systems and services, may differ from a technical process in following:

The boundaries of a transaction may differ in the business process and technical process.

The model (atomic vs. long-running) of a transaction differs in the business process and the technical process.

In Table 2, below, we summarize the design of technical and business processes for the described process aspects.

Table 2. A summary of design aspects for business and technical processes

	Functional	*Behavioural*	*Informational*	*Transactional*
Design task	Decompose into activities	Sequencing of activities	Structure message and process information	Select transaction model and transaction boundaries
Business process design	Decompose according to business activities	Governed by business rules and contracts	According to business concepts	Keep consistent business states
Technical process design	Decompose according to operations in existing services	Governed (partly) by features of the existing systems	Business concepts split/extended according to service needs	Keep systems and data in a consistent state

Based on the identified differences in realizations of business processes and technical processes, in the following section, we discuss abilities for integration of those two perspectives in a software system.

4 Architectures

Software architectures need to support both business and technical perspectives. Ideal would be to have a software architecture that includes both the business and technical perspectives of executable processes. Architectures that combine business and technical perspectives of processes must be able to handle all, or at least some of the process design differences that where discussed in Section 3. An example of a difference is that a technical and a corresponding business process might contain different number of activities (i.e. they differ in the functional aspect). Even though this difference exists, the architecture must be able to convey the state of the business process, for example by making it possible to instantly identify the activities that are currently executing on the business level.

When constructing an architecture that support both business and technical aspects there are basically three possible approaches:

a) Use the designed technical process as a starting point and add the features of the business process. We call this architecture the Layered architecture.

b) Use the business process as a fundament, and add technical aspects. This will result in what we call an Aspect oriented architecture.

c) Combine business and technical aspects by identifying “modules” that encapsulates technical or business behaviour. This architecture is named the Domain service architecture.

In the following sections we will examine how each of the three architectures enables a closer integration between business and technical processes. We base our

examination on the four process aspects (functional, behavioural, informational and transactional) described in section 3.

4.1 The Layered Architecture

In a layered architecture, the business process is layered on top of the technical process (see Figure 2). The executable business process reflects the states in the business, while technical issues are dealt with in the technical process. The two layers of processes is kept synchronized, for example by implementing a publish-subscribe event system.

Generally the business process will represent an abstraction of the underlying technical process. This is in-line with the common architectural principle of abstraction layers [14]. The business process layer is dependent on the technical process for executing systems actions, thus this dependency can be categorised as a "use" dependency [15].

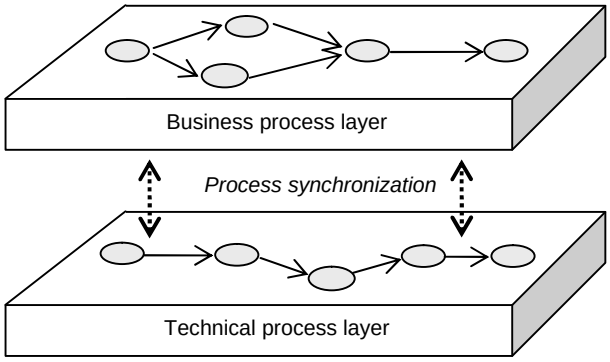

Fig. 2. Overview of the layered architecture

A prerequisite to using this architecture is that it should possible to perform process synchronization between the two layers. In essence the design of all of the four aspects of the technical process must be done such that it is possible to keep the four aspects of the business process unaltered.

The *functional aspect* of a process contains its activities and the pre-and post conditions of those activities. In the technical process layer it is possible to break down a business activity into several technical activities without disrupting the synchronization. Following principles from object-oriented programming [16] pre conditions in the technical process can also be weakened in the technical process, while post-conditions can be made stronger. However, to maintain synchronization between the two layers, the functional aspects must be designed such that an activity in the business process corresponds to at least one (unique) activity in the technical process. For example, if two activities in the business process correspond to a single technical activity, it is not possible to determine which of the business activities that are executing.

