Service-Oriented Enterprise Architectures: Evolution of Concepts and Methods

Gregor Engels\textsuperscript{1,2}, Martin Assmann\textsuperscript{1}
\textsuperscript{1}Department of Computer Science
University of Paderborn
Warburger Straße 100, 33098 Paderborn, Germany
\{engels, martin.assmann\}@upb.de
\textsuperscript{2}sd&m Research
Carl-Wery-Str. 42, 81739 München, Germany

Abstract

This paper depicts the evolution of enterprise architectures to their today often used service-oriented form and presents a state-of-the-art development process for this kind of architecture. The development process covers both the development of business architecture as well as the appropriate software architecture. While showing up a possible form of further evolution of enterprise architectures, we identify the major challenges for future development methods of enterprise architectures.

1. Introduction

In the last years, service-oriented architecture (SOA) has grown from a hype to a serious enterprise topic. Service-orientation has become more and more popular, as it is mainly a reaction to an upcoming demand in enterprises—efficient and quick reactions of the IT on changing business processes or in one word flexibility.

Enterprises of all sizes, such as Deutsche Post, Credit Suisse, and Wincor Nixdorf now foster or claim that they will foster the development of a service-oriented enterprise. This means many of them begin with the development of service-oriented software. But, to reach a service-oriented enterprise architecture it is not sufficient to just buy SOA software suites from the big players like Oracle, IBM, or Microsoft. These software suites are more or less pure technology and they can never automate the process of enterprise transformation. A crucial part of this transformation process is to establish governance mechanisms and to develop services with the existing software and the new development efforts.

We regard service-orientation as a style for enterprise architectures. It is the result of an evolutionary process over the last decades concerning the style of enterprise architectures. And while the architecture style evolves the corresponding development methods have to be adapted, too.

This is the main intention of our paper: we want to identify the challenges concerning development methods that arise with the further evolution of the enterprise architecture style.

To identify these challenges we depict the historical evolution of the enterprise architecture style. Then, we present a development method for the design of service-oriented enterprise architectures, namely the Quasar Enterprise (QE) method [4].

We will go on by anticipating how the evolution of service-oriented architectures will continue. We will point out the general challenges that have to be overcome by the evolution of development methods.

Before we start with describing enterprise architectures, we define them and their relation to service-orientation.

\begin{center}
\begin{figure}
\includegraphics[width=\textwidth]{fig1.png}
\caption{Service-oriented Enterprise architecture}
\end{figure}
\end{center}

Enterprise Architecture addresses nearly the whole enterprise (cf. Fig. 1). It comprises business and IT
2. Evolution of Enterprise Architectures in the Past

We regard SOA as a concept that is a further step in the evolution of enterprise architecture style reaching back to the early 70’s. It combines many of the concepts developed before. Therefore, we suggest understanding the historical development of the enterprise architecture style in order to better understand what SOA is. Usually, there is a major concept behind each evolution step, often being itself a hype topic in the past.

To illustrate the evolution of enterprise architecture styles we present a running example. For every evolution step we will give reasons why the systems have been built the way they are and why they did not suffice anymore, so that the next evolution step was initiated.

2.1 Beginning with Monolithic Applications

In our running example, we will begin with a very simple setting being depicted in Fig. 2. There is a business process reaching from a customer request to production up to payment handling. All steps involve human work and IT systems which are used via a graphical user interface (GUI). The monolithic IT systems, which nowadays would be called a Customer Relationship Management system (CRM) and an Enterprise Resource Planning system (ERP), are characterized by integrated GUIs, huge functionality and no direct communication with other systems.

The monolithic systems developed in the beginning were often developed directly on purpose, because efficiency was a major design driver. Furthermore, the life cycle of a business process was much longer in the past. Thus, processes were implemented directly in the applications leading to a strong coupling of functionality in these systems. Also, users and developers were either the same or had close contact.

The reasons why these systems did not suffice anymore are relatively simple. Over time the monolithic systems had to be adapted to (slowly) changing processes. Maintenance effort was relatively high because every change in the system required the test of the whole system. In addition, every update made a system less structured and could be regarded as another balcony that was added. The more balconies the more the implementation of new updates was hindered. Not only IT-responsible persons recognized that maintenance was a growing cost factor that should be reduced by improving the architecture of the steadily growing enterprise systems.

2.2 Component Based Architecture

The novelty in Fig. 3 is the component structure within each application. Each component realizes a part of the functionality of an application and has an interface for communication with other components. As in principle the functional expansion or replacement of a component is easier than the one of a huge...
monolithic system, these component based systems facilitated maintenance compared to monolithic systems.

