Patterns for Business Object Model Integration in Process-Driven and Service-Oriented Architectures

Carsten Hentrich
CSC Deutschland Solutions GmbH
Abraham-Lincoln-Park 1
65189 Wiesbaden, Germany

chentrich@csc.com

Uwe Zdun
Distributed Systems Group
Information Systems Institute
Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040 Vienna, Austria

zdun@acm.org

ABSTRACT

Service-oriented architectures often have the goal to integrate various systems of one or more organizations in a flexible way to be able to quickly react on business changes. Integration based only on services, however, falls short in reaching this goal because the application-specific business object models of multiple external systems (especially legacy systems) need to be integrated into the service-oriented system. When multiple business object models must be integrated into one system, serious data integration issues might arise. Examples of such problems are incompatible data definitions, inconsistent data across the enterprise, data redundancy, and update anomalies. We present patterns that address these issues and describe how to integrate the application-specific business object models of various external systems into a consistent process-driven and service-oriented architecture.

1. INTRODUCTION

Service-oriented architectures (SOA) are an architectural concept in which all functions, or services, are defined using a description language and have invokable, platform-independent interfaces that are called to perform business processes [1, 2]. Each service is the endpoint of a connection, which can be used to access the service, and the interactions are relatively independent from each other (e.g., stateless services are favoured over stateful services). On top of the various layers implementing the foundations of a SOA, we find in many SOAs a Service Composition Layer that deals with service orchestration, coordination, federation, and business processes based on services [8]. In this paper, we consider architectures in which the Service Composition Layer provides a process engine (or workflow engine) that invokes the SOA services to realize individual activities in the process (aka process steps, tasks in the process).

The most important goal for using a SOA is often to integrate heterogeneous systems in a flexible manner so that organizations can quickly react on changes in the business. One important aspect in this respect is that usually the SOA is used for integrating a number of external systems. With this term we refer to systems that are not yet integrated into the SOA. External systems include systems of the organisation that realizes the SOA or systems of other organisations. Typically, many of the external systems are "legacy systems". But there are many other kinds of external systems, for instance, standard systems like SAP or other third party systems. One of the key ideas in recent SOA definitions is to save the investment that has been made in existing IT infrastructure and applications and provide flexible means for integrating them. This, however, is difficult, as most of these external systems have been independently developed, or at least there is a certain level of independence in their historical evolution. For this reason, they often implement heterogeneous data models.

This is not necessarily a problem because this is where stateless services can help. In a SOA, the most important conceptual pattern of integration is to offer SERVICES [3] that provide the integration of an external system. To assume that services alone are sufficient to design a larger SOA, however, is not enough. When various business object models need to be integrated into a SOA, often a purely SERVICE-based integration is infeasible or impossible because of data integration issues. Examples are incompatible data definitions, inconsistent data across the enterprise, data redundancy, data incompleteness, data availability issues, data ownership issues, or update anomalies. All these problems can only be addressed at a broader scope than a single service. In practice, often massive hand-coding efforts are used to resolve these issues, which require a lot of time and are often hard to maintain in the long run. Instead of using such "ad hoc solutions" it is advisable to follow a more systematic approach – both in terms of the refactoring processes and the architectural solutions.

As a real world example, consider an automobile rental company that has grown in the last years, has merged with two other companies, and now consists of three independently working territorial branches. Each branch represents a company being acquired over the years to serve a territorial market. Transparent business processes shall now be implemented, following a SOA approach that allows renting cars via the Internet, independent of the territorial assignment. The data models in the various branches are different, as each branch uses independently grown systems. Moreover, customer data is redundant in these systems: They use inconsistent automobile identification mechanisms,

PLoP '06, October 21–23, 2006, Portland, OR, USA.

there is inconsistent formatting of data, and there are incorrect or incomplete values in the data fields. If common business processes shall be implemented for these branches, these data issues must be resolved first.

Certainly, the cost for resolving these issues needs to be balanced with the business case associated to improving the business processes. However, in this paper we assume that this business case has been made and concentrate on the solutions of resolving these problems. The discussion concerning the business case should be made separately and prior to starting an engagement or project in this direction. For this reason, we will not consider these aspects any further. On the other hand, the problems and solutions provided in this paper can be used to lead such a discussion and to reason about cost issues in relation to a business case. In this paper we primarily present how to deal with these issues and thus make a project successful.

In this paper, we explain proven practices – in patterns form – for dealing with these crucial problems of systems integration. The patterns interpret the data models of external systems, as well as the data models defined in the service architecture, from an object-oriented (OO) perspective, and hence we call these data models business object models. When integrating systems via a process-driven and service-oriented approach, application-specific business object models need to be consolidated somehow and integrated via the process flow.

Please note that the process-oriented and service-oriented perspectives advocate a more behavioural, stateless view on the system than objects. However, they usually perform operations on data. This data can be represented in many different ways. We assume the use of an object-oriented model of the access to data in a process-driven SOA to follow the business object concept. This is a proven practice, especially for larger process-driven SOAs (for details see [4]).

Often it is necessary to adapt or change given data models to understand them from an object-oriented perspective, for instance, if a legacy system offers a procedural interface to its data model. Because there are many different building blocks used for representing state and/or access of business data, such as objects or procedures that access data in a database, below we generally use the term entity to refer to the different kinds of building blocks of external systems (following the ENTITY pattern from [3]).

The patterns contained in this paper, offer solutions that allow to integrating various business object models. We present three refactoring patterns that explain basic alternative steps for consolidating two individual business object models. And we also describe three architectural patterns that allow you to build a consistent large scale architecture that is able to consolidate multiple business objects.

In fact, data seems to be a forgotten child in SOA approaches. One could ask, why we propose an approach considering OO while also being service-oriented. Do these approaches not contradict each other? We are convinced, the answer is no, as services need to deal with data structures to describe and define the input and output parameters of the services. These parameters are usually not simple data types but rather represent complex structures that can be interpreted as objects. In our opinion, SOA and OO are, for this reason, complementary approaches. We apply OO concepts to tackle the issues related to the "data" perspective in SOA that is rather a functional than a data-driven approach. OO offers suitable concepts for describing data structures, which fits very well with current programming languages and technology used in conjunction with SOA, such as J2EE or .NET. Object-oriented languages are still leading edge in these recent technology approaches related to SOA. As a result, we propose an OO approach for tackling the data related issues in SOAs. The patterns in this paper thus contribute to solving data issues in SOA.

We present an example at the end of the paper to demonstrate the application of all patterns and to outline the pattern relationships. Please note that it might be useful for the reader to jump to the example from time to time while reading the patterns to grasp a concrete example of a pattern that is currently investigated.

2. PATTERNS OVERVIEW

In this paper, we first present three refactoring patterns that explain basic alternatives for how to change a system in the situation that a single business object model of an external system should be integrated into a process-driven architecture:

- WRAP SERVICE AS ACTIVITY – explains a refactoring solution that introduces one or more services for an application-specific business object model. The pattern's solution is to wrap one or more of these services using a process activity type that can be flexibly assembled in process models.
- RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL – explains a refactoring solution that restructures a specific business object model of an integrated external system. The external system restructuring is done in a stepwise, minimal manner until the external system meets the new requirements introduced by the process-oriented architecture. WRAP SERVICE AS ACTIVITY can be used to offer service interfaces to the restructured system.
- SYNTHESIZE BUSINESS OBJECT MODELS – explains a refactoring solution that synthesizes a specific business object model of an integrated external system and a common business object model of the process-oriented architecture.