The *behavioural aspect* of a process refers to the order of execution of its activities. The layered architecture enables the technical process to contain advanced parallel execution and synchronisation of activities, this enable for example gathering information from several back-end systems. Just as for the functional aspect, the behavioural aspects of the technical process cannot deviate too much from the behaviour of the business process. A basic criteria for proper synchronization of the processes is that the order of the activities in the technical process follows the same order as in the business process. Similarly, parallel flows of activities cannot be employed on the technical level if the corresponding activities on the business level must be executed in sequence. Conditional branches in the business process flow should be represented with at least the same amount of branches in the technical process. Without following these basic criterias, the business process will behave inconsistent.

Synchronizing the *informational aspect* of the two processes mean that business information present at the business layer must be possible to extract from the technical layer. The technical layer can contain additional concepts for handling technical issues, such as transaction identifiers. In the technical layer it is also possible to duplicate the information used in the business process, this might be useable if the information is stored in several systems. A basic requirement of the layered architecture is however that the technical layer must be able to represent all the concepts of the business process.

The *transactional aspects* of the two layers describe the transaction model of the processes, as well as transaction boundaries. By using two layers it is possible to introduce two-phase commit protocols (atomic transactions) on the technical layer, even thought the business level are designed by using long-running transactions. To perform proper transaction synchronization between the layers, the transaction model in the technical layer must support the same, or a higher level of transaction model compared to the business process. For example, if the business process is designed using long running transactions, the technical process must support atomic transactions or long running transactions. Another limitation of this architecture is that the transaction boundaries of the two levels must be designed such that they do not overlap. For example, two business activities grouped together into a transaction should not be executed as two unrelated transactions on the technical level.

The main advantage of this architecture is its clear separation between business and technical issues. This separation makes it possible to represent a graphical overview of the business process, while still enabling detailed monitoring of the technical process. However, as described above, the synchronization of the levels can introduce severe limitations of the process design. Simply put, the business level must be a “pure” abstraction of the technical level. In many cases it is not possible to realize the technical level without affecting the business level. An example of this is the business and technical processes shown in Figure 1. In the example, the two processes differ too much in the behavioural aspect to be able to coexist in a layered architecture (the order of execution of the activities differs).

6.2 The Aspect Oriented Architecture

While the layered architecture has the technical process as the basic building block, the aspect oriented architecture starts with the business process as the fundament. The main idea of this architecture is to add orthogonal technical features to the business process without affecting its design. Commonly these orthogonal technical features can be expressed as non-functional requirements [17] such as security, transaction handling and error handling.

When the business process is executed the functionality of the aspects is combined with the process. This process of at runtime extending code with aspects is commonly called “Weaving” [18].

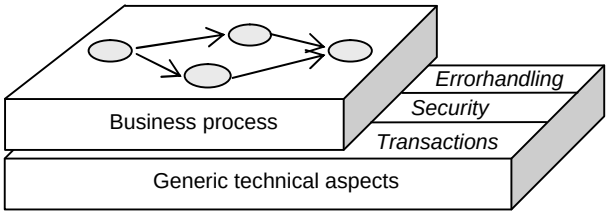

Fig. 3. Overview of the aspect architecture

Aspects such as those presented in Figure 3 are not the only types of functionality that can be expressed as aspects. Any functionality that is generic enough to be independent of the actual design of the business process can be added as an aspect. Just as for the layered architecture, the aspect architecture cannot be applied to implement all business processes. In the following we examine each of the four process aspects to single out possible uses of the aspect oriented architecture, and to find its limitations.

The *functional and behavioural aspects* of a process can partly be realized by using the aspect oriented architecture. As an example, the technical process depicted in Figure 1 contains activities that handle FTP and HTTP communication. The use of these two protocols affect both the functional (number of activities) and the behavioural (the conditional process flow) aspect of the process. By constructing an aspect that handles protocol selection and invocation it is possible to remove both the functional and (some of) the behavioural differences of the processes shown if Figure 1. The constructed aspect would be invoked whenever a message should be sent to the customer. The aspect then selects, dependent on the customer information, the appropriate protocol to use (FTP or HTTP). However, aspects cannot replace the domain specific parts of the technical process. For example, it is not feasible to construct an aspect that handles the difference in the ordering of the activities in the process.