But, often proprietary interface technologies were used so that it was still hard to exchange components or to integrate them with other components. This means that components and thus systems were still tightly coupled.

With the time, requirements from the business process side concerning the integration of different applications arose. Building ever new interfaces between applications implemented with different technologies became a serious cost problem. A simple reason for this is the n:n communication pattern instead of the 1:n communication pattern. In case of diverse interface technologies, n different interfaces have to be implemented for each of the n components in the worst case.

A solution for this problem was offered by the introduction of middleware technology. Now, each component offers an interface in the middleware technology, which allows to communicate with all other applications connected to the middleware. Thus, a 1:n communication became possible.

### 2.3 Enterprise Application Integration (EAI)

Enterprise application integration has been a hype topic several years ago [9]. Its goal was to integrate several applications with each other. The main concept to achieve this was middleware technology like CORBA or MQSeries which strongly decreased the integration effort in enterprise systems.

![Fig. 3: Component based applications](image)

Fig. 4. illustrates the running example with the newly introduced middleware. Now, the third and fourth human actors do not have to access the CRM system anymore. The order data they need for procurement and giving work orders is transferred automatically to the ERP system in a nightly batch run. This saves a lot of effort and reduces the number of possible mistakes in a business process.

In general, this means that a middleware eases the integration of different applications and reduces the human effort that is needed to retrieve information scattered over different applications. A drawback of middleware is that a new technology is required within the enterprise systems.

An additional problem occurred as often applications, especially older ones, offer their functionality only via a graphical user interface. Thus, the middleware can not adapt to it and automation was hindered once again. To overcome this, the GUI design had to be changed.

### 2.4 Separation of GUI

The separation of GUls increased the flexibility of systems because all functionality was now reachable for automation purposes via the middleware. Of
course, the separation increased the design and implementation effort slightly. Fig. 5 shows the separation of graphical user interfaces. Like other components, GUIs have to use middleware interfaces to be connected to applications.

But, unfortunately, ongoing flexibility demands from the business side were still not satisfactorily fulfilled. Those flexibility demands were increasing, as business process lifecycle times were steadily decreasing. Especially since the beginning of the new decade, this topic has been discussed by IT and business experts extensively. Business experts more and more realized that IT is the key technology to implement their processes and not only some minor support technology. At the same time, they realized that the granularity and functional tailoring of applications is not supporting the implementation of new business processes in an optimal way.

From the view of IT experts, an IT landscape with separated GUIs and Enterprise Application Integration might already look very flexible. For business experts it is not that perfect because the tailoring of functionalities within the IT system was made with respect to technical and maybe organizational circumstances. This means that the building blocks of functionality that are preferably be reused from the business process view are not the ones that were developed by IT-experts. This insight might be the initiation of the service-orientation hype.

Business experts that want to be able to react on steady process changes demand the adequate tailoring of business functions that are implemented by IT. In other words, they want to have the alignment of business and IT optimized. For this purpose, the service concept was introduced. In its simplest form a service is a business function that is implemented by IT. For a service it is only interesting what it does and not how it does something. This especially means that a service implementation is not bound to a certain application.

### 2.5 Business-IT Alignment - Basic Services

Fig. 6 depicts an IT-landscape that offers services that are consumed by the GUIs or directly by applications. For example, the procurement service is executed in a batch run every 24 hours. The figure shows an ideal landscape were all functions are exposed by services. However, this is not necessary. Especially during the transformation phase to a service-oriented enterprise there will exist both forms.

There is a new concept showing up at this stage, the Service Registry. It keeps all the information about services that are required to use them. This includes behavioral descriptions, usage costs, availability etc. Potential users of services are enabled to search for services and use them afterwards.

![Fig. 6: IT-landscape with basic services](image)

The services represent a new form of abstraction bringing up advantages and also drawbacks. First advantage is that now functions can be offered that span several applications, which increases the IT-Business alignment. Secondly, the reusability of services is increased if they are tailored well, documented in the Service Registry and if the Service Registry and thus the services themselves are used properly by developers. The drawbacks occur in form of increased effort for identifying, implementing, and maintaining services as well as the related Service Registry.

The role of the GUIs is again important in this scenario, as in case of changing business processes the GUIs have to be transformed, too. For this purpose the use of web-based interfaces, e.g. implemented with portal and portlet technology, can save effort when changing a process and therefore increases flexibility.

