SOA principles applied in current methodologies

Describing guidelines for using RUP and AIM for implementing SOA solutions

Version control

Version	Date	Short description changes
0.1	13-12-07	Start Rapport/Whitepaper
0.2	21-01-08	Created TOC
1.0	<datum>	Final document

Name author: Rutger Cobben

SOA principles applied in current methodologies

Name author: Rutger Cobben

Preface

This document was created as the final thesis of the Computer Science study program, which is mandatory in the final year of the master program Information Architecture. The project will be conducted at Capgemini.

For the topic of this research project I looked for a combination of literature study and something I could put into practice during my work as an Information architect and with added value to my employer. I think the outcome of this research project contributes to these goals: I evaluated the suitability of two methodologies for creating a Service Oriented architecture according to the SOA principles and used related frameworks to support these methodologies.

I would like to thank my supervisor prof. dr. J.G.L. Dietz and my referent drs. H. Vermeulen for their reviews and their useful information. I would also like to thank everyone who participated in the interviews: you provided me with a lot of useful information.

Table of Contents

1 Executive Summary

Service-oriented architecture(SOA), an approach where service will be used as building blocks for architectural design, will be used in part in more than 50% of new, mission-critical applications and business processes designed in 2007, and in more than 80% by 2010 (0.7 probability).[1] This new approach requires new design methodologies for implementing those. This thesis will identify guidelines for RUP and AIM, as examples of design methodologies, to design a SOA solution. These guidelines help to understand the power of SOA and will create a change mindset in from implementing software from a process central orientation to a more reusable, composable and adaptable orientation. This orientation will create a more user central approach.

To understand the power of SOA we first take a look at the principles of SOA described by Thomas Erl. In this paper we identify guidelines based on these principles. Reusable, composable, loosely coupled and standardized services are examples of the principles and create a basis for a SOA design. This thesis will try to relate these principles to creating services in a SOA environment with current design methodologies.

One of the design methodologies is RUP and stands for Rational Unified Process. This methodology uses the UML language as basis for their designs. The RUP methodology has developed a new approach for identifying the SOA services, called SOMA. It identifies the services through goal-services modeling, domain decompositions and looking at existing assets. These methods are used during the analysis and design phase and will help to identify reusable and composable services. The Use Cases in combination with the activity and sequence diagrams from RUP generate a powerful method to develop a SOA. These models describe the process from a user central point and describe the system in a business perspective and in an IT perspective. These models describe the decomposition of the system, and can be layered to create a structured overview of the system. This approach will deliver a reusable and structured SOA design.

The AIM methodology, for implementing package applications, creates services during the creation of the process logic. This methodology has different methods to identify reusable, composable parts. The composable part of the services is mostly prescribed by the package application. The reusable parts are identified through expert knowledge on existing services in the modules and the creation of an architectural design for services. A market reference model and a reference architecture model are often used to make this approach more standardized and fundamental. Also this approach requires, when building a SOA solution, a strong emphasis on finding reusable parts during the analysis and design phase.

To gain more efficiency and control over a SOA architecture the IAF framework and the Sonic Maturity Model are used in combination with the RUP methodology. The IAF framework, an architectural framework, helps identifying services on business and IT level with the creation of the IAF model. The Sonic MM will generate a roadmap for the development of a SOA solution.

---

[1] SOA Advances, Gartner, 17 November 2006

Concluding to gain agility and cost-effective architecture, RUP and AIM can be used with additional guidelines to build an SOA architecture. In this architecture every asset is modeled as a service.

2 Introduction

2.1 Background

In today's market, many companies present themselves to the world through their internet site. They sell their products or services using web enabled technologies, manage and control their accounting and information exchange using electronic means. Companies use internet technologies for sharing knowledge, decision making, coordination and project control. As applications and systems increased in number and complexity, the need for a clear and consistent view of the complete picture, together with a structured approach to integration became apparent. Gradually, the term architecture was extended to include all areas described above. Over the years, IT has evolved from delivering point solutions to a complex, interrelated landscape of applications, interfaces and infrastructure that support the business processes of an organization and the productivity of its people. Recently, this has started to include an architectural view of business change through an alignment of business and IT to deliver the business goals. Recent research and innovations have shown that Service Oriented Architectures (SOA), a way of architecture design, can be seen as the new architecture design with agility and reusability as their core values[2]. SOA prescribes the development of applications as modular business services that can be easily integrated, secured, and administered.

Capgemini recognized the value of architecture in the early nineties, developing the first components of its architecture framework and approach, the Integrated Architecture Framework (IAF), as early as 1993. IAF has been steadily enhanced since to become a mature and comprehensive framework. Within this framework a method is developed for building SOA solutions.