Technical additions in the *informational aspects* can be handled by weaving aspects into the business process. An example of this could be that certain processes need to be identified with a process identifier on the technical level. An aspect can be constructed such that upon instantiation of the process a unique process identifier is

generated. This process identifier can subsequently be used as and identifier when connecting to existing back-end systems. However, adding information structures like this is delimited to generic, domain agnostic information. For example, adding the creation of an order number is not as simple as creating a “technical” identifier. The creation of an order number might be tightly governed by business rules, and thus cannot be handled on a generic level.

The *transactional aspect* concerns transaction model and transaction boundaries. Generic handling of transactions is a typical example of where the aspect oriented approach can play a major role. In fact, technologies that provide automatic transactions handling is already in wide use. Examples are Enterprise Java Bean (EJB) component servers and Microsoft enterprise services (formerly COM+). These technologies effectively remove the burden of handling transactions from the developer. For example automatic transformation from atomic to long-running transaction can be provided by generic aspects. The design of the transaction boundaries, however, still needs to be decided and implemented by the process designer.

The main advantage of this architecture is that technical features can be moved into generic aspects, or middleware products. These aspects and products can then be applied to a wide range of business processes.

Overall the aspect-oriented architecture is a very promising approach to remove domain-independent, generic technical functionality out of the core business process. While there exist special aspect-oriented languages, such as AspectJ [19], the aspect can also be provided by middleware servers. Business Process Management Systems (BPMS) provide basic aspects such as scalability and persistence. In the future use of the Business Process Execution Language for Web Services (BPEL4WS) will enable middleware servers to add more features (like secure communication) to processes.

6.3 The Domain Service Architecture

The two previously described architectures started with either the technical process or the business process as a fundament. A third approach to creating an architecture that support both business and technical perspectives is to combine technical and business requirements at the design stage. A viable solution is to let technical and business requirements guide the creation of modules that later can be combined using an executable process language. This approach is commonly presented as an option when building process-based systems [20], [21], [22]. Figure 4 contains a schematic overview of a domain service architecture where the modules sales, production planning and logistics are coordinated by a process layer.

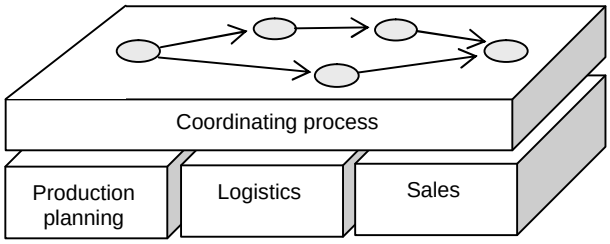

Fig. 4. Domain service architecture

The advantage of this architecture is that it is possible to hide technical details in the modules, and thereby only expose a façade to the modules functionality. The critical point when employing this architecture is to select a principle that governs the decomposition of functionality into modules. Selecting modules according to existing business functions are proposed by some authors [20], [23]. This approach is illustrated in Figure 4. Another option commonly used when construction component-based systems is to partition the modules according to central business concepts [24][25]. For companies with a large set of legacy systems, existing systems might also govern the creation of modules. In the following we will examine possible ways to implement the four aspects of processes using the service domain architecture.

The ability to represent the *functional* and *behavioural* aspects of a business process is highly dependant on the criteria of decomposition. The granularity of the modules and their interfaces decide the level of control the coordinating process can have over the modules. For example, a module might encapsulate several business activities and represent them as a single operation in the module interface. This encapsulation will affect the design of the coordinating process, the level of control will be lover at the process level. The example also holds for the behavioural aspect, where complex activity dependencies can be encapsulated in a module. The key issue here is to design the modules such that they encapsulate technical functions and behaviour, rather than hiding high-level business functions and behaviour.

