University of Sofia "St Kliment Ohridski"
Faculty of Mathematics and Informatics
Department: Information technologies

# Master thesis

## "Process and realization of SOA centralized system"

**Student**: Velichko Ginev Sarev
**MSc program:** Computer science – Software Engineering
**FN:** M21608

**Scientific supervisor:** Silvia Ilieva, Assoc. Prof., PhD

Sofia, 2007

CONTENTS

# 1 Introduction

In the last twenty years the software industry has grown tremendously. It has conquered and occupied every part of human lives – business, economics, communications, health, science and entertainment.
The software products become more and more complex, as well as more and more critical to everyday life. The public needs for quality and flexible software products of all kind of areas are constantly increasing.
The need to respond to changing business demands with quality and flexible IT solutions has led many businesses to Service-Oriented Architectures (SOAs). SOA is an IT framework that combines individual business functions and processes, called services, to implement sophisticated business applications and processes. SOA is an approach to IT that considers business processes as reusable components or services that are independent of applications and the computing platforms on which they run. It allows you to design solutions as assemblies of services in which the assembly description is a managed, first-class aspect of the solution, and hence, amenable to analysis, change, and evolution. You can then view a solution as a choreographed set of service interactions. The idea of viewing enterprise solutions as federations of services connected via well-specified contracts that define service interfaces is gaining widespread support. The ultimate goal of adopting an SOA is to achieve flexibility for the business and within IT.

## 1.1 Goals

The goal of this master thesis is to summarize the process and to demonstrate the realization of SOA centralized system and its methodology. It will start with concept of SOA system, after that the thesis continues with primary three steps that covers every SOA project:

- CBM – Component Business Model
- SOMA – Service Oriented Modeling Architecture
- SOA – Service Oriented Architecture

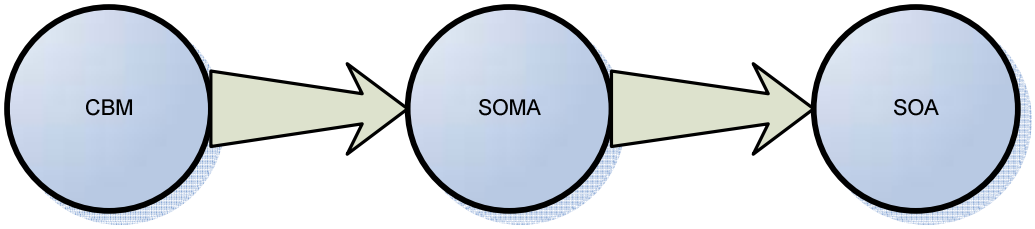


*Figure 1 SOA approach*

The scope of this master thesis will covers also prove of concept project which will presents practical and realization of SOA centralized systems.
Tasks that should be performed to reach the goal:

- To introduce the emerging Service Oriented Architecture (SOA) concepts
- To summarize the Methodology of SOA
- To describe the Business Model and Architecture of SOA systems
- To explain of the SOA tools in term of SOA solution stack
- To create the SOA project in order to apply the described theory in practice

## 1.2 Benefit of the master thesis

During the evolution of IT technologies the world go through many technologies and best practices such us procedure programming, object oriented programming and etc. Now the most popular programming in the world is SOA. It provides the ability to develop the software and think in terms of services. The base things in SOA programming are services. The services give us business and technical benefits such us reusability for business and technical services, well known protocols to access these services. The most popular services in SOA are web services which can be consumed trough soap/http protocol. However when we talk for services into SOA it doesn't mean that all the services are web services. There are several types of well know bindings in order to consume SOA services. These bindings are:

- Soap/http binding
- JMS binding
- SCA binding
- Soap/jms binding

These services are basis for SOA programming. They are orchestrated by technical business processes which fallow real business processes that are used from the customers. When we deliver such kind of IT infrastructure and software to our clients they will be flexible enough to build their own business processes, rules, entities and etc.
This master thesis illustrates the IBM methodology and could be used as a guideline that needs to be followed during each SOA project.

## 1.3 Structure of the master thesis

The master thesis consists of six mainly chapters – Introduction, SOA Concept, SOA methodology and architecture, IBM tools for SOA development, Prove of concept and Conclusion:

- **Chapter 1 "Introduction"** – gives short description of SOA
- **Chapter 2 "Concept of SOA system"** – describes why SOA is important and what the benefit of its usage is.

- **Chapter 3 "Methodology and architecture of SOA system"** – describes Rational Unified Process (RUP) for SOMA and Model-Driven development of SOA systems
- **Chapter 4 "Component Business Model" -** describes methodologies and methods which can be successfully applied to each SOA project.
- **Chapter 5 "Business processes and modeling" –** describes business architecture and modeling.
- **Chapter 6 "Service oriented modeling and architecture" –** describes SOMA method.
- **Chapter 7 "SOA solution stack" –** describes different layer of SOA centralized system.
- 
- **Chapter 8 "IBM based tools for development of SOA projects"** – this section covers the most popular IBM tools used for SOA projects.
- **Chapter 9 "Demonstration through practical solution"** – developing SOA solution based on methodology and tools that are mentioned above.
- **Chapter 10 "Conclusion"** – final conclusion and descriptions of possibilities for future development

# 2 Concept of SOA systems

Architecture is not tied to a specific technology. It may be implemented using a wide range of technologies, including SOAP, RPC, DCOM, CORBA and Web Services. SOA can be implemented using one or more of these protocols and, for example, might use a file system mechanism to communicate data conforming to a defined interface specification between processes conforming to the SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without the service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks.

SOA can also be considered as a style of information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service. SOA-based systems can therefore be independent of development technologies and platforms (such as Java, .NET etc). Services written in C# running on .NET platforms and services written in Java running on Java EE platforms, for example, can both be consumed by a common composite application (or client). Applications running on either platform can also consume services running on the other as Web services, which facilitates reuse. Many COBOL legacy systems can also be wrapped by a managed environment and presented as a software service. This has allowed the useful life of many core legacy systems to be extended indefinitely no matter what language they were originally written in. SOA can support integration and consolidation activities within complex enterprise systems, but SOA does not specify or provide a methodology or framework for documenting capabilities or services.

High-level languages such as BPEL and specifications such as WS-CDL and WS-Coordination extend the service concept by providing a method of defining and supporting orchestration of fine grained services into more coarse-grained business services, which in turn can be incorporated into workflows and business processes implemented in composite applications or portals.

## 2.1 Introduction to SOA

Service Oriented Architecture (SOA) is an evolution of distributed computing and modular programming. SOAs build applications out of software services. Services are relatively large, intrinsically unassociated units of functionality, which have no calls to each other embedded in them. They typically implement functionalities most humans would recognize as a service, such as filling out an online application for an account, viewing an online bank statement, or placing an online book or airline ticket order. Instead of services embedding calls to each other in their source code, protocols are defined which describe how one or more services can talk to each other. This architecture then relies on a business

process expert to link and sequence services, in a process known as orchestration, to meet a new or existing business system requirement.

Relative to earlier attempts to promote software reuse via modularity of functions, or by use of predefined groups of functions known as classes, SOA's atomic level objects are 100 to 1,000 times larger, and are associated by an application designer or engineer using orchestration. In the process of orchestration, relatively large chunks of software functionality (services) are associated in a non-hierarchical arrangement (in contrast to a class's hierarchies) by a software engineer, or process engineer, using a special software tool which contains an exhaustive list of all of the services, their characteristics, and a means to record the designer's choices which the designer can manage and the software system can consume and use at run-time.

Underlying and enabling all of this is metadata which is sufficient to describe not only the characteristics of these services, but also the data that drives them. XML has been used extensively in SOA to create data which is wrapped in a nearly exhaustive description container. Analogously, the services themselves are typically described by WSDL, and communications protocols by SOAP. Whether these description languages are the best possible for the job, and whether they will remain the favorites going forward, is at present an open question. What is certain is that SOA is utterly dependent on data and services that are described using some implementation of metadata which meets two criteria. The metadata must be in a form which software systems can consume to dynamically configure to maintain coherence and integrity, and in a form which system designers can understand and use to manage that metadata.

The goal of SOA then is to allow fairly large chunks of functionality to be strung together to form ad-hoc applications which are built almost entirely from existing software services. The larger the chunks the fewer the interface points required to implement any given set of functionality. This is at odds with very large chunks of functionality which are not granular enough to be easily reused. Since each interface brings with it some amount of processing overhead, there is a performance consideration in choosing the granularity of services. The great promise of SOA though, is that in this world, the marginal cost of creating the nth application is zero, as all of the software required already exists to satisfy the requirements of other applications. Only orchestration is required to produce a new application.

The key here is there are no interactions between the chunks which are specified within the chunks themselves (where functions specify their calls to other functions or classes specify ownership of their member functions as well as calls as part of the source code). By contrast, the interaction of services, all of whom are unassociated peers, is specified by humans in a relatively ad-hoc way with the intent du jour driven by newly emergent business requirements. Again we see the need for services to be much larger units of functionality than traditional

functions or classes, lest the sheer complexity of thousands of such granular objects overwhelm the application designer. The services themselves continue to be made using classical languages like Java, C#, C++, C or COBOL.

SOA services are therefore loosely coupled, in contrast, for example, to the functions a linker binds together to form an executable, a DLL, or an assembly. SOA services also run in "safe" wrappers such as the .NET environment, which manages memory allocation and reclamation, allows ad-hoc and late binding, and some degree of indeterminate data typing.

It is important to note that increasingly there are third party software companies which offer software services for a fee. Going forward, many SOA systems may be composed of services, only some of which were created in-house. This has the potential to spread costs over many customers, and customer uses, and promotes standardization both in and across industries. The travel industry in particular now has a well-defined and documented set of services, and the data they consume, sufficient to allow any reasonably competent software engineer to create travel agency software using entirely off the shelf software services. Other industries, such as the finance industry, are also making significant progress in this direction.

There is no widely agreed upon definition of SOA other than its literal translation. It is an architecture that relies on service-orientation as its fundamental design principle. In an SOA environment independent services can be accessed without knowledge of their underlying platform implementation.[2] These concepts can be applied to business, software and other types of producer/consumer systems.

## 2.2 Why SOA?

Enterprise architects believe that SOA can help businesses respond more quickly and cost-effectively to changing market conditions. This style of architecture promotes reuse at the macro (service) level rather than micro (classes) level. It can also simplify interconnection to - and usage of - existing IT (legacy) assets.
In some respects, SOA can be considered an architectural evolution rather than a revolution and captures many of the best practices of previous software architectures. In communications systems, for example, there has been little development of solutions that use truly static bindings to talk to other equipment in the network. By formally embracing a SOA approach, such systems are better positioned to stress the importance of well-defined, highly inter-operable interfaces.
Some have questioned whether SOA is just a revival of modular programming (1970s), event-oriented design (1980s) or interface/component-based design (1990s)[citation needed]. SOA promotes the goal of separating users (consumers) from the service implementations. Services can therefore be run on various distributed platforms and be accessed across networks. This can also maximize reuse of services

## 2.3 Benefit of usage

The following guiding principles define the ground rules for development, maintenance, and usage of the SOA:

- Reuse, granularity, modularity, compos ability, componentization, and interoperability
- Compliance to standards (both common and industry-specific)
- Services identification and categorization, provisioning and delivery, and monitoring and tracking

The following specific architectural principles for design and service definition focus on specific themes that influence the intrinsic behavior of a system and the style of its design:

- Service Encapsulation - A lot of existing web-services are consolidated to be used under the SOA Architecture. Many a times, such services have not been planned to be under SOA.
- Service loose coupling - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other
- Service contract - Services adhere to a communications agreement, as defined collectively by one or more service description documents
- Service abstraction - Beyond what is described in the service contract, services hide logic from the outside world
- Service reusability - Logic is divided into services with the intention of promoting reuse
- Service compos ability - Collections of services can be coordinated and assembled to form composite services
- Service autonomy – Services have control over the logic they encapsulate
- Service optimization – All else equal, high-quality services are generally considered preferable to low-quality ones
- Service discoverability – Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms

In addition, the following factors should also be taken into account when defining a SOA implementation:

- SOA Reference Architecture covers the SOA Reference Architecture, which provides a worked design of an enterprise-wide SOA implementation with detailed architecture diagrams, component descriptions, detailed requirements, design patterns, opinions about standards, patterns on regulation compliance, standards templates etc.
- Life cycle management SOA - Introduction to Services Lifecycle introduces the Services Lifecycle and provides a detailed process for

services management though the service lifecycle, from inception through to retirement or repurposing of the services. It also contains an appendix that includes organization and governance best practices, templates, comments on key SOA standards, and recommended links for more information.

- Efficient use of system resources
- Service maturity and performance

## 2.4 Requirements for a SOA

In order to efficiently use a SOA, one must meet the following requirements:

- Interoperability between different systems and programming languages - The most important basis for a simple integration between applications on different platforms is a communication protocol, which is available for most systems and programming languages.
- Clear and unambiguous description language - To use a service offered by a provider, it is not only necessary to be able to access the provider system, but the syntax of the service interface must also be clearly defined in a platform-independent fashion.
- Retrieval of the service - To allow a convenient integration at design time or even system run time, a search mechanism is required to retrieve suitable services. The services should be classified as computer-accessible, hierarchical or taxonomies based on what the services in each category do and how they can be invoked.

# 3 Methodology and Architecture of SOA systems

In this chapter we will concentrate mainly on two different methods for development of SOA solutions:

- RUP for SOMA
- Model-Driven Development

## 3.1 RUP for SOMA

The roots of Rational Process go back to the original spiral model of Barry Boehm. The Rational Approach was developed at Rational Software in the 1980s and 1990s.
In 1995 Rational Software acquired the Swedish Company Objectory AB. The Rational Unified Process was the result of the merger of the Rational Approach and the Objectory process developed by Objectory founder Ivar Jacobson. The first results of that merger was the Rational Objectory Process, designed to an Objectory-like process, but suitable to wean Objectory users to the Rational Rose tool. When that goal was accomplished, the name was changed. The first version of the Rational Unified Process, version 5.0, was released in 1998. The chief architect was Philippe Kruchten.
RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs.
The latest version of RUP 7.1 that incorporates additional information on building SOA-based solutions was released with the announcement of the IBM Rational Method Composer. This information is a combination of SOA material that was part of earlier versions of RUP, and a lot of content from IBM's service-oriented modeling and architecture technique is used by IBM consultants. Actually there were two updates on RUP before. The current update of RUP 7.1 is focused upon Service-Oriented Architecture (SOA). This update represents a major milestone in the RUP guidance around SOA as it provides a unified method combining previous RUP for SOA content with content from the IBM Global Business Services (GBS) Service-Oriented Modeling and Architecture (SOMA) method. The SOMA method has been successfully used by IBM in a number of client engagements, and while it was initially developed leveraging the existing IBM Global Services Method (GS Method) it was felt that in the area of SOA both IBM and our customers would benefit more from a unified method approach than having two separate methods. When looking at the two methods it is clear that the authors had very similar aims in mind and structured the methods in similar ways - in fact the two teams did meet in 2004 and made some changes to both methods to align terminology. While this alignment is not necessarily surprising as both methods are focused on the pragmatic activities of developing a service-oriented solution it was noted that a generic framework could be extracted that would be able to describe both methods at a high level.

### 3.1.1 Method Integration Overview

The following diagram (fig. 2) illustrates the framework mentioned above. It represents a method neutral set of activities required of any process for the development of service-oriented solutions. Now, this diagram is significantly simplified from the content of either method but clearly represents the key activities of both methods -- Service Identification, Service Specification and Service Realization. In the area of work products, it was clear that there was a lot of conceptual alignment. Similar work products were required with similar roles and stakeholders, but in some cases were realized differently; for example, as either a UML model or a Word document.

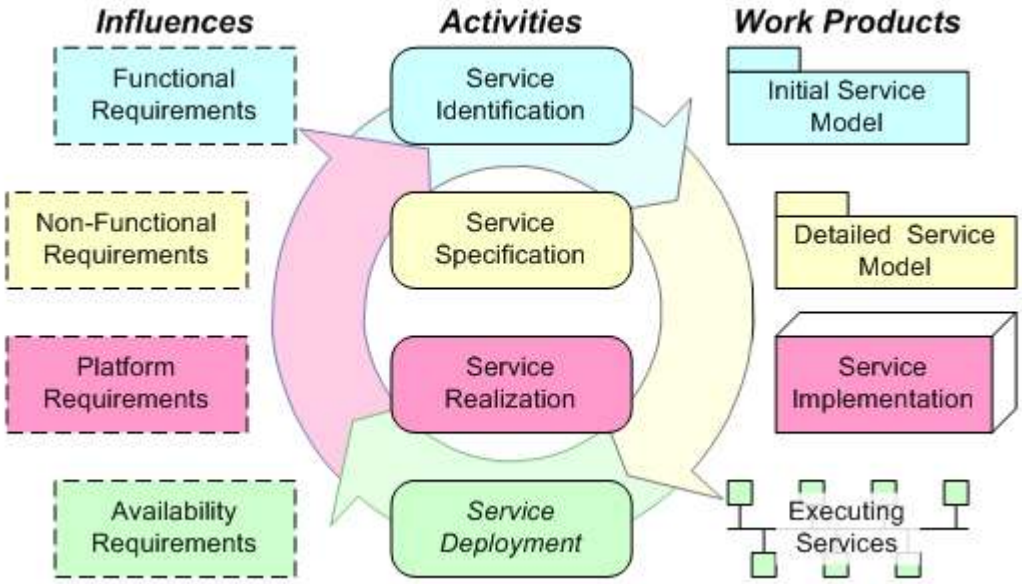


*Figure 2 SOMA approach*

Rational Unified Process 7.1 has the scope of introducing guidance for the Software Architect and Designer in developing a Service Model, a model representing a portfolio of services that can be used as the basis for implementation tasks already in the RUP. It is also our intent to describe the connection between business modeling and the services model. Many Service-Oriented Architecture (SOA) projects use business-process models in understanding their business, functional requirements, and the services required to support a process.

The scope of this update was addressed briefly in the introduction, but here are the set of requirements and scoping statements used to guide the project.

- Leverage the existing RUP; in this case it means that where possible we should describe new tasks and work products in relation to existing ones in the RUP and not unnecessarily add new concepts. Also, new elements should be added such that they fit into the overall flow of the RUP.
- Look forward to future tool capabilities; although the RUP is not tool dependent, it should be understood that there is no point in developing content in areas where no tooling will ever exist and then, there is no need to not write a topic because the tool is not in the market but we can expect it to be soon.
- Integration of other IBM experience in SOA; it is clear that other groups within IBM have experience that can be leveraged; harvested; and added to the concepts, guidelines, and practice we are introducing.
- Scope changes to Analysis & Design; we have looked at the literature that describes the application of SOA to business design and the implication of SOA on business models, operational organization, and business integration. We have also looked at the differences SOA tends to make on implementation, deployment, and operational management. This is too broad a scope for the first iteration so we have only focused on architecture and design issues.
- Deliver a foundation; this is the first iteration. We expect that additional guidance can be added to the framework presented here and the connection made between this content and the rest of the RUP in subsequent iterations.
- Look to changes that need to be made in the base, but defer to future release; we know that some concepts we introduce would fit better with terminology or other minor changes to the base RUP. While it would be desirable to change the RUP, that would have wider implications and would take far longer.

### 3.1.2 Phases

The RUP lifecycle is an implementation of the spiral model. It has been created by assembling the content elements into semi-ordered sequences. Consequently the RUP lifecycle is available as a work breakdown structure, which could be customized to address the specific needs of a project. The RUP lifecycle organizes the tasks into phases and iterations.
A project has four phases usually, which are described below:

- **Inception phase** - The overriding goal of the inception phase is to achieve concurrence among all stakeholders on the lifecycle objectives for the project. The inception phase is of significance primarily for new development efforts, in which there are significant business and requirements risks which must be addressed before the project can proceed. For projects focused on enhancements to an existing system,

the inception phase is more brief, but is still focused on ensuring that the project is both worth doing and possible to do.

- **Elaboration phase** - The goal of the elaboration phase is to baseline the architecture of the system to provide a stable basis for the bulk of the design and implementation effort in the construction phase. The architecture evolves out of a consideration of the most significant requirements (those that have a great impact on the architecture of the system) and an assessment of risk. The stability of the architecture is evaluated through one or more architectural prototypes.
- **Construction phase** – The goal of the construction phase is clarifying the remaining requirements and completing the development of the system based upon the baseline architecture. The construction phase is in some sense a manufacturing process, where emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality. In this sense the management mindset undergoes a transition from the development of intellectual property during inception and elaboration, to the development of deployable products during construction and transition.
- **Transition phase** - The focus of the Transition Phase is to ensure that software is available for its end users. The Transition Phase can span several iterations, and includes testing the product in preparation for release, and making minor adjustments based on user feedback. At this point in the lifecycle, user feedback should focus mainly on fine tuning the product, configuring, installing and usability issues, all the major structural issues should have been worked out much earlier in the project lifecycle.

Each of these four phases can be spitted into sub-phases, but this decision depends of project milestones and development.

### 3.1.3 RUP Workflows

There are nine core process workflows (disciplines) in the Rational Unified Process, which represent a partitioning of all workers and activities into logical groupings (fig. 3).

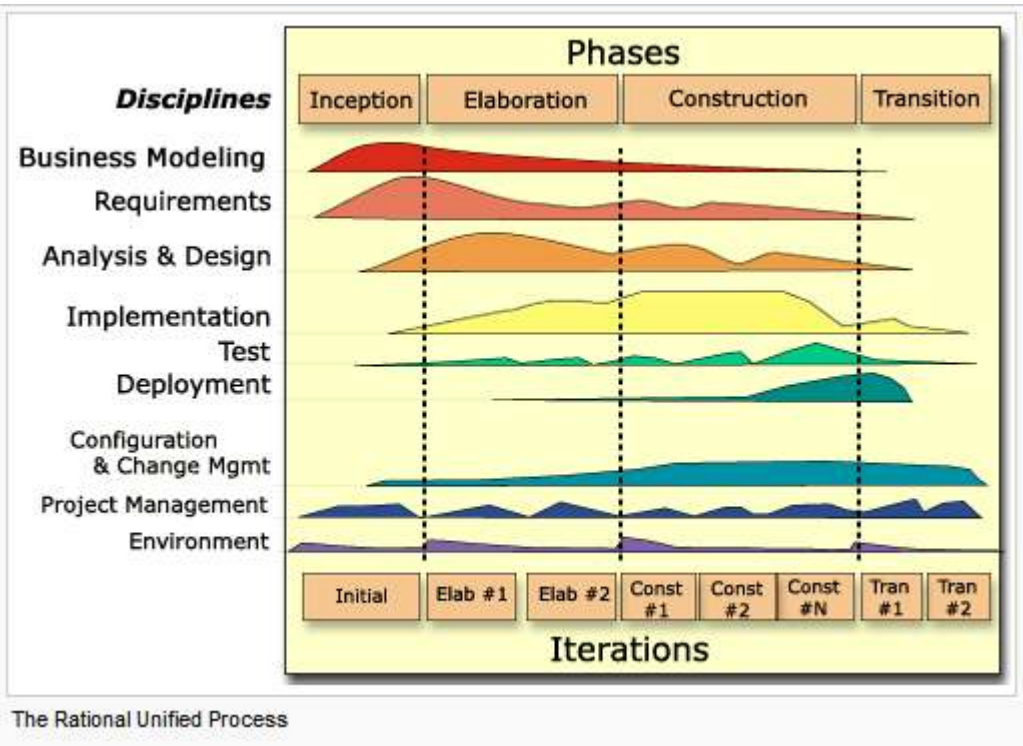


*Figure 3 RUP phases*

The nine core process workflows will be shortly presented now.

Workflows are revisited again and again throughout the lifecycle. The actual complete workflow of a project interleaves these nine core workflows, and repeats them with various emphasis and intensity at each iteration.

**Business Modeling**
One of the major problems with most business engineering efforts is that the software engineering and the business engineering community do not communicate properly with each other. This leads to that the output from business engineering is not used properly as input to the software development effort, and vice versa. The Rational Unified Process addresses this by providing a common language and process for both communities, as well as showing how to create and maintain direct traceability between business and software models. In Business Modeling we document business processes using so called business use cases. This assures a common understanding among all stakeholders of what business process needs to be supported in the organization. The business use cases are analyzed to understand how the business should support the business processes. This is documented in a business object-model. Many projects may choose not to do business modeling.

**Requirements**
The goal of the Requirements workflow is to describe what the system should do and allows the developers and the customer to agree on that description. To achieve this, we elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions. A Vision document is created, and stakeholder needs are elicited. Actors are identified, representing the users, and any other system that may interact with the system being developed. Use cases are identified, representing the behavior of the system. Because use cases are developed according to the actor's needs, the system is more likely to be relevant to the users.

**Analysis & Design**
The goal of the Analysis & Design workflow is to show how the system will be realized in the implementation phase. Analysis & Design results in a design model and optionally an analysis model. The design model serves as an abstraction of the source code; that is, the design model acts as a 'blueprint' of how the source code is structured and written. The design model consists of design classes structured into design packages and design subsystems with well-defined interfaces, representing what will become components in the implementation. It also contains descriptions of how objects of these design classes collaborate to perform use cases. The design activities are centered around the notion of architecture. The production and validation of this architecture is the main focus of early design iterations. Architecture is represented by a number of architectural views. These views capture the major structural design decisions. In essence, architectural views are abstractions or simplifications of the entire design, in which important characteristics are made more visible by leaving details aside. The architecture is an important vehicle not only for developing a good design model, but also for increasing the quality of any model built during system development.