These three refactoring patterns explain basic alternatives for refactoring a single business object model into a "harmonized" model of a process-oriented architecture. However, in larger systems, it is necessary to consider multiple refactorings of business object models and their interdependencies from the perspective of the whole process-driven SOA. This cannot be explained in terms of a single refactoring process, but must be addressed at the architectural level. We present three architectural patterns that are applied in this context:

- INTEGRATED BUSINESS OBJECT MODEL – explains an overall architectural solution that allows you to implement a harmonized business object model. Each of the three refactoring patterns can be applied when it is most appropriate. But still a consistent architecture is produced.

- DATA TRANSFORMATION FLOW – explains an architectural solution based on a process subflow for data transformation that maps different application-specific business object models to a common business object model. The goal is to enable flexible integration of various external systems.
- BUSINESS OBJECT POOL – explains an architectural solution in which a central pool for the business objects enables processes that have logical interdependencies. The processes can hence interact with each other without comprising their technical independence.

Figure 1 shows an overview of the pattern relationships. There are a number of external patterns that play a role in the patterns introduced in this paper. We present thumbnails for these patterns in an appendix at the end of the paper.

3. WRAP SERVICE AS ACTIVITY

External systems, i.e., systems that have so far not been part of the process-driven SOA, should be integrated in to a process-driven SOA. In many cases, the external systems are legacy systems.

◆◆◆

Existing interfaces of external systems often do not reflect the requirements of a process-driven architecture. Loose coupling – a main goal of any SOA – for instance is often not well supported because the external system only offers stateful interfaces. Or, the required communication protocols of a process-driven system are not supported by the external system. However, flexible interfaces to external systems are required to flexibly assemble processes involving external system invocations from within a process design tool – which is a central goal of a process-driven SOA.

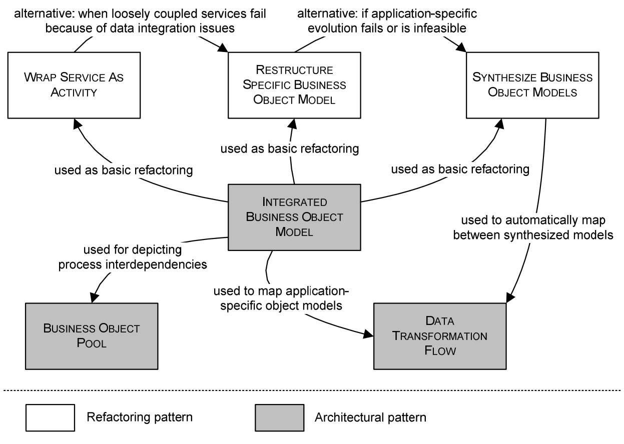

Figure 1. Patterns overview

In a SOA, the most important pattern of integration is to offer SERVICES [3] that provide the integration of an external system. A SERVICE is an operation offered as an interface, without encapsulating state. SERVICE interfaces solve the basic problem of how to represent loosely coupled interfaces. However, loose coupling is hard to achieve, if the external system design forces us to hard-code dependencies to stateful interfaces or communication protocol details in the process models or integration code. For a connection to the process-oriented layer, we must also meet the requirements of the process-oriented SOA, but most often the external system does not fulfil them a priori. Again, we do not want to hard-code them in the process models, which should be kept flexible, changeable, and understandable to the domain expert.

Typically, a central requirement is that the SERVICES can be used to integrate any kind of system in the same way and allow process designers to flexibly assemble processes from the SERVICES offered by the external systems. The SERVICES should hide all details of the communication with the external system from the process designer. Consider, for instance, integrating a mainframe that only supports batch processing. From the perspective of the process designer this system should be integrated in the same way as a Web Service that was specifically written for this task. However, different service developers use different approaches to design SERVICES and integrate them into process models. This means, the desired information hiding is hard to achieve, and process designers must cope with these differences.

An inhouse guideline for SERVICES development can solve this problem only partially. For instance, if services are used that are not developed inhouse (e.g., services offered by an external standard systems like SAP), guidelines on their design cannot be imposed.

◆◆◆

Refactor the external system and the process-driven SOA using the following steps: For each entity in the external systems that needs to be exposed to the process-driven architecture, define one or more stateless SERVICES on top of the existing interfaces of the external system. Define a special SERVICE activity type in the process engine that wraps invocations to external services. This way, SERVICE invocations are represented as atomic activities in the process flow. The SERVICE activity type can be used in business processes to flexibly assemble services, because all details of the communication with the external system are hidden in the wrapper activity. Instantiate and use the SERVICE activity type in process models whenever an external system needs to be invoked.

The main task of the SERVICE is to translate a service-based invocation into the interface of the external system and translate the responses back into a service-based reply. Hence, the relevant interfaces of external systems are integrated into the SOA using SERVICES, exposing a view on the external systems that reflects the requirements of the process-driven SOA.

The goal of decoupling processes and individual process activities, realized as SERVICES, is to introduce a higher level of flexibility into the SOA: Pre-defined services can be flexibly assembled in a process design tool. The technical processes should reflect and perhaps optimize the business processes of the organization. Thus the flexible assembly of services in processes enables developers to cope with required changes to the organizational processes, while still maintaining a stable overall architecture.

In cases, where a service exists or can be built that equals the required meaning of a process activity, an activity can be mapped to exactly one service. However, in reality this is not always possible. For instance, an activity in a process might need to wrap a whole set of application services because each service only fulfils a part of the overall functionality requested by the more coarse-grained process activity. The main driving factor for the integration of services and process activities should always be that the process activity type needs to be understandable in the context of the process models. A one-to-one integration between service and activity is very easy to build and maintain. Hence it should be chosen if possible, but only if its meaning fits well into the context of the process model. There are other driving factors for the integration of services and process activities, such as reusability of services in different activity types or design for foreseeable future changes.

Very often more than one application needs to be wrapped to fulfil the goal of the activity (as shown on the right hand side of Figure 2). Consequently, designing and implementing the integration of the activity with application services is not trivial and introduces a whole new set of problems. These problems are addressed in more detail by the PROCESS BASED INTEGRATION ARCHITECTURE pattern [4]. This pattern provides an architectural concept for achieving that integration. Especially, the MACROFLOW INTEGRATION SERVICE pattern [4] – a typical part of the PROCESS BASED INTEGRATION ARCHITECTURE – is very important in this respect, as it depicts the functionality requested by a process activity as a one service, which is composed of more fine grained services. These patterns thus allow developers to solve issues that arise when the services cannot be directly designed and implemented according to the requirements of process activities and directly invoked via the process flow.

Figure 2 illustrates the refactoring from a process model and applications that offer only stateful interfaces to a process model that wraps services of those applications in its activities. There are two possible options for the mapping:

- Services can be designed and implemented to represent requirements of process activities directly.
- Application services can only be designed and implemented to fulfil parts of the process activities.