The *informational* aspect of processes in the service domain architecture concerns the information content that each module exposes to the coordinating process. Using this architecture, it is simple to encapsulate technical information in each module. A risk when using this architecture is that technical information concepts will affect the coordinating process. An example of this is a module that requires a costumer's digital certificate to send a notification. The certificate must be supplied by the process, since communication between modules would break the architecture. This implies moving certificate handling to the process level, which might be undesired.

Handling the *transactional aspects* can be done by firstly implementing module-internal transactions, and secondly by letting each module implements a transactional interface. The transactional interface lets the coordinating process coordinate transactions on the business level. A key issue of this architecture is to handle differences in transactional models between modules. Basically, conversion of transactional protocols can be handled either in the process on in each module.

While the service domain architecture provides a good way to isolate technical details into modules, it relies heavily on the selection of decomposition criteria for the modules. If technical considerations play a major role when designing the modules, it will affect the coordinating process. Since all communication go through the coordinating process there is a risk that technical features “creep” into the process. Note that this problem is not evident in the layered architecture, since that architecture enables communication on the technical level.

7 Selecting Architecture

The three architectures to some extent represent extremes, when selecting architecture the best option is to combine features of the architectures.

The layered architecture can be used to provide monitoring capabilities for the business, while enabling complex technical processes on a separate layer. As discussed before, the needed process synchronization is the Achilles heel of this architecture. In practice it might not be possible to implement his architecture in its pure form. However, if the process synchronization is made less strict, the architecture is applicable in more cases. A “loosened” synchronization can be applied by allowing inconsistencies on the business level process. This would yield an executable process that indicates the status of the process, rather than to directly correspond to the executing process.

The aspect oriented architecture is applicable as a powerful tool for introducing system-wide technical properties. However, domain specific details should not be implemented as aspects, since there is not point in adding the complexity of an aspect to implement functionality that only concerns a single activity.

The service domain architecture provides a traditional fundament to build systems by dividing the functionality into modules. This is of course applicable to all processes. However, the architecture lacks the clear separation of processes provided by the layered architecture. The architecture also lacks the possibility to add generic technical functionality.

Given the above discussion, we propose that the features of all the architectures should be combined in the following order:

1) Maximize the use of aspects by identifying technical concerns that affects a large part of the activities. Use an aspect oriented language, or an existing middleware product to supply these aspects.
2) Construct a domain service architecture by building modules (services) that represent existing business concepts or functions. This enables the coordinating process to be as close to the business process as possible.
3) If needed, supply a business process monitor by implementing a second process layer depicting a simplified business process. To make this possible, the synchronization criteria of the processes must be relaxed.

Applying the features of the architectures in the above order makes it possible to maximize the advantages of the architectures.

8 Conclusion

In this paper we initially identified two types of processes, business and technical. These processes cater to partially different needs, and are thus designed differently. The possible differences in process design where examined using four aspects of process design. Furthermore, we used the identified process differences to examine three architectures that aim to unify business and technical aspects. The three architectures represent three different ways to deal with the integration of technical and business issues in executable processes. While these architectures can be applied directly, we proposed simple guidelines on how to combine the architectures.

Having a clear separation between business and technical issues is important when building an executable process. Our definitions of process types and the examination of their design differences can contribute to a better separation of the process types, and thus aid in the design process. The presented architectures and guidelines can aid this design process further. The architectures are not novel, our contribution is rather to examine them in the context of executable processes.

The three architectures represent three distinct ways to unify technical and business perspectives in a single architecture. The layered architecture and the aspect oriented architecture focused on the business and technical aspects respectively. The domain service architecture is an architecture where technical and business aspects can be "mixed" together. These architectures are on an abstract level, and must thus be combined with other architectural principles to build large scale enterprise systems. In particular, we have not discussed the possibility to "mix" technical and business issues in more than two abstraction layers.

Further work entails examining lover-level architectural principles that can be applied when designing executable processes.