2.2 Problem description

In the world of standard enterprise application packages SAP and Oracle dominate the market. These packages software vendors have a lot of pressure on their applications from enterprises through the demand for SOA applications. The demand for an increase in flexibility and extendibility controlled by the SOA applications needs a better foundation of the package applications. The introduction of SOA hasn't yet reached the design methodologies and is now based on experience instead of well formed rules. RUP and AIM, current methodologies for implementing and designing enterprise applications aren't yet suitable for this approach. They aren't designed to implement a SOA architecture. The SOA architecture approach requires a new way of thinking for designing and implementing enterprise systems. The first step is to understand

---

[2] E – Supply Chain Orchestration Using Web Service Technologies:A case using BPEL4WS, Simon Samwel Msanjila, 2003, http://portal.acm.org/ft_gateway.cfm?id=1089636&type=pdf&coll=GUIDE&dl=GUIDE&CFID=46560301&CFTOKEN=73758637

the principles of SOA. These principles are the basis of creating a SOA solution. In this thesis guidelines are described on how SOA principles can be applied to transform RUP and AIM to a methodology where SOA solutions can be built from. The guidelines in this thesis will help to identify the problems concerning the design of a SOA solution and help to build a solution space for the methodologies.

2.3 Intended audience

This research is intended for two groups of people:

- "Technically focused people":
  - IT architects (develop architecture framework and manage the implementation of the IT-structure)
  - Technical consultants (configure and implement IT-structures)
  - Developers (implements IT-structures)
- "Business focused people":
  - IT architects
  - Clients (determine the scope and goals of an IT-project)
  - functional designers (set up requirements)
  - business analysts (design business processes)

For the "technically focused people" this will create technical information about how to integrate services and explain the difficulties.

For "business focused people" this will create new insight information about integrating different platform services and the value of using SOA for this.

2.4 Context of the project

This section describes the context of this thesis. The context start with describing the definitions. In the architecture world many definition are vague and immature. To prevent ambiguity this thesis includes the formal definitions about the subject. The next section describes the history of the SOA, which will go into detail about the basic principle 'separation of concerns' of SOA, why enterprise architecture is needed in the current enterprises. The final section of this chapter gives the preconditions of the project.

2.4.1 Definitions

In this section a clear separation is made between business and IT, because in practice there is a lot of discussion and ambiguity about it.

Business process[3]: A business process is a set of linked activities that create value by transforming an input into a more valuable output. Both input and output can be artifacts and/or information and the transformation can be performed by human actors, machines, or both.

[3] Wikipedia, 2007, http://en.wikipedia.org/wiki/Business_process

Business service[4]: A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services

IT process[5]**:** An IT process is an instance of a computer program that is being executed.

IT service[6]**:** A service is a unit of solution logic to which service-orientation has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguish a unit of logic as a service from units of logic that may exist only as objects or components.

Enterprise Architecture[7] **(IAF)**: Details the structure and relationships of the Enterprise, its business models, the way an organization will work, and how and in what way Information, Information Systems and Technology will support the organization's business objectives and goals. Enterprise Architecture provides an all-encompassing, holistic, end-to-end view of the business in terms of people, process, governance and technology, within (and external to) the business that supports these objectives and goals. Enterprise Architecture is often likened to "City Planning."

SOA[8]: Service-oriented architecture represents an architectural model that aims to enhance the agility and cost-effectiveness of an enterprise while reducing the overall burden of IT on an organization. It accomplishes this by positioning services as the primary means through which solution logic is represented. SOA supports service-orientation in the realization of the strategic goals associated with service-oriented computing.

Design principle[9]: A highly recommended guideline for shaping solution logic in a certain way and with certain goals in mind.

2.4.2 Separations of concerns

This section describes how separations of concerns can be seen as the basis of the SOA ideology. This principle shows us that dividing different concerns over different design levels is needed to create a structure in the chaos of software engineering. If we are attempting to separate concern A from concern B, then we are seeking a design that provides that variations in A do not induce or require a corresponding change in B (and usually, the converse). If we manage to successfully separate those concerns, then we say that they are "decoupled", for obvious reasons. As a simple example, a car's side view mirror controls are

---

[4] Service-oriented architecture (SOA) definition, Barry & Associates, Inc., 2008, http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html
[5] Process computing, Wikipedia, 2008, http://en.wikipedia.org/wiki/Process_%28computing%29
[6] SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service.asp
[7] Enterprise, Business and IT Architecture and the Integrated Architecture Framework, Capgemini, 2007
[8] SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_oriented_architecture.asp
[9] Design Principle, Thomas Erl, 2007, http://www.soaglossary.com/design_principle.asp

decoupled from its accelerator, permitting us to adjust our mirrors without the fear that the car will speed up or slow down.