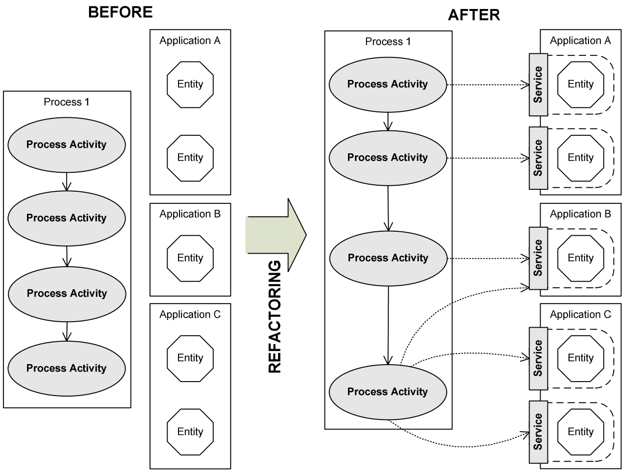

Figure 2. Refactoring to services that are wrapped by activities

Actually, this wrapping implies important design decisions, as the process activities will be designed in dependency with the services. Ideally, the application services can be designed according to the requirements of a process activity. However, on the other hand, processes might change and thus the requirements might change. For this reason, it is often better to provide the services in terms of self-contained functions of an application that are based on the entities of the application. That is, the services are designed according to the specific business object model applied by an application. The consequence is that processes and application services are more loosely coupled and thus more flexible. There is the trade-off, however, that larger integration effort and greater complexity for implementing the integration is required.

In this respect, the MACRO-MICROFLOW pattern [4] can be used to conceptually decouple the fine grained application services that are required within the integration context from long-running processes. Following MACRO-MICROFLOW, the fine grained application services are orchestrated in a microflow, i.e., a more fine grained technical integration process. The PROCESS BASED INTEGRATION ARCHITECTURE pattern provides flexible means for implementing both the one-to-one and the one-to-many relationship between process activities and application services.

4. RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL

External systems, i.e., systems that have so far not been part of the process-driven SOA, should be integrated into a process-driven SOA. In many cases, the external systems are legacy systems.

◆◆◆

When integrating systems into a process-driven architecture, the first choice should be to follow WRAP SERVICE AS ACTIVITY. This, however, might fail because the external system is a legacy system that is not structured in a suitable way to allow for offering an object-oriented business object model via SERVICES. Or the business processes might require an integration of data from two or more application-specific business object models, and service-based access to the data is not enough to deal with the data integration problems. Or the external system does not even allow services to access the data.

Some legacy systems only offer unsuitable interfaces that are hard to map to an (object-oriented) business object model design or to a service-oriented design. Consider, for instance, a legacy system has a procedural design that can be understood as an object-oriented business model. Or the legacy system does not offer session abstractions that can be used for aligning interdependent stateless service invocations, and hence the performance of interdependent invocations is weak.

If the data types of two external systems are incompatible and cannot (easily) be mapped, it might be necessary to think about a better solution than performing individual mappings within wrapper SERVICES (maybe over and over again). In addition to data mapping problems, it might be possible that an external system does not offer appropriate interfaces to access the relevant data at all via a pure wrapper SERVICE. Sometimes the data is accessible, but not in a suitable way. Consider for instance a legacy system that offers only a batch interface. It might be possible that the performance of this interface is not good enough for an integration task. Or the data model and the interfaces require repetitive invocations via the wrapper SERVICE which downgrades the performance of the overall system. In other words, often the external system was designed without having the requirements of integration in a SOA in mind, and thus cannot fulfil the requirements of the SOA.

Such data integration issues can arise even when the developers only need to integrate two interfaces. Consider a simple point-to-point integration between two systems is needed. In this simple case, the interfaces between the two integrated systems need to be mapped to exchange data. This is only possible in simple wrapper SERVICES if the mapping of (data) types can be completely performed in the service implementation.

In a larger SOA with a dedicated service orchestration layer things get even more complicated. The reason for this is that the different business object models of the involved external systems need to be consolidated somehow to achieve a flexible orchestration within the process flow.

◆◆◆

Refactor the external system and the process-driven SOA using the following steps: First assess whether a restructuring is possible according to the following criteria. The system evolution should be as non-intrusive and minimal as possible. It should not break existing client code. Substantial portions of the system should remain unchanged. If the assessment is positive, restructure the application-specific business model of an integrated external system by evolving the system to meet the new requirements introduced by the process-oriented architecture. Next, offer service interfaces so that the business process can access the evolved external system following WRAP SERVICE AS ACTIVITY.

Before applying a restructuring of an application-specific business model it is necessary to consider that it may not be possible at all or with acceptable effort to restructure the business object models of legacy applications such that they work consistently together. The requirements of the business processes need to be considered by a business object model designer so that the business object model is suitable for representing the domain architecture of the business processes. Also, it is necessary to consider changing requirements, e.g., in case another legacy application needs to be integrated in a process flow. It is important to consider whether a restructuring can be done with minimal changes so that existing assets are preserved and existing client code is not broken. That is, existing external interfaces should remain compatible.

A restructuring should only be performed, if all these considerations lead to the conclusion that it is possible to restructure the application-specific business object model of an external system. If additionally the restructuring is possible with acceptable effort, it should be considered before considering integration following SYNTHESIZE BUSINESS OBJECT MODELS. This is because RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS will be quite effective: Most often it is easier to make local changes to a system's data in the system itself then to evolve the data in an external mapping component (which is part of the business process).

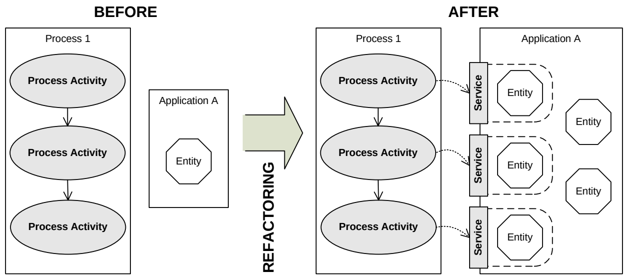

Figure 3. Refactoring by restructuring an application-specific business object model

Figure 3 illustrates a refactoring process based on a restructuring of an application-specific business object model: One monolithic entity is split into a number of entities. Some of them are exposed as services. These services are then integrated following the WRAP SERVICE AS ACTIVITY pattern. Please note that this is just an example of a restructuring. Many other restructurings are also possible. The goal is to preserve the existing assets as far as possible and not break existing client code.

Applying RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS is often the only way to be able to integrate two business object models. In some cases, it is relatively easy and not much work. However, the restructuring might also be infeasible or inapplicable. The evaluation whether the pattern is infeasible or inapplicable might be non-trivial. In some cases, to RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS might be a big effort and sometimes the effort is underestimated.

5. SYNTHESIZE BUSINESS OBJECT MODELS

External systems, i.e., systems that have so far not been part of the process-driven SOA, should be integrated into a process-driven SOA. In many cases, the external systems are legacy systems.

◆◆◆

Consider integrating systems into a process-driven architecture using WRAP SERVICE AS ACTIVITY fails because of data integration issues, and RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS proves to be difficult, infeasible, or even impossible, because the external systems cannot or should not be changed or adapted. Local, independent changes in the application-specific business object models are often not enough to resolve data integration issues, such as incompatible data definitions, inconsistent data across the enterprise, data redundancy, and update anomalies.