ACKNOWLEDGMENTS

The authors would like to thank Sandvik for providing the input to the example case presented in Section 2. This work is a part of the Serviam project, partly funded by the Swedish Agency for Innovation Systems.

References

1. Piccinelli G., Zirpins, C., Lamersdorf W.: The FRESCO Framework: An Overview. Proceedings of the 2003 Symposium on Applications and the Internet Workshops. IEEE Computer Society (2003), 120-126
2. BEA, IBM, Microsoft, SAP and Siebel. Business Process Execution Language for Web Services (BPEL4WS). http://www-106.ibm.com/developerworks/library/ws-bpel/, (June 9, 2004)
3. Sharp, A., McDermott, P.: Workflow Modeling. Artech House, Inc., Boston, USA (2001)
4. Hohpe, G. et al.: Enterprise Integration Patterns, Addison. Wesley (2003)

5. Henkel, M., Zdravkovic, J., Johannesson, P., "Service-based Processes - Design for Business and Technology", Accepted for the International Conference on Service Oriented Computing, New York (2004)
6. Bubenko, J.A., Jr., Wangler, B.: Objective driven capture of business rules and of information systems requirements. In Proceedings of the IEEE Systems Man and Cybernetics Conference. Le Touquet, France (1993), 670-677
7. Grover, V., Fiedler, K., Teng, J.T.C.: IEEE Transactions on Engineering Management, Vol. 41, Iss. 3 (1994). 276–284
8. Rolland, C.,Prakash, N.: Bridging the Gap Between Organisational Needs and ERP Functionality. Journal of Requirements Enigneeering, Vol 5, Iss. 3 (2000). 180-193
9. Kleppe, A., Warmer, J., Bast, W.: MDA Explained, Addison-Wesley (2003)
10. White, S.: Business Process Modeling Notation Version 1.0, The Business Management Initiative (2004)
11. Jablonski, S.: A Software Architecture for Workflow Management Systems. In Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA'98). Vienna, Austria. IEEE Computer Society (1998). 739-744.
12. Rausch-Scott, S.: TriGSflow – Workflow Management Based on Active Object-Oriented Database Systems and Extended Transaction Mechanisms. PhD Thesis, University at Linz (1997)
13. Garcia-Molina, H.: Modeling Long-Running Activities as Nested Sagas. IEEE Data Engineering Bulletin, Vol. 14, Iss. 1, (1991). 14–18
14. Bass, L., Clements, P. and Kazman, P.: Software architecture in practice. Addison Wesley (1998)
15. Parnas, D.: Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering, (March 1979). 128-138
16. Meyer, B.: Applying Design by Contract. IEEE Computer, Vol. 25, Iss. 10, (1992). 40–51
17. Cysneiros L., do Prado Leite J.: Non-Functional Requirements: From Elicitaion to Modelling Languages. International Conference on Software Engineering (2002)
18. Elrad T., Filman, R., Bader A.: Aspect-oriented Programming an Introduction. Communications of the ACM, Vol. 44, Iss. 10 (2001)
19. AspectJ, www.aspectj.org. (April 2003)
20. Wald, E., Stammers, E.: Out of the Alligator Pool: A Service-Oriented Approach to Application Development. EAI Journal (March 2001)
21. Yang, J., Papazoglou, M.: Interoperation Support for Electronic Commerce. Communications of the ACM, Vol 6, Iss. 43 (2000). 39-47
22. P. Johannesson, B. Wangler, P. Jayaweera,: Application and Process Integration - Concepts, Issues, and Research Directions. Information Systems Engineering Symposium, Springer Verlag (2000)
23. Channabasavaih, K., Holley, K., Tuggle, E. M.: Migrating to a service-oriented architecture, Part2. IBM developerworks, http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa2/, (December 2003)
24. Herzum, P., and Sims, O.: Business Component Factory. OMG Press (2000)
25. Cheesman, J., and Daniels, J.: UML Components. Addison-Wesley (2001)