Now, we have a very premature kind of service-oriented architecture, as it uses the service concept for the first time. But flexibility regarding the implementation of new processes has not yet reached its climax. This is because until now the control of the process flow is distributed over humans and the whole application landscape. Humans can learn new things quite easily, but applications have to be reprogrammed in their specific language. This is tedious because the programmer has to find the code fragments that change the desired process control flow among other uninteresting code fragments. Consequences of code changes are at least recompiling, testing, and down time for restarting. If there is a way to extract process flow information and make it available in an explicit way, the effort of changing it could be reduced drastically.
2.6 Hard-wired Service Orchestration

From the previous step we already know basic services, representing simple business functions. But for the realization of a process it is not sufficient to just have the single business functions. Each service operation has to be invoked. To automate and make them accessible in a relatively pure form is called orchestrating services. If this is done with services itself it is called hard-wired orchestration.

Fig. 7: Basic and orchestrated services

Fig. 7 shows the orchestrated incoming order service. It represents the process part that begins after the “Accepted” event and ends with the “Material available” event. The incoming order orchestration contains the process control flow information of the named process part and the calls of the subordinate basic services. Such a hard-wired service orchestration can be realized with any technology that is able to call the other services and itself can be used like a service operation, for example a web service that is deployed on an application server. This means the process control flow is now made more explicit as it is separated from other implementation details. Though, it still exists in form of a programming language, which is the reason why we call this a “hard-wired” orchestration.

Orchestrations can also help to reduce data redundancies in systems. Often, data is held several times in different applications and the synchronization is done in time intervals. A service orchestration, as done in the procurement service, easily offers the possibility to avoid redundant data and to retrieve data directly from other applications.

This example shows another advantage of service-orientation known from component-based systems. Both solutions for the procurement service do not bother any users of it, as its interface remains unchanged. Only the hidden part, the implementation is exchanged.

In this example, we have shown two examples for hard-wired orchestrations. The procurement service that might be regarded as reusable, becomes service operation of a basic service, and the incoming order service that embraces a bigger process part and is less likely to be reused. Just like the process part it represents, it is more likely to be changed. In both variants the process control flow was made explicit in the service layer, thus decreasing the effort for process changes. The price to pay for this flexibility is the increasing number of services that have to be maintained.

Until now, service orchestrations could be used to easily implement the process control flow in a relatively well isolated form. But still one has to read the code of several services to gain knowledge about the process control flow. Furthermore changing the flow requires reprogramming several services. In addition, there will also be information in the code that is not relevant for the service orchestration. What are of interest for the business user are the parallel and alternative paths within process models that are implemented in the services. To meet decisions for the alternative paths, parameters of services operations are evaluated and forwarded. To close the gap between business and IT would mean enabling a business analyst to implement new processes. As a business analyst will not write any code, a further desirable enhancement regarding the quick implementation of new processes is to have a visual process modeling language that combines concepts of process models and that is interpretable at the same time.

2.7 Soft-wired Service Orchestration

Fig. 8: IT-landscape with orchestration engine

Fig. 8 introduces a new component of a service-oriented architecture, the Orchestration Engine. It is able to interpret and execute special process models. The most common language for these process models
is the Business Process Execution Language (BPEL) [10].

The concept behind orchestration is not new and known since a longer time as workflow management. It first brought up the idea of making process control flow more explicit and modeling it in an interpretable language. Now, in combination with the middleware used it is possible to link process activities to realized services. At the same time, the business process-oriented tailoring of services eases the modeling of executable process models.

In Fig. 7, we introduced the hard-wired incoming order service, which is now replaced in Fig. 8 by a soft-wired orchestration. It is called soft-wired because it is not programmed, but modeled in a language such as BPEL. Such a model makes it easier to change. In the example, the “give work orders” process step is added to the orchestration. Compared to Fig. 7, where this step was triggered by a human person, it is now automated and integrated in the orchestration. Furthermore, the orchestration is now much nearer to the process description (similar modeling concepts like activities, forks, and joins), but still executable as it is interpretable by the orchestration engine.

2.8 Summary of concepts

We have begun with the introduction of component based software that mainly reduces maintenance costs. To integrate the growing number of applications and their components a middleware has been established (EAI). To be able to use all functionalities of applications within the IT-landscape some had to be decoupled from their only interface – the graphical user interface. This separation of GUI was a precondition for establishing services which offer business functions instead of IT functions. Thus, they improve the business IT-alignment. A service that uses these basic services is a simple hard-wired service orchestration. This means the process control flow lies within the service code. If it is made explicit in an orchestration engine, we call this soft-wired orchestration.