Data integration issues, such as incompatible data definitions, inconsistent data across the enterprise, data redundancy, and update anomalies, can occur when integrating data or interfaces of two or more systems into a process-driven architecture. These issues can often not be resolved in a suitable way using only wrapper SERVICES. Usually, in such cases one should try to apply RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS next. But consider a legacy system where the source code is not available. Or no experts for the languages or platforms used by a legacy system are working for the company anymore. Or a significant investment is needed to make changes to the legacy system, and the extra costs should be avoided. Such situations are highly unwanted, but nonetheless they occur.

Let us consider the other case; to apply RESTRUCTURE SPECIFIC BUSINESS OBJECT MODELS is possible and feasible. The pattern might, however, be still not applicable, if a "global" perspective is needed for data integration. Consider for instance two or more application-specific business object models need to be integrated in a process flow. Sometimes data integration issues cannot be (effectively) solved by only changing the local applications. For instance, if one data model depicts an address as a custom data record, and the other one as a string, we need to write conversion code between the two incompatible data types at the "global" level. That is, we create a "global" view based on the combination of the information in the different application-specific business object models.

◆◆◆

Refactor the system using the following steps: Design a synthesized business object model that consolidates the structures of the involved business object models. Map the relevant parts of the application-specific business object models into the synthesized business object model, and perform the data integration tasks at the global level. The synthesized business object model depicts the requirements of the related business processes, i.e., it provides a process-related, global view on the application-specific business object models.

The parts of the application-specific business object models that are subject to exposed services are mapped into the synthesized business object model. The exposed services are usually integrated into the process flow using wrapper SERVICES that are invoked by activities in the process flow.

The application-specific business object models can be mapped to the synthesized business object model by some well-defined mapping rules to automate the mapping, for instance following the DATA TRANSFORMATION FLOW pattern.

Figure 4 shows a business process design and two applications that can be accessed via service interfaces (e.g., external wrapper services). Consider that the two applications cannot be changed and data integration issues arise. The figure illustrates the refactoring process from this situation to the introduction of a synthesized business object model. The synthesized business object model provides a consolidated model of the two application-specific models. It especially fulfils the requirements of the business processes.

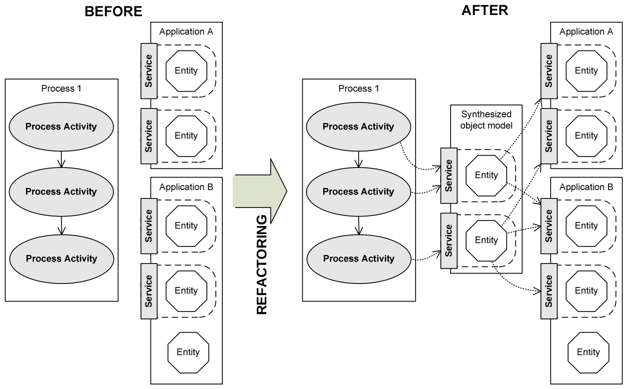

Figure 4. Refactoring to a synthesized business object model

The synthesized business object model design has to consider all requirements of the process domain, in terms of the services that the processes need to expose. The model must be consistent with all integrated applications and with the service requirements of the processes.

6. INTEGRATED BUSINESS OBJECT MODEL

External systems, i.e., systems that have so far not been part of the process-driven SOA, should be integrated into a process-driven SOA. In many cases, the external systems are legacy systems.

◆◆◆

The three refactoring patterns WRAP SERVICE AS ACTIVITY, RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL, and SYNTHESIZE BUSINESS OBJECT MODELS explain alternatives and considerations for integrating a single business object model interface into a process-oriented SOA. If multiple external applications and business object models need to be considered, often none of the three alternatives alone provides a suitable solution. Also, the process flow might be offered itself as a service and needs to provide a harmonized, consistent view on the integrated application-specific business object models. The different integration solutions must be managed and offered in way that they can be flexibly assembled from a process design tool.

The process flow needs to operate with a business object model, i.e., the business objects being associated to the process and being manipulated by the process. Moreover, often the process is a function itself and represents a service. The input and output parameters of this service relate to the business object model of the process. The requirements on the business object model of a process and the business object models of external systems integrated in the process usually vary. That means all the business object models under consideration are usually not consistent – and need to be harmonized.

The various business object models implemented by external systems will thus be reflected by the parameters of the application services that are used to access them. These services simply reflect the interfaces in terms of the business objects used as input and output.

As a result, one has to deal with the problem of harmonizing the business object models of the various applications to integrate them via a configurable process in some way. The problem even gets worse if multiple processes need to be integrated. In this case many requirements of these processes need to be represented in the corresponding business object models. Consequently, greater conflicts will be observed between the business object models of the processes and those of the external systems.

◆◆◆

Provide an INTEGRATED BUSINESS OBJECT MODEL for a process-oriented SOA as an architectural solution. In the design of the INTEGRATED BUSINESS OBJECT MODEL use the following guideline: For each application-specific business object model first try to WRAP SERVICE AS ACTIVITIES. If this does not work for an interface of an application-specific business object model because of data integration issues, assess whether an integration solution based on RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL or SYNTHESIZE BUSINESS OBJECT MODELS (or both) would work better, and then follow the chosen refactoring pattern. Integrate the result of the refactoring using WRAP SERVICE AS ACTIVITIES into the process model. The INTEGRATED BUSINESS OBJECT MODEL uses appropriate metadata description mechanisms to keep the model flexible concerning changing requirements.

The INTEGRATED BUSINESS OBJECT MODEL pattern introduces an architecture which allows developer to use each of the three refactoring patterns when it is most appropriate. The "standard" solution of a SOA, to use the SERVICES pattern and to wrap it with an activity in the process flow, should always be the first choice, because this solution is simple and offers loose coupling. When WRAP SERVICE AS ACTIVITIES alone is not sufficient, one has to check whether SYNTHESIZE BUSINESS OBJECT MODELS can be achieved and is of less effort than restructuring. The mapping between application-specific and synthesized business object models takes computational time and thus may imply a performance issue. Performance in this respect is often the driving factor to consider following RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL.

Flexible aspects of the INTEGRATED BUSINESS OBJECT MODEL should be described by metadata mechanisms. An abstraction from concrete structures to more abstract structures, defined by metadata, helps to manage a synthesized business object model centrally. For instance, flexible data structures within business objects can be defined via XML. What areas are subject to change is detected by an analysis of application-specific business object models and design issues detected in the business process requirements.

Figure 5 illustrates how an INTEGRATED BUSINESS OBJECT MODEL is designed. The INTEGRATED BUSINESS OBJECT MODEL integrates all involved business object models, and the business processes are defined on top this model. The integrated object model – if designed using appropriate metadata mechanisms – is open for integrating additional external business object models.

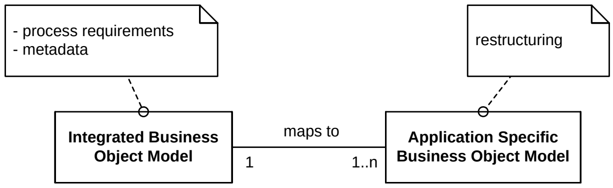

Figure 5. Integrated business object model

Unanticipated changes to the INTEGRATED BUSINESS OBJECT MODEL might occur during the evolution and lead to some restructuring. In fact, taking the right level of design abstraction with metadata that anticipates future changes and, at the same time, provides enough concrete structures is still rather an art than a science.