In the IT world, in particular with distributed systems, we often talk about the importance of the "separation of interface from implementation". Enterprises try nowadays to design their distributed system using this principle. Chapter 4 will go into detail how most of the SOA principles are formed from this principle.

2.4.3 SOA

This section gives an introduction into the field of SOA. This generates the first ideas about what SOA is and how this can be realized. SOA is not a new concept, but is now playing a major role in most IT related magazines and the CIO's agenda. But what is SOA really? In spite of the diversity of the thoughts about what SOA is, the two previous sections tell us about the origin of SOA. The basic elements where SOA is created from are the separation of different concerns and the creation of a fundamental enterprise architecture. According to the definition used in section 2.4.1 SOA is an architectural approach which is used to enhance the business agility and cost effectiveness. This approach contains services as basic elements of the architecture and uses those services, which represent solution logic, to support the strategic goals of the business according to the previously noted principles. Does this really mean that every element in this architecture can be seen as a service? Yes, it does, a reusable and easily connectable service. But what is a service really? A service is[10]:

An automated ' unity of behavior', provided by a provider and used by one or more consumers. It has a clear goal, a clear result and adds value. Implementation and specification are separated. Consider a broad perspective. A service is not always a web service. A Web service is an information service that can be contacted using http and has a definition published in a WSDL. There are a lot of different type of service technologies and service types: Portlets, FTP service, Database link, API's, part of a legacy application, Java, Forms, etc.

Concluding: SOA stands for an architectural approach with composable services as the basis element to create agility and cost-efficiency in the enterprise.

2.4.4 Enterprise Architecture

This section describes why it is important to have an enterprise architecture in current enterprises.

According to the second law of thermodynamics, any closed system cannot increase its internal order by itself. In fact, any activity that is geared toward ordering the system will increase its overall disorder (called entropy). In many respects, this law is also applicable to enterprise software, which often has very similar characteristics. Consequently, outside intervention is continually required to help create a higher order and to ensure that development efforts are not lost.

---

[10] Service Oriented Architecture, Joost van der Vlies, 12 February 2008 (presentation)

In enterprise software, the architect takes on the role of an outside influencer and controller. It is his responsibility to oversee individual software projects from the strategic point of view of the overall organization, as well as from the tactical, goal-oriented viewpoint of the individual project. He has to balance different requirements while attempting to create an enduring order within the enterprise software landscape. The enterprise software architecture is the architect's most important tool at hand. Software architects are constantly confronted with changes to and expansion of functionality that increase system complexity and reduce efficiency. By re-factoring current solutions, architects constantly strive to reduce complexity and thereby increase the agility of the system conformity to their enterprise architecture: 'the big picture'.

2.4.5 Preconditions

In this section the preconditions are defined. These preconditions are based for the use of enterprises which are the major clients of Capgemini. The following list demarcates the subject, by describing the boundaries of the subject:

- Enterprise Architecture:
  - Describe only the design methodology from business process to definition of services.
  - Don't describe the design for the configuration of standard packages.
  - Don't describe any design for Business Intelligence.
  - Don't describe any security nor governance design.
  - Describe the methodology from a conceptual and on logical view.
- Business process:
  - Describe business process on functional and on organizational level.
- Services:
  - Don't describe any implementation details of services.
  - Service should be modeled conform the SOA principles.
- SOA:
  - Only consider SOA architectures for long term usages
  - Only consider SOA for enterprises.
- Principles:
  - Only consider the following design principles of SOA:
    - Standardized Service Contracts
    - Service Loose Coupling
    - Service Abstraction
    - Service Reusability
    - Service Autonomy
    - Service Statelessness
    - Service Discoverability
    - Service Composability
    - Service-Oriented Interoperability

2.5 Problem definition

This section demarks the problem definition. From the previous sections we can identify that SOA has aroused much attention in the enterprise architecture market. The question that is still open is how architects can build a SOA solution

using the current methodologies like RUP and AIM. Thomas Erl, has stated eight design principles to create a SOA. These design principles are the leading viewpoint how to apply SOA solutions in this thesis. During my research I will show how these principles can be used as guidelines for RUP or AIM. The main question this thesis will answer is this:

How to safeguard the SOA principles, stated by Thomas Erl, when designing an Enterprise Architecture solution using the RUP or the AIM methodology?

2.6 Subquestions

This section describes the sub questions which should be answered first to answer the main question to provide a basis for designing a good SOA.

- What is the influence of the IAF framework when applying RUP on the SOA principles?
- What is the influence of SOA maturity model from Sonic on the SOA principles?
- Which tasks, qua responsibility, are changed through the introduction of SOA principles?