The concepts of an SOA that are realized so far in at least some enterprises can be summarized as follows

1. Component Based (software) Architecture
2. Enterprise Application Integration
3. Separation of GUI
4. Business-IT-Alignment
5. Hard-wired service orchestration
6. Soft-wired service orchestration

This outline of the historical development of enterprise architectures has shown, too, that the service concept and in particular the right, business-oriented choice of a service is crucial in reaching flexible service-oriented enterprise architectures. Thus, an appropriate development method for modeling and realizing enterprise architectures is needed.

3. Developing Service-Oriented Enterprises with the Quasar Enterprise Method

This chapter summarizes the Quasar Enterprise (QE) Method [4,6] as a representative for a development method for service-oriented enterprise architectures. While there are also other approaches for realizing service-oriented architectures (e.g. as described in [8] by Krafzig, [5] by Erl, [3] by Dostal, and [11] by Woods), the Quasar Enterprise method provides detailed guidelines for designing and realizing services which are driven by business goals and business requirements.

Fig. 9 shows the main steps of the QE Method as a roadmap within the generic Integrated Architecture Framework (IAF) [7]. The 5 arrows represent the 5 main development steps:

1. Analysis of Business Architecture
2. Definition of an Ideal Application Landscape
3. Evolution Planning
4. Integration of Components in Application Landscape
5. Selection of Integration platforms

Step one analyses and defines the business architecture. The QE method suggests starting with business goals and deriving architectural guidelines from that. Afterwards, business services are identified and described.

The second step defines an ideal application landscape as destination for a sequence of transformation steps starting with the existing IT landscape. It begins with the definition of domains as underlying business-driven structure of IT applications [6]. Application services are derived from the high-
level business services, are assigned to domains and grouped in suitable (logical) application components. The components’ interfaces and operations are identified and the communication between components is described as coupling architecture.

Step three of the QE method provides guidelines how to structure the evolution process from an existing IT landscape into the direction of the identified ideal landscape. On the basis of an evaluation of the as-is architecture and the ideal architecture the target architecture for each evolution step is determined.

The fourth step takes care of the integration architecture. At first the integration demand between logical components is identified. If done so, the architect has to decide whether he wants to realize the integration in a conventional way or with orchestration. The conventional way can still be the way of choice, e.g. when performance is critical for mass data exchange. An orchestration, especially a soft-wired one, is usually performing worse than a conventional integration. If there are still physical interfaces left that have not been covered with the integration patterns, these are defined now. Afterwards the physical coupling is defined, which means the use of technical services is determined.

The fifth and last main step takes care of the concrete integration platform with which the integration planned in the step before is realized. At first, by a technical analysis, the existing integration technologies are examined and then integration technologies that shall be used in the future are determined. From the integration architecture from the previous step the method picks the integration tasks and formulates integration scenarios. Both, the technical analysis and the integration scenarios are used to define technical services. Now, the existing integration products that are suitable for the realization of single products are selected. Afterwards the deployment of the integration platform can begin.

This rough description of the QE method gives an overview of the most important aspects that have to be taken into account when planning a service-oriented enterprise architecture. The next chapter will point out what else will have to be taken into account when the enterprise architecture style will evolve in the future.

4. Future evolution and development challenges

With the QE method we have shown a way to develop a service-oriented landscape in an enterprise. Now we will continue with our running example from chapter two and thereby depict our view on the future development of enterprise architectures. For every evolution step we point out the drawbacks and changing requirements that might lead to the next evolution step.

4.1 Services with Human Interaction

The evolution of the past (compare Fig. 8) provides a high flexibility for IT-processes. With the means provided by now human interaction is a serious problem because it cannot be integrated in any service orchestration. Until now there is no way to invoke a task on a human and no way for the orchestration engine to register when and how a task with human interaction is fulfilled.

The second person from the left in Fig. 8 just initiates the orchestration. In this scenario orchestrations are only interpretable if no humans are involved. Every time a human interacts in the process the process control flow held by the orchestration engine is lost. But human interaction can also be seen as part of a service implementation. “Send order confirmation” is a business function and therefore a possible service operation. On the one hand its implementation could be an employee that writes letters and sends them via mail to the clients address, on the other hand this could be fully automated with an application that retrieves the required data and sends the confirmation via email.

To generalize the example we suggest treating human work just as any other service. If so, the orchestration engine is able to interpret any kind of process. Theoretically nothing speaks against human interaction within service implementation; the problem lies in the technical realization.