The DATA TRANSFORMATION FLOW pattern provides an architecture design approach for designing and implementing the necessary mapping from application-specific business object models to INTEGRATED BUSINESS OBJECT MODEL.

When the model is implemented, the actual business objects will be stored in a CENTRAL BUSINESS OBJECTPOOL.

The CANONICAL DATA MODEL [6] represents a similar approach to designing a data model that is independent from specific applications. The INTEGRATED BUSINESS OBJECT MODEL can be viewed as a specialisation of it within a process-driven SOA context. SERVICES are used to access the external system from a SOA.

7. DATA TRANSFORMATION FLOW

Systems need to be integrated via a business-process driven and service-oriented approach, and the systems have heterogeneous business object models.

◆◆◆

Consider a transformation between the business object models of two systems integrated into a SOA is needed. Major goals of a SOA are loose coupling and flexibility. These properties should not be compromised by hard-coding data integration details. In a process-oriented SOA, it is additionally necessary to map the data integration steps conceptually to the process flow to be able to easily configure data integration changes from process design tools.

In SOAs, the systems have usually been independently developed and have changed over time. As a result it is usually not trivial to depict the business objects provided as input and output parameters of one system onto the business object model used by the target system. Consequently, some kind of mapping and transformation will be necessary. The structures and the semantics of the business object models must map somehow.

In this context mapping means that business objects and the attributes of them need to be projected onto business objects and corresponding attributes of the target model. This mapping must be maintainable, and the mapping architecture must be extensible. It should be possible to react on typical change requirements, such as an increased workload, a business object model change, or that a new application needs to be integrated with minimum effort.

This means especially that no programming effort should be necessary to change (minor) details of the data integration. Somehow we need to depict and configure data integration between business object models in the process so that it is possible to use process design tools for the mapping process and for rapidly changing the mapping.

◆◆◆

Implement the data transformation as a process subflow (a microflow) that uses mapping components that are based on configurable transformation rules to project one business object model on another. Technology that supports rule-based data transformation is used to change the transformation rules at runtime. Perform the mapping steps as activities of a process subflow to make the data transformations configurable from the process design tool.

The mapping logic to project one business object model onto another is encapsulated in a component that performs the transformation. The mapping logic is implemented by configurable mapping rules associated to a component. There may be several of these components in the DATA TRANSFORMATION FLOW.

In a process-driven and service-oriented architecture, the DATA TRANSFORMATION FLOW is actually depicted by a MICROFLOW ENGINE [4], and the mapping components are represented as (reusable) process flows in the engine. The process flows perform the transformation of the business object models. The individual activities in the process flow represent transformation steps. As a result, the structural model of a DATA TRANSFORMATION FLOW can be defined as shown in Figure 6. The actual conceptual mapping is done by specialized microflows that are invoked as sub-microflows to realize the transformation.

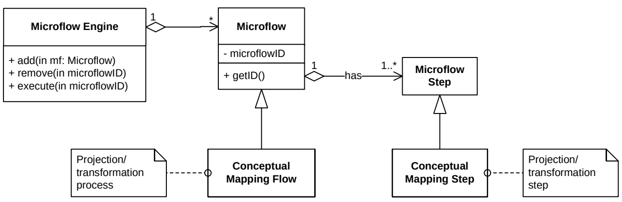

Figure 6. Conceptual mapping as special sub-microflows

Figure 7 illustrates one possible realization in a flow model: A MICROFLOW EXECUTION SERVICE [4] exposes an integration microflow as a service that can be invoked by process activities. All data transformation is done in data transformation sub-flows. The MICROFLOW EXECUTION SERVICE thus realizes the composition of the mapping functionality according to the requirements of the integration process.

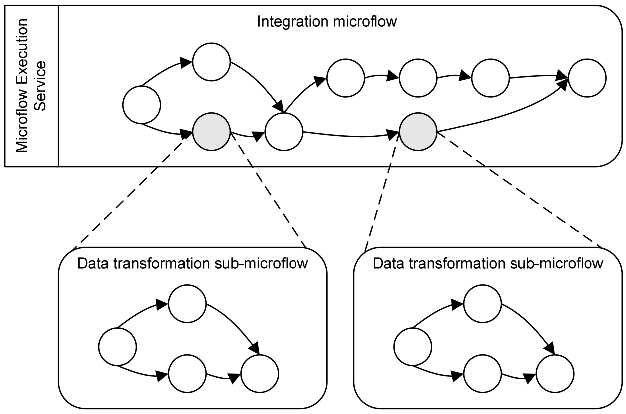

Figure 7. Conceptual mapping flows as sub-microflows.

This DATA TRANSFORMATION FLOW pattern realizes the transformations from application-specific to synthesized models, when SYNTHESIZE BUSINESS OBJECT MODELS is applied.

When realizing the transformation in a mapping flow, message transformation patterns will be applied, e.g., MESSAGE TRANSLATOR, CONTENT ENRICHER, and CANONICAL DATA MODEL [6]. A conceptual mapping microflow represents a mapping component in the spirit of MESSAGING MAPPER [6]. The DATA TRANSFORMATION FLOW pattern can be realized as part of an ENTERPRISE SERVICE BUS [8]. The MACRO-MICROFLOW pattern [4] can be used for structuring processes: In the context of this pattern the mapping flows refer to the microflow level.

The DATA TRANSFORMATION FLOW pattern leads to an architecture in which the mapping flows are encapsulated in maintainable units that can be flexibly composed.

Appropriate technology is required to implement the mapping flows. For instance a message broker with transformation functionality can be used to achieve this, or another integration middleware. The mapping may cause performance issues, if the logic gets complicated and/or storage functions are required to keep the transformed objects in databases. Thus, this pattern may only be suitable in larger SOA contexts, where this kind of flexibility is actually required.

8. BUSINESS OBJECT POOL

Business processes are executed on a process engine.

◆◆◆

Business processes are very often interdependent in their flow logic. That is, a running process may have effects on other processes being executed in parallel. Technically each process has its own data space that carries the control data for executing a business process and is thus independent of other processes. On the one hand, we need to implement the logical interdependencies between processes, but on the other hand, we need to retain the technical independence – which means interdependences should be avoided.

Business processes in execution have their own data space, i.e., the data spaces of business processes running in parallel are disjoint. Actually, this is necessary to provide a business process instance with full control over the execution of the instance –

from a technical point of view. Logically, however, business processes are interdependent. That means processes are often depending on the results of other processes – or even on events being generated by other processes. For instance, consider a business process handles an order and during this process, the customer decides to cancel the order. This is an event being generated outside the control of the actual order fulfilment process, but the order fulfilment should react accordingly to this event, i.e., by stopping the fulfilment or rolling back certain things that have already been done.

The other way round, one might consider a point in the order fulfilment process which is a point of no return. That means at some point in the fulfilment process, the order cannot be cancelled any more. Consequently, the order fulfilment process generates the respective status of the order. If the customer wants to cancel the order, the order cancellation process needs to consider this point of no return, for instance, by informing the customer that the order cannot be cancelled anymore.