**Implementation**
The purposes of implementation are:

- To define the organization of the code, in terms of implementation subsystems organized in layers.
- To implement classes and objects in terms of components (source files, binaries, executables, and others).
- To test the developed components as units.
- To integrate the results produced by individual implementers (or teams), into an executable system.

The system is realized through implementation of components. The Rational Unified Process describes how you reuse existing components, or implement new components with well defined responsibility, making the system easier to maintain, and increasing the possibilities to reuse.

**Test**
The purposes of testing are:

- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure defects are addressed prior to the deployment of the software.

The Rational Unified Process proposes an iterative approach, which means that you test throughout the project. This allows you to find defects as early as possible, which radically reduces the cost of fixing the defect. Test are carried out along three quality dimensions reliability, functionality, application performance and system performance. For each of these quality dimensions, the process describes how you go through the test lifecycle of planning, design, implementation, execution and evaluation.
Strategies for when and how to automate test are described. Test automation is especially important using an iterative approach, to allow regression testing at then end of each iteration, as well as for each new version of the product.

**Deployment**
The purpose of the deployment workflow is to successfully produce product releases, and deliver the software to its end users. It covers a wide range of activities including:

- Producing external releases of the software.
- Packaging the software.
- Distributing the software.
- Installing the software.
- Providing help and assistance to users.

In many cases, this also includes activities such as:

- Planning and conduct of beta tests.
- Migration of existing software or data.
- Formal acceptance.

Although deployment activities are mostly centered around the transition phase, many of the activities need to be included in earlier phases to prepare for deployment at the end of the construction phase.

**Project Management**
Software Project Management is the art of balancing competing objectives, managing risk, and overcoming constraints to deliver, successfully, a product which meets the needs of both customers (the payers of bills) and the users. The fact that so few projects are unarguably successful is comment enough on the difficulty of the task.
This workflow focuses mainly on the specific aspect of an iterative development process. Our goal with this section is to make the task easier by providing:

- A framework for managing software-intensive projects.
- Practical guidelines for planning, staffing, executing, and monitoring projects.
- A framework for managing risk.

It is not a recipe for success, but it presents an approach to managing the project that will markedly improve the odds of delivering successful software.

**Configuration & Change Management**
In this workflow we describe how to control the numerous artifacts produced by the many people who work on a common project. This workflow provides guidelines for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site-specific development policies.
We describe how you can manage parallel development, development done at multiple sites, and how to automate the build process. This is especially important in an iterative process where you may want to be able to do builds as often as daily, something that would become impossible without powerful automation. We also describe how you can keep an audit trail on why, when and by whom any artifact was changed. This workflow also covers change request management, i.e. how to report defects, manage them through
their lifecycle, and how to use defect data to track progress and trends.

**Environment**
The purpose of the environment workflow is to provide the software development organization with the software development environment—both processes and tools—that are needed to support the development team. This workflow focuses on the activities to configure the process in the context of a project. It also focus on activities to develop the guidelines needed to support a project. A step-by-step procedure is provided
Describing how you implement a process in an organization.
The environment workflow also contains a Development Kit providing you with the guidelines, templates and tools necessary to customize the process.

### 3.1.4 The Best Practices of RUP

The Rational Unified Process describes how to effectively deploy commercially proven approaches to software development for software development teams. These are called “best practices” not so much because you can precisely quantify their value, but rather, because they are observed to be commonly used in industry by successful organizations. The Rational Unified Process provides each team member with the guidelines, templates and tool mentors necessary for the entire team to take full advantage of among others the following best practices:

- **Develop Software Iteratively** - Given today’s sophisticated software systems, it is not possible to sequentially first define the entire problem, design the entire solution, build the software and then test the product at the end. An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations. The

Rational Unified Process supports an iterative approach to development that addresses the highest risk items at every stage in the lifecycle, significantly reducing a project's risk profile. This iterative approach helps you attack risk through demonstrable progress - frequent, executable releases that enable continuous end user involvement and feedback. Because each iteration ends with an executable release, the development team stays focused on producing results, and frequent status checks help ensure that the project stays on schedule. An iterative approach also makes it easier to accommodate tactical changes in requirements, features or schedule.

- **Manage Requirements** -The Rational Unified Process describes how to elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions; and easily capture and communicate business requirements. The notions of use case and scenarios proscribed in the process has proven to be an excellent way to capture functional requirements and to ensure that these drive the design, implementation and testing of software, making it more likely that the final system fulfills the end user needs. They provide coherent and traceable threads through both the development and the delivered system.

- **Use Component-based Architectures** - The process focuses on early development and baselining of a robust executable architecture, prior to committing resources for full-scale development. It describes how to design a resilient architecture that is flexible, accommodates change, is intuitively understandable, and promotes more effective software reuse. The Rational Unified Process supports component-based software development. Components are non-trivial modules, subsystems that fulfill a clear function. The Rational Unified Process provides a systematic approach to defining an architecture using new and existing components.

- **Visually Model Software** - The process shows you how to visually model software to capture the structure and behavior of architectures and components. This allows you to hide the details and write code using "graphical building blocks." Visual abstractions help you communicate different aspects of your software; see how the elements of the system fit together; make sure that the building blocks are consistent with your code; maintain consistency between a design and its implementation; and promote unambiguous communication.

- **Verify Software Quality** - Poor application performance and poor reliability are common factors which dramatically inhibit the acceptability of today's software applications. Hence, quality should be reviewed with respect to the requirements based on reliability, functionality, application performance and system performance. The Rational Unified Process assists you in the planning, design, implementation, execution, and

evaluation of these test types. Quality assessment is built into the process, in all activities, involving all participants, using objective measurements and criteria, and not treated as an afterthought or a separate activity performed by a separate group.

- **Control Changes to Software** - The ability to manage change-making certain that each change is acceptable, and being able to track changes-is essential in an environment in which change is inevitable. The process describes how to control, track and monitor changes to enable successful iterative development. It also guides you in how to establish secure workspaces for each developer by providing isolation from changes made in other workspaces and by controlling changes of all software artifacts (e.g., models, code, documents, etc.). And it brings a team together to work as a single unit by describing how to automate integration and build management.

## 3.2 Model-Driven Development of SOA

Service-oriented architectures (SOA) will form the basis of future information systems. Basic web services are being assembled to composite web services in order to directly support business processes. As some basic web services can be used in several composite web services, different business processes are influenced if for example a web service is unavailable or if its signature changes. Yet the range of such a change is often ambiguous due to a missing overall SOA service model pointing out the influence of services on business processes. Here we will present a SOA service model defined as a UML-based metamodel and its integration into a model-driven service development approach.

With the evolution of service-oriented architecture (SOA) the focus in software development changes from applications to reusable services. These (atomic) services that offer coarse-grained functionality required for accomplishing the business processes and are then being assembled in a process-oriented way to composite services implementing fully automated and reusable parts of business processes. This approach allows for flexible adjustments in quickly changing business processes. Web services with the Web Service Description Language (WSDL) for interface description and SOAP as communication protocol are the most promising technologies for the implementation of SOA, but also other technologies like for instance CORBA are conceivable. Concerning the development process for SOA a model-driven approach is commonly embraced. More precisely, various approaches for the mapping of business processes to an SOA-based IT support have been proposed. Thereby, business processes are formally described in a notation which allows the automated mapping to an execution language and the execution by a process engine. As these kinds of execution language mainly facilitate the possibility for composing services in a process-oriented way, the development is also referred to as programming-in-the-large. In the web service context, especially the Business Process Modeling Notation (OMG-BPMN) supports such a programming-in-thelarge by introducing

an adequate metamodel for specifying executable business processes [EW+06]. In case of BPMN, the abovementioned automatic mapping is already defined for the Business Process Execution Language (BPEL) [OASIS-BPEL], which represents the most prominent execution language for specifying executable business processes. Typically, an SOA has to support multiple business processes, which currently are specified by means of several independent BPMN models.
The following subsection describes SOA service metamodel and process oriented methodology in more details:

- The SOA service metamodel consisting of the conceptual part, the deployable part and the formal definition as a UML profile.
- Process-oriented methodology for designing an SOA on basis of our metamodel.

### 3.2.1 SOA Service Metamodel

In this section our SOA service metamodel is introduced. This metamodel is supposed to allow a comprehensive modeling of SOA, including atomic and composite services along with the components implementing them. Furthermore, a distinction is drawn between a solely conceptual service model and a deployable service model which extends the conceptual model by deployment-specific information, like for instance the actual service endpoints. The following picture (fig. 4) shows the conceptual part of the SOA service metamodel:

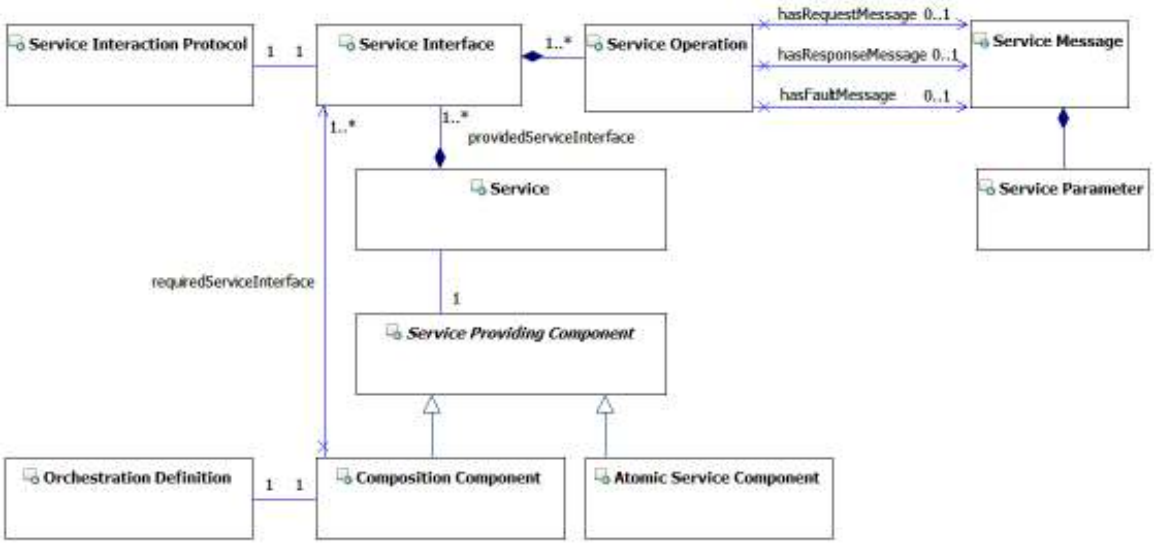


*Figure 4 Conceptual Part of the SOA Service Metamodel*

As stated before, within an SOA a general distinction is drawn between composite and atomic services. Composite services use the explicit composition functionality of SOA.

The central element of our metamodel represents the Service. It provides a set of Service Interfaces each of them consisting of Service Operations. This containment relation is strictly enforced. The provision of Service Interfaces is modeled using the association “providedServiceInterface”. The usage view on a service is defined via the signatures of its Service Operations and the corresponding Service Messages which can be of type Request Message (incoming) or Response Message (outgoing). This is modeled using associations between Service Operation and Service Message. In this way, Service Messages can be used both in different Service Operations and in different contexts: they can be Request Messages for one Service Operation and Response Message for another Service Operation. As a constraint, a Service Operation does either have to have a Request Message or a Response Message. Additionally, a Fault Message can be defined for each Service Operation which is used if an error occurs. Each Service Message consists of a set of Service Parameters; at least one has to be defined.

So far, we defined the external view on a service. The aforementioned elements do not describe the Service’s functional part (i.e. the Service Providing Component) yet. This is why we put a n:1 association between a Service and its implementation, the (abstractly defined) Service Providing Component. In case of atomic services, this Service Providing Component is an Atomic Service Component, which basically relates to a traditional component artifact. The previously introduced elements allow the modeling of atomic services. Neither the specification of services composition nor the explicit modeling of dependencies between the composite and the included atomic services is supported yet. For this purpose, we introduce the Composition Component as a Service Providing Component that provides the implementation for the composite service. Unlike atomic services, the composition service’s application flow (i.e. orchestration) is implemented using explicit SOA composition technology. The required information is held as the executable Orchestration Definition, for instance based on the Business Process Modeling Notation or UML Activity Diagrams as defined in UML Superstructure [OMG-Super]. In order to execute these definitions they have to be transformed to an executable language like BPEL, which are then being deployed on a BPEL engine. Concerning the Service Interface, there are no considerable differences between compositions and atomic services. Just as well, regular Service Operations are provided. However, regarding long-running compositions, for example, a Service Interaction Protocol has to be additionally defined. This protocol is also referred to as the abstract process or orchestration and defines the sequences of operation invocations. Note that atomic services may be implemented as stateful services and therefore also require such a protocol specification. One essential feature of composite services represents the composition of already existing services to more complex services. Thereby, the included services may be either atomic or composite. Hence, the metamodel has to support the modeling of dependencies between service compositions and the included services. For this purpose, we introduce the association “requiredServiceInterface” which allows the linkage of a

Composition Component with the required Service Interfaces. The Orchestration Definition in turn refers to the imported Service Operations, in case of BPMN for instance within the scope of embedded receive, reply or service tasks.

#### 3.2.1.1 Enhancing SOA Service Metamodel with Deployment Information

At this point, we are able to model atomic as well as composite services including the relationships between them in a purely conceptual way. The services are fully specified regarding their offered functionality along with the service providing components. However, in order to operate the services, additional deployment information is required. In consequence, for each conceptually specified Service there may be several Deployable Services. With the word “deployable” we express that the service model comprises additional deployment-relevant information, but the services do not have to be actually deployed yet. However, the deployment enhanced metamodel may form the basis for a corresponding (operational) deployment model, parts of which could be generated automatically. The figure below (fig 5) illustrates the extensions of the previously presented metamodel required for specifying Deployable Services. The newly introduced elements extend their conceptual counterpart by deployment relevant information. Note that the associations shown in (fig 2) are actually inherited from the conceptual elements, which – for the sake of clarity – are hidden in this diagram. A Deployable Service Interface for instance inherits all features of the related conceptual service interface, but is extended by the supported binding type and the service’s endpoint reference. Each specified deployable element is associated with one distinct conceptual element via a designated association (e.g. hasConceptual). Compiling a model of the deployable services several constraints apply depending on the used element. These constraints do not only apply to the elements of the metamodel (M2 level according to the UML 4-layer metamodeling hierarchy (OMG-Infra), but also on the instantiated models (M1 level).

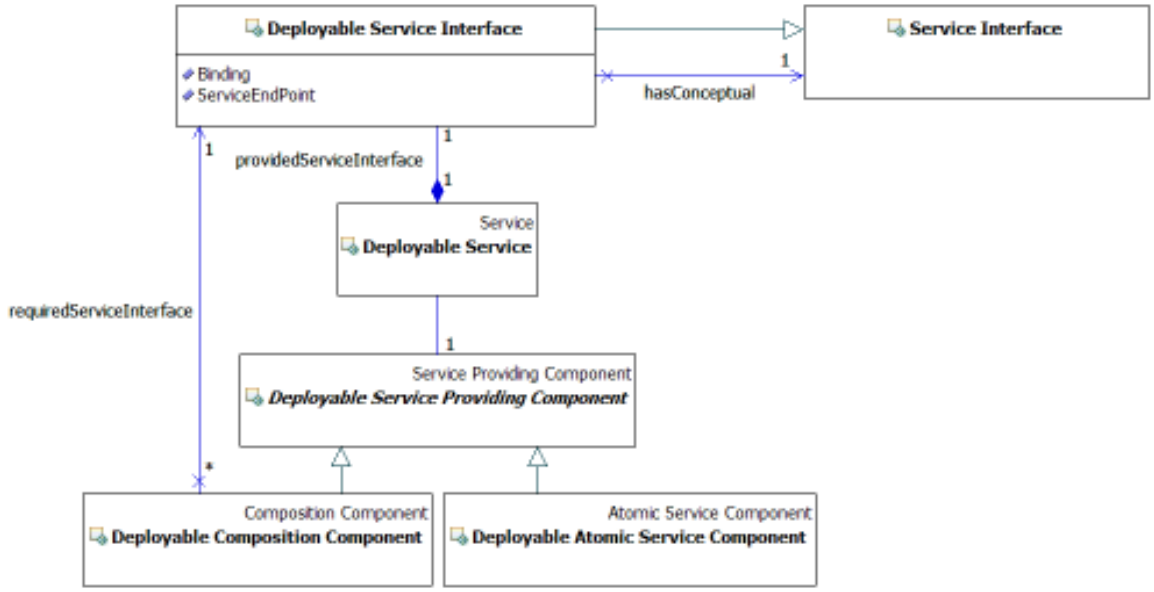


*Figure 5 Deployable Part of the SOA Service Metamodel*

- On the M1 level, a Deployable Service Interface which is associated with a Deployable Service has to comprise exactly the same features as the conceptual Service Interface belonging to the associated conceptual Service. For example, a Deployable Service Interface has to offer exactly the same Service Operations as the associated conceptual Service Interface on the M1 level. The same applies for Deployable Atomic Services and Deployable Composition Components.
- In case of a Deployable Composition Component only Deployable Service Interfaces may be included via the (inherited) association requiredServiceInterface.
- On the M1 level, for each conceptual Service Interface the corresponding Composition Component includes via the association requiredServiceInterface, the respective Deployable Composition Component requires exactly one Deployable Service Interface that corresponds to the included conceptual Service Interface. For example, if a Composition Component “c1” requires a Service Interface “s1” and there are two Deployable Service Interface for “s1”, namely “s1,1” and “s1,2”, a corresponding Deployable Composition Component “dc1,1” requires exactly one of them.

Using this extension for the (conceptual) SOA service model we are able to bridge the gap between a pure design model and an operational model. Furthermore, this approach conforms to the distinction that is drawn between the abstract and the concrete part within WSDL (W3C-WSDL). Accordingly, the specified atomic Deployable Services hold all information needed for an automated generation of a fully fledged WSDL along with skeletons for the specific implementation. In case of composite services, the specific BPEL

deployment descriptor holding the binding information about the included partner's endpoints can also be generated via parsing all corresponding associations of type requiredServiceInterface.

### 3.2.1.2 UML profiles

To be able to apply our service model in an UML-based software development process, it is a prerequisite to define a UML-Profile deriving our concepts to meta-classes defined by UML superstructure (UML-Super). Thereby, we followed related approaches to SOA modeling and basically specified an extended component model. According to Table 1, we regard all kinds of (Deployable) Service Providing Components as specific types of components known from the UML metamodel. Thus, all the required stereotypes in this case – directly or indirectly – extend the UML meta-class Component. The (Deployable) Service itself extends the UML meta-class Port. As a service from an engineering point of view is often defined as a software entity that offers functionality in a standardized way [Le03], we regard the semantics of the element Port as most suitable. But in contrast to the UML component diagram, where several Ports may be attached to one Component, a Service Providing Component may only offer one Service. A Service on the other hand may be comprised of several Service Interfaces, which in turn offer several Service Operations. These stereotypes extend the corresponding UML meta-classes interface and operation.

| Stereotype in Service Model | Extension of Metaclasses of UML Superstructure |
|---|---|
| (Deployable) Service | Port |
| (Deployable) Service Interface | Interface |
| Service Operation | Operation |
| (Deployable) Service Providing Component, (Deployable) Atomic Service Component, (Deployable) Composition Component | Component |
| hasConceptual, hasRequestMessage, hasResponseMessage, hasFaultMessage | Association |
| providedServiceInterface | Provided interface |
| requiredServiceInterface | required interface |
| Service Interaction Protocol | Sequence Diagram or Protocol State Machine |
| Orchestration Definition | Activity Diagram or BPMN.BPD |
| Service Message, Service Message Parameter | Class |

*Figure 6 UML Profile for the SOA Service Metamodel*

Due to the fact that a Service Operation in our case refers to different Service Messages, we did not directly use the respective UML metaclass. Consequently, we also had to define a new stereotype for Service Interface, as this element may only provide such Service Operations. The same holds for the newly introduced associations "*providedServiceInterface*" and "*requiredServiceInterface*", which extend the UML metaclasses "provided interface" and "*required interface"*. All the remaining custom associations are derived from the UML Kernel metaclass

Association. In contrast to these straightforward profile extensions, for the elements Service Interaction Protocol and Orchestration Definition in each case several feasible options are conceivable. According to the Interaction Protocol may be specified through a protocol state machine offered by UML. As an alternative to this approach, propose the employment of UML sequence diagrams for this purpose. Within this paper we limit the scope to stateless services, which do not require such an Interaction Protocol. A final decision in this matter is part of our future work. The orchestration definition on the other hand may for instance be specified by means of UML activity diagrams. So the stereotype would extend the UML metaclass Activity Diagram. Unfortunately, activity diagrams are designed for a very general purpose. Unlike BPMN, the specific semantics of orchestration models is not regarded. But if BPMN were used for modeling orchestrations, these models could not be part of an integrated UML profile. Nevertheless, the different models could be synchronized through adequate transformations. This would be rather complex approach. With the introduction of the BPDM these discrepancies might be resolved.

### 3.2.2 Process-Oriented Methodology for Developing an SOA

In this section we will introduce a well known process-oriented methodology for developing an SOA using our previously introduced SOA service metamodel. This methodology is common for each SOA specific project. The following steps need to be performed:

- Identification and Modeling of Executable Business Processes – Definition of business processes, sub-processes and their relationship need to be defined.
- Identification and Modeling of the Atomic Services - Having identified and modeled the executable processes, the next step comprises the identification and modeling of the required atomic services. Within the process models we already pointed up the necessary service operations. After equivalent operations have been identified, these consolidated operations have to be grouped to services. This grouping can for instance be accomplished by creating a service for each involved legacy system. If the services grow too large a further segmentation may be performed by means of the process they support, the coarse-grained modules of an existing application they belong to, or along the involved business objects in terms of Create, Read, Update and Delete (CRUD) operations.
- Specification of Composite Services - Services are combined to composite services that usually appear as component.
- Specification of Deployable Services - The final step in the SOA service modeling represents the extension of the previously created conceptual services by deployment information. In consequence, different ServiceEndPoints are specified for the two Deployable Service Interfaces. The Binding in both cases is set to SOAP. The Deployable Atomic Service Components may be as well extended deployment-specific information, like for instance the additional information required to generate an

deployment descriptors for application servers like BEA WebLogic, IBM WebSphere or Redhat JBoss AS. As one can observe, each deployable element is connected with its corresponding conceptual element via the association hasConceptual. On basis of this association, the two different models can be synchronized in case of changes, like for instance the specification of a further Service Operation within the conceptual part.

- Mapping to BPEL - Using the process that we defined we can automatically generate the relevant BPEL code. To execute a BPEL process a BPEL engine is needed which parses the BPEL code and executes the contained instructions. Examples of existing BPEL engines are Oracle BPEL Process Manager and ActiveBPEL by ActiveEndpoints. All engines have in common that the BPEL process, which has to be deployed itself as well, needs to be supplemented. Of course the general BPEL code is always the same regardless which BPEL engine is used because it is standardized. But in practice the deployable BPEL packages differ from engine to engine. For instance, an engine-specific so-called deployment descriptor is additionally needed in order to execute the process. Using our approach, we can automatically generate the necessary deployment descriptors along with the required wrapper services, which extend the original WSDL by BPEL-specific information about the provided partner links.

# 4 Component Business Model

Component business models offer a proven approach to driving a specialized focus, both internally and externally. Internally, components help firms rethink the leverage they can achieve with the assets and capabilities they own. Externally, components help firms to archive specialized capabilities that they cann't feasibly create themselves. These are specific industry components that firms cann't create by the themselves (For example the firm can create it by external company or some public organization). Combining these types of specialization allows firms to redefine their competitive positions in the face of the sweeping changes in their industries, while simultaneously achieving the competing benefits of scale, flexibility and efficiency. CBM allows firms to evaluate the goals and strategy of the entire enterprise to take simultaneous advantage of internal and external specialization. Without increasing complexity, the model allows an organization to expand and evolve while reducing risk, driving business performance, boosting productivity, controlling costs and improving capital efficiency and financial predictability.

### 4.1.1 CBM Framework

As we have seen, components aggregate business activities into discrete modules that can be shared across the firm. But these components can work together within the context of an overall business model. CBM provides a framework for organizing these components. These component represent internal and external specialization by organizing activities by competency and accountability level. The fig 7 represents a CBM map with internal and external components.