In Fig. 10 human actions are triggered like any other service operation. Of course some extra effort is required to provide this possibility. Generally if a service with human interaction is triggered, then the potential actors have to be informed. The task has to be delegated to an employee who will actually execute the action. After completing the task, he has to transfer the results back to the orchestration engine, which then regains control of the process flow. A role concept for human actors is very helpful to be able to inform those and only those employees being able to fulfill the task.
BPEL4People is an extension of the Business Process Execution Language that provides support for human interaction [1]. It comprises a role concept, task delegation and support for scenarios like escalation and the four eyes scenario.

The integration of human work in service operations is laborious but allows to model whole processes in an interpretable language like BPEL. The integration in development methods like the QE method should make things easier as before, as the design of services is implementation agnostic. Not till then the implementation of services starts, the question arises whether they are better implemented with IT or with humans. Without a modeling support of human interaction, services finally implemented with human resources can not completely be treated like services (e.g. in orchestration demands). To realize human interaction the interface between any kind of human action and the orchestration engine has to be mastered and on top of that a role model should be developed.

An enterprise following the architecture style in Fig. 10 can implement new processes quite fast and efficient. But how useful is it to be able to change a process within minutes if the process itself lasts for days? Due to ever stronger pressure in the market, process execution times have become a critical aspect and should also be fostered by the enterprise architecture.

At first, potential time savings that can be addressed by an architectural style have to be identified. Often time is wasted between actions and not during them. In the running example there is only one process, but the real world is a little more complicated. There are a lot of processes within an enterprise as no one should be able nor should have interest in modeling all activities of an enterprise in a single executable process model. There will always be several processes and they have to be invoked when certain conditions are fulfilled. These conditions are mostly distributed over the enterprise or base on external events. Events can also be regarded as a condition that is fulfilled and maybe it has to be reacted on in a certain time. Generally, there are two possibilities to check these conditions.

First one is checking them on a regularly basis. But, this might mean that one loses nearly a whole cycle length if the conditions get fulfilled just after they were checked.

Second one is to create notifications when something changes. These notifications have to be monitored and then reactions can be invoked immediately when all conditions got fulfilled. This is not as easy as it might sound, so we will see which consequences for the enterprise architecture style arise.

4.2 Event-Driven Architecture (EDA)

In Fig. 11 several new concepts are introduced. On top of the service layer, a complex event processor (CEP) is added that receives all events (notifications) generated by event producers (any services or applications may create events). The CEP also holds subscriber lists. Any application can subscribe for a type of event. If so, the dispatcher forwards the event to the subscriber immediately after its occurrence.

The event correlator is a component of the CEP that receives a copy of all events. It checks the flow of events on occurrence patterns where several events are involved in a certain time. These patterns are specified in correlation rules. An example is if an error occurs in a process usually nothing is done, because a single error is not regarded as problem that requires intervention. But if the error occurs ten times within an hour, a process for quality improvement should be invoked. The event correlator can recognize this easily, but on the basis of single events this would not be possible. If such a rule is fulfilled the correlator fires a
new (complex) event, which then is treated as any other event by the dispatcher.

Another very important event consumer is the orchestration engine. It can invoke new processes if a certain event occurs. This means a further decoupling of the IT-landscape. By soft-wired orchestration the process control flow within processes was made explicit. With an event-driven architecture and an event listening orchestration engine the control flow between different processes can also be made explicit. Without events any kind of application or human had to check some conditions in the system until a reactive process is launched.

For example, a batch script might check the log files for errors every night. As it is executed every 24 hours, only a delayed reaction is possible. If it finds more than ten, it invokes a quality insurance process. In order to do so, it has to know its information source and any kind of application or address where the process can be invoked. This means that the control flow logic is hidden in an application or in human minds. With the event handler of the orchestration engine this control flow can be made explicit and a lot of time can be saved as reactions are executed immediately.

For the development process of enterprise architectures this means that complex and simple events have to be identified and also implemented.

This kind of architecture is very flexible and allows a quick process execution. But with the frequent business process changes a new drawback occurs in enterprises. A business process model, just like a piece of code or any other result of human work, is rarely perfect. During its lifetime it often undergoes several (minor) changes to improve its quality. One of the problems with the quality assurance of processes is the recognition of errors. The reason for this is that every business process can have different characteristics that mirror its quality. The present architecture does not support the monitoring of processes, so that their weaknesses cannot be recognized as soon as possible. But due to ever more frequent changing business processes, this becomes a driving demand in enterprises.