The first sub question is raised, because during my research I encountered a lot of architecture frameworks which help creating an architecture. Capgemini has made their own version of such a framework and this can be used as starting point for building a SOA.

The second question is raised, because during my research I encountered a lot of problems about failure rates in SOA projects. These projects failed, because they had done too many architecture steps at once. The Sonic MM helps to structure the development of SOA and creates a roadmap for a solution.

The third question is raised, because during my research I encountered problems about responsibility for it-assets and aligning them to the current architecture.

2.7 Research objective

To create added value for Capgemini and the Technical University of Delft clear objectives have to be describe. The following list identifies those goals:

- Understand the SOA ideology
- Create insight how RUP and AIM can be used to create SOA solutions
- Develop guidelines for implementing a SOA using RUP and AIM.

2.8 Structure of the report

To find guidelines for the RUP and AIM method, I first introduce the methods to create a fundamental basis for the problem. In the following chapter I introduce the principles described by Thomas Erl where the methodologies need to be mapped to.

In the following chapter the RUP and AIM are described. The RUP methodology is mostly used for customizations, while AIM is used for package application

from Oracle. These methodologies give a good overview how software is developed nowadays.

The next chapter describes the IAF framework. This framework is used to support current methodologies with an extra view about why, what, how and with what the system in a business should be configured.

The next chapter describes the Maturity model. This model visualizes how SOA implementations can be done incremental.

The next chapter describes the mapping between the SOA principles and the current methodologies to help identifying guidelines for these methodologies. This chapter also describes how the IAF model and Sonic MM model helps identifying services.

The next chapter describes how the validation of this research is conducted.

The final chapters describe the conclusion, discussion and further research of this thesis. In these chapters the conclusion and open issues are described.

2.9 Research Approach

This chapter describes the scientific basis of this thesis. For this research I wanted to do a case study to include practical information , but I found out that a case study about this subject was impossible, because only a few enterprises where on the level of this subject and none of them had room for an extra investigation. So I used the following scientific elements to conduct the study:

- Scientific literature
- Interviews with IT-experts
- Survey

2.9.1 Scientific literature

Most of the scientific literature used in this thesis has its basis from scientific journals on the web. Other sources for the literature came from Capgemini internal knowledge bank for all their scientific literature and of their partners. The last source for the scientific literature was knowledge banks of the university of Delft which provided the literature through their extensive libraries.

2.9.2 Interviews with IT-Experts

A source of information were the IT-experts in the field. Those people have a lot of practical experience and have insight information about new technologies and problems in this field. First I used open interviews to create new ideas for the thesis. After this I validate the result by using structured interviews. The interviews gave me the most added value.

2.9.3 Survey

The last source of information was the survey. In the survey questions were asked to IT experts to validate the founded proposition. The survey is based on a scientific approach and is processed with statistical methods to validate the propositions scientifically.

3 SOA Principles

This chapter describes the principles of the SOA. They form the basis for this thesis and will be used to see how different methodologies will be able to apply them. These principles are the guidelines to design a SOA implementation and improve the SOA maturity. The principles of SOA should be valid in a SOA architecture to gain the maximal benefit within the boundaries of a SOA architecture. The principles don't provide one certain solution, only a solution space where many solutions can be found for one problem. The following principles are the basis of a SOA[11]:

- Standardized Service Contracts
- Service Loose CouplingCreation of a service contract which is decoupled from technology and implementation.
- Making a functional service context independent on their outside logic.
- Minimal consumer coupling requirements.
- Service AbstractionServices consistently abstract specific information about technology, logic, and function.
- Create meta data for communication with the service.
- Outside of what is documented in the service contract all the other information about the service should be hidden for consumers.
- Service Reusability
- Service Autonomy
- Service StatelessnessDefine highly business process-agnostic logics so that the services are not designed to retain state information for any specific parent business process.
- Increase amounts of interpretative programming routines capable of parsing a range of state information delivered by messages and responding to a range of corresponding action requests.
- Service Discoverability
- Service Composability
- Service-Oriented Interoperability

3.1.1 Standardized Service Contracts

The standardized service contract design principle is essentially about who, what, why and when someone can use a service. This is made public through a service's interface and its goals will be published as part of a service's official

[11] Principles of service Design, pag 71-74, Thomas Erl, July 2007

contract. The goal of standardized service contracts is to minimize data transformation operations.

It's important to note, however, that there are two different types of contracts. First, there is the legal agreement between two business entities, and second, there is the technical relationship. The legal agreement of two business entities looks exactly like a normal service legal agreement. In these agreements things like payment, delivering and quality are described. The technical relationship is, in the same way legal contracts are documents, a set of executable rules and agreements written in a language that both parties can understand. This information should be documented externally (registry) for each participant that provides the information to each participant that needs the information.