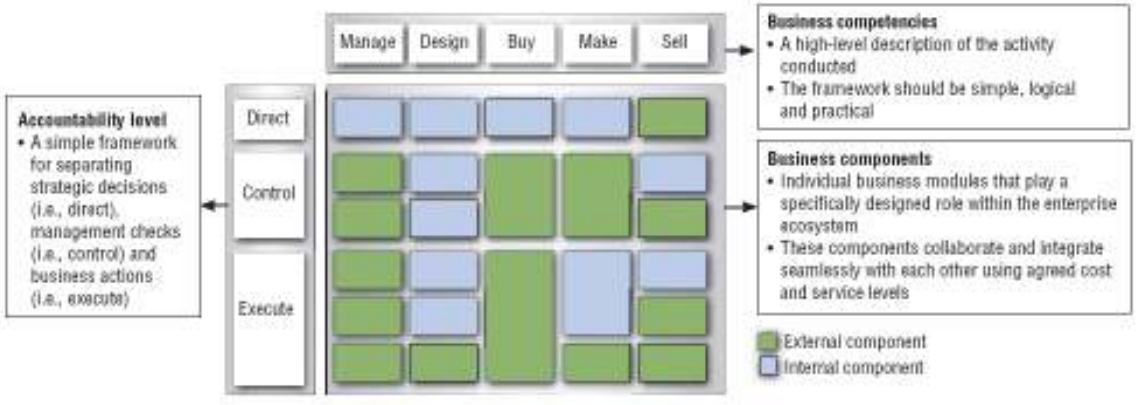


*Figure 7 CBM map*

By employing this framework, executives can begin to envision how current business activities might function as an interlocking set of modules. Categorizing

activities by business competency yields a high-level view of components according to the type of value they provide to the enterprise. Different firms in different industries will model their competencies differently, but, in every case, each activity should line up under a particular competency. Assigning each activity to one of three accountability levels – direct, control and execute – can also help executives begin to flesh out the component vision. The level of a given component should be intuitive, although exceptions will exist.

- Direct. Components at this level provide strategic direction and corporate policy to other components. They also facilitate collaboration with other components.
- Control. These mid-tier components serve as checks and balances between the “direct” and “execute” levels. They monitor performance, manage exceptions and act as gatekeepers of assets and information.
- Execute. These “boots on the ground” components provide the business actions that drive value creation in the enterprise. They process assets and information for use by other components or the end customer.

The three accountability levels imply different priorities. At the “execute” level, for example, the emphasis is on keeping people fully occupied and productive.
Components at this level tend to be structured in ways that make information easily available. From a technology standpoint, speed of data entry and real-time availability are very important. When customers go to an ATM, for instance, they want a simple interface that provides accurate information in a straightforward format: how much money is in my account?
Contrast this with activities related to the “direct” tier, where such high-level activities as launching new products are handled. This level houses a small number of people who have a very large impact on shareholder value, so the design imperatives are nearly the opposite of those at the “execute” tier. Launching a new product requires collaboration among several elements, including marketing, risk, finance, regulatory and credit. Input from all of these stakeholders is needed to make the launch a success, so workflow is a key requirement. From a technology standpoint, activities typically require people to discern patterns and trends from rich, multidimensional data, usually stored in a data warehouse. So, systems at the direct level are not designed for speed of data entry, but rather for ease, breadth and depth of analysis. Real-time interfaces are not needed, as data is often months old and processed in batches.

### 4.1.2 Business Components

Business components are the modular building blocks that make up the specialized enterprise. Each component encompasses five dimensions:

- A component’s business purpose is the logical reason for its existence within the organization, as defined by the value it provides to other components.
- Each component conducts a mutually exclusive set of activities to achieve its business purpose.
- Components require resources, the people, knowledge and assets that support their activities.

- Each component is managed as an independent entity, based on its own governance model.
- Similar to a standalone business, each business component provides and receives business services.

**Example of Business Component:** Bank decides to gather its credit decision activities into a single component. To realize efficiency gains, it centralizes all of the associated people, processes and assets that used to be spread across several business units. It also consolidates financial databases from across the firm, boosting the quality
of information on which its decision activities rely. Keeping the information in a single place also allows credit appraisers to make better choices when it comes time to assess portfolio information across accounts (say, when a checking customer applies for a credit card). With a much clearer picture of a customer's credit risk, the company can cross-sell its financial products much more effectively.

### 4.1.3 CBM Strategy road map

CBM is not simply a way to imagine the future of the organization. It can also be used to put theory into action and drive the evolution toward a specialized enterprise, both internally and externally. This process involves three dimensions:

- Developing a component view of the existing organization based on analysis of the business and the market environment.
- Evolving toward specialization based on a reinvention plan within the context of changing industry dynamics.
- Advancing the organizational and operational infrastructure toward component-based enterprise optimization.

#### 4.1.3.1Developing a component view of the enterprise

A company can begin to develop a component view of the enterprise by using the CBM framework as an analytical tool to identify the gaps and redundancies it must resolve on the way to becoming a component-based enterprise. A good way to start is by mapping the current business as a network of components. The initial analysis involves identifying and grouping cohesive activities into discrete units and testing the overall logic. The result is a "component map."
The next component map (fig. 8) shows the retail industry. Of course, every business will have its own, unique perspective on its component structure, despite substantial commonality with other players in its industry.

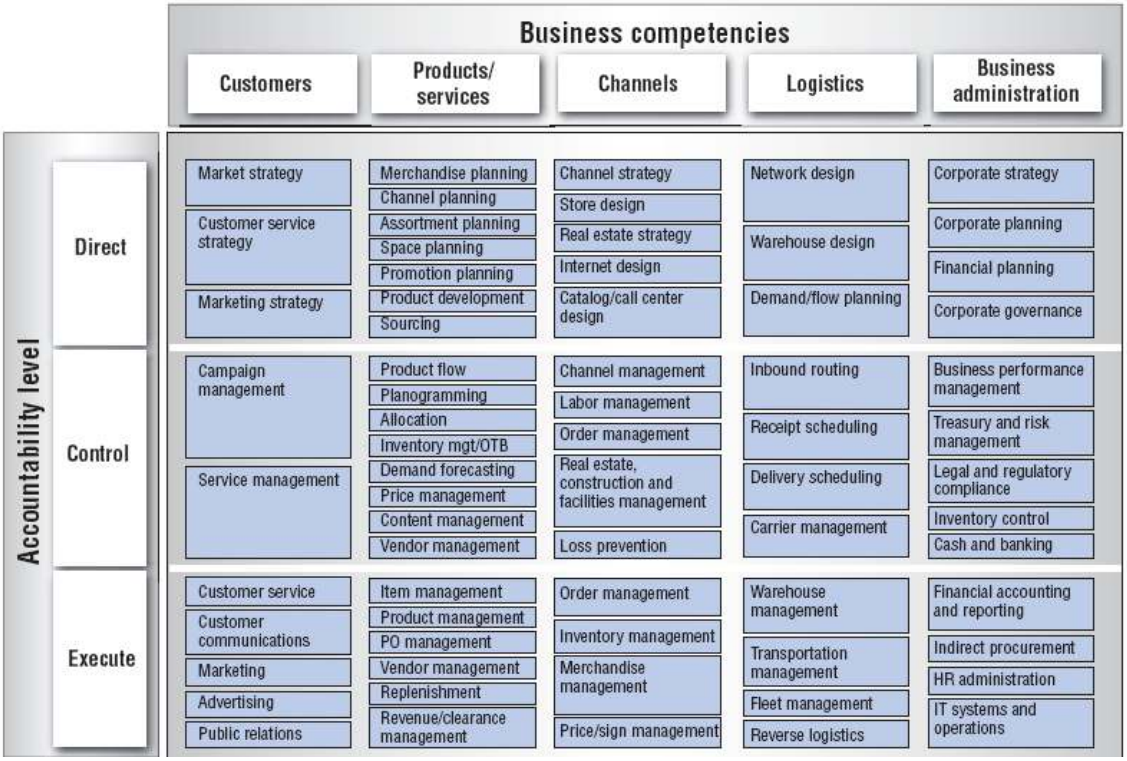


*Figure 8 Mapping the enterprises as network building model.*

The component map provides a basis for developing strategic and operating insights for the business. By gauging the relative business value of different areas of the map, executives can determine which components demand immediate attention. The next component map illustrates this type of analysis yields a “heat map” that highlights the components that represent the greatest economic value.

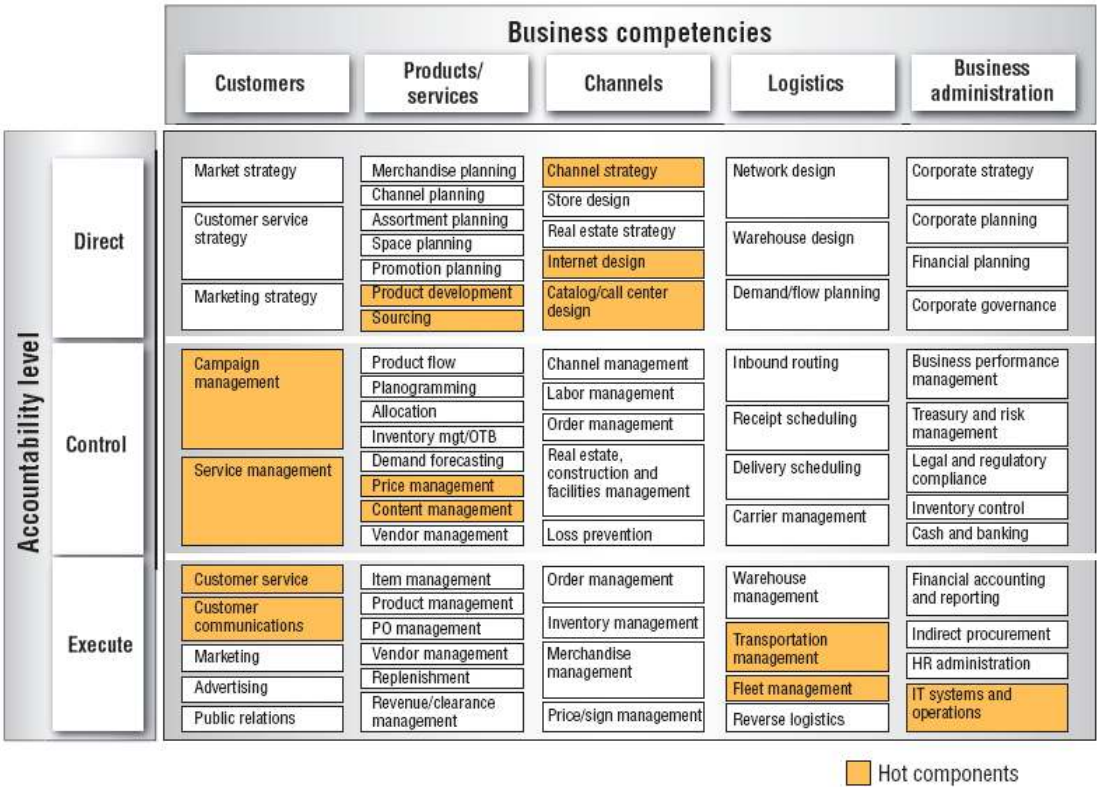


*Figure 9 Heat map identify "hot" areas to exploit business value.*

To determine heat map priorities, executives will typically consider the following questions. Which components differentiate them most significantly in the marketplace?

Which components have the most dramatic impact on their ability to maintain and grow margins? Which components offer significant cost and capital optimization opportunities?

For example, near-term changes that enhance the firm's strategic differentiators are likely to be designated as "hot" areas. Parts of the business that already resemble components, such as shared service centers, may also be early priorities. Quick wins are typically found when disparate and duplicate functions are consolidated into true operational components. Efficiencies gained in the first round of componentization can be used to support subsequent change initiatives.

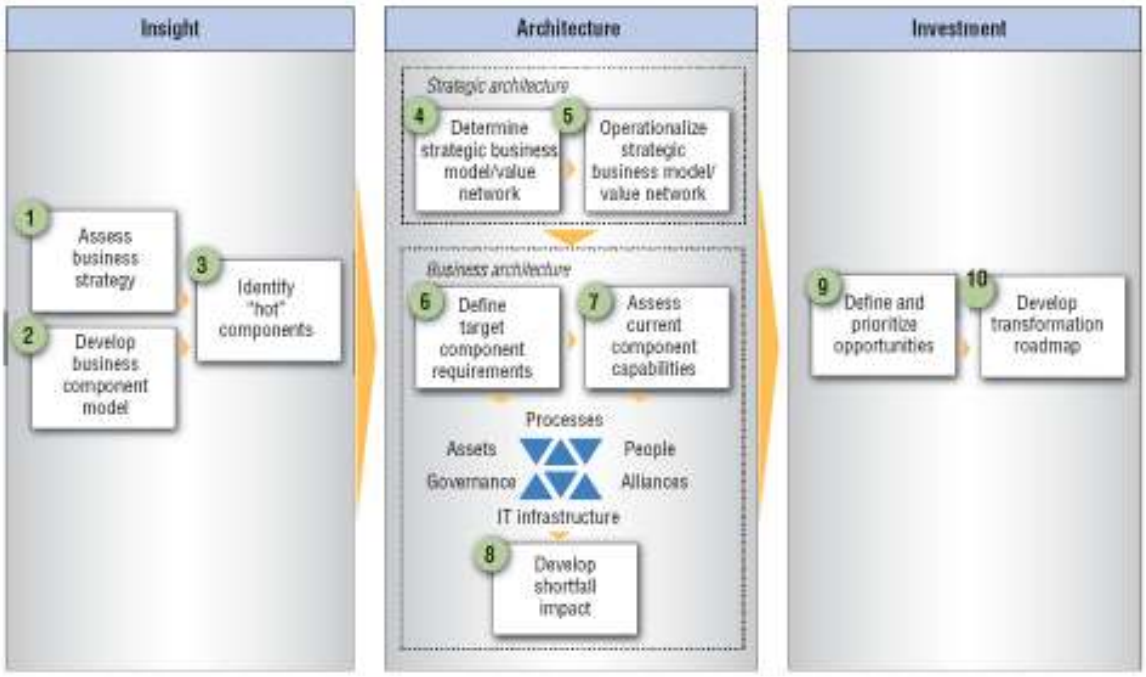


*Figure 10 Three phases of CBM analysis: Insight, Architecture, Investment*

After the insight phase of CBM analysis comes the architecture phase (fig 10). Here, the firm overlays the heat map onto the existing business. The goal is to identify gaps between the “to-be” vision of the componentized business and the “as-is” view – a representation of how the firm presently organizes its people, processes and technology. To capture the full scope of the firm’s current capabilities and market positioning, this “as-is” representation must be firmly grounded in empirical data, such as organization charts, cost drivers, application portfolios, technology investments, key performance metrics and existing processes. Finally, in the investment phase, the firm decides how to close the gaps: How big a leap can the firm take? How much change can be absorbed? Which areas should the company focus on fist? Where are the quick wins?

### 4.1.3.2Evolving toward CBM-based specialization

An enterprise can evolve toward its component-based vision by developing a reinvention plan. The good news is that many firms have already begun the CBM journey. Process reengineering and outsourcing have provided enterprises with modest levels of internal and external specialization. Most firms today have a blended process optimized and partnered model and now need to decide where to go.

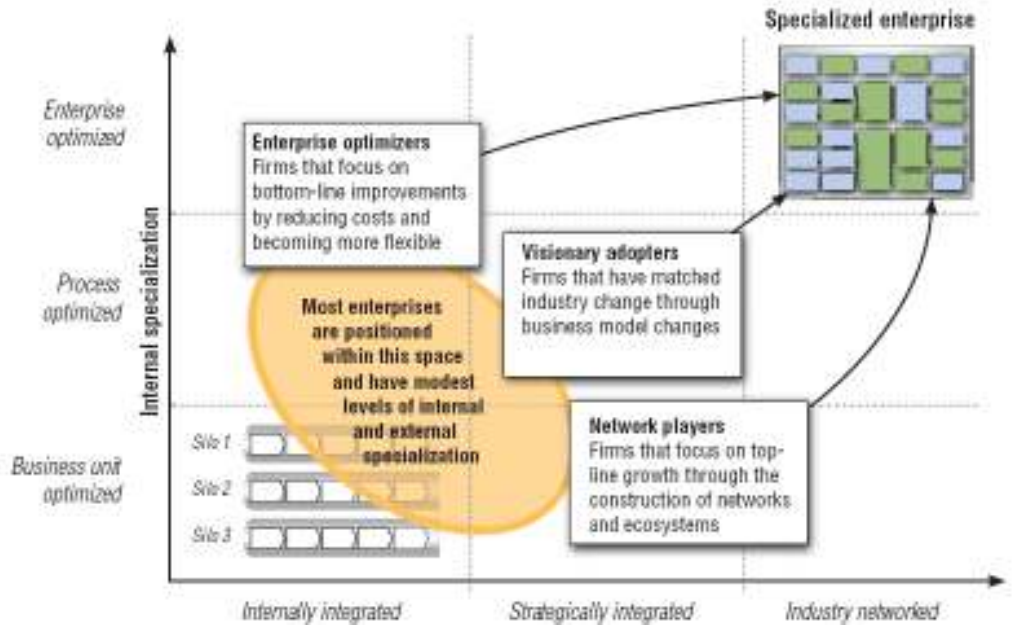


*Figure 11 External Specialization*

As shown into Figure 11 above, a firm can mature toward specialization by considering the role (or, in the vast majority of cases, the blend of roles) that provides the greatest competitive advantage in the marketplace. “Network players” focus on external specialization, focusing on top-line growth through the construction of networks and ecosystems. “Enterprise optimizers” drive internal specialization, focusing on bottom-line improvements by reducing costs and becoming more flexible. “Visionary adopters” strike a balance between the two, aligning internal and external strategies with industry trends to move more directly toward a specialized enterprise.

These three roles do not represent mutually exclusive approaches. Rather, they highlight the external, internal and blended aspects that all companies must take into account as they evolve toward specialization. Over time, the emphasis placed on any particular aspect will tend to vary depending on the firm, the industry and the current level of specialization. Most firms will find they must iterate between the external and internal dimensions strategically, selecting priorities that position them for further progress toward full specialization. At every stage, the enterprise should align its migration strategy with opportunities that create the most value most quickly.

### 4.1.3.3Building a component infrastructure

Components are autonomous in the sense that they are freed from the constraints of hardwired processes and organizational silos. But they do not operate in a strategic vacuum. To effectively serve the firm, components must work together toward a common goal – the delivery of sustainable value to the firm’s stakeholders. Achieving this alignment of ends requires the right

organizational model, process view and connectivity platform. Successful component-based organizational models balance the need for flexibility and discipline. To be responsive, the governance structure must be tied strongly to the customer value proposition, yet must also provide a clear context of defined relationships and measurable expectations as a basis for component interaction. Value networks should similarly be flexible and resilient, leveraging variable pricing and supply to support fluctuating demand while improving business continuity. Job descriptions should also be variable – based on organizational roles, rapid resource deployment and established methods for sharing knowledge and developing deep capabilities – rather than fixed around departmental structures. Finally, the organization's culture should provide a collaborative work environment that empowers employees to engage in fact-based decision-making. In addition to a flexible, disciplined organizational model, a successful component infrastructure also requires processes that are responsive across a sequence of components. Under CBM, processes are represented as sequences of activities performed via networks of collaborating components. The placement and timing of decision points that define the course of a process must be appropriate to the requirements of the organizational model. Recognizing and anticipating potential exceptions allows the enterprise to be more resilient.

Finally, the infrastructure should leverage the full power of the global connectivity platform to support the firm's evolution toward a specialized enterprise. Fortunately, trends in this area continue to be favorable. The combination of high-performance connectivity, widespread technology platforms and open protocols boosts collaboration and reduces the costs of coordination, both within firms and externally with partners.

Technologies like broadband, wireless, instant messaging and voice-over-IP streamline collaboration by offering realtime access to information and seamless connectivity beyond traditional boundaries. Complex enterprise activities are increasingly optimized by standard software like enterprise resource planning solutions. Hardware, software and storage costs continue to decline, even as application functionality and processing speeds increase. Open standards like Linux®, XML and service oriented architecture and programming (SOAP) help organizations tap the resources of the global connectivity platform while leveraging faster and cheaper plug-and-play substitution.

# 5 Business Processes and Modeling

It is an obvious, but often ignored statement that IT systems should support solutions to business problems. The SOA-led approach aligns IT solutions to business needs and constraints more directly than traditional techniques. The service-led approach implicit in any SOA changes the way IT thinks when it looks to provide automated solutions. Prior to implementing a solution, the business has to decide what the problem is, and the business value in solving that problem. This relates back to the wider problem of business strategy and business alignment. By aligning IT to the strategic business goals and values, SOA-based solutions can lead to much focused deliveries from IT.
All the information about business processes and modeling is provided by IBM from their red book Building SOA solution using Rational SDP chapter 5,6 and 7.
Unfortunately I cann't provide this information inside of my master thesis, but however the most interested readers can find it here: http://www.redbooks.ibm.com/abstracts/sg247356.html?Open

# 6 Service oriented modeling and architecture

Service oriented modeling and architecture covers a broader scope and implements service-oriented analysis and design through the identification, specification and realization of services, components that realize those services and flows that can be used to compose services.

SOMA includes an analysis and design method that extends traditional object-oriented and component-based analysis and design methods to include concerns relevant to and supporting SOA. It consists of three major phases of identification, specific and realization of the three main elements of SOA, namely, services, components that realize those services and flows that can be used to compose services.

SOMA is an end-to-end SOA Method for the identification, specification, realization and implementation of services (including information services), components, flows (processes/composition). SOMA builds on current techniques in areas such as domain analysis, functional areas grouping, variability-oriented analysis process modeling, component-based development, object-oriented analysis and design and use case modeling. SOMA introduces new techniques such as goal-service modeling, service model creation and a service litmus test to help determine the granularity of a service.

The three fundamental constructs of the SOA models are services, service components (implementations that realize those services), and flows (or processes) that orchestrate the services. SOMA was created specifically to address the analysis and design of all three constructs. As documented within the Rational® Unified Process (RUP), SOMA essentially consists of three major steps:

- Service Identification: Derives and defines the candidate services.
- Service Specification: Establishes and validates service exposure decisions and derivation of the high-level service model.
- Service Realization: Attributes and extends the high-level service model in terms of overall component design.

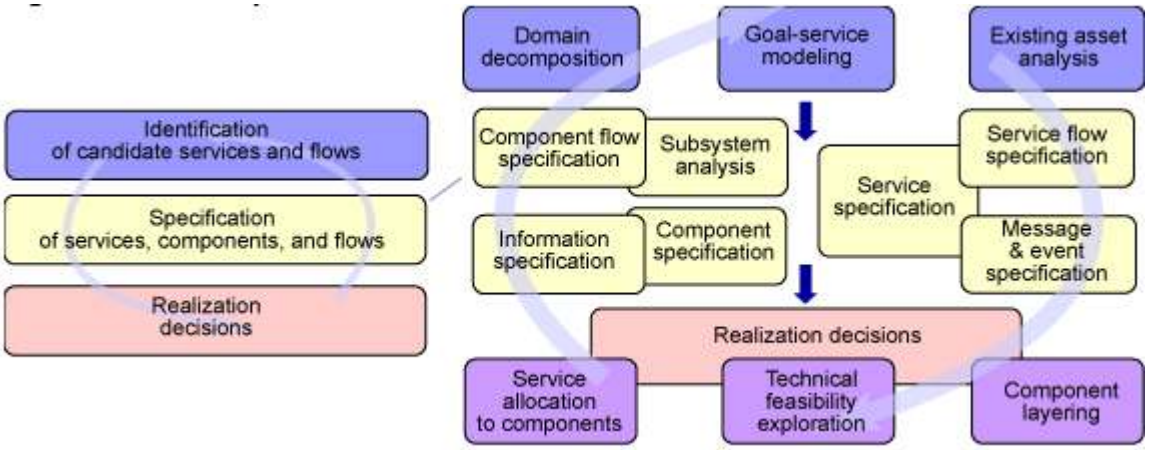


*Figure 12 SOMA Method*

### 6.1.1 Service Identification

The key output of the Service Identification step is a set of candidate business services. The list of candidate services is generated in Service Identification via the three techniques:

- A top-down approach known as domain decomposition.
- A bottom-up IT-centric approach focusing on discovery and characterization of existing IT assets.
- A meet-in-the-middle approach referred to as goal-service modeling.

Domain decomposition provides the top-down, business-driven technique aimed at capturing information about significant business domains, functions, conceptual subsystems, and business processes for an organization. Domain decomposition results from the specification of business requirements originating from the business component design. Additionally, domain decomposition can be enabled via the creation and validation of process models using tools like the WebSphere Business Modeler.

Business process modeling often takes place following the identification of key business activities arising from business component design. Process modeling usually starts with a business analyst using tooling to model the As-Is (current) state of the business process. Within the process model, analysts represent work activities or tasks as steps in the process. As the process model evolves and is reviewed by other business stakeholders, the "tasks" become analogous to candidate services. In some modeling tool implementations such as the WebSphere Business Modeler, business analysts can design As-Is and To-Be models, and can simulate the process to determine run time characteristics including costs, resource requirements, and process bottlenecks. Some tools also support the definition and specification of business key performance indicators (KPI). An example of a business KPI for Account Opening might be stating the average time needed to open an account should be less than 18

hours. Through ongoing design and simulation, process modeling links business requirements and business services to the identification of candidate services.
The existing asset analysis technique is enabled via tooling, as well as by the review of existing documentation and knowledge of existing IT assets. This activity examines existing IT assets that might be considered for implementing functionality required by the To-Be process. Sources of existing assets might include:

- Mainframe-based (for example, CICS/IMS/Batch) transactions.
- Commercial application (for example, SAP, Siebel) via API, messaging or service interfaces.
- Custom in-house applications, such as J2EE, .Net, and client/server applications.
- Services and interfaces for external services and components available through partners.

One tool that might be used to support this function is the WebSphere Asset Transformation Workbench. This tool assists IT personnel with the extension, reuse, and transformation of existing applications, and dependencies analysis of applications within mainframe and/or distributed environments.