4.3 Business Process Monitoring

The novelty in Fig. 12 is the business process monitor. For every business process it holds several performance indicators, like the execution time between specific steps or the number of used resources during process execution. In order to gain knowledge about these it has to retrieve the information from the IT-landscape. If it would hold an interface to every component that provides information required for a performance indicator, it would relatively soon deliver wrong results, as the sources for specific pieces of information can change their locations over time. Furthermore, it would hardly be possible to get informed about the exact point in time when an action was executed.

A more sophisticated method to retrieve the required information is to get it delivered. The events we introduced in the previous section are the method of choice for this task. Any performance indicator has to be calculated with any kind of event and some rules what do to with several events. The example with the number of used resources requires a “resource used” event that is added up with every occurrence in the context of a certain process. For the execution time, start and end events of the according steps are read and the duration time is calculated. Furthermore, a performance indicator can have thresholds and upon the crossing of a threshold an alert is created, maybe in form of an event or visualization on a dashboard.

By this, the quality characteristics of a business process are monitored in real-time, enabling business analysts to react immediately on quality problems. With low thresholds and indicator trend analyses even pro-active reaction can be invoked.

Of course, this new feature has to be integrated in development methods like QE. Just as events performance indicators have to be identified at first. These indicators are heavily influenced by the business goals the enterprise aims at. The realization of those performance indicators has to be planned, that means which events are required to measure it and how have the events to be processed.

5. Conclusion and Outlook

In this paper, a possible scenario for the evolution of the enterprise architecture style is depicted. Additionally, several challenges for the development of
development methods for enterprise architecture are pointed out.

Firstly, human interaction has to be supported, so that it can be treated just like any other service. For the realization, some kind of software is needed that builds a configurable interface between an orchestration engine and the human actor. It has to provide the information required to fulfill the actor’s task and to receive and to forward the results. In addition, it should be able to distribute the tasks among the users. A role model is more than helpful for this task. Both the role model and the configuration of the interfaces between human actors and orchestration engines have to be taken into account during the development process.

Secondly, an event driven architecture requires the identification of events, locating their producers and their subscribers, the planning of their correlation rules, and the invocation of process orchestrations on certain events. The generation of events takes place all over the IT-landscape and might not be a simple task. Moreover, the middleware platform has to ensure the immediate and reliable transmission of events.

Thirdly, the monitoring of processes has to be installed. It requires an event-driven architecture. For a development method it means that it has to support EDA plus the identification of performance indicators and how these can be computed with the help of events. Events that have to be implemented especially for the process monitoring have to be fitted in the development process of the other events. Additionally, the process monitor itself has to be planned with the help of a development method.

The three mentioned challenges are the ones that are deduced from the anticipated evolutorial development of enterprise architecture style. In addition, we identified two further challenges that are not yet addressed in detail enterprise architecture development methods. Neither in Quasar Enterprise, nor in those as e.g. of Erl [5] or Krafitz [8].

The first additional challenge concerns the tailoring of huge enterprise applications like the ones from SAP or Oracle. Their applications have to be tailored into services, too, but often this is hardly possible as the huge applications do not allow tailoring as desired. The reason is that it is hardly traceable what is happening within the system if a certain action is executed. This means decoupling of functions is not possible, because the side-effects of an action are not known properly. Moreover, those systems can have their own small workflow management components, which might be hardly integrable with the orchestration engine, which is an enterprise-wide workflow management system.

The second additional challenge arises when the SOA-style is established and the interaction with other enterprises for shared business processes becomes of greater interest. The scenario involving several service-oriented enterprises is named choreography because enterprises act on their own and are not steered by a central unit like the services in an orchestration. There will be much potential in coordinating the use of services within shared business process. BPMN4Chor is a BPMN based approach, suggested by Decker et al. in [2], which already addresses this problem on the modeling level, but the concrete realization of shared business processes requires also the contractual and the technical realization.

References
[2] Decker, Gero; Kopp, Oliver; Leymann, Frank; Pfitzner, Kerstin; Weske, Mathias: Modeling Service Choreographies Using BPMN and BPEL4Chor, CAiSE 2008, p.79-93.
[4] Engels, Gregor; Hess, Andreas; Humm, Bernhard; Juwig, Oliver; Lohmann, Marc; Richter, Jan-Peter; Voß, Markus; Willkomm, Johannes. Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten, dpunkt: Heidelberg, 2008.