It is necessary and useful that the data spaces of each process instance are disjoint – to keep the processes instances as separate and autonomous entities. But this makes it hard to depict the interdependencies of the processes. In any case the behaviour of the process must be deterministic. The process logic has to consider all possible events that may occur and depict those events by some decision logic and the corresponding paths of execution.

◆◆◆

Keep the business objects in a central pool which can be accessed in parallel by all processes of the process domain. Attribute changes to objects in the pool can then be used as triggers to corresponding behaviour in interrelated business processes. The processes can access the central pool during their execution and react on those attribute values.

Treating the business objects as central resources and allowing access to those centralized business objects enables, in principle, parallel processes to read and write the data of the business objects. One process might write certain attributes of a business object, e.g., a change in the status of the object. Another parallel process might then read the status information and react to the attribute values correspondingly. Often, the pool of business objects is realized as a central Repository [3].

Process instances can use their disjoint data spaces to store information that is only relevant for the process instance but which is of no interest for other process instances, such as data to implement the decision points in control flow logic. This data is generally of no relevance to other processes but only the instance itself. Information that has central relevance will be stored in a central business object kept in the Business Object Pool.

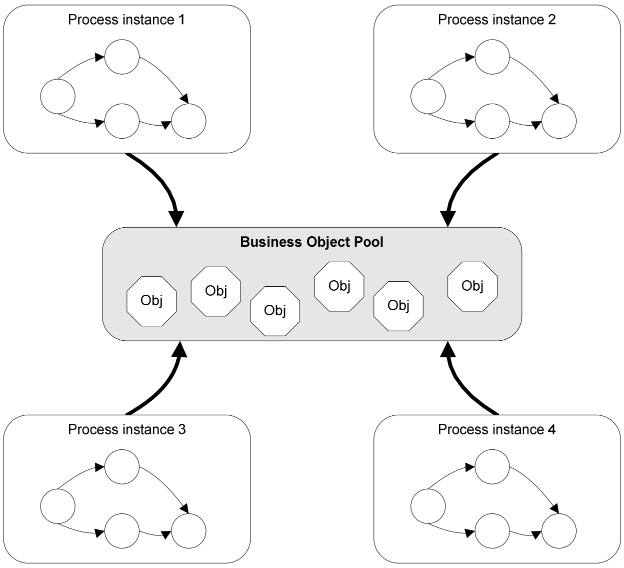

Figure 8. Central business object pool

Concurrency issues may occur in case several process instances have write access on the same business object, for instance. Traditional locking mechanisms can be used to solve some of these issues. Accessing the business objects takes some additional computational time, and, in case large amounts of data need to be read, caching mechanisms might be suitable.

The access to business objects in the Business Object Pool from the data space of a process instance can be realized via Business Object References [5] that point to objects in a central Repository [3]. The Repository is often necessary for revision and reporting purposes to store the business objects manipulated in business processes for historical reasons. To allow for controlled modifications of central business objects, the Private-Public Business Object pattern [7] can be used. This pattern offers a solution to the problem of hiding modifications to business objects as long as the process activities that manipulate the objects are not yet finished. The business object pool may be a representation of an Integrated Business Object Model.

By accessing the Business Object Pool and observing attribute values of those objects, a process instance may react in its control logic on an attribute value. The attribute value might have been set by another process running in parallel. Hence the pattern allows the process logic and its data spaces to be defined independently from other process, but still logical interdependencies can be depicted.

However, the process model must exactly define on what events it is able to react, and the business objects must be accessed via process activities. Sometimes representing process interdependencies only by using central business objects is not enough. Then usually new services or processes must be defined to realize the (more complex) interdependent behaviour.

9. EXAMPLE AND KNOWN USES

The patterns have been applied in various integration and SOA projects within the project scope of IBM. For instance, in a SOA project for a telecommunications customer in Germany, these patterns have been applied to build a larger SOA architecture based on an ENTERPRISE SERVICE BUS [8]. The architecture has been based on IBM WebSphere technology. WebSphere Business Integration Message Broker has been used as the MICROFLOW ENGINE [4] to depict the conceptual mapping flows and the service bus.

The project has focused on restructuring the business model for order management and depicting redesigned business processes on the SOA platform. We have followed the SYNTHESIZE BUSINESS OBJECT MODELS pattern to form a synthesized object model to process various types of orders. For historical reasons many different systems have been involved in the ordering and fulfilment of products, as new products have been developed over time and quick tool support has been implemented. There has been redundant data in these various systems.

An integrated and business process oriented approach needs to take the overall process perspective of ordering products and integrating the various systems involved in the business processes into account. Hence, the data models of these systems to be integrated have been mapped to business object models and a synthesized business object model for the overall business processes has been developed.

In order to achieve this, the redundancies of data in the systems have been identified by looking for the same conceptual entities in each system. For instance, the customer, or information on related contracts to the customers could be found in many of these systems. However, the data associated to these conceptual entities have not been the same in all the systems. There was some overlap, and this overlap needed to be identified to define a representation in the INTEGRATED BUSINESS OBJECT MODEL. The second step was thus to identify the overlaps and to depict the commonalities in the INTEGRATED BUSINESS OBJECT MODEL. The common representation had to be chosen in a way that allows to integrating the systems by DATA TRANSFORMATION FLOWS. Following the SYNTHESIZE BUSINESS OBJECT MODELS pattern it was thus possible to extract the redundancies and to develop a synthesized object model for the business processes systematically. The synthesized business object model thus did not contain redundant data but consolidates the views of the systems involved in the business processes. This INTEGRATED BUSINESS OBJECT MODEL has been implemented in a separate DB/2 datastore, used by the executed business process that also represented a BUSINESS OBJECT POOL. That means, the DB/2 database served as the technology for realizing the BUSINESS OBJECT POOL. The various business processes running in parallel were thus able to access the business objects concurrently, and the objects were realizing all requirements of the overall business processes.

One critical factor of flexibility regarding the object model was the products being ordered by customers. To provide reduced time to market, the processes needed to be designed in a way that products being ordered and processed are easy to change. For this reason, the notion of product has been designed in the INTEGRATED BUSINESS OBJECT MODEL via metadata description mechanisms in XML. The mandatory and optional attributes of a product could be flexibly specified using an XML-based language.

The DATA TRANSFORMATION FLOWS have been implemented using message transformation mechanisms of the WebSphere Business Integration Message Broker. This broker offers functionality for defining reusable message transformation flows that served as the DATA TRANSFORMATION FLOWS to map object models. The messages have been transported via WebSphere MQ.

The WRAP SERVICE AS ACTIVITY pattern has been applied as well. In some cases it was even possible to directly integrate the application service in the process flow, as both mapped one-on-one. One example is the integration of a legacy customer application. This application basically is a database containing a customer table and some related tables. In case of a larger business customer there is a whole hierarchy of sub-customers, for instance, representing different geographical locations. The customer table as an entity has been wrapped by services offering read/write access to the customer repository. Additionally, more simple services have been implemented, such as checking whether a customer already exists in the customer repository. This is a simple service that just returned a Boolean value. However, no persistent data needed to be stored in a business object in this case, as the process logic depicts the corresponding path of execution for the Boolean values true or false.