As with domain decomposition, the result of the asset analysis activity is a list of potential candidate services. It should be explicitly stated that assets discovered (as well as the first iteration of potential candidate services) are not equal to services. In fact, most operations are fine-grained even when they are composed services, such as an IDOC or BAPI interactions through SAP. These candidate services are usually suboptimal in terms of conformance to SOA design principles and will likely be encapsulated by higher level services.

Lastly, goal-service modeling is derived from top-down and bottom-up approaches using the business and IT requirements to drive identification of additional candidate services. This activity helps in the identification of business-aligned services and ensures that services have not been identified during domain decomposition or existing asset analysis. Goal-service modeling starts with an identification of business goals, breaks them into sub-goals, and then determines which services are needed to fulfill these sub-goals.

#### 6.1.1.1 Service classification or categorization

This activity is started when services have been identified. It is important to start service classification into a service hierarchy, reflecting the composite or fractal nature of services: services can and should be composed of finer-grained components and services. Classification helps determine composition and layering, as well as coordinates building of interdependent services based on the hierarchy. Also, it helps alleviate the service proliferation syndrome in which an increasing number of small-grained services get defined, designed, and deployed with very little governance, resulting in major performance, scalability, and

management issues. More importantly, service proliferation fails to provide services, which are useful to the business, that allow for the economies of scale to be achieved.

#### 6.1.1.2Subsystem analysis

This activity takes the subsystems found above during domain decomposition and specifies the interdependencies and flow between the subsystems. It also puts the use cases identified during domain decomposition as exposed services on the subsystem interface. The analysis of the subsystem consists of creating object models to represent the internal workings and designs of the containing subsystems that will expose the services and realize them. The design construct of “subsystem" will then be realized as an implementation construct of a large-grained component realizing the services in the following activity.

### 6.1.2 Component Specification

In the next major activity, the details of the component that implement the services are specified:

- Data
- Rules
- Services
- Configurable profile
- Variations

Messaging and events specifications and management definition occur at this step.

#### 6.1.2.1Service allocation

Service allocation consists of assigning services to the subsystems that have been identified so far. These subsystems have enterprise components that realize their published functionality. Often you make the simplifying assumption that the subsystem has a one-to-one correspondence with the enterprise components. Structuring components occurs when you use patterns to construct enterprise components with a combination of:

- Mediators
- Façade
- Rule objects
- Configurable profiles
- Factories

Service allocation also consists of assigning the services and the components that realize them to the layers in your SOA. Allocation of components and services to layers in the SOA is a key task that requires the documentation and resolution of key architectural decisions that relate not only to the application

architecture but to the technical operational architecture designed and used to support the SOA realization at runtime.

### 6.1.3 Service Specification

The second major phase in SOMA is Service Specification. This technique contains a number of distinct steps, but for the purpose of this article, I will distill the discussion down to two major activities:

- Applying the Service Litmus Test to determine which candidate services are appropriate to expose.
- Specification of the service model in terms of dependencies, composition, non-functional requirements, message definition, and state management requirements.

The Service Litmus Test is a defined set of criteria to resolve whether a candidate service should be exposed. The criteria fall into four major areas:

- Business alignment: Focusing on business relevance of the service, the presence of a funding model to support development and maintenance, and the ability to share the service across the organization.
- Composability: Focusing on consistency with non-functional requirements at the composite level, consideration of state management aspects, identifying service dependencies, and supporting technology/platform neutrality.
- Externalized service description: Focusing on the presence of an external service description (such as WSDL), the ability to support service discovery and binding via the service description, and providing meta data as part of the service description.
- Redundancy elimination: Focusing on the ability to reuse the candidate service across multiple composite scenarios where the specific function is needed.

Through this set of questions and optional extensions and customizations, as appropriate for the specific organization, the design team can make appropriate architectural decisions regarding which services should be developed, exposed, and managed as service implementations.

The definition of the service model consists of multiple steps and normally results in the creation of UML work products. This step is facilitated through the use of architecture tooling such as the Rational Software Architect and the UML Profile for Software Services. During this step, the service model is designed through documenting service dependencies, defining service composition, documenting the service non-functional aspects, defining the high-level service message model, and specifying state management requirements.

### 6.1.4 Service Realization

As the last major activity in SOMA, Service Realization defines the allocation of services to a component and the allocation of these components to an implementation solution. For example, a service might be realized through an EJB that we expose as a Web service; another service may be realized through the wrappering of one or more CICS transactions, and another may be realized through an adapter providing a J2C interaction pattern.

# 7 SOA solution stack

SOA allows business and IT convergence through agreement on a set of business-aligned IT services that collectively support an organization's business processes and goals. Not only does it provide flexible, decoupled functionality that can be reused, but it also provides the mechanisms to externalize variations of quality of service; for example, in declarative specifications such as WS-Policy and related standards.
SOA "solution stack" includes reference architecture (an architectural template, or blueprint) for a Service-Oriented Architecture. It provides a high-level abstraction of an SOA factored into layers, each of which addresses specific value propositions within SOA. Underlying this layered architecture is a Meta model consisting of layers, architectural building blocks (ABBs), relations between ABBs and layers, interaction patterns, options, and architectural decisions. These will guide the architect in the creation of the architecture.
The layers facilitate separation of concerns and assist the process of creating an SOA in conjunction with methods such as the Service-Oriented Modeling and Architecture (SOMA) method. The SOA solution stack defines a blueprint that can be used to define the layers, architectural building blocks within the layers, options available at each layer, and typical architectural decisions that need to be made.
The SOA solution stack has nine layers that are designed to reinforce SOA business value. For each layer, there are two aspects: logical and physical. The logical aspect includes all the architectural building blocks, design decisions, options, key performance indicators, and the like; the physical aspect of each layer covers the realization of each logical aspect using technology and products.
The logical aspect of the solution stack addresses the question, "If I build a SOA, what would it conceptually look like and what abstractions should be present?" The solution stack enumerates the fundamental elements of an SOA solution and provides the architectural foundation for the solution.

As shown into a picture bellow (fig 13), the meta model of solution stack includes the following elements:

- Layer. An abstraction of the nine layers of the SOA solution stack that contains a set of characteristics, including architectural building blocks, architectural decisions, interactions among components, and interactions among layers.
- Option. A collection of possible options available in each layer that impacts other artifacts of a layer. Options are the basis for architectural decisions within and between layers.
- Architectural decision. A conclusion derived from options. The architectural decision involves architectural building blocks, key performance indicators, and nonfunctional requirements to provide information on configuration and usage of architectural building blocks.

Existing architectural decisions can also be reusable by some layers or architectural building blocks.

- Method activity. A collection of steps that involve architectural building blocks to form a process in a layer.
- Architectural building blocks. Reside in a layer and contain attributes, dependencies, and constraints as well as relationships with other architectural building blocks in the same layer or different layers.
- Interaction pattern. An abstraction of the various relationships among architectural building blocks; for example, patterns and diagrams.
- Key performance indicator (KPI). A constraint on architectural building blocks.
- Nonfunctional requirement (NFR). A constraint on architectural building blocks.
- Enabling technology. A technical realization of architectural building blocks in a specific layer.
- Externalized business solution element. A business service entity in a specific layer to be exposed to external consumers.
- Business solution connection. An adaptor for utilizing external services.
- Data model. Models data content associated with architectural building blocks, including data exchange between layers and external services.

### 7.1.1 SOA solution stack assumptions

There are several SOA solution stack assumes that are listed below:

- A set of requirements (service requirements) exists that collectively establishes the objective of the SOA. These requirements are both functional and nonfunctional in nature. Nonfunctional service aspects include security, availability, reliability, manageability, scalability, and latency.
- A service requirement is the documented capability that a service is expected to deliver. The provider view of a service requirement is the business and technical capability that a specific service needs to deliver given the context of all of its consumers. The consumer view of a service requirement is the business and technical capability that the service is expected to deliver in the context of that consumer alone.
- The fulfillment of any service requirement may be achieved through the capabilities of one layer or a combination of layers in the SOA solution stack.
- For each layer there is a specific mechanism by which the service requirements influence that layer.
- The identification of service requirements and the mapping of those requirements to each of the layers of the solution stack is a key aspect in developing an SOA for an enterprise.

### 7.1.2 Layers of the SOA reference architecture

The architectural diagram shown in picture (fig 13) below depicts an SOA as a set of logical layers. Note that the SOA solution stack is a partially layered architecture. One layer does not solely depend upon the layer below it. ; for example, a consumer can access the business process layer as a service or the service layer directly, but not beyond the constraints of the SOA architectural style. Further, a given SOA solution may exclude a business process layer and have a consumer's layer that interacts directly with the services layer. Such a solution would not benefit from the business value associated with the business process layer; however, that value could be achieved at a later stage by adding the layer. The degree to which a given organization realizes the full SOA solution stack will differ according to the level of service integration maturity it requires.

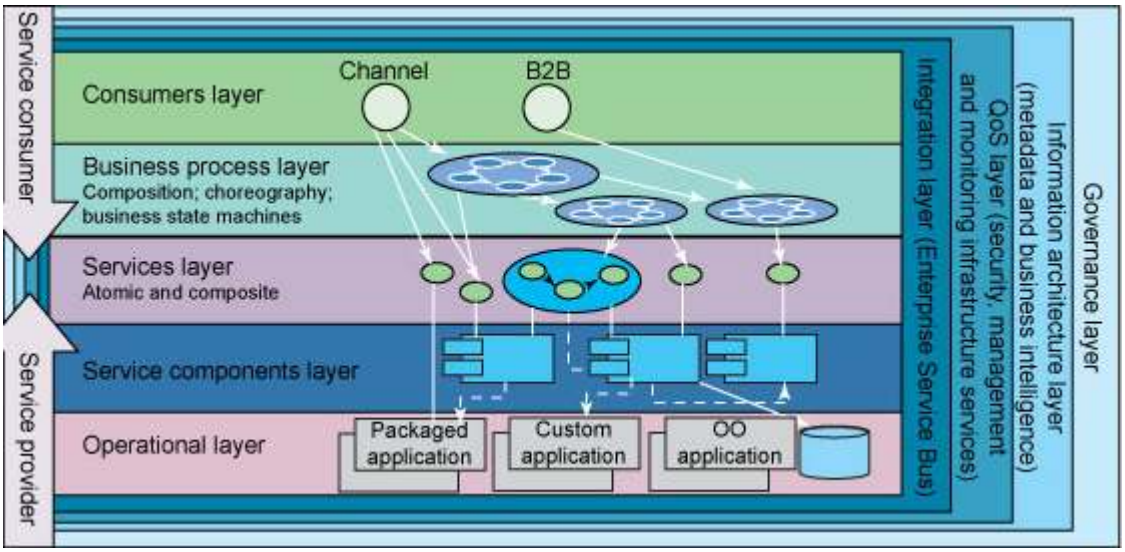


*Figure 13 SOA solution stack*

The picture above (fig 13) illustrates the multiple separations of concern in the nine layers of this reference architecture. Although the provider and the consumer can belong to the same organization -- and they usually do -- the SOA solution stack does not assume this to be the case. The main point of the provider/consumer separation is that there is value in decoupling one from the other along the lines of a business relationship. Organizations may have different lines of business that use this architectural template (where one organization is the consumer and another is the provider), customizing it for their own needs and integrating and interacting between organizations. In such cases there is still real value in maintaining a decoupled consumer/provider relationship. The lower layers (services, service components and operational layer) are concerns for the provider, and the upper ones (services, business processes, and consumers) are concerns for the consumer. In the remainder of this article we describe each layer and, in subsequent sections, describe the relationships between the layers.

There are five horizontal layers that relate to the overall functionality of the SOA solution. The vertical layers are nonfunctional in nature and support various concerns that cut across the functional layers.

#### 7.1.2.1 Operational layer (Layer 1)

This layer includes all custom or packaged application assets in the application portfolio running in an IT operating environment, supporting business activities.
The operational layer is made up of existing application software systems; thereby, it is used to leverage existing IT investments in implementing an SOA solution. This directly influences the overall cost of implementing the SOA solution, which can help free up budget for new initiatives and development of new business-critical services. A number of existing software systems are part of this layer. Those systems include:

- Existing monolithic custom applications, for example J2EE™ and Microsoft® .NET® applications
- Legacy applications and systems
- Existing transaction processing systems
- Existing databases
- Existing packaged applications and solutions, including enterprise resource planning (ERP) and customer relationship management (CRM) packages (such as SAP and Oracle solutions and etc.)

#### 7.1.2.2 Service component layer (Layer 2)

This layer contains software components, each of which provide the implementation for, realization of, or operation on a service, which is why it's called a service component. Service components reflect the definition of a service, both in its functionality and its quality of service. Service components may comply with the Service Component Architecture (SCA) and Service Data Objects (SDO).
The service component layer conforms to service contracts defined in the services layer; it guarantees the alignment of IT implementation with service description.

Each service component:

- Provides an enforcement point for "faithful" service realization to ensure quality of service and adherence to service-level agreements (SLAs).
- Enables business flexibility by supporting the functional implementation of IT flexible services as well as their composition and layering.
- Enables IT flexibility by strengthening decoupling in the system. Decoupling is achieved by hiding volatile implementation details from consumers.

The picture below (fig 14) illustrates these concepts and shows service A implemented using a combination of behavior from the third-party Package X and Application Y. Application B, the consumer, is coupled only to the description of the exposed service. The consumer must assume that the realization of the service is faithful to its published description (thereby providing service compliance), and it is the providers' responsibility to ensure that it is. The details of the realization, however, are of no consequence to Application B. Service Component A acts as a service implementation facade, aggregating available system behavior and giving the provider an enforcement point for service compliance.

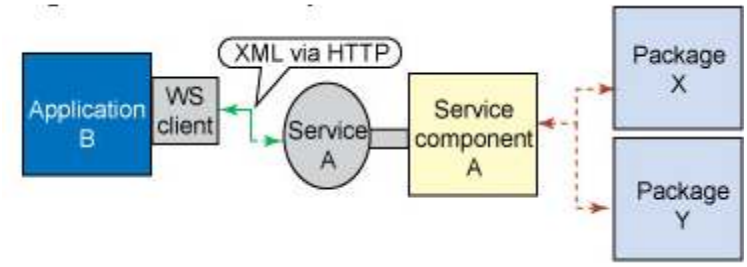


*Figure 14 Service component as a facade*

Subsequently, the provider organization may decide to replace Package X with Package M or some other application. Service component A encapsulates the required modifications with the result that there is no impact on any consumers of service A. This example illustrates the value of the service component layer in supporting IT flexibility through encapsulation.

### 7.1.2.3Services layer (Layer 3)

This layer consists of all the services defined within the SOA. For the purposes of this reference architecture, a service is considered to be an abstract specification of a collection of (one or more) business-aligned IT functions. The specification provides consumers with sufficient detail to invoke the business functions exposed by a provider of the service; ideally this is done in a platform-independent manner. The service specification includes a description of the abstract functionality offered by the service similar to the abstract stage of a Web Services Definition Language (WSDL). This information is not necessarily written using WSDL.The service specification may also include:

- A policy document
- SOA management descriptions
- Attachments that categorize or show service dependencies

Some of the services in the service layer may be versions of other services, implying that a significant successor-predecessor relationship exists between them.

Exposed services reside in this layer; they can be discovered and invoked or possibly choreographed to create a composite service. Services are functions that are accessible across a network through well-defined interfaces of the services layer. The service layer also takes enterprise-scale components, business-unit-specific components, and project-specific components and externalizes a subset of their interfaces in the form of service descriptions. Thus, the components provide services through their interfaces. The interfaces are exported as service descriptions in this layer, where services exist in isolation (atomic) or as composite services.

This layer contains the contracts (service descriptions) that bind the provider and consumer. Services are offered by service providers and are consumed by service consumers (service requestors).

Services and their underlying building blocks are defined according to the service identification activities defined through three complementary techniques:

- Domain decomposition
- Existing asset analysis
- Goal-service modeling

These techniques are part of Service-Oriented Modeling and Architecture (SOMA) method for the identification, specification, and realization of services, components and flows. They represent, therefore, the heart of the SOA value proposition -- improved agility from the decoupling of business and IT. The quality of these service definitions has a significant impact on the benefit of a given SOA effort.

Services are accessible independent of implementation and transport. This capability allows a service to be exposed consistently across multiple customer-facing channels such as the Web, interactive voice response (IVR), Seibel client (used by a customer service rep), and so on. The transformation of responses to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done through XSLT functionality supported through Enterprise Service Bus (ESB) transformation capability in the integration layer.

It is important to acknowledge that service components may consume services to support integration. The identification and exposure of this type of service (that is, internal services) does not necessarily require the same rigor as is required for a business service. While there may be a compelling IT-related reason behind the use of such services, they are not generally tied to a business process. As such, they do not warrant the rigorous analysis required for business services.

This set of requirements and services contained by this layer can be used to better leverage the various capabilities provided by a mix of different vendors. This is because the requirements enable the objective identification of SOA

infrastructure requirements. The solution stack provides a well-factored decomposition of the SOA problem space, which allows architects to focus on those parts of an SOA solution that are important in the context of the problem they are solving and to map the required capabilities to vendor product capabilities. This is preferred to trying to reverse-engineer SOA solution architecture from the capability of a particular vendor's products. So, in addition to being an important template for defining an SOA solution at a logical level, this layer of the SOA reference architecture is also a useful tool in the design of vendor-neutral SOA solutions.

The picture below (fig 15) magnifies the services layer shown into picture above, and it shows that the services layer can be further divided into sub layers. It includes the services that will be delivered by a given architecture, including both composite and atomic services. The picture above (fig. 15) shows how to build an SOA solution using the underlying middleware and infrastructure services provided and categorized in following picture (fig 15):

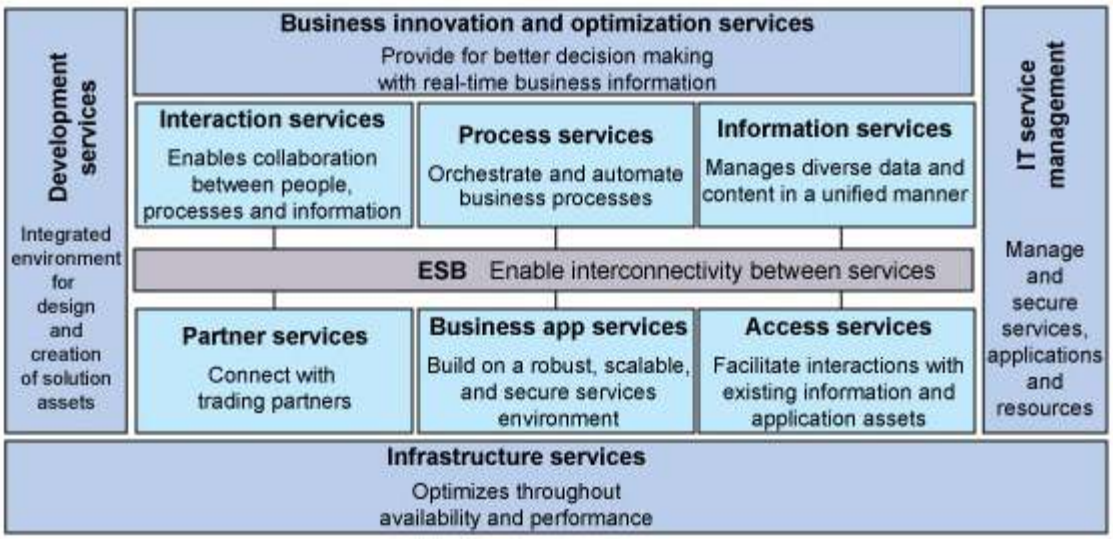


*Figure 15 The middleware view of the SOA reference architecture.*

## 7.1.2.4Business process layer (Layer 4)

Compositions and choreographies of services exposed in layer three are defined in this layer. We use service composition to combine groups of services into flows, or we choreograph services into flows, thereby establishing applications out of services. These applications support specific use cases and business processes. To do this, visual flow composition tools can be used for design of application flows. The picture below (fig 16) shows how a business process P can be implemented using services A, B, C, and D from the services layer. Process P contains the logic for the sequence in which the services need to be invoked and executed. The services that are aggregated as a business process,

or flow, can be individual services or composite services made up of individual services.

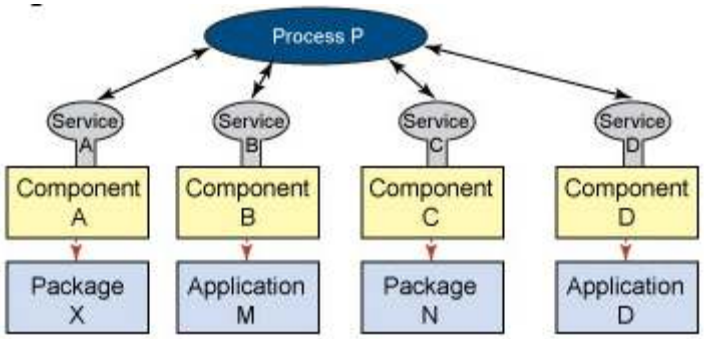


*Figure 16 Services orchestration*

The business process layer covers the process representation, composition methods, and building blocks for aggregating loosely coupled services as a sequencing process aligned with business goals. Data flow and control flow are used to enable interactions between services and business processes. The interaction may exist within an enterprise or across multiple enterprises.

This layer includes information exchange flow between participants (individual users and business entities), resources, and processes in a variety of forms to achieve the business goal. Most of the exchanged information may also include nonstructural and no transactional messages. Business logic is used to form service flows as parallel tasks or sequential tasks based on business rules, policies, and other business requirements. The layer also includes information about data flows within the enterprise or across multiple enterprises.

The life-cycle management for business process orchestration and choreography is also covered in this layer. In addition to the run-time process engine (for example, WS4BPEL engine), this layer covers all aspects of composition, collaboration, compliance, process library, process service, and invocation elements.

On demand building process blocks allow a change from high-volume transactional supporting technologies to a sophisticated, much smaller footprint and less-expensive applications. In today's business solutions, business processes play a central role in bridging the gap between business and IT.

A business process captures the activities needed to accomplish a specific business goal. Typically, an enterprise uses both top-down and bottom-up approaches to assure proper business process definition. Using the top-down approach, business processes are defined by business analysts based on customers' requirements. To optimize the business process for better IT implementation, it is componentized as a reusable service that can be modeled, analyzed, and optimized based on business requirements such as quality of

service (QoS) described in layer 7, flow preference, price, time of delivery, and customer preferences. Using a bottom-up approach, after creating a set of assets, we would try to leverage them in a meaningful business context to satisfy customer requirements. The flexibility and extensibility of services composition guided by business requirements and composition rules help make business process into an on demand entity for addressing different types of customer pain points by reusing services assets.

The business process layer communicates with the consumer layer (also called the presentation layer) to communicate inputs and results from the various people who use the system (end users, decision makers, system administrators) through Web portals or business-to-business (B2B) programs. Most of the control-flow messages and data-flow messages of the business process may be routed and transformed through the integration layer. The structure of the messages is most often defined by the information architecture layer. The key performance indicators (KPIs) for each task or process could be defined in the QoS and business intelligence layers. The design of service aggregations is guided by the governance layer. Of course, all the services should be represented and described by the services layer in the SOA solution stack.

From a technical perspective, dynamic and automatic business process composition poses critical challenges to researchers and practitioners in the field of Web services. Business processes are driven by business requirements, which typically tend to be informal, subjective, and difficult to quantify. Therefore, it is critical to properly formulate the descriptive and subjective requirements into quantifiable, objective, and machine-readable formats in order to enable automatic business process composition. In addition, the current Web services specifications generally lack the facility to define comprehensive relationships among business entities, business services, and operations. These relationships may be important to optimize business process composition. Clearly specifying search requirements to discover the most appropriate Web services candidates remains a challenge. Last, a typical business process generally requires multiple Web services to collaborate in order to serve business requirements. Therefore, each service not only needs to satisfy individual requirements, but must also coexist with other services to fit within the overall composed business process. This suggests that the entire business process needs to be optimized prior to execution.

Clearly, the business process layer in the SOA solution stack plays a central coordinating role in connecting business-level requirements and IT-level solution components through collaboration with the integration layer, QoS, and business intelligence layer, as well as the information architecture layer and services layer. Addressing the challenging issues that come up in the business process layer can further differentiate the proposed SOA solution stack from conceptual reference architectures proposed by other vendors.

### 7.1.2.5Consumer layer (Layer 5)

The consumer layer, or the presentation layer, provides the capabilities required to deliver IT functions and data to end users to meet specific usage preferences. This layer can also provide an interface for application to application communication. The consumer layer of the SOA solution stack provides the capability to quickly create the front end of business processes and composite applications to respond to changes in user needs through channels, portals, rich clients, and other mechanisms. It enables channel-independent access to those business processes supported by various application and platforms. It is important to note that SOA decouples the user interface from the components. Some recent standards such as Web Services for Remote Portlets (WSRP) Version 2.0 can be used to leverage Web services at the application interface or presentation level. Other suitable standards include SCA components, portlets, and Web Services for Remote Portlets (WSRP).