Contracts and Policies

Given that the contract is the key to enable service communication, which elements are in such a service contract? The following list describes the content of a service contract[12]:

- Service contracts should describe **functional requirements**. That is, what a service provider will make available for its consumer. The contract should define its functionality and the type of input data and type of the output data. These fields can be empty.
- Service contracts must also specify **nonfunctional requirements** that provide details about the operation of a service. This includes information about the responsibility, availability, security, and other quality of services.
- Service contracts also specify the **rules of engagement** between consumers and providers, known as policies, that govern who can access a provider, what security procedures the participants must follow, and any other rules that apply to exchange information.

A contract describes the expressions of the visible aspects of a service behavior. What contracts never include are the data that providers and consumers actually exchange. In addition, since consumers vary just as much as providers, there might be multiple contracts for a single service.

A point of confusion in this subject is the difference between contract and policy. The definitions of these terms can vary from one company to another. In general the difference is that a contract is described as above and a policy *is a set of conditions that can apply to any number of contracts, from none of them to all of them*.[12] For example, a policy might state that all interactions with services must be SSL-secured, and that policy would then apply to all service contracts in an organization.

The following guidelines help developing standardized service contracts[13]:

---

[12] Zapthink, Ronald Schmelzer, August 2005, http://www.zapthink.com/report.html?id=ZAPFLASH-2005824

- A service contract is provided with the service.
- The service contract is standardized through the application of design standards.

3.1.2 Service Loose Coupling

SOA principles place a strong emphasis on decoupling service consumers from service providers. One underlying concept behind decoupling is that changes to the service provider should not necessarily correspond in changes of the service consumer. For example, a decision to redeploy a service currently running on a particular operating system to a different platform wouldn't affect the services consumer. This can be interpreted as autonomous versionability[14]. If you can't version your consumers, providers and brokers separately then you don't have loosely coupled services.

Another concept of loosely coupling is that the service consumer should be isolated as far as possible from the details of the business logic implemented by the service provider. Again, careful service design is required in order to achieve this decoupling.

Generally service coupling is about the dependencies of services on other services. The lower the dependency of a service on another service is the looser the coupling. The advantages of this are keeping the service abstract to improve reusability and changeability.

The following guidelines help developing loosely coupled services[15]:

- Creation of a service contract which is decoupled from technology and implementation.
- Making a functional service context independent on their outside logic.
- Minimal consumer coupling requirements.

3.1.3 Service Abstraction

According to Thomas Erl service abstraction is: "*To hide information about a program not absolutely required for others to effectively use that program.*"[16]

There are two concepts in service abstraction. One is to hide information unnecessary for consumers and the other is to describe unambiguously the functions of the provider.

---

[11] SOA principles, Thomas Erl, july 2007, pag 130
[14] SOA Zone, Dan Foody, August 2005, http://www.soa-zone.com/index.php?/archives/16-Defining-loose-coupling.html
[11] SOA principles, Thomas Erl, july 2007, pag 168
[11] Principles of Service design, pag. 245, Thomas Erl, July 2007

Hide unnecessary information
Services can use logic to support their execution. There is no limit to the amount of logic a service can represent. A service may be designed to perform a simple task or it may be positioned as a gateway to an entire automation solution. There is also no restriction as to the source of automation logic a service can draw upon. Abstraction is about hiding this logic.

Why do you want to hide this information? By limiting information about the service logic designers aren't bothered with implementation details and can use the service according to its contract.

Describe unambiguously the functions
But how can service providers describe unambiguously the functions of the service in a service contract? This is described in standard language (like WSDL Web Service Description Language) that every consumer can understand. This description may include textual descriptions intended for human consumption or keywords which the service can be referenced to.

The following guidelines help to develop abstract services[17]:

- Services consistently abstract specific information about technology, logic, and function.
- Create meta data for communication with the service.
- Outside of what is documented in the service contract all the other information about the service should be hidden for consumers.

3.1.4 Service Reusability

According to Thomas Erl service reusability is: "*Services contain and express agnostic logic and can be positioned as reusable enterprise resources.*"[18]

In theory, reuse is a pretty straight-forward idea: simply make a software program useful for more than just one single purpose. The reasons for doing this are also quite evident. Whereas something that is useful for a single purpose will provide value, something that is repeatedly useful will provide repeated value and is therefore a more attractive investment.

Reuse is one of the core principles of service-orientation. This influences the design, analysis and models of SOA and can result in huge savings in application development cost and time. On the other hand the uniqueness of specific applications and systems can be a major obstacle for reusing components.