As WebSphere MQ Workflow and the integrated application had MQ messaging interfaces only some simple transformation was necessary in terms of DATA TRANSFORMATION FLOWS. The DATA TRANSFORMATION FLOWS basically performed the mapping of different data structures and types between the customer application and the services.

A concrete example for these data transformations can be found in the context of a service that allows retrieving customer data. The customer repository had information split across many tables, such as the basic customer data like name and address in one table, contract data of the customers in another table, and the customers account data in separate table, as a customer may have several accounts. The service represents the retrieval of all this data in a consolidated way as this was the requirement of the corresponding business process activity. For this reason, transformation flows implement the consolidation of the basic customer data, the contract data, and the account data to make them available by a single service. The consolidated data have been put in an XML message representing the output of the service.

Figure 9 provides an overview of the INTEGRATED BUSINESS OBJECT MODEL. The model represents the order domain and the product domain and the relations between products and orders. Moreover, the model shows that no specialized classes have been designed for dedicated products. The special products have been configured in XML – the example below shows the definition of the product DSL/ISDN.

The XML product definitions have been stored in terms of a product catalogue. An order only references the products by their product code, as we can see in Figure 9 – the Product class contains the product code as an attribute. The product code is basically an ID of a product to identify it in the product catalogue. The product catalogue and the products may thus be easily changed without modifying the INTEGRATED BUSINESS OBJECT MODEL where the business objects themselves have been stored in a BUSINESS OBJECT POOL represented by a DB/2 database.

```xml
<ProductType name="BundleDSLOnline" id="ProductBundleDSLOnline" sellable="true">
    <Documentation>
        <ShortDescription>This is the product bundle ISDN / DSL and Online </ShortDescription>
        <DetailedDescription>Detailed description...</DetailedDescription>
    </Documentation>
    <ProductRef name="ISDN/DSL" ref="ProductIsdnDSL" />
    <ProductRef name="Online" ref="ProductOnline" />
    <AttributeRef name="Customer class" type="CustomerClass" />
    <AttributeRef name="Installation price" type="Number" />
    <AttributeRef name="Tariff" type="Tariff" />
</ProductType>

<ProductType name="ISDN/DSL" id="ProductIsdnDSL" sellable="false" marketingName="-">
    <Documentation>
        <ShortDescription>This is the type definition of the product ISDN / DSL</ShortDescription>
        <DetailedDescription>Detailed description...</DetailedDescription>
    </Documentation>
    <AttributeRef name="Tariff" type="Tariff" />
    <AttributeRef name="Upstream bandwidth" type="Bandwidth" />
    <AttributeRef name="Downstream bandwith" type="Bandwidth" />
    <AttributeRef name="Damping" type="Damping" />
    <RuleRef name=" UpDownBandwidthConstraint " ref="UpDownBandwidthConstraint" />
</ProductType>

<ProductType name="Online" id="ProductOnline" sellable="false" marketingName="Online">
    <Documentation>
        <ShortDescription>This ist the type definition of the product Online</ShortDescription>
        <DetailedDescription>Detailed description...</DetailedDescription>
    </Documentation>
    <AttributeRef name="Tariff" type="Tariff" />
    <AttributeRef name="ImDSLBundle" type="Boolean" />
```

The corresponding user interfaces for data entry and for processing the products could thus be designed generically, as the metadata structure could be interpreted and the user interfaces were constructed generically. Implementing a new or improved product was thus basically an act of configuration. Though, some amendments and enhancements in the business processes also needed to be designed and implemented in this case. The SOA approach provided an effective means to do that. However, the effort was minimised as the design has considered the notion of product to be variable construct and changes have been limited to a minimum. The INTEGRATED BUSINESS OBJECT MODEL thus had to depict the domain of orders considering the requirements of the redesigned business processes and the integrated applications.

Furthermore, recent technologies directly support these patterns. For instance, IBM WebSphere InterChange Server and WebSphere Process Server conceptually support the concept of synthesized object models. Application specific object models addressed by application adapters can be mapped via tool support to the synthesized object model. Consequently, the patterns have shown much relevance as they are more and more supported by development tools. However, the patterns are not restricted to WebSphere technology. They are also applicable with other platforms that support process-driven and service-oriented approaches, such as Staffware. The problems addressed by the patterns actually do not depend on any particular platform.

There are other known uses of the patterns in the banking industry. In finance we usually deal with old legacy systems,

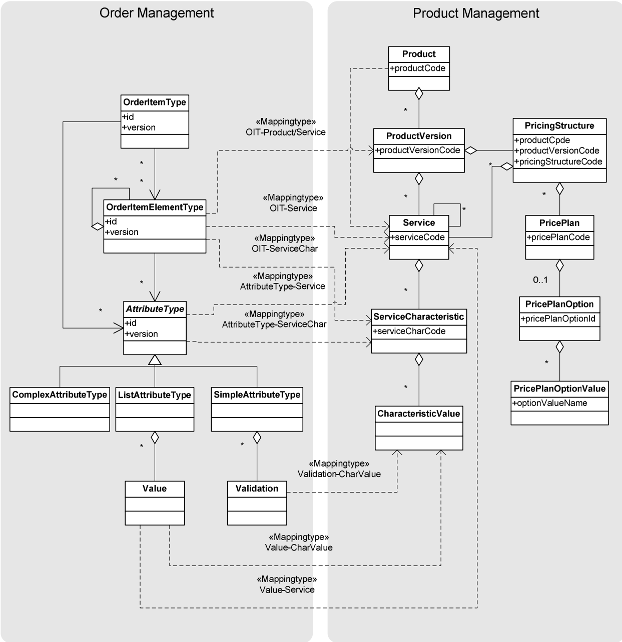

Figure 9. Example of an Integrated Business Object Model

implemented in Cobol, running on large mainframe computers. These systems represent a huge investment that needs to be protected, not at least because of their reliability and stability. The SOA approach is very interesting for the financial industry, because most of the processes are rather strongly formalised and SOA promises an approach for integration and flexibility.

Moreover, there are other known uses in the automotive industry, especially in supply chain management, where we will find the problems addressed in this paper. In supply chain management we usually deal with business processes that run across different departments, involving various stakeholders, and even across companies (suppliers). In such supply chain contexts, heterogeneity of the system landscape involved in the business processes is rather the norm than the exception.

The patterns in this paper address common problems arising in SOA projects that are built considering existing and historically grown legacy systems, or – more generally speaking – systems being developed independently. Often these legacy systems represent island solutions for requirements that needed to be implemented quickly and in an evolutionary context. The problems also occur in situations where no broader IT strategy is defined and where systems grow independently. When taking a business process driven and service-oriented perspective, some of the data integration issues, discussed in this paper, arise, such as data redundancies. This is due to the broader and integrated view taken by the SOA approach. SOA often forces developers to solve these – sometimes long known – issues in a systematic way. The problems addressed by the patterns are often inherent and most probably predictable in projects that extend system boundaries and take an enterprise-wide view.

For this reason, SOA rather offers a systematic approach for tackling data integration issues that are often very well known and existing for years. SOA, as an architectural concept, is not the solution to these well known integration problems, but it provides a means to approach them systematically and effectively. It is rather the systematic detection and the solutions aligned with business goals represented by the business process oriented approach that makes these patterns valuable.