Adopting proven front-end access patterns (for example, portals) and open standards (such as WSRP) can decrease development and deployment cycle times through the use of pre-built, proven, and reusable front-end building blocks. Use of these patterns also reduces complexity and maintenance costs through use of those common building blocks. This practice promotes a single unified view of knowledge presentation as well as a single unified entry point to the supported business processes and applications. This unified entry point integrates with other foundational services, such as security (single sign-on, for example) and trust, and significantly improves the usability of the business process and application. More specifically, it allows for the plug and play of content sources (for example, portlets) with portals and other aggregating Web applications. As a result, adopting common front-end patterns standardizes the consumption of Web services in portal front ends and the way in which content providers write Web services for portals.

Scenarios that motivate WSRP-like functionality include:

- Portal servers providing portlets as presentation-oriented Web services that can be used by aggregation engines
- Portal servers consuming presentation-oriented Web services provided by portal or non-portal content providers and integrating them into a portal framework

The same functionality can also be obtained through non-portal environments. WSRP allows content to be hosted in the environment most suitable for its execution while still being easily accessed by content aggregators. The standard enables content producers to maintain control over the code that formats the presentation of their content. By reducing the cost for aggregators to access their content, WSRP increases the rate at which content sources may be easily integrated into pages for end users. It should be noted that Asynchronous JavaScript and XML (Ajax), which is used to exchange XML contents over HTTP

without refreshing Web browsers, can be used to enhance SOA interaction capability with Web users.

### 7.1.2.6Integration layer (Layer 6)

The integration layer is a key enabler for an SOA because it provides the capability to mediate, route, and transport service requests from the service requester to the correct service provider. This layer enables the integration of services through the introduction of a reliable set of capabilities. These include modest point-to-point capabilities for tightly coupled endpoint integration as well as more intelligent routing, protocol mediation, and other transformation mechanisms often provided by an enterprise service bus (ESB). Web Services Description Language (WSDL) specifies a binding, which implies the location where a service is provided. An ESB, on the other hand, provides a location-independent mechanism for integration.

The integration that occurs here is primarily the integration of layers 2 thru 4. This is the layer that provides communications, invocation, and quality of service between adjacent layers in an SOA. For example, this layer is where binding of services occurs for process execution, allowing a service to be exposed consistently across multiple customer-facing channels such as Web, IVR, Seibel client, and the like. The transformation of response to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done using XSLT functionality supported through ESB transformation capability in the integration layer.
As shown into picture below (fig 17), the integration layer does the following:

- Provides a level of indirection between the consumer of functionality and its provider. A service consumer interacts with the service provider by way of the integration layer. As a result, each service specification is only exposed through the integration layer (such as an ESB and WMB), never directly.
- Decouples consumers and providers, allowing for integration of disparate systems into new solutions.

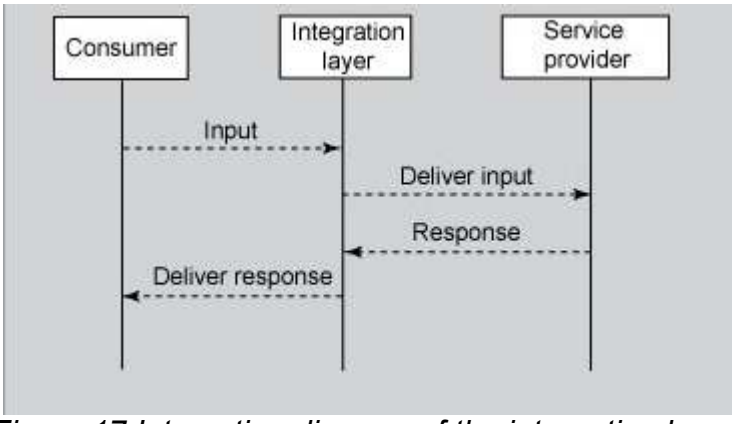


*Figure 17 Interaction diagram of the integration layer.*

#### 7.1.2.7 Quality of service layer / QoS (Layer 7)

Inherent in SOA are characteristics that exacerbate existing QoS concerns in computer systems. Among those characteristics are:

- Increased virtualization
- Loose coupling
- Widespread use of XML
- The composition of federated services
- Heterogeneous computing infrastructures
- Decentralized SLAs
- The need to aggregate IT QoS metrics to produce business metrics

These characteristics create complications for quality of service that clearly require attention within any SOA solution.

The QoS layer provides an SOA with the capabilities required to realize nonfunctional requirements (NFRs). It must also capture, monitor, log, and signal noncompliance with those requirements relating to the relevant service qualities associated with each SOA layer. This layer serves as an observer of the other layers and can emit signals or events when a noncompliance condition is detected or, preferably, when a noncompliance condition is anticipated.

Layer 7 establishes non-functional requirement related issues as a primary feature or concern of SOA and provides a focal point for dealing with them in any given solution. This layer provides the means of ensuring that an SOA meets its requirements with respect to reliability, availability, manageability, scalability, and security. Finally, it enhances the business value of SOA by enabling businesses to monitor the business processes contained in the SOA with respect to the business KPIs that they influence.

#### 7.1.2.8 Information architecture and BI layer (Layer 8)

The information architecture and business intelligence layer ensures the inclusion of key considerations pertaining to data architecture and information architectures that can also be used as the basis for the creation of business intelligence through data marts and data warehouses. This includes metadata content, which is stored in this layer, as well as information architecture and business intelligence considerations.

Especially applicable to industry-specific SOA solutions, this layer captures cross-industry and industry-specific data structures, XML-based metadata architectures (that is, XML schema), and business protocols of exchanging business data. Some discovery, data mining, and analytic modeling of data are also covered in this layer.

#### 7.1.2.9Governance layer (Layer 9)

The governance layer covers all aspects of business operational life-cycle management in SOA. It provides guidance and policies for making decisions about an SOA and managing all aspects of an SOA solution, including capacity, performance, security, and monitoring. It enables SOA governance services to be fully integrated by emphasizing the operational life-cycle management aspect of the SOA. This layer can be applied to all the other layers in the SOA solution stack. Since it helps enforce QoS and make appropriate application of performance metrics, it is well connected with layer 7.

This layer can speed the SOA solution planning and design process. The governance layer provides an extensible and flexible SOA governance framework that includes solution-level service-level agreements based on QoS and KPIs, a set of capacity planning and performance management policies to design and tune SOA solutions, and solution-level security enablement guidelines from a federated applications perspective. The architectural decisions in this layer are encapsulated in consulting practices, frameworks, architectural artifacts, documentation of SOA capacity planning, any SOA-solution SLAs, SOA performance-monitoring policies, and SOA solution-level security-enablement guidelines.

Note that we do not have a separate layer for business rules and policies. Business rules cut across all layers. For example, business process and governance layers intersect in defining the rules and policies for the business process. Consumer layer validation rules, and input and output transformations from and to that layer, must abide by some rules. These lie at the intersection point between the consumer and governance and policy layer.

# 8 IBM Based tools for development of SOA projects

This section covers IBM based tools for development of SOA projects. In terms of "phases" most of SOA project has the following areas:

- Component business model area
- Service oriented modeling and architecture area
- Service oriented architecture area

The following table (tab 1) provides mapping information between IBM well known tools (that can be use in SOA projects) and each area that was defined above:

| Areas | Tools |
|---|---|
| CBM | • CBM Tool |
| SOMA | • SOA-IF<br>• SOMA-ME |
| SOA | • Websphere Business Modeler<br>• Websphere Process Server<br>• UDDI Registry<br>• WS Registry<br>• Websphere ESB<br>• Websphere Message Broker<br>• Websphere Integration Developer |

*Table 1 Mapping table between IBM tools and methods*

I would like to mention that there are lots of other tools (such as Tivoli Family for monitoring, rational tools for testing and UML modeling, LDAP and etc.) that can be used into SOA projects, but I will be focused only in the tools that are listed into table above (tab 1). Those tools are selected based on my practical experience and available free information.

## 8.1 SOA-IF

IBM's SOA Integration Framework project will be a major source of next generation SOA Best Practices. IBM's SOA Integration Framework, brings to pilot users "a set of Best Practices, templates, recipes and scripts for how to best use IBM SOA technologies, including Tivoli, Websphere and Rational.

## 8.2 Websphere Business Modeler

WebSphere Business Modeler products help organizations fully visualize, comprehend, and document their business processes. Rapid results can be obtained through the collaboration functionality, where subject matter experts team to clearly define business models and eliminate inefficiencies. You can

model business processes, then deploy, monitor, and take actions based upon KPIs, alerts, and triggers for continuous optimization. Business processes then get tightly linked with strategic corporate objectives. WebSphere Business Modeler products can drive much more granular business insight and knowledge, where knowledge equates to competitive advantage.

## 8.3 Websphere Process Server

WebSphere Process Server is a high-performance business engine to help form processes to meet your business goals. Technically, WebSphere Process Server is mounted on top of WebSphere Application Server and extends the WebSphere Enterprise Service Bus. It uses the WebSphere Integration Developer as development tool. Their components are:

- Service Components

- Business State Machines - A business state machine is a way of modeling a business process, representing it as a sequence of states and events.
- Business Processes - The business process component in WebSphere Process Server implements a WS-BPEL compliant process engine. Users can develop and deploy business processes with support for long and short running business processes and a robust compensation model in a highly scalable infrastructure. WS-BPEL models can be created in WebSphere Integration Developer or imported from a business model that has been created in WebSphere Business Modeler.
- Human Tasks - Human tasks in WebSphere Process Server are stand-alone components that can be used to assign work to employees or to invoke any other service. Additionally, the Human Task Manager supports the ad-hoc creation and tracking of tasks. Existing Lightweight Directory Access Protocol (LDAP) directories (and operating system repositories and the WebSphere user registry) can be used to access staff information. WebSphere Process Server supports multi-level escalation for human tasks, including e-mail notification. WebSphere Process Server also includes an extensible Web client that can be used to work with tasks or processes. This Web client is built based on a set of reusable Java Server Faces (JSF) components that can also be used to create custom clients or embed human task functionality into other Web applications.
- Business Rules - Business rules are a means of implementing and enforcing business policy through the externalization of business. This allows dynamic changes of a business process for a more responsive business environment. Business rule authoring is supported by an Eclipse-based desktop tool. WebSphere Process Server also includes a Web-based runtime tool for business analysts so that business rules can be updated as business needs dictate without affecting other SCA services.

- Supporting Services

- Interface Maps
- Business Object Maps - Used to translate one type of business object into another type, these maps can be used in a variety of ways (for example, as an interface map to convert one type of parameter data into another).
- Relationships
- Selectors - Different services that all share the same interface can be selected and invoked dynamically by a selector.
- Adapters - Supports both WebSphere Business Integration (WBI) Adapters and JCA 1.5 Compliant WebSphere Adapters.

- SOA Core

- Service component architecture
- Business Objects
- Common Event Infrastructure

## 8.4 UDDI Registry

The Universal Description, Discovery & Integration (UDDI) specification provides (fig 18) a platform independent way of describing and discovering Web services and Web service providers. The UDDI data structures provide a framework for the description of basic service information, and an extensible mechanism to specify detailed service access information using any standard description language.
A business entity will be created in the UDDI registry in order to group a set of services implemented in the project. All services will be published under this business entity. The services will have unique names. A separate directory exposed via a web server will be used as a repository for the services WSDL files. Service records within UDDI will contain pointers to these WSDL files. Also they will contain actual services' URLs (access–points).
In the Websphere stack IBM UDDI registry can be installed on top of Websphere Application Server, Websphere Portal Server and etc.

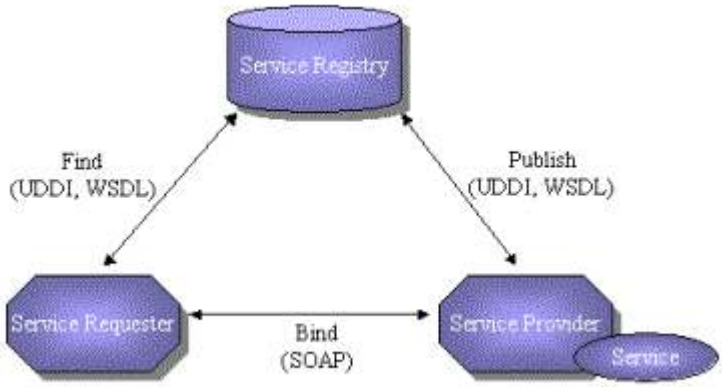


*Figure 18 Registry relationship*

## 8.5 WS Registry and Repository

WebSphere Service Registry and Repository is a system for storing, accessing and managing information, commonly referred as service metadata, used in the selection, invocation, management, governance and reuse of services in a successful SOA. WebSphere Service Registry and Repository provides both Java and Web services interface for searching, updating, creating and deleting service description and associated metadata. In other words, it is where you store information about services in your systems, or in other organizations' systems, that you already use, plan to use, or want to be aware of. For example, an application can check the Registry & Repository just before invoking a service to locate the service instance best satisfying its functionality and performance needs. Registry & Repository also play a role in other stages of the SOA lifecycle.
Our view of a Registry & Repository encompasses:

- Service Registry that contains information about services, such as their interfaces, operations and parameters
- Metadata Repository providing a robust, extensible framework to suit the diverse nature of service usage.

## 8.6 Websphere ESB

An Enterprise Service Bus (ESB) is a flexible connectivity infrastructure for integrating applications and services. An ESB can power your service-oriented architecture (SOA) by reducing the number, size, and complexity of interfaces between those applications and services.
An ESB performs the following functions:

- Route messages between services
- Convert transport protocols between requester and service
- Transform message formats between requester and service
- Handle business events from disparate sources

An ESB should allow customer organizations to focus on it core business needs rather than the IT infrastructure required for connecting the programs together. An ESB should allow you to add new services or make changes to existing services with little or no impact to the use of existing services.

## 8.7 Websphere Message Broker

WebSphere Message Broker delivers an advanced Enterprise Service Bus to power your service-oriented architecture. It provides connectivity and universal data transformation for both standard and non-standards-based applications and services. The following functionalities are also included:

- Transforms and enriches in-flight information to provide a level of intermediation between applications that use different message structures and formats
- Enriches and distributes real-time information from disparate sources of information through a network of access points, and provides a powerful new means to unify organizations
- Integrates with multiple sources of data such as databases, applications, and files to perform any type of data manipulation, including logging, updating, and merging
- Simplifies the integration of existing applications with Web services by transforming and routing SOAP messages, as well as logging of Web services transactions
- Includes the functionality for customers requiring publish and subscribe capabilities and other distribution capabilities using multiple protocols without the powerful and adaptable transformation futures.
- WebSphere Message Broker with Rules and Formatter Extension extends the capabilities of the Message Broker to offer continuity and compatibility for customers who require IBM Rules and Formatter nodes.

## 8.8 Websphere Integration Developer

Websphere Integration Developer is a common tool for building SOA-based integration solutions across WebSphere Process Server, WebSphere ESB, and WebSphere Adapters.

- Simplifies integration with rich features that accelerate the adoption of service-oriented architecture by rendering existing IT assets as service components, encouraging reuse and efficiency
- Enables integration developers to assemble complex business solutions -- processes, mediations, adapters, or code components -- requiring minimal skills
- Enables construction of process and integration solutions using drag-and-drop technology without having a working knowledge of Java
- Enables rapid assembly of business solutions by wiring reusable service components

- Integrates testing, debugging, and deployment for solution development
- Enables Business-Driven Development, fully integrating with WebSphere Business Modeler to import models for rapid implementation
- Improves reuse and efficiency with online modules and libraries
- Operating systems supported: Linux, Windows
- Service Component architecture (fig 19)

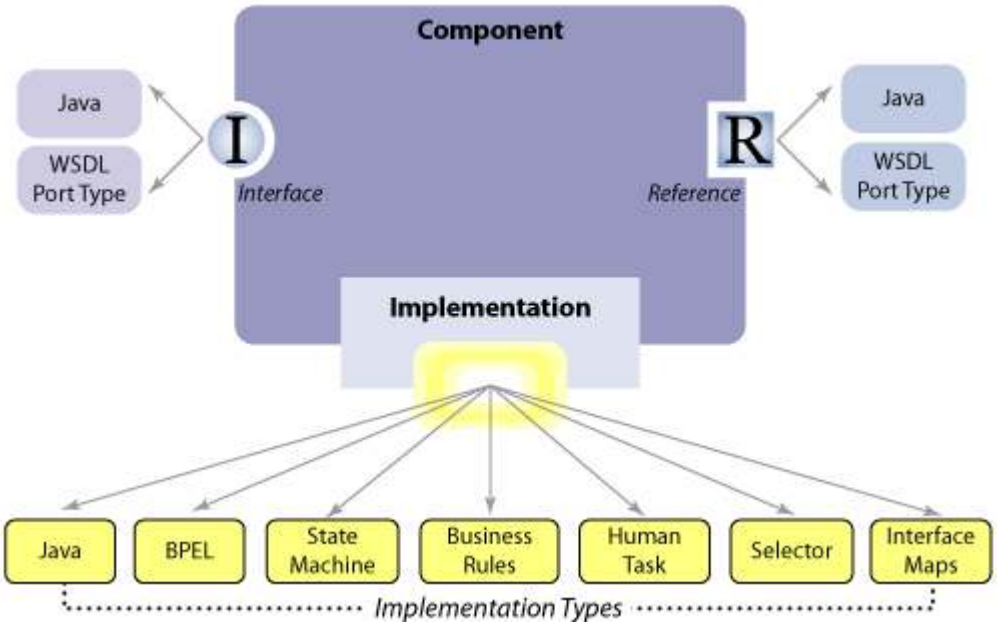


*Figure 19 Service component architecture*

# 9 Practice solution

In this section we will go through the primary three methods that are described above (CBM, SOMA and SOA). We will concentrate primary on SOA implementation part. For CBM and SOMA parts we will just describe what is the input/output from/to them.
Our practice solution will starts with typical SOA problem statement definition. During the problem statement definition we will define customer expectations, customer business needs and CBM map (section 4). After that we will proceed with process decomposition (sections 3.2.2 and 5). The main idea of process decomposition is to help us with list of candidate services. When we have a list with all candidate services we can proceed with service identification(section 6) where we will applied service litmus tests. When service identification is done we will continue with service specification. For service specification we will use Rational Software Architect and Rational Data Architect. When the services are specified we will develop them with tools that we was defined (section 8). All services, processes, components and database can be directly map to the layers from SOA solution stack (section 7).

## 9.1 Problem statement and CBM

The Good Insurance Company (GIC) is focusing on Claims Processing for this SOA based initiative. There were a few business sessions between senior consultant from IBM and business person from GIC, in order to capture information about the business context where GIS is interested. The following notes summarize discussions during those work sessions.

After these sessions the senior consultant from IBM identified several competencies (fig 20) for GIC. In this section we will concentrate on Claims Management Competency. This competency covers Claim Strategy, Claim Monitor and Claim Processing business components.

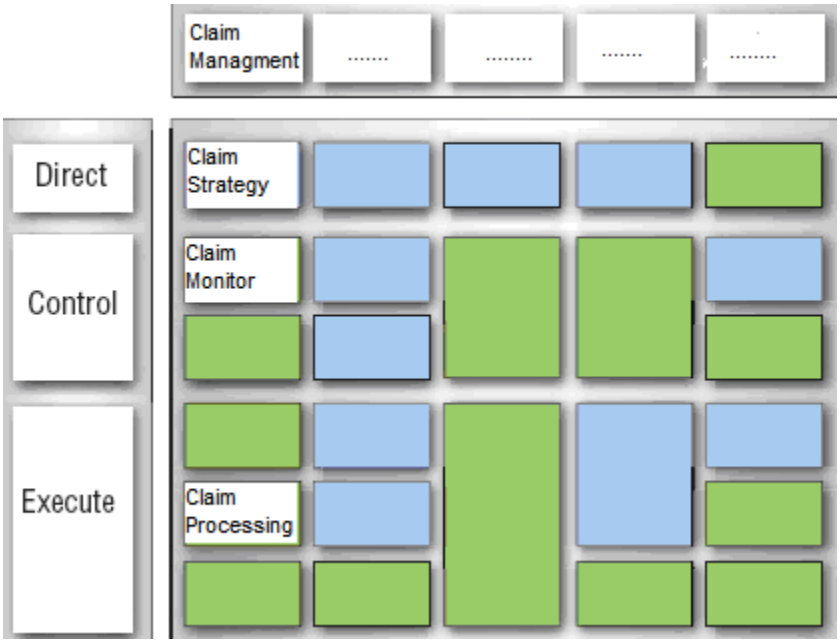


*Figure 20 Identified CBM map for Good Insurance Company*

The focus of the work sessions is the Claim Processing business component that is part of the Claims Management Competency.

| Level | Name | Description |
|---|---|---|
| Competency | Claims Management | Provide benefit to the claimant under the terms of the insurance agreement. The life-cycle of a claim ranges from the recognition of something that may result in a claim (an incident for pre-authorization or a requested service) to the final settlement with the client and, where applicable, the recovery of money from re-insurers or third parties. |
| Business Component | Claim Processing | Receive a claim and record the relevant information such as the claimant, the insurance agreement number, the loss claimed, and the incident to which the claim relates. This set of activities is usually considered to represent the opening of the claim. |

*Table 2 GIC CBM description Table*

A customer's policies provide coverage that covers various types of claims. Coverage for a claim is provided by lines of business such as auto and home. Many claims follow a traditional claims processing process, although some claims are now classified as express claims when they meet certain criteria such as type of damage, and can be processed more quickly.

Responsibilities of the Claim Processing business component include the administration of first notice of loss for auto claims, and receiving claim

notifications of various types. Claims are recorded as soon as a claim notification is received, along with claim details, and the “loss event” related to the claim. When claim processing has not completed by a set period of time, a customer can make a request for benefit prepayment. When that happens, a prepayment request is recorded in the system for the related claim. Some types of claims require special recording procedures such as “Life Death” claims, new auto and home claims.

The “Administer Claim” process begins when a claim notification is received and takes the claim through verification and analysis steps to its ultimate acceptance or rejection. Recording a claim involves recording claim details and a record of the loss event. Once a claim has been recorded, a complex verification process takes place, followed by a claim analysis process.

Verification involves accepting loss coverage, allocating an investigation, allocating the loss to the claimants coverage, analysis of claim history, creating reserves to cover the loss, determining investigation requirements, evaluating the value of the claim, identifying the specific policy that applies to the claim, retrieving claim history, reviewing the loss event, verifying coverage and verifying the policy.

Verification is followed by analysis of the claim. Analysis is also a complex process that involves allocating a claim cost code, analyzing injuries and loss circumstances, estimating the loss amount, determining if there is reason to consider the claim to be fraudulent, determining liability for loss or injury, evaluating damage, recording damage details, reviewing pertinent legislation, possibly initiating litigation related to the claim and recording the results of the investigation.
Following verification and analysis, a claim is either accepted or rejected.

## 9.2 Solution Overview

In this scenario we will consider the following instructions:

- Review the problem statement and CBM.
- Create a Domain Decomposition diagram that subdivides the claims management domain into the functional areas, and identify a list of functions associated with the functional area(s).
- Based on the narrative in the background information that senior consultant collect from GIC, appropriate business person from IBM will decompose “Administer Claim” process.
- Based on “Administer Claim” process decomposition architect from IBM will create domain model.
- Based on “Administer Claim” process decomposition architect from IBM will create a list of identified services.
- Specify identified services for “Administer Claim” process

- Implementation of specified services

## 9.3 SOMA

When the business consultant collect all the information that he need he starts with customer business analysis. For “Claim Processing” the architect decides to use Domain Decomposition (fig 21) which is a top down approach that analyzes a high level business view of the enterprise to identify candidate services and flows.