In a SOA, the only characteristic of a service that a requesting application needs to know is the public interface. The functions of an application or system, described in these interfaces, (including legacy systems) can be dramatically easier to access as a service in a SOA than in any other architecture.

---

[11] SOA principles, Thomas Erl, july 2007, pag 216
[18] SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_reusability.asp

How is reuse promoted in SOA? When people need a service, first people must look into the registry if they can find an appropriate service. If so, re-use is preferable over making a new one. This keeps services in the repository non-duplicated, increases reusability and makes the architecture maintainable. The other very important design issue for service design is to keep your service functions as small as it can be to promote reuse.

The following guidelines help developing reusable services[19]:

- The service is highly generic
- The service is defined for a functional context
- The service has an extendable contract
- The service can be accessed by multiple consumers

3.1.5 Service Autonomy

According to Thomas Erl service autonomy is where: "*Services exercise a high level of control over their underlying runtime execution environment.*"[20]

This definition tell us that Autonomy has two keywords: control and execution. Combining those two words will end up in controlling their execution. This implies that autonomy is achieved by making the service independent giving it access to control its own execution process and process logics. Something that is autonomous has the freedom to control and make its own decisions without the need for external approval or involvement. A reduction in dependencies takes place by increasing the amount of control supporting the loosely coupling principle.

The following two common autonomies are common in a SOA environment: [21]

- service-level autonomy
- pure autonomy

Service-level autonomy

Service boundaries are clearly distinct from each other, but the service can use other resources. For example, a wrapper service that adapts a legacy environment and at the same time can operate independently has service-level autonomy. It governs the legacy system and shares resources with other legacy clients. This is autonomy from a functional point of view.

---

[11] SOA principles, Thomas Erl, july 2007, pag 259
[11] SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_autonomy.asp
[21] The principles of service-orientation, Thomas Erl, May 2006, http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1192369,00.html

Pure autonomy
Pure autonomy means that the underlying logic is under complete control and ownership of the service. This is typically the case when the logic is built up from the ground up in support of the service. This is autonomy from an operational point of view.

Conclusion
Two kinds of autonomy are used for services. Service-level autonomy is used when the service uses resources from other services, but can be used independently for other usage. Pure autonomy is used when a service hasn't any influence on other services and controls all of its actions. Pure autonomy can be seen as the most dedicated autonomy to SOA, because of its full independency. However in most SOAs both service-level -and pure autonomous services coexist.

The following guidelines help to develop autonomous services[22]:

- Services should have a well-defined functional boundary that should not overlap with other services.
- Services should have a high level of control over their own functions.

3.1.6 Service Statelessness

A "stateful" service is a service which retains some information (state) about transaction steps in its memory. This can happen with a complex transaction that requires several steps to complete a service transaction and the client must return to the same service for the next step. This might result in a delay if many clients are using the same service or in a transaction failure if the service hosting the service fails between steps.

A better design for using complex transaction is when services don't have states within the transaction, but are "stateless." For a multi-step transaction the following properties should be included[23]:

- At the end of each intermediate step the service should supply enough information back to the client, the state information, to complete the transaction which each other available service with can complete the transaction.
- The client must continue with transactions by using whatever service is selected by the orchestration process.
- The selected service should be able to handle the transaction even if it didn't start the transaction.

---

[11] SOA principles, Thomas Erl, july 2007, pag 296
[11] Principles of service design, Thomas Erl, 2007, chapter 11

The figure below shows how you can maximize your statelessness. States are stored in the data repository when not needed at the specific time, which makes more resources available.

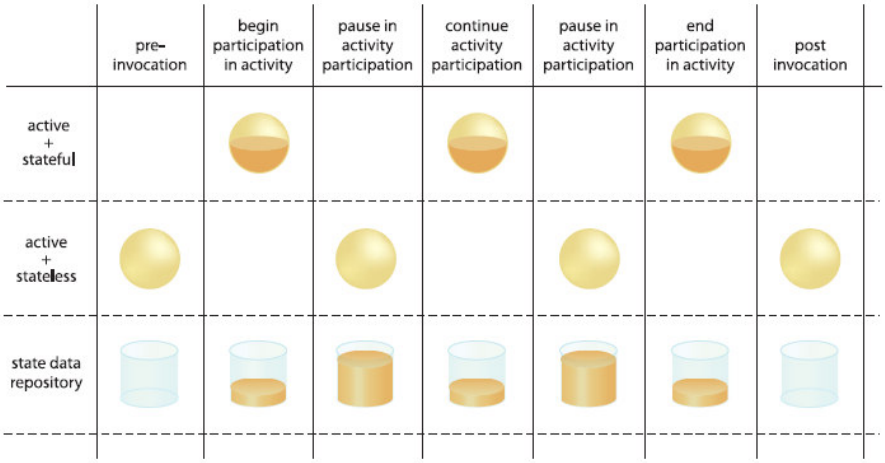

Figure 1 the service with fully deferred state management maximizes its opportunities to be in a stateless condition. Even when stateful, it defers state data when possible.