10. CONCLUSION

In this paper, we have presented patterns in the realm of data integration in process-oriented SOAs. The first three patterns offer alternatives for single refactoring design decisions about the integration of specific business object models: WRAP SERVICE AS ACTIVITY, RESTRUCTURE SPECIFIC BUSINESS OBJECT MODEL, and SYNTHESIZE BUSINESS OBJECT MODELS. Besides the description of these patterns in the process-oriented SOA domain, this paper describes architectural patterns to use these patterns in a larger context. An architecture which supports the use of each of the refactoring patterns, when it is most appropriate, is introduced by the INTEGRATED BUSINESS OBJECT MODEL pattern. Additionally we have described a process-oriented solution for data mapping and transformation, the DATA TRANSFORMATION FLOW pattern. Finally, the BUSISNESS OBJECT POOL pattern supports the harmonization of business object models, as the pattern introduces a central pool for business objects which can be accessed in parallel by independent processes.

11. ACKNOWLEDGEMENTS

We like to thank Wolfgang Keller, our PLoP 2006 shepherd, for his very valuable comments on this paper. Further thanks go to Axel Emmer for helping with the revision of the paper for ACM DL.

12. REFERENCES

[1] D. K. Barry. Web Services and Service-oriented Architectures, Morgan Kaufmann Publishers, 2003

[2] K. Channabasavaiah, K. Holley, and E.M. Tuggle. Migrating to Service-oriented architecture – part 1, http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa/, IBM developerWorks, 2003

[3] E. Evans. Domain-Driven Design – Tackling Complexity in the Heart of Software", Addison-Wesley, 2004.

[4] C. Hentrich, U. Zdun. Patterns for Process-Oriented Integration in Service-Oriented Architectures, Proceedings of EuroPLoP 2006, Universitätsverlag Konstanz, 2006.

[5] C. Hentrich. Six Patterns for Process-Driven Architectures, Proceedings of EuroPLoP 2004, Universitätsverlag Konstanz, 2004

[6] G. Hohpe and B. Woolf. Enterprise Integration Patterns. Addison-Wesley, 2003.

[7] T. Köllmann, C. Hentrich. Synchronization Patterns for Process-Driven and Service-Oriented Architectures. Proceedings of EuroPLoP 2006, Universitätsverlag Konstanz, 2006.

[8] U. Zdun, C. Hentrich, and W. van der Aalst. A survey of patterns for service-oriented architectures. International Journal of Internet Protocol Technology, 1(3):132-143, 2006.

13. APPENDIX: OVERVIEW OF REFERENCED RELATED PATTERNS

There are several important related patterns referenced in this paper, which are described in other papers, as indicated by the corresponding references in the text. Table 1 gives an overview of thumbnails of these patterns in order to provide a brief introduction to them for the reader. For detailed descriptions of these patterns please refer to the referenced articles.

Table 1. Thumbnails of referenced patterns.

Pattern	*Problem*	*Solution*
BUSINESS OBJECT REFERENCE [Hentrich 2004]	How can management of business objects be achieved in a business process, as far as concurrent access and changes to these business objects is concerned?	Only store references to business objects in the process control data structure and keep the actual business objects in an external container.
CANONICAL DATA MODEL [Hohpe et al. 2003]	How to minimize dependencies when integrating applications that use different data formats?	Design a CANONICAL DATA MODEL that is independent from any specific application. Require each application to produce and consume messages in this common format.
CONTENT ENRICHER [Hohpe et al. 2003]	How do we communicate with another system if the message originator does not have all the required data items available?	Use a specialised transformer, a CONTENT ENRICHER, to access an external data source in order to augment a message with missing information.
ENTERPRISE SERVICE BUS [Zdun et al. 2006]	How is it possible in a large business architecture to integrate various applications and backends in a comprehensive, flexible and consistent way?	Unify the access to applications and backends using services and service adapters, and use message-oriented, event-driven communication between these services to enable flexible integration.
ENVELOPE WRAPPER [Hohpe et al. 2003]	How can existing systems participate in a messaging exchange that places specific requirements, such as message header fields or encryption, on that message format?	Use an Envelope Wrapper to wrap application data inside an envelope that is compliant with the messaging infrastructure. Unwrap the message when it arrives t the
MACROFLOW INTEGRATION SERVICE [Hentrich et al. 2006]	How can the functionality and implementation of process activities at the macroflow level be decoupled from the process logic that orchestrates them, in order to achieve flexibility, as far as the design and implementation of these automatic functions are concerned?	The automatic functions required by macroflow activities from external systems are designed and exposed as dedicated MACROFLOW INTEGRATION SERVICE with well-defined service interfaces.
MACRO-MICROFLOW [Hentrich et al. 2006]	How is it possible to conceptually structure process models in a way that makes clear which parts will be depicted on a process engine as long running business process flows and which parts of the process will be depicted inside of higher-level business activities as rather short running technical flows?	Structure a process model into macroflow and microflow.
MESSAGE TRANSLATOR [Hohpe et al. 2003]	How can systems using different data formats communicate with each other using messaging?	Use a special filter, a MESSAGE TRANSLATOR, between other filter or applications to translate one data format into another.
MESSAGING MAPPER [Hohpe et al. 2003]	How do you move data between domain objects and the messaging infrastructure while keeping the two independent of each other?	Create a separate MESSAGING MAPPER that contains the mapping logic between the infrastructure and the domain objects.
MICROFLOW ENGINE [Hentrich et al. 2006]	How is it possible to flexibly configure IT systems integration processes in a dynamic environment, where IT process changes are regular practice, in order to reduce implementation time and effort?	Delegate the microflow aspects of the business process definition and execution to a dedicated MICROFLOW ENGINE that allows to configuring microflows by flexibly orchestrating execution of microflow activities.
MICROFLOW EXECUTION	How to expose a microflow as a coherent	Expose a microflow as a MICROFLOW EXECUTION SERVICE

Pattern	*Problem*	*Solution*
SERVICE [Hentrich et al. 2006]	function with defined in- and output parameters without having to consider the technology specifics of the MICROFLOW ENGINE being used, in order to decouple the engine's technology specifics from the actual functionality that is has to offer to execute concrete microflows?	that abstracts the technology specific API of the MICROFLOW ENGINE to a standardised well-defined service interface and encapsulates the functionality of the microflow.
PRIVATE-PUBLIC BUSINESS OBJECT [Köllmann et al. 2006]	How can business object modifications be hidden from other users as long as the process activity during which the changes are made is not finished?	Introduce private-public business objects, which expose two separate images, a private and a public image of the contained data.
PROCESS-BASED INTEGRATION ARCHITECTURE [Hentrich et al. 2006]	What architecture design concepts for process-driven backend systems integration are necessary, in order for the architecture to be scalable, flexible, and maintainable?	Provide a multi-layered PROCESS-BASED INTEGRATION ARCHITECTURE to connect macroflow business processes and the backend systems that need to be used in those macroflows.
REPOSITORY [Evans 2004]	Exposure of technical infrastructure and database access mechanisms complicates the client.	Delegate all object storage and access to a REPOSITORY.
SERVICE [Evans 2004]	Some domain concepts are hard to model as objects because they have no state.	Define one or more related operations as a standalone interface declared as a SERVICE and make the SERVICE stateless.