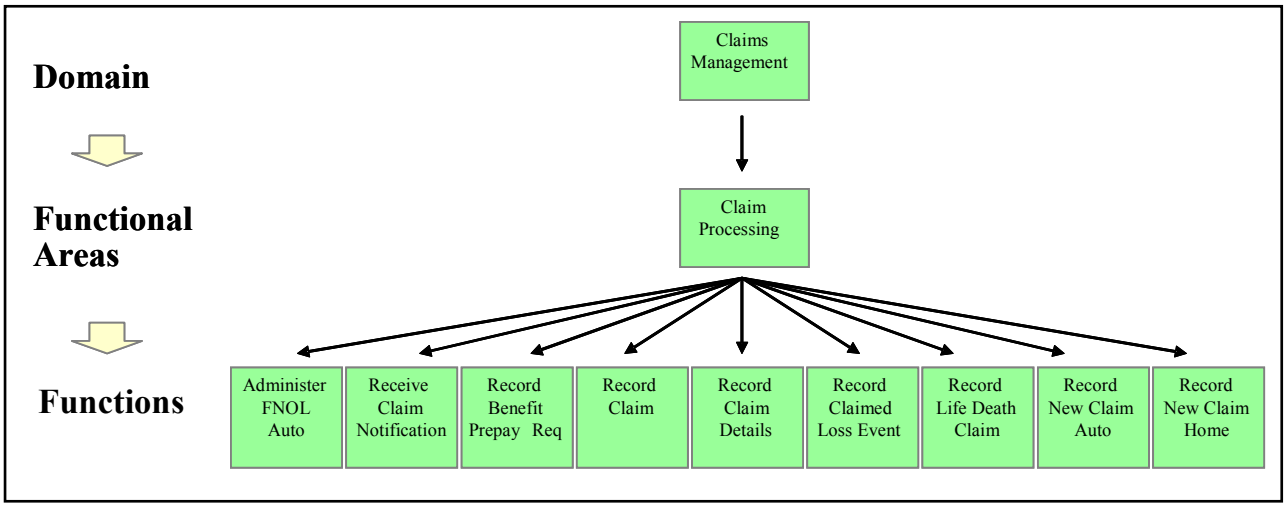


*Figure 21 Domain Decomposition Diagram*

The table below describes each function that belongs to Claim Processing business component:

| **Domain:** Claims Management<br>**Functional Area:** Claims Processing | |
|---|---|
| **Functions** | **Description** |
| **Administer FNOL Auto** | Customer party desires to report damage on insured car (first notice of loss information) to the financial institute. Customer has an insured car which has received damage to the windshield. Customer reports the damage on the financial services provider’s internet website in order to open a claim. Customer enters the information about the loss and about the auto policy. The information is stored in the financial institute’s system for further processing and the customer receives a loss reference number. |
| **Receive Claim Notification** | A claim notification is received and recorded. |
| **Record Benefit** | Records the request to receive a prepayment of the |

| **Domain:** Claims Management<br>**Functional Area:** Claims Processing | |
|---|---|
| **Functions** | **Description** |
| **Prepayment Request** | benefit. In the case where after a preset period from claim notification, the claim handling process has not been completed, then the claimant can be allowed to request a prepayment of part of or complete amount of the claimed benefit. A benefit prepayment is considered as an advance payment to the benefit payment. This activity is a variation of record claim notification with a reference to the claim for which the prepayment is requested. |
| **Record Claim** | Record all elements about a claim by a party on an insurance company. The claim description may be received by various intermediary (for example, agent or broker) or directly by the company (for example, a letter, fax, or phone call). In some cases the intermediary can ask the claimant to provide additional information in the claim description before introducing the claim to the insurance company. This process description considers that the process starts when the company receives the claim.<br>Process outcome is either a “recorded claim” - after this step the claim must be recorded or “request for additional information” - if the information is incomplete a request for additional information is issued. |
| **Record Claim Details** | Record the details about a claim by a party against an insurance company. This includes recording the claimant’s details and claim circumstances, and checking any references to the agreements to ensure the basic information necessary for the claim to proceed is present. This also includes recording the details of the claimed conditions, recording the details of the activities, recording the details of the benefit that is requested by the claimant, recording of the details reported via a medical bill; it registers the claimed medical treatment and its cost as represented on the bill. |
| **Record Claimed Loss Event** | Record the details of the claimed loss event. |
| **Record Life Death Claim** | A beneficiary reports the death of one of the insured’s covered by a life insurance policy. |
| **Record New Claim Auto** | The customer (insured) has an insured auto which has received damage to the windshield. He/she reports the damage to the insurance company to open a claim. |
| **Record New** | The customer (insured) has an insured home which |

| **Domain:** Claims Management<br>**Functional Area:** Claims Processing | |
|---|---|
| **Functions** | **Description** |
| **Claim Homeowners** | has suffered hail damage to the roof of the home. He/she reports the damage to the insurance company to open a claim. |

*Table 3 Claim processing description*

## 9.3.1 Process decomposition

Process decomposition (fig 22) is a very important part of Service Oriented Modeling and Architecture. Its primary role is to prepare and decompose business processes (base of function that are described into previous section) three of four layers down for each business component. There are zero or more processes that will be created for each function.

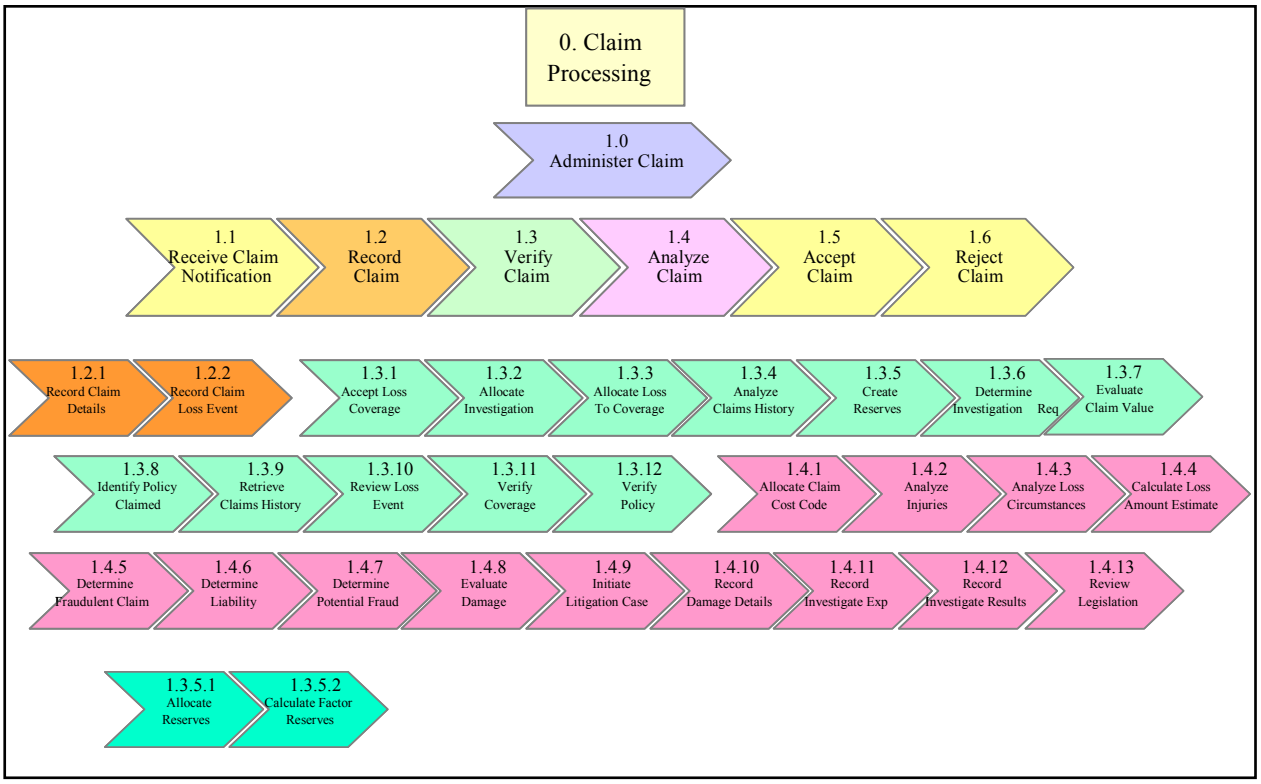


*Figure 22 Process decomposition of claim processing – administer claim*

## 9.3.2 Service identification

Based on decomposed processes the key output of the Service Identification step is a set of candidate business services. In our scenario the following list of candidate services (service portfolio) was identified (fig 23):

1 Administer Claim
1.1 Receive Claim Notification
1.2 Record Claim
1.3 Verify Claim
1.4 Analyze Claim
1.5 Accept Claim
1.6 Reject Claim
1.2.1 Record Claim Details
1.2.2 Record Claim Loss Event
1.3.1 Accept Loss Coverage
1.3.2 Allocate Investigation
1.3.3 Allocate Loss to Coverage
1.3.4 Analyze Claims History
1.3.5 Create Reserves
1.3.5.1 Allocate Reserves
1.3.5.2 Calculate Factored Reserves
1.3.6 Determine Investigation Requirements
1.3.7 Evaluate Claim Value
1.3.8 Identify Policy Claimed
1.3.9 Retrieve Claims History
1.3.10 Review Loss Event
1.3.11 Verify Coverage
1.3.12 Verify Policy
1.4.1 Allocate Claim Cost Code
1.4.2 Analyze Injuries
1.4.3 Analyze Loss Circumstances
1.4.4 Calculate Loss Amount Estimate
1.4.5 Determine Fraudulent Claim
1.4.6 Determine Liability
1.4.7 Determine Potential Fraud
1.4.8 Evaluate Damage
1.4.9 Initiate Litigation Case
1.4.10 Record Damage Details
1.4.11 Record Investigation Expense
1.4.12 Record Investigation Results
1.4.13 Review Legislation

*Figure 23 Service Portfolio after Process Decomposition*

Initially one functional area was identified for the Claims Management domain. When this single functional area was applied to the service portfolio it became evident that one functional area was too broad and needed to be split into additional functional areas:

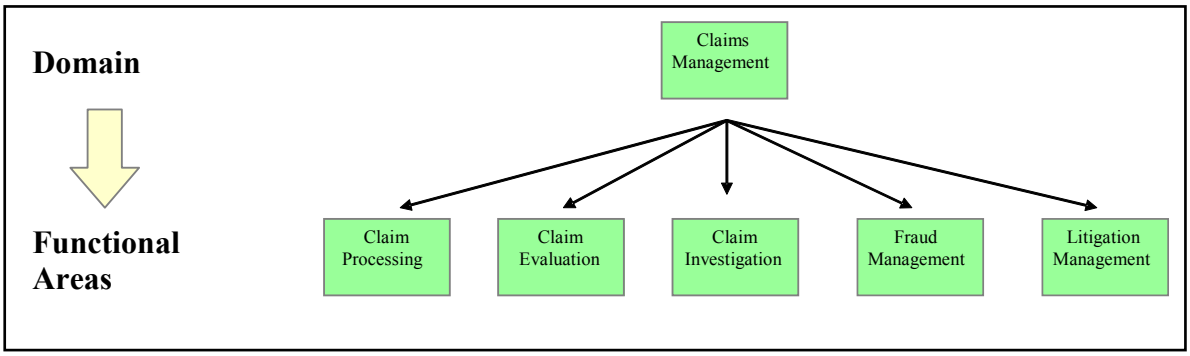


*Figure 24 Refactored Functional Areas*

This is reflected in the service hierarchy shown below (fig 25):

**Claims Processing**

- 1 Administer Claim
- 1.1 Receive Claim Notification
- 1.1.1 Receive Express Claim Notification
- 1.1.2 Receive Traditional Claim Notification
- 1.1.3 Receive Auto Claim Notification
- 1.1.4 Receive Home Claim Notification
- 1.2 Record Claim
- 1.5 Accept Claim
- 1.6 Reject Claim
- 1.2.1 Record Claim Details
- 1.2.2 Record Claim Loss Event
- 1.3.5 Create Reserves
- 1.3.5.1 Allocate Reserves
- 1.3.5.2 Calculate Factored Reserves
- 1.3.8 Identify Policy Claimed
- 1.3.9 Retrieve Claims History

**Claims Evaluation**

- 1.3 Verify Claim
- 1.4 Analyze Claim
- 1.3.1 Accept Loss Coverage
- 1.3.3 Allocate Loss to Coverage
- 1.3.4 Analyze Claims History
- 1.3.7 Evaluate Claim Value
- 1.3.10 Review Loss Event
- 1.3.11 Verify Coverage
- 1.3.12 Verify Policy
- 1.4.2 Analyze Injuries
- 1.4.3 Analyze Loss Circumstances
- 1.4.8 Evaluate Damage

**Claims Investigation**

- 1.3.2 Allocate Investigation
- 1.3.6 Determine Investigation Requirements
- 1.4.11 Record Investigation Expense
- 1.4.12 Record Investigation Results

**Fraud Management**

- 1.4.5 Determine Fraudulent Claim
- 1.4.7 Determine Potential Fraud

**Litigation Management**

- 1.4.6 Determine Liability
- 1.4.9 Initiate Litigation Case
- 1.4.13 Review Legislation

*Figure 25 Service hierarchy*

After identification of candidate services we need to apply the Service Litmus Test to determine which candidate services are appropriate to expose. The Service Litmus Test is a defined set of criteria to resolve whether a candidate service should be exposed. The criteria fall into four major areas:

- **Business alignment SLT1**: Focusing on business relevance of the service, the presence of a funding model to support development and maintenance, and the ability to share the service across the organization.
- **Composability SLT2**: Focusing on consistency with non-functional requirements at the composite level, consideration of state management aspects, identifying service dependencies, and supporting technology/platform neutrality.
- **Externalized service description SLT3**: Focusing on the presence of an external service description (such as WSDL), the ability to support service

discovery and binding via the service description, and providing meta data as part of the service description.

- **Redundancy elimination SLT4**: Focusing on the ability to reuse the candidate service across multiple composite scenarios where the specific function is needed.

Through this set of questions and optional extensions and customizations, as appropriate for the specific organization, the architect can make appropriate architectural decisions regarding which services should be developed, exposed, and managed as service implementations.
When the architect applies and restructure all services the following list of identified services was prepared (tab 4):

| Service | SLT1 | SLT2 | SLT3 | SLT1 | Expose |
|---|---|---|---|---|---|
| 1. Submit claim notification | Pass | Pass | Pass | Pass | Yes |
| 2. Register Claim | Pass | Pass | Pass | Pass | Yes |
| 3. Verify Claim | Pass | Pass | Pass | Pass | Yes |
| 4. Analyze Claim | Pass | Pass | Pass | Pass | Yes |
| 5. Register Customer | Pass | Pass | Pass | Pass | Yes |

*Table 4 Litmus test table*

### 9.3.3 Service specification

The definition of the service specification consists of multiple steps and normally results in the creation of UML work products. This step is facilitated through the use of architecture tooling such as the Rational Software Architect and the UML Profile for Software Services. During this step, the service model is designed through documenting services and their operations, defining service composition, documenting the service non-functional aspects, defining the high-level service message model, and specifying state management requirements.

#### 9.3.3.1Domain model

A domain model can be thought of as a conceptual model of a system which describes the various entities involved in that system and their relationships. The domain model is created to document the key concepts and the vocabulary of the system. The model displays the relationships among all major entities within the system and usually identifies their important methods and attributes. This means that the model provides a structural view of the system which is normally complemented by the dynamic views in Use Case models. An important benefit of a domain model is to describe and constrain system scope.
The domain model can be used at a low level in the software development cycle since the semantics shown therein can be used in the source code. Entities become classes, while methods and attributes can be carried directly to the source code; the same names typically appear in the source code relationship.
In our scenario (for Good Insurance Company) the following domain model (fig 26) was specified into Rational Software Architect v7.0:

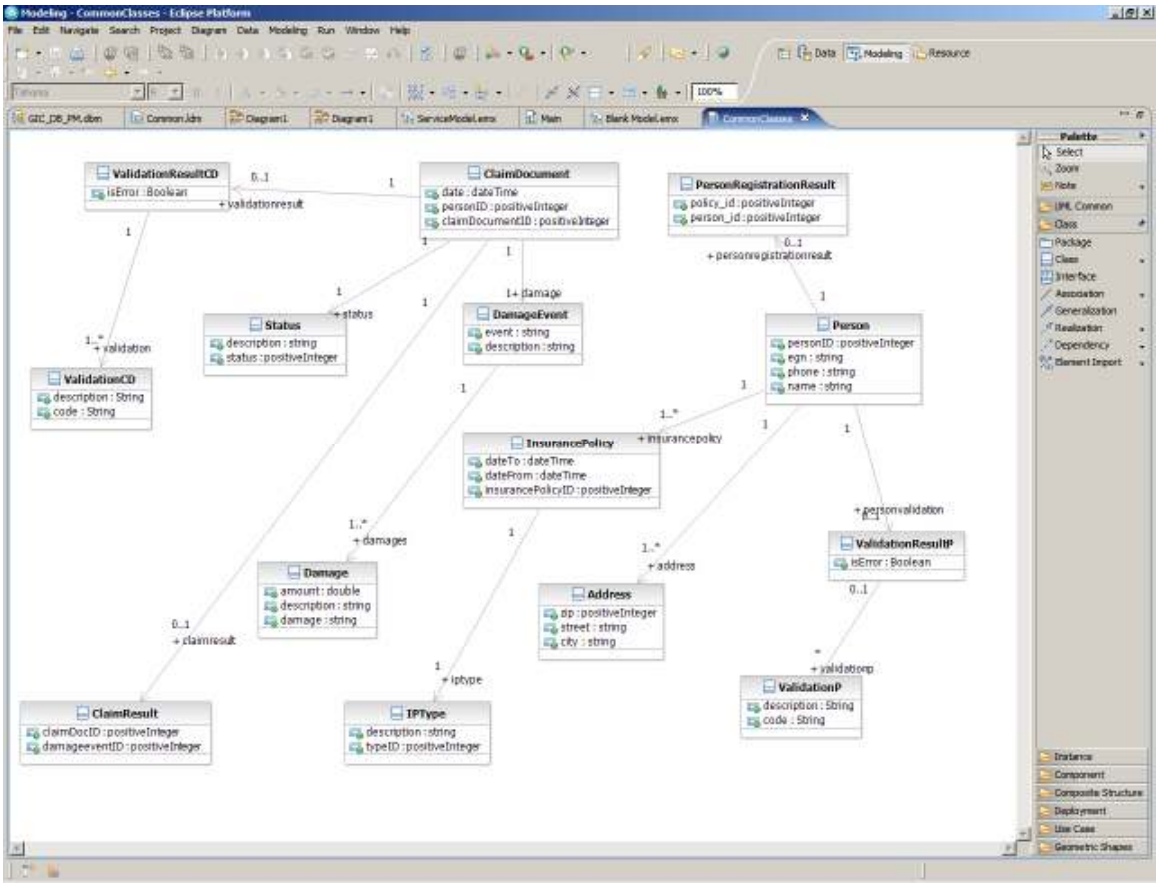


*Figure 26 Domain model diagram*

- ClaimDocument – This complex business object (BO) provides all necessary data that is needed for Administer Claim process.
- Status – This BO holds the current status of ClaimDocument.
- DemageEvent – This complex BO contains the information about damage event that was appearing.
- Damage – It has a list with damages that was happened thought damage event.
- ClaimResult – It contains all IDs that has been generated during Claim Registration
- ValidationResultCD – It contains all errors that may appear during execution of ClaimValidation service.
- ValidationCD – It holds each validation data that can appear.
- Person – It contain human personal information.
- InsurancePolicy – This complex BO has information about insurance policy.
- PersonRegisrtationResult – It contains all IDs that will be generated during person registration service
- IPType – It contains information about insurance policy type (Auto and Homeowners).

- Address – It holds the address data for Person BO.

- ValidationResultP – It contains list with errors that appear during validation process.
- ValidationP – It contains code and description if there is an error during validation process

### 9.3.3.2Components & Interface Specification

This layer contains software components, each of which provide the implementation for, realization of, or operation on a service, which is why it's called a service component. Service components reflect the definition of a service, both in its functionality and its quality of service.
When the architect goes through the different use cases which describe Administer Claim business process he identified few components and their services. All components and services are described below.

#### 9.3.3.2.1 User Component

This component (fig 27) has one Registration service. It contains one operation registerUser. This operation takes as input Person BO and returns as output operation state (successful/unsuccessful).

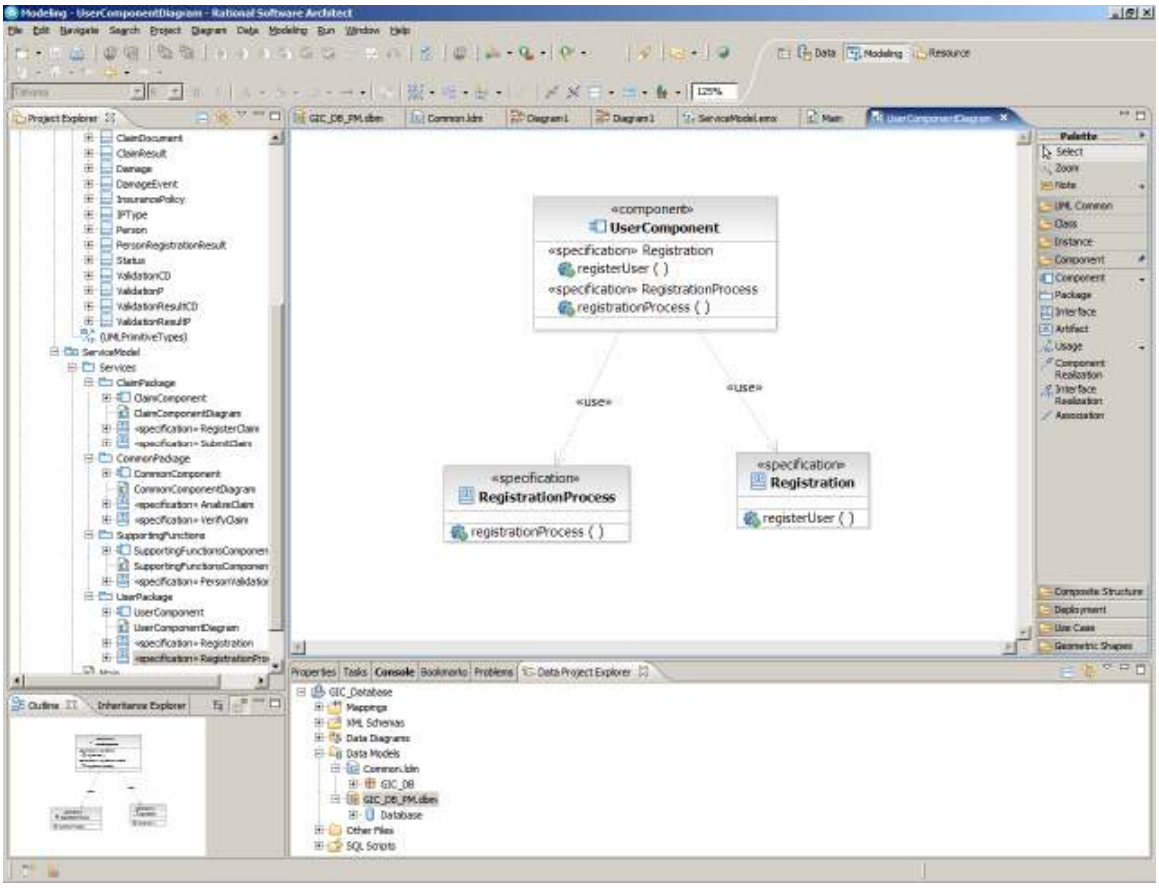

*Figure 27 User component diagram*

#### 9.3.3.2.2 Common Component

This component has two services (fig 28). The first one is AnalizeClaim. It contains one operation analizeClaimDocument. This operation takes as input ClaimDocument BO and returns as output ClaimDocument. The second service is VerifyClaim. It has one operation verifyClaimDocument. This operation takes ClaimDocument as input and returns output operation state (successful/unsuccessful).

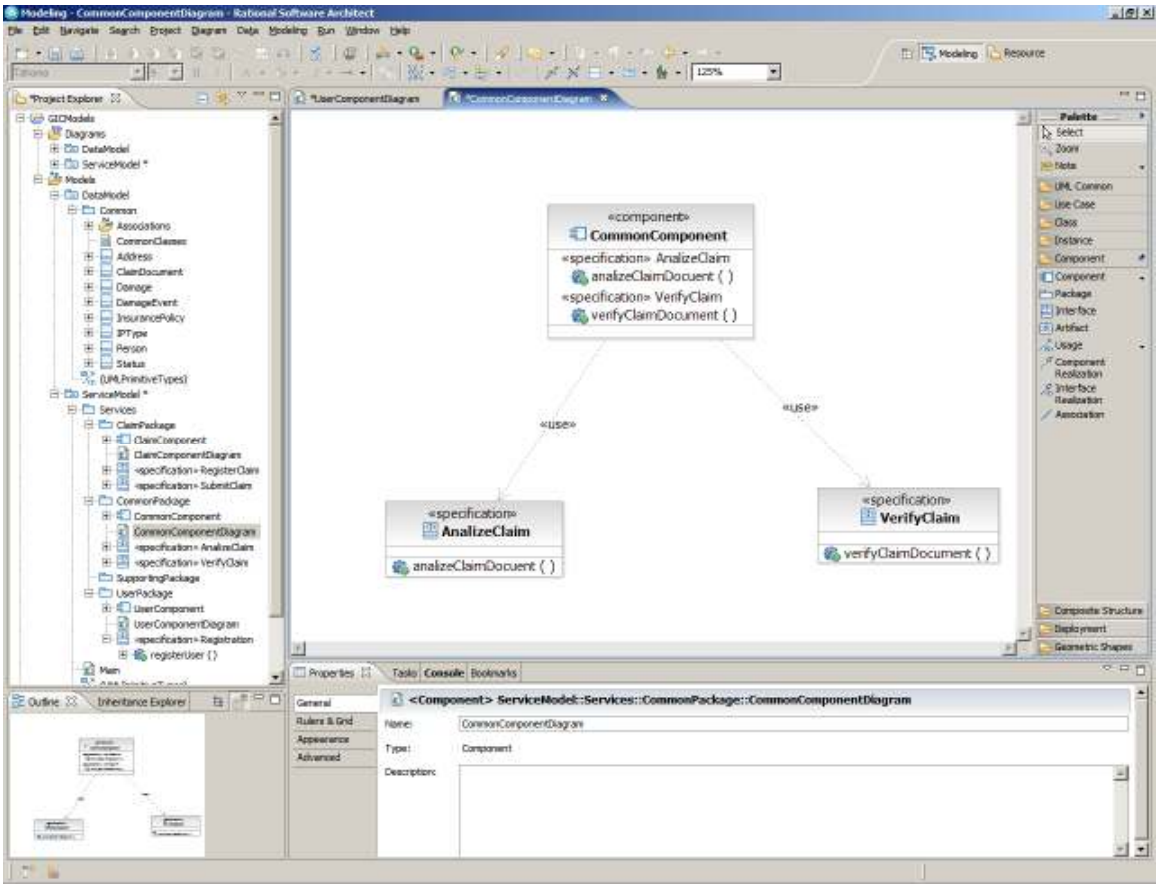


*Figure 28 Common component diagrams*

#### 9.3.3.2.3 Claim Component

This component has two services (fig 29). The first one is RegisterClaim. It contains one operation registerClaimDocument. This operation takes as input ClaimDocument BO and returns as output ClaimDocument. The second service is SubmitClaim. It has one operation submitClaimNotification. This operation takes ClaimDocument as input and returns as output ClaimDocument.