Activity in the service with data from the state

Activity in the service without data from state

The usage of memory on the state data repository

The following guidelines help developing stateless services[24]:

- Define highly business process-agnostic logics so that the services are not designed to retain state information for any specific parent business process.
- Increase amounts of interpretative programming routines capable of parsing a range of state information delivered by messages and responding to a range of corresponding action requests.

[11] SOA principles, Thomas Erl, july 2007, pag 333

3.1.7 Service Discoverability

The principle of discoverability creates guidelines to help services to be founded by the use of their metadata. The metadata of a service do not describe only the service's purpose, but also the functionality offered by its individual operations, because each service operation provides a potentially reusable piece.

Services in SOA must always be as much discoverable as possible. The reasoning behind this is that, even if there is no need for a service to be registered, a service portfolio will grow in size. Also the evolutionary governance of those services can be better managed when each service is equipped with sufficient metadata to properly communicate its functionality.

There are three key elements for discoverability[25]:

- Awareness
- Willingness
- Reachability

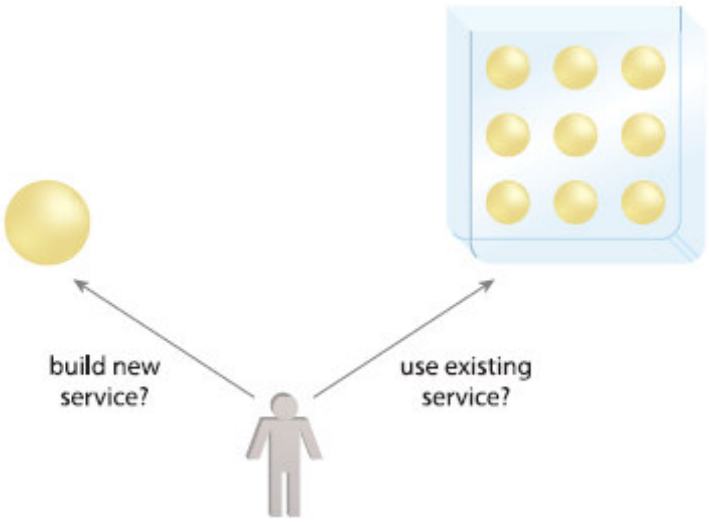

Figure 2 Service discoverability

[25] Reference model for Service Oriented Architecture, C. Matthew Mackenzie, August 2006, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

Awareness
Awareness is that the service provider and the service consumer should know each other's existence. The initiator of a service interaction is responsible for the awareness and this is the service consumer usually.

For a service consumer to discover a service, the service provider should provide a service description and the policies of the service. The service consumer should be able to receive this kind of information. This information can be pulled by the consumer or pushed by the provider. How much information is pulled or pushed depends on many factors (advertising, broadcasting, and searching). Generally a service consumer should get enough information about the service provider to find out if the service provider provides the right service.

Willingness
The willingness of a service is the cooperation of a service, described in its policies. The policies contain the preconditions for the consumer on how to use the service, policies are the security constrains of a service.

Reachability
When services are aware of their existence a service should also be reachable. This request seems obvious, but is an essential pre-request of communicating with the other. It can be that a service has all the information from the other, but that still no physical path (like an internet connection path) exists to communicate.

Conclusion
When a service is aware, willing and reachable it can be called discoverable. A discoverable service is closely linked to reusable. When a service can be found easily it can be configured to be reused. This makes governance of services easier and can speed up development time.

The following guidelines help developing discoverable services[26]:

- Service contracts are equipped with metadata that will be needed for discovery purposes.
- Services should describe in humanly readable terms their functions in their contracts.
- Create services for registration purposes.

[11] SOA principles, Thomas Erl, july 2007, pag 369

3.1.8 Service Composability

According to Wikipedia (2007) service composability is: "*Collections of services can be coordinated and assembled to form composite services.*"[27] This principle is about how services can cooperate with each other and has been evolved from the OO paradigm where people programmed classes as components. These components are nowadays evolved to services. The term orchestration is used to talk about assembling services. Through the use of standard interfaces it's easy to connect services with each other. To ensure maximal composability service should be created as generic as possible. In the figure below a metaphor is chosen to illustrate the power of composability, where the LEGO blocks define services. If all services are connected in a well structured way, a whole enterprise can be built, like a whole building can be build.