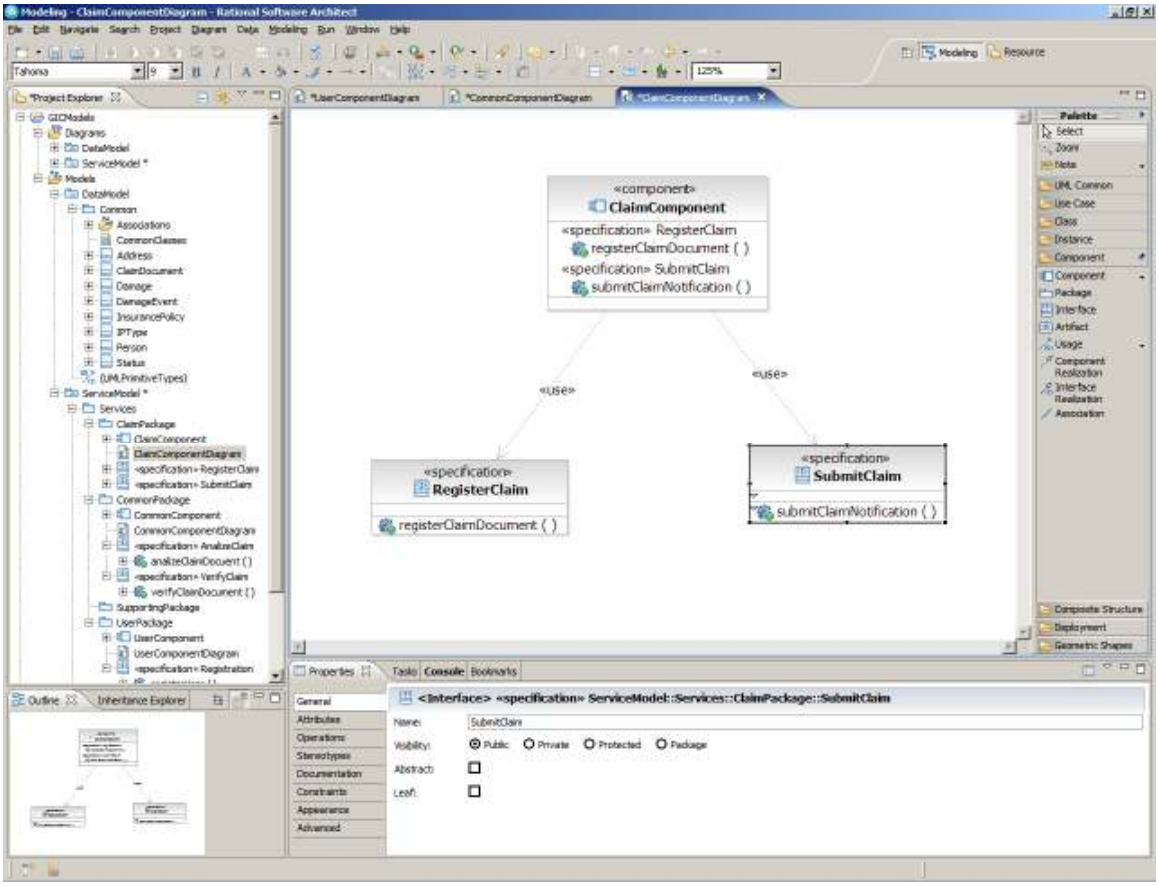


*Figure 29 Claim component diagram*

#### 9.3.3.2.4 Supporting Functions Component

This component has one PersonValidation service (fig 30). It contains one operation - isEGNValid. This operation takes as input personal number and checks whether is valid or not. It returns as output state (successful/unsuccessful).

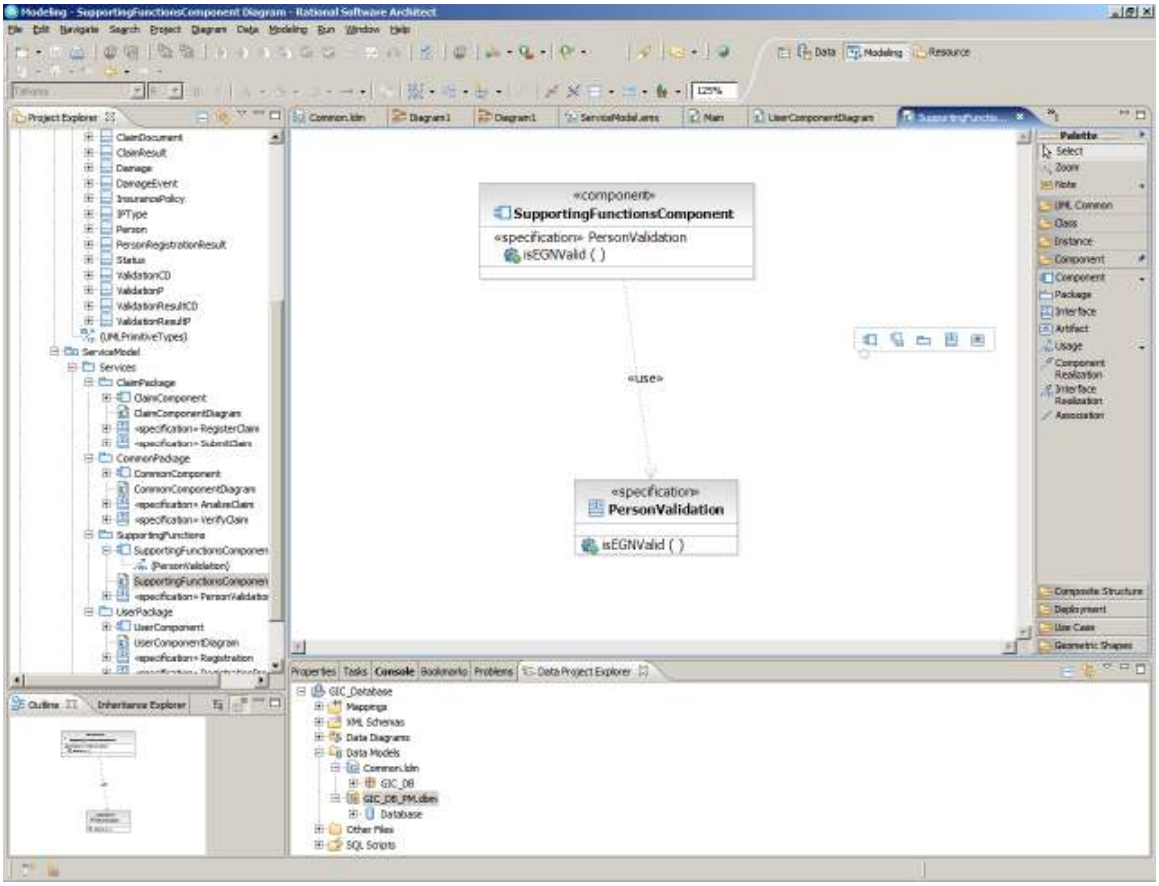


*Figure 30 Supporting function component diagram*

### 9.3.3.3 Database

This scenario use database as persistency layer in order to store all needed information during execution of the different business processes. The logical database was generated with Rational Data Architect from existing classes that was defined above. The logical model was transformed into physical data model by appropriate transformation. The physical database diagram (fig 31) illustrates the database.

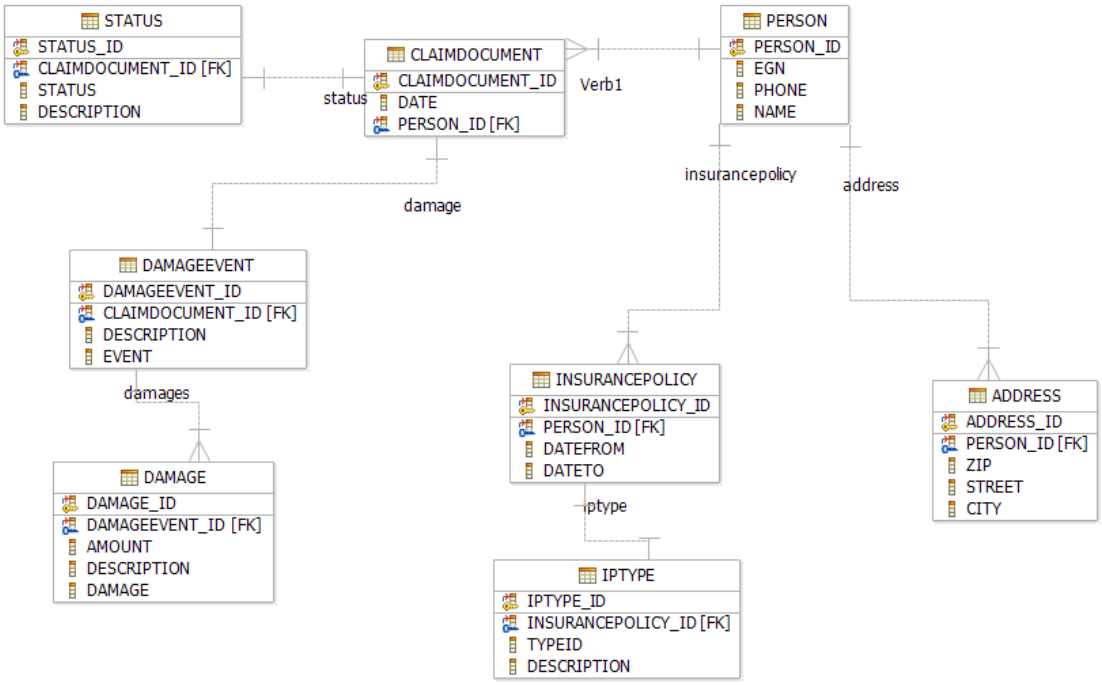


*Figure 31 Physical database model*

## 9.4 Service components (SCA)

Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture. SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services.

SCA encourages an SOA organization of business application code based on components that implement business logic, which offer their capabilities through service-oriented interfaces and which consume functions offered by other components through service-oriented interfaces, called service references. SCA divides up the steps in building a service-oriented application into two major parts:

- The implementation of service components which provide services and consume other services.
- The assembly of sets of components to build business applications, through the wiring of service references to services.

SCA emphasizes the decoupling of service implementation and of service assembly from the details of infrastructure capabilities and from the details of the access methods used to invoke services. SCA components operate at a business level and use a minimum of middleware APIs.

SCA supports service implementations written using any one of many programming languages, both including conventional object-oriented and procedural languages such as Java™, PHP, C++, COBOL, XML-centric languages such as BPEL and XSLT, and also declarative languages such as SQL and XQuery. SCA also supports a range of programming styles, including asynchronous and message-oriented styles, in addition to the synchronous call-and-return style.

SCA supports bindings to a wide range of access mechanisms used to invoke services. These include Web services, Messaging systems and CORBA IIOP. Bindings are handled declaratively and are independent of the implementation code. Infrastructure capabilities, such as Security, Transactions and the use of Reliable Messaging are also handled declaratively and are separated from the implementation code. SCA defines the usage of infrastructure capabilities through the use of Policies, which are designed to simplify the mechanism by which the capabilities are applied to business systems.

SCA also promotes the use of Service Data Objects to represent the business data that forms the parameters and return values of services, providing uniform access to business data to complement the uniform access to business services offered by SCA itself.

The SCA specification is divided into a number of documents, each dealing with a different aspect of SCA. The Assembly Model deals with the linking of components through wiring. The Assembly Model is independent of implementation language. The Client and Implementation specification deals with the implementation of services and of service clients -- each implementation language has its own Client and Implementation specification, which describes the SCA model for that language.

### 9.4.1 Business Processes

Two business processes was identified for the scope of this scenario:

- Administer Claim (fig 32).

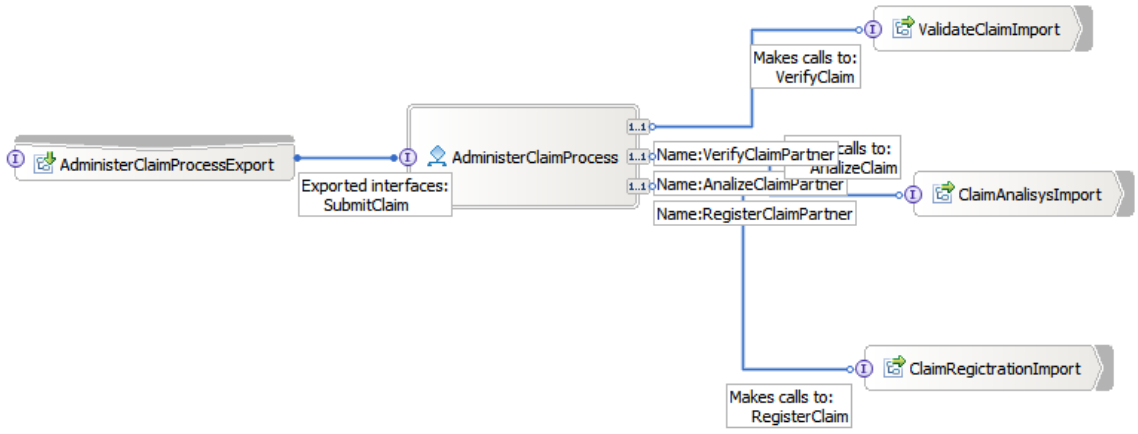


*Figure 32 Administer claim business process*

- Person Registration (fig 33)

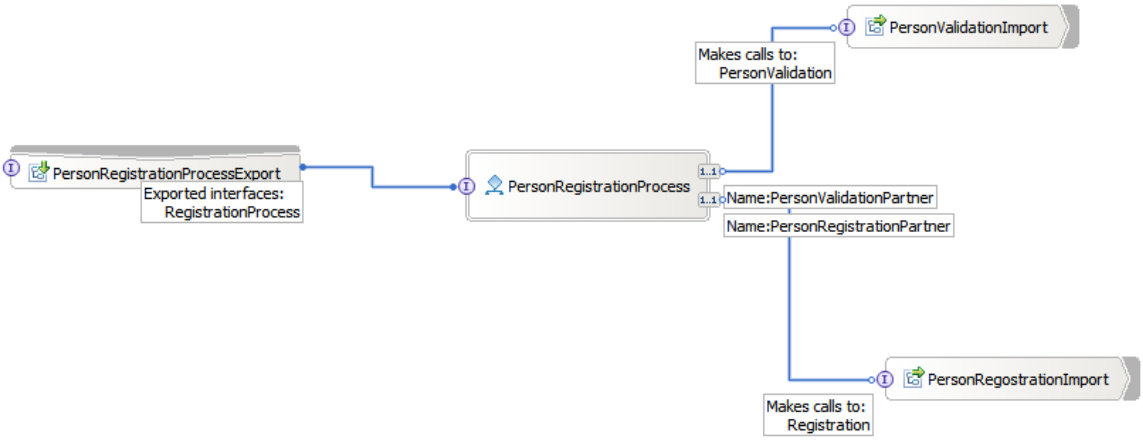


*Figure 33 Person registration business process*

## 9.4.2 Person Registration

Person registration business process has two main steps. The first one is person validation and the second one is person registration. If there is validation error during its first step, the process returns the list XML keys and their error description. The following figure (fig 34) illustrates the main logic of the process.

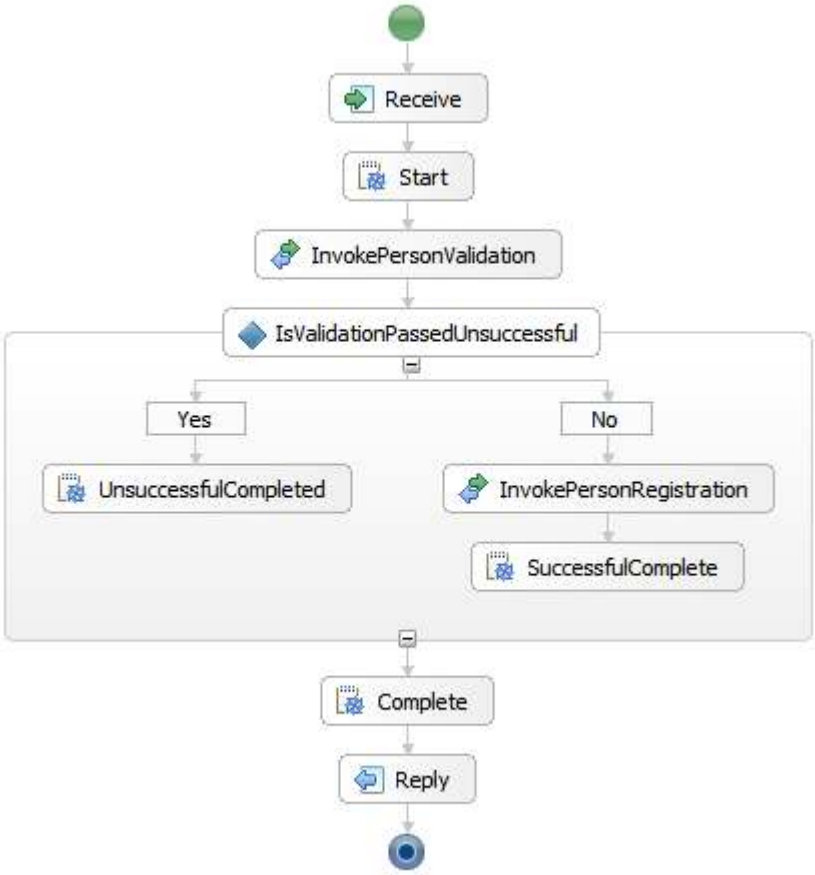


*Figure 34 Person registration logic*

### 9.4.3 Administer Claim

Administer claim business process has three main steps. The first one is *"claim validation".* If there is validation error during its first step, the process returns the list XML keys and their error description. The second step is "claim analysis" where analyze service evaluate all the damages that will be passed with the claim. The following figure (fig 35) illustrates the main logic of the process.

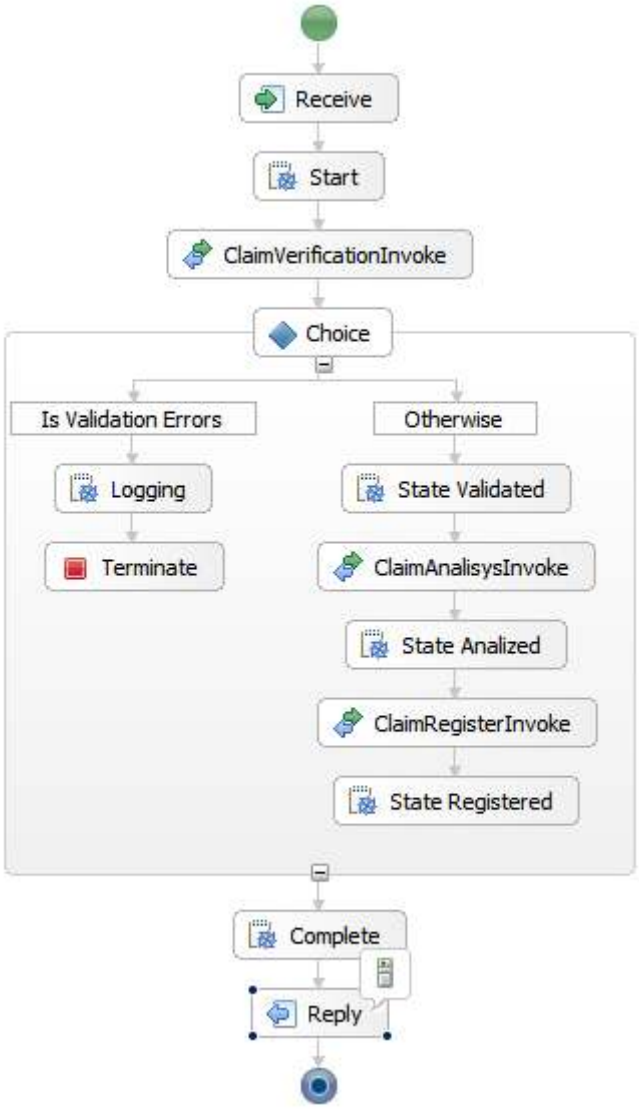


*Figure 35 Administer claim logic*

### 9.4.4 Claim ragistration

For the purpose of this scenario all five services was implemented into Websphere Integration Developer as Service Component Architecture (SCA) components with java implementation (fig 36).

*Figure 36 Claim registration component*

The following was implemented into ClaimRegistration SCA component:

```
public DataObject registerClaimDocument(DataObject inClaimDocument) {

DB2Connector connector = null;
try
{
	//Save claim document data

	connector = new DB2Connector();
	connector.createConnection();
	int claimDocumentID = connector.saveClaimDocument (inClaimDocument.getDate
	("date") ,inClaimDocument.getInt("personID"));
	DataObject status = inClaimDocument.getDataObject("status");
	int status_id = connector.saveStatus(claimDocumentID,status.getInt("status"),
	status.getString("description"));
	DataObject damageEvent = inClaimDocument.getDataObject("damage");
	int damageEventID = connector.saveDamageEvent(claimDocumentID,
	damageEvent.getString("description"),damageEvent.getString("event"));
	List damagesList = (List)damageEvent.get("damages");

	//Create ClaimResult BO
	ServiceManager serviceManager = new ServiceManager();
	BOFactory bof = (BOFactory)serviceManager.locateService
	("com/ibm/websphere/bo/BOFactory");
	DataObject claimResult = bof.create("http://Common/", "ClaimResult");
	claimResult.setInt("claimDocID",claimDocumentID);
	claimResult.setInt("damageeventID",damageEventID);
	int damageID = 0;
	for(int i=0;i<damagesList.size();i++)
	{
		DataObject damage = (DataObject)damagesList.get(i);
		damageID = connector.saveDamages (damageEventID,
		damage.getDouble("amount"),
		damage.getString("description"),damage.getString("damage"));

	}
	//Set updated values to ClaimResult BO
	inClaimDocument.setDataObject("claimresult",claimResult);
	inClaimDocument.setInt("claimDocumentID",claimDocumentID);
}
catch(Exception ex)
{
	ex.printStackTrace();
}
finally
{
	if(connector != null)
		connector.closeConnection();
}
return inClaimDocument;
}
```

# 10 Conclusion

In this Master thesis will explore an SOA architecture, which provides roadmaps and guidelines for architectural, design, and implementation decisions. Additionally, it provides patterns and insights for integrating these aspects. As part of that reference architecture, reusable assets are being created to enable end-to-end, SOA-based business solutions that cover enterprise modeling, business process modeling, service modeling, as well as integration and management of business applications.

## 10.1 SOA solutions overview

Project-based SOA solutions (fig 37) typically result from a bottom-up, technical focus, providing an entry point for SOA design and development, but offering minimal benefits from an enterprise architecture standpoint.
Designing SOA solutions with a business focus links business requirements and the IT development process at the enterprise level. Defining SOA as the key enabling architecture provides the foundation platform for enterprise solution development. This realization of SOA as a core enterprise solution approach lets requirements be defined and scoped based on the core business competencies of the organization. With these business and IT requirements as input, SOA solutions can be optimally designed through service development methodologies such as SOMA. This approach to designing SOA solutions can provide for an enterprise-level view of services, and offers the ability to decrease time-to-market for new applications, while reducing IT resource exposure through service reuse. More importantly, the design of SOA solutions with a business focus ensures the relevancy and the value of SOA to the organization.

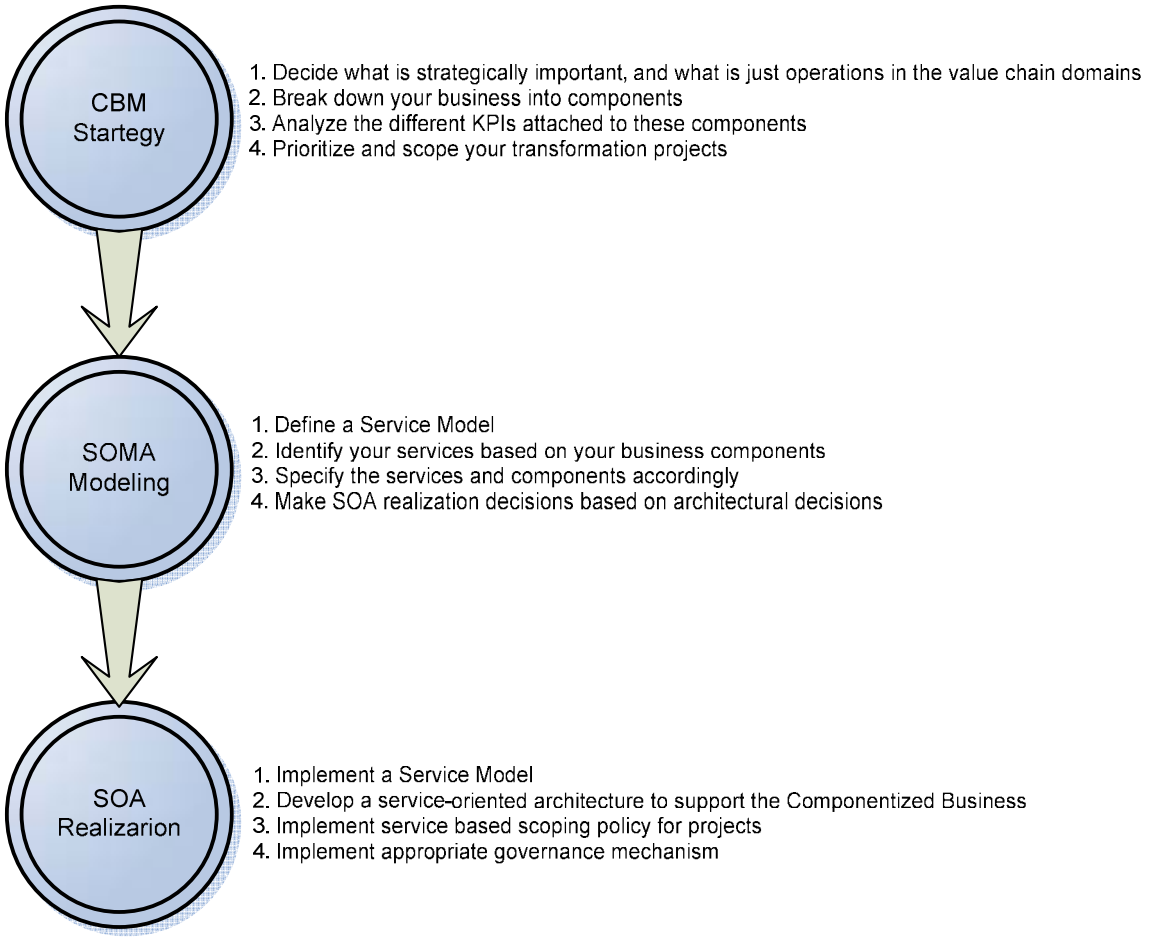


*Figure 37 SOA approach*

## 10.2 Possible feature work

There section covers some feature extension that was identified from the author of this master thesis:

- The goal of the SOA solution stack is to provide templates and guidelines to help architects facilitate and automate the process of modeling and documenting the architectural layers, building blocks, options, product mappings, and architectural and design decisions that contribute to the creation of an SOA.
- There is no defined pattern about service identification for SOMA. Let's assume that we have candidate system that needs to be analyzed (existing assets analysis) and this system has standard three layers architecture.
- There is no way to generate flat WSDL file. The tools need to be more flexible. Some tools have such kind of constraints. They can only work with flatten WSDL files.
- Ability to connect to COM components in order to exchange data with them.

- There is no tool where you can model CBM, SOMA and SOA. It will be better all Business Analysis, Architects and Developer to work together.
- There is no way to generate screens from domain model. It will be better if developer can generate screens based on XSD schemas. If we have that all constraint that are part from XSD schemas will reflect to the screans.
- There is no way to perform reverse engineering between CBM, SOMA, SOA services.

# Abbreviations

[1] CBM – Component Business Model
[2] SOMA – Service Oriented Modeling and Architecture
[3] SOA – Service Oriented Architecture
[4] SCA – Service Component Architecture
[5] COM – Component Object Model
[6] WSDL - Web Services Description Language
[7] XSD - XML Schema Definition
[8] XML - Extensible Markup Language
[9] JMS – Java Messaging Service
[10] PHP - PHP Hypertext Preprocessor
[11] SQL - Structured Query Language
[12] XQuery - XML Query Language
[13] BO – Business Object
[14] ESB – Enterprise Service Bus
[15] UDDI - Universal Description, Discovery & Integration
[16] WBI - WebSphere Business Integration
[17] JSF – Java server Faces
[18] BPEL - Business Process Execution Language
[19] KPI – Keep Performance Indicator
[20] NFRs - Nonfunctional Requirements
[21] HTML – Hyper Text Markup Language
[22] WSRP - Web Services for Remote Portlets
[23] B2B – Business to Business
[24] SLAs - Service-Level Agreements
[25] ABB - Architectural Building Blocks
[26] API – Application Interface
[27] RUP – Rational Unified Process
[28] SOAP – simple Object Access Protocol
[29] UML Unified Modeling Language
[30] RPC - Remote Procedure Call
[31] DCOM - Distributed Component Object Model

# References

Books:

[1] Rational Unified Process 7.1

[2] Designing SOA with a business focus, Scott Simmons, 2007

[3] Components Business Model, 2005

[4] Model-Driven Development of SOA Services, Christian Emig, Karsten Krutz, Stefan Link, Christof Momm, Sebastian Abeck

[5] Analysis and Design Techniques for Service-Oriented Development and Integration, Olaf Zimmermann, Niklas Schlimm, Günter Waller, Marc Pestel

Web resources:

[6] http://en.wikipedia.org/wiki/Service-oriented_architecture

[7 ] http://www.ibm.com/developerworks/library/ar-archtemp/

[8] http://www.redbooks.ibm.com/redbooks/pdfs/sg247356.pdf

[9] http://www.ibm.com/developerworks/library/specification/ws-sca/

[10] http://www.idevnews.com/IntegrationNews.asp?ID=172

[11] http://www.tusc.com.au/utilities/solution_details.php?id=28

[12] www.ibm.com/software/integration/wps/

[13] www.oasis-open.org/committees/uddi-spec/doc/tns.htm

[14] www.ibm.com/software/integration/wsrr/library/faqs.html

[15] www.ibm.com/software/integration/esb

[16] www.ibm.com/software/integration/wbimessagebroker/

[17] www.ibm.com/software/integration/wid/

[18] www.ibm.com/software/awdtools/architect/swarchitect/

[19] http://www-306.ibm.com/software/data/integration/rda/

[20] www.ibm.com/db2

# Appendix A – Run Administer Claim Process

This appendix illustrates the execution of Administer Claim business process. The picture bellow (fig 38) illustrates the appropriate way how to test our business processes into Websphere Integration Developer (WID). To do that we need to starts WID integrated test client. After that into right side we should specify the Module, Component, Interface and Operation that we would like to test.

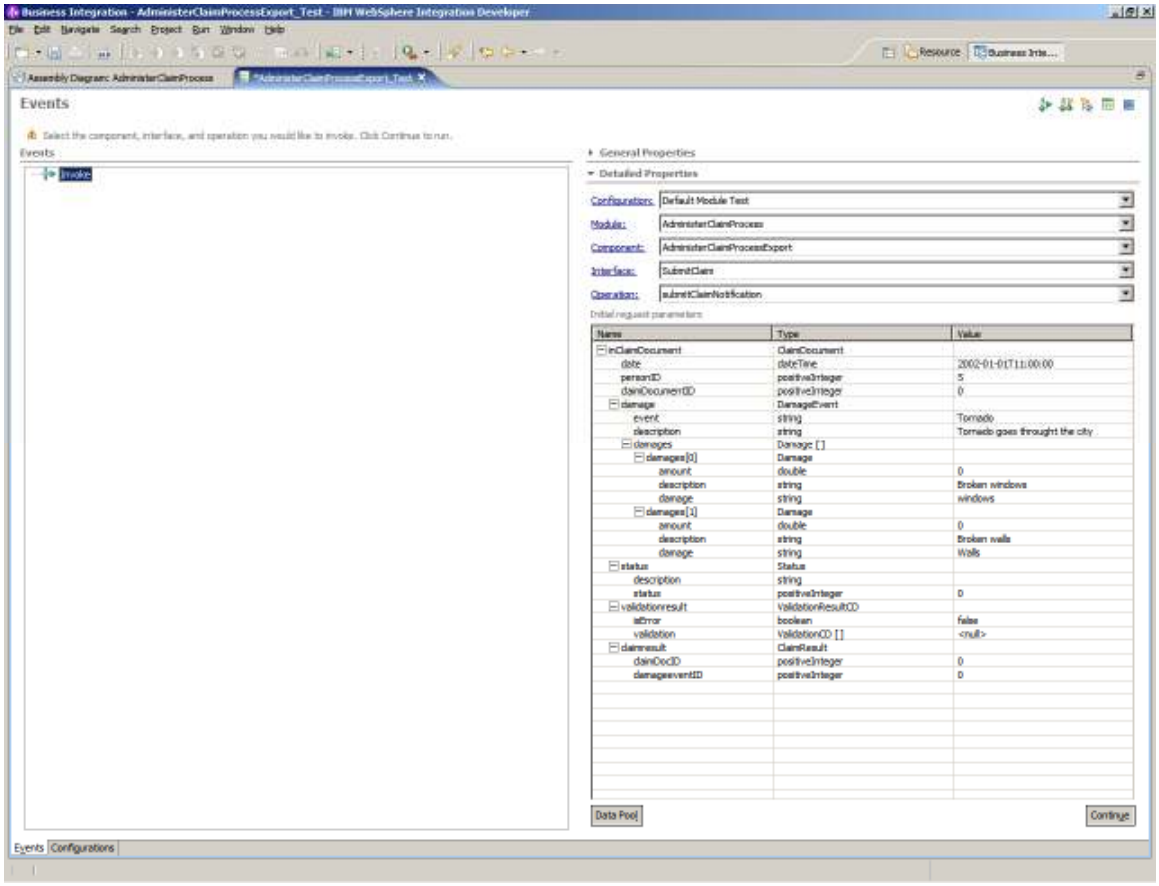


*Figure 38 Integrated test client request screen*

When we enter the required request data for Administer Claim business process, the following result screen appear (fig 39):

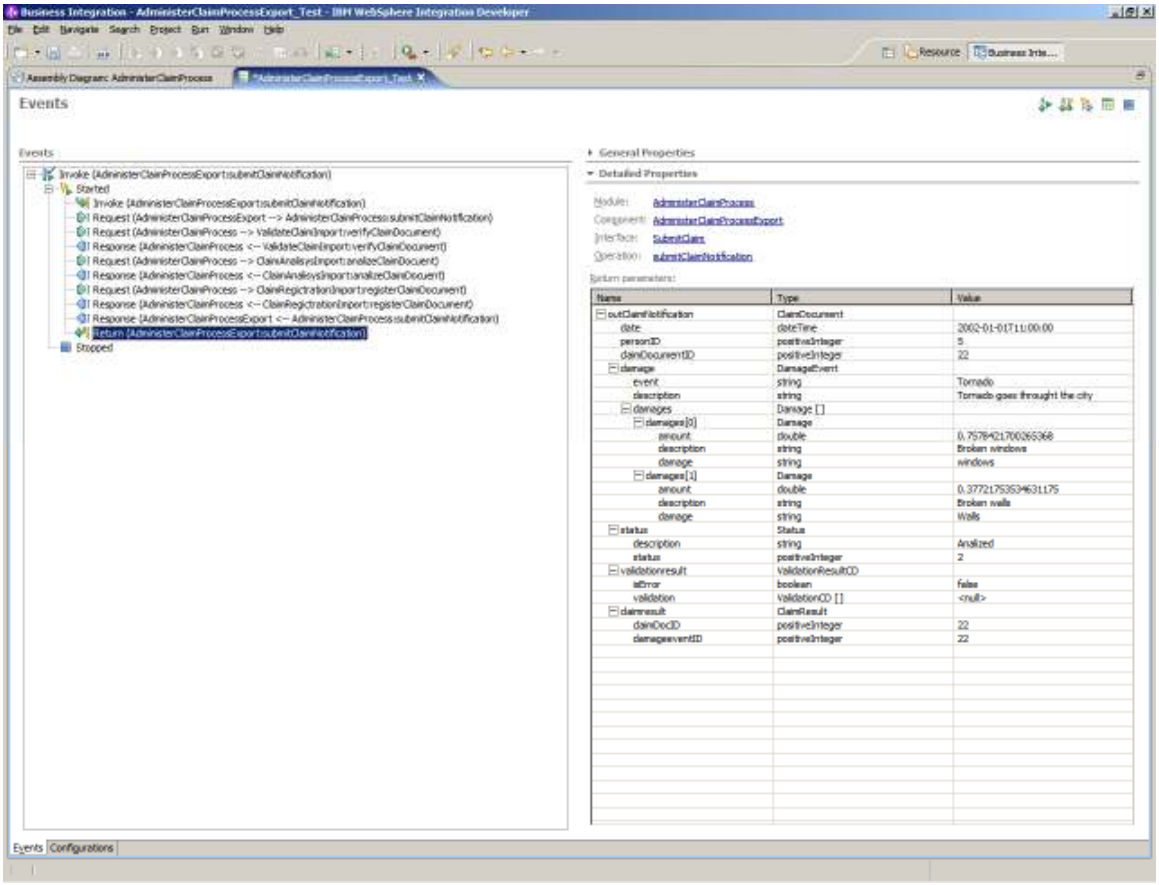


*Figure 39 Integrated test client response result*

# Appendix B – SCA Components Implementation

```
public DataObject isEGNValid(DataObject inPerson) {

String egn = inPerson.getString("egn");
DataObject personValidation = null;
ServiceManager serviceManager = new ServiceManager();
BOFactory bof =
(BOFactory)serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
DataObject validationP = null;
List validationList = new ArrayList();
personValidation = inPerson.getDataObject("personvalidation");
if(egn.length() != 10)
	{
		personValidation.setBoolean("isError",true);
		validationP = bof.create("http://Common/", "ValidationP");
		validationP.setString("code","Person.egn");
		validationP.setString("description","The lenght of the field is not
ten");
		validationList.add(validationP);
	}
try
{
	Long.parseLong(egn);
}
catch(Exception ex)
{
	personValidation.setBoolean("isError",true);
	validationP = bof.create("http://Common/", "ValidationP");
	validationP.setString("code","Person.egn");
	validationP.setString("description","This field should contains numbers
only");
	validationList.add(validationP);
}
if(validationList.size() > 0)
	personValidation.set("validationp",validationList);
return inPerson;
}

public DataObject registerUser(DataObject person) {

DB2Connector connector = null;
try
	{
```

```
                // Save person data
                connector = new DB2Connector();
                connector.createConnection();
                int person_id =
connector.savePerson(person.getString("egn"),person.getString("name"),person.
getString("phone"));
                List addresses = person.getList("address");
                DataObject address = (DataObject)addresses.get(0);
                int address_id =
connector.saveAddress(person_id,address.getInt("zip"),address.getString("street
"),address.getString("city"));
                List policies = person.getList("insurancepolicy");
                DataObject insurancepolicy = (DataObject)policies.get(0);
                int insurance_id =
connector.saveInsurance(person_id,insurancepolicy.getDate("dateFrom"),insura
ncepolicy.getDate("dateTo"));
                DataObject iptype = insurancepolicy.getDataObject("iptype");
                int iptype_id =
connector.saveIPType(insurance_id,iptype.getInt("typeID"),iptype.getString("desc
ription"));

                // Set IDs
                person.setInt("personID", person_id);
                insurancepolicy.setInt("insurancePolicyID",insurance_id);

                // Create result business object
                ServiceManager serviceManager = new ServiceManager();
                BOFactory bof =
(BOFactory)serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
                DataObject validationP = null;
                DataObject personRegistrationResult =
bof.create("http://Common/", "PersonRegistrationResult");
                personRegistrationResult.setInt("person_id",person_id);
                personRegistrationResult.setInt("policy_id",insurance_id);
                System.out.println("--->Person registration complete successful");
        }
        catch(Exception ex)
        {
                ex.printStackTrace();
        }
        finally
        {
                if(connector != null)
                        connector.closeConnection();
        }
return person;
```

```
}

public DataObject verifyClaimDocument(DataObject inClaimDocument) {
	DataObject damageEvent = inClaimDocument.getDataObject("damage");
	ServiceManager serviceManager = new ServiceManager();
	BOFactory bof =
(BOFactory)serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
	DataObject validationResultCD = null;
	DataObject validationCD = null;
	DataObject validationResult =
inClaimDocument.getDataObject("validationresult");
	List validationList = new ArrayList();
	if(damageEvent.getString("event").equals(""))
		{
			validationResult.setBoolean("isError",true);
			validationCD = bof.create("http://Common/", "ValidationCD");

			validationCD.setString("code","ClaimDocument.DamageEve
			nt.event");
			validationCD.setString("description","Event property doesn't
contain appopriate value.");
			validationList.add(validationCD);
			validationResult.set("validation",validationList);

		}
return inClaimDocument;
}

public DataObject analizeClaimDocuent(DataObject inClaimDocument) {
		DataObject damageEvent =
inClaimDocument.getDataObject("damage");
		List damages = (List)damageEvent.get("damages");
		Random generator = new Random(500);
		for(int i=0;i<damages.size();i++)
		{

			((DataObject)damages.get(i)).setDouble("amount",generator.
			nextDouble());
		}
return inClaimDocument;
}
```

# Appendix C – DB2 Connector

This appendix provides Java code about database persistency into DB2. The connection to the database is defined as JNDI resource into Websphere Process Server.

```
public class DB2Connector {
	private Connection conn = null;
	private Context ctx= null;
	private DataSource ds = null;
	private Statement statement = null;

	public void createConnection()
	{
		try
		{
			System.out.println("--->Resource lookup jdbc/CIG_DB_DS");
			// Create context for JNDI
			ctx = new InitialContext();
			// Get DataSource object
			ds =(DataSource)ctx.lookup("jdbc/CIG_DB_DS");
			// Get connection
			conn = ds.getConnection("db2admin", "db2admin");
		}
		catch(NamingException ex)
		{
			ex.printStackTrace();
		}
		catch(SQLException ex)
		{
			ex.printStackTrace();
		}
	}

	public void closeConnection()
	{
		try
		{
			if(ctx != null)
			{
				ctx.close();
			}
			if((conn != null) || (!conn.isClosed()))
			{
				conn.close();
			}
		}
		catch(SQLException ex)
```

```
		{
			ex.printStackTrace();
		}
		catch(NamingException ex)
		{
			ex.printStackTrace();
		}

	}

	public int savePerson(String egn,String name,String phone)
	{
		int result = 0;
		PreparedStatement stmt = null;
		try
		{
			String query = "INSERT INTO GIC.PERSON (EGN, NAME,
PHONE)"
			  + "VALUES(?,?,?)";
			stmt                                                    =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
			stmt.setString(1, egn);
			stmt.setString(2, name);
			stmt.setString(3, phone);
			int res = stmt.executeUpdate();
			if(res > 0)
			{
				ResultSet rs = stmt.getGeneratedKeys();
				while(rs.next())
				    result = rs.getBigDecimal(1).intValue();
			}
		}
		catch(SQLException ex)
		{
			ex.printStackTrace();
		}
		finally
		{
			try
			{
				if(stmt != null)
					stmt.close();
			}
			catch(SQLException ex)
			{
				ex.printStackTrace();
```

```
                }
        }
        return result;
}

public int saveAddress(int person_id,int zip,String street,String city)
{
        int result = 0;
        PreparedStatement stmt = null;
        try
        {
                String query = "INSERT INTO GIC.ADDRESS
                (PERSON_ID, ZIP, STREET, CITY)" + "VALUES(?,?,?,?)";
                stmt = conn.prepareStatement(query,Statement.
                RETURN_GENERATED_KEYS);
                 stmt.setInt(1, person_id);
            stmt.setInt(2, zip);
            stmt.setString(3, street);
            stmt.setString(4, city);
            int res = stmt.executeUpdate();
            if(res > 0)
            {
                    ResultSet rs = stmt.getGeneratedKeys();
                    while(rs.next())
                         result = rs.getBigDecimal(1).intValue();
            }
        }
        catch(SQLException ex)
        {
                ex.printStackTrace();
        }
        finally
        {
                try
                {
                        if(stmt != null)
                                stmt.close();
                }
                catch(SQLException ex)
                {
                        ex.printStackTrace();
                }
        }
        return result;
}
```

```
	public int saveInsurance(int person_id,Date dateFrom,Date dateTo)
	{
		int result = 0;
		PreparedStatement stmt = null;
		try
		{
			String query = "INSERT INTO GIC.INSURANCEPOLICY
(PERSON_ID, DATEFROM, DATETO)"
			  + "VALUES(?,?,?)";
			stmt =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
			stmt.setInt(1, person_id);
			stmt.setDate(2, new java.sql.Date(dateFrom.getTime()));
			stmt.setDate(3, new java.sql.Date(dateTo.getTime()));
			int res = stmt.executeUpdate();
			if(res > 0)
			{
				ResultSet rs = stmt.getGeneratedKeys();
				while(rs.next())
				    result = rs.getBigDecimal(1).intValue();
			}
		}
		catch(SQLException ex)
		{
			ex.printStackTrace();
		}
		finally
		{
			try
			{
				if(stmt != null)
					stmt.close();
			}
			catch(SQLException ex)
			{
				ex.printStackTrace();
			}
		}
		return result;
	}

	public int saveIPType(int insurancePolicyID,int typeID,String description)
	{
		int result = 0;
		PreparedStatement stmt = null;
		try
```

```
		{
			String query = "INSERT INTO GIC.IPTYPE (INSURANCEPOLICY_ID, TYPEID, DESCRIPTION)"
			  + "VALUES(?,?,?)";
			stmt = conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
			stmt.setInt(1, insurancePolicyID);
			stmt.setInt(2, typeID);
			stmt.setString(3, description);
			int res = stmt.executeUpdate();
			if(res > 0)
			{
				ResultSet rs = stmt.getGeneratedKeys();
				while(rs.next())
				    result = rs.getBigDecimal(1).intValue();
			}
		}
		catch(SQLException ex)
		{
			ex.printStackTrace();
		}
		finally
		{
			try
			{
				if(stmt != null)
					stmt.close();
			}
			catch(SQLException ex)
			{
				ex.printStackTrace();
			}
		}
		return result;
	}

	// Claim document

	public int saveClaimDocument(Date date,int personID)
	{
		int result = 0;
		PreparedStatement stmt = null;
		try
		{
			String query = "INSERT INTO GIC.CLAIMDOCUMENT (DATE, PERSON_ID)"
```

```
                    + "VALUES(?,?)";
                stmt =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
                stmt.setDate(1, new java.sql.Date(date.getTime()));
                stmt.setInt(2, personID);
                int res = stmt.executeUpdate();
                if(res > 0)
                {
                        ResultSet rs = stmt.getGeneratedKeys();
                        while(rs.next())
                            result = rs.getBigDecimal(1).intValue();
                }
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
            finally
            {
                try
                {
                    if(stmt != null)
                        stmt.close();
                }
                catch(SQLException ex)
                {
                    ex.printStackTrace();
                }
            }
            return result;
        }

        public int saveStatus(int claimDocument,int status, String description)
        {
            int result = 0;
            PreparedStatement stmt = null;
            try
            {
                String query = "INSERT INTO GIC.STATUS
(CLAIMDOCUMENT_ID, STATUS, DESCRIPTION)"
                    + "VALUES(?,?,?)";
                stmt =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
                stmt.setInt(1, claimDocument);
                stmt.setInt(2, status);
                stmt.setString(3, description);
```

```
                int res = stmt.executeUpdate();
                if(res > 0)
                {
                        ResultSet rs = stmt.getGeneratedKeys();
                        while(rs.next())
                            result = rs.getBigDecimal(1).intValue();
                }
            }
            catch(SQLException ex)
            {
                    ex.printStackTrace();
            }
            finally
            {
                    try
                    {
                            if(stmt != null)
                                    stmt.close();
                    }
                    catch(SQLException ex)
                    {
                            ex.printStackTrace();
                    }
            }
            return result;
        }

        public int saveDamageEvent(int claimDocument,String description, String
event)
        {
                int result = 0;
                PreparedStatement stmt = null;
                try
                {
                        String query = "INSERT INTO GIC.DAMAGEEVENT
(CLAIMDOCUMENT_ID, DESCRIPTION, EVENT)"
                    + "VALUES(?,?,?)";
                    stmt                                                    =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
                    stmt.setInt(1, claimDocument);
                    stmt.setString(2, description);
                    stmt.setString(3, event);
                    int res = stmt.executeUpdate();
                    if(res > 0)
                    {
                            ResultSet rs = stmt.getGeneratedKeys();
```

```
                    while(rs.next())
                        result = rs.getBigDecimal(1).intValue();
                }
            }
            catch(SQLException ex)
            {
                ex.printStackTrace();
            }
            finally
            {
                try
                {
                    if(stmt != null)
                        stmt.close();
                }
                catch(SQLException ex)
                {
                    ex.printStackTrace();
                }
            }
            return result;
        }

        public int saveDamages(int damageEvent,double amount, String
description, String damage)
        {
            int result = 0;
            PreparedStatement stmt = null;
            try
            {
                String query = "INSERT INTO GIC.DAMAGE
(DAMAGEEVENT_ID, AMOUNT, DESCRIPTION, DAMAGE)"
                  + "VALUES(?,?,?,?)";
                stmt =
conn.prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
                stmt.setInt(1, damageEvent);
                stmt.setDouble(2, amount);
                stmt.setString(3, description);
                stmt.setString(4, damage);
                int res = stmt.executeUpdate();
                if(res > 0)
                {
                    ResultSet rs = stmt.getGeneratedKeys();
                    while(rs.next())
                        result = rs.getBigDecimal(1).intValue();
                }
```

```
                }
                catch(SQLException ex)
                {
                        ex.printStackTrace();
                }
                finally
                {
                        try
                        {
                                if(stmt != null)
                                        stmt.close();
                        }
                        catch(SQLException ex)
                        {
                                ex.printStackTrace();
                        }
                }
                return result;
        }

}
```