Figure 3 Service Composability[28]

The following guidelines help developing composable services[29]:

[27] Wikipedia August 2007, http://en.wikipedia.org/wiki/Service-oriented_architecture

[28] Lego, 2007, http://factory.lego.com

- The service contract needs to be flexible so that it can facilitate different types of data exchange requirements for similar functions. This typically relates to the ability of the contract to exchange the ame type of data at different levels of granularity.
- Create services for a single task.

3.1.9 Service-Oriented Interoperability

The SOA vision of interaction between clients and loosely-coupled services means widespread interoperability. In other words, the objectives for clients and services are to communicate and understand each other independent on the platform they run. This objective can be met only if clients and services have a standard way of communicating with each other, a way that is consistent across platforms, systems, and languages. In fact, web services provide exactly that. Web services have protocols and technologies that are widely accepted and used, and that are platform, system, and language independent. In addition, these protocols and technologies work across firewalls, making it easier for business partners to share vital services.

[11] SOA principles, Thomas Erl, july 2007, pag 394

4 Current Methodologies

4.1 Introduction

This chapter describes two methodologies. The first one the RUP methodology. is chosen because this methodology is used by many system integrators and yields the advantage to have a wide range of expert literature and example cases. Also I have some experience with UML, which is an important part of the RUP methodology. The development of RUP, which is originally designed by Rational, has been taken over by IBM in 2002. This methodology focuses on the UML modeling set and has proven itself over the years by many successful implementations. The second methodology I describe is AIM methodology. This methodology is designed by Oracle and has its focus on the design and implementation of Oracle package application software. This methodology is therefore more focused on package application. These two methodologies will give a good overview of how to design a SOA solution in different environments.

4.2 RUP

IBM's Rational Unified Process, RUP[30], is an iterative process framework that provides best practices for software and systems delivery and implementation and effective project management. 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 Unified Process was designed from the start to include both a generic, public domain process (known as the Unified Process), and a more detailed specification known as the Rational Unified Process which could be marketed as a commercial product. RUP is built from six principles, has nine disciplines and has four phases. From these disciplines and phase only those are described that focus on the analysis and design of an enterprise solution, which are the first three disciplines and the first two phases.

[30] RUP, IBM, 2008 http://www-306.ibm.com/software/awdtools/rup/

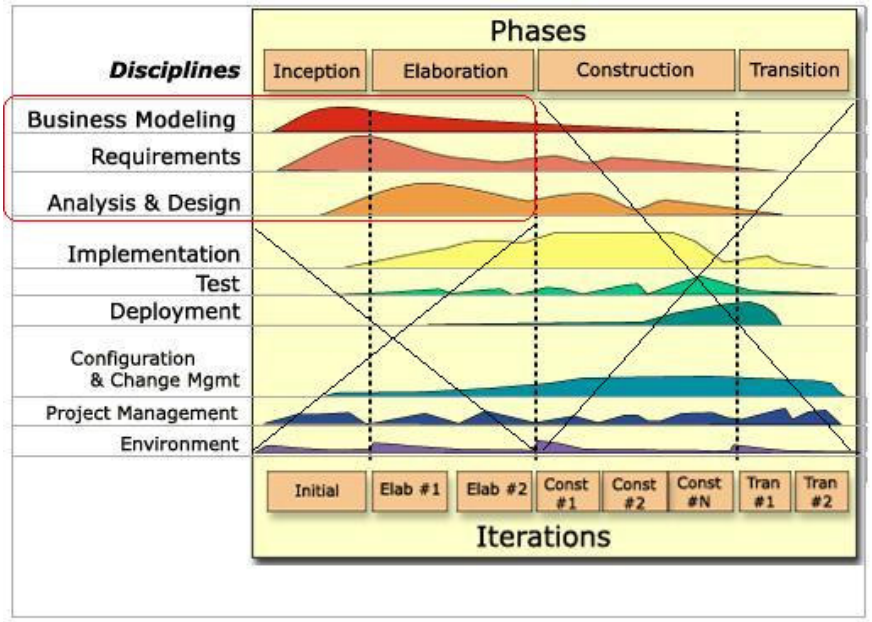

Figure 4 RUP methodology[31]

4.2.1 Principles

The six key principles for business-driven development are[31]:

1. **A**dapt the process
2. **B**alance stakeholder priorities
3. **C**ollaborate across teams
4. **D**emonstrate value iteratively
5. **E**levate the level of abstraction
6. **F**ocus continuously on quality

Adapt the process
Project can be of different sizes or format, and this should be aligned with the processes of the project. RUP provides pre-configured process templates for small, medium and large projects, which can be used for easier adoption. The context of the process should reflect the goals of the RUP phases. Adapting a process stimulates the improvement of a process in an organization.

[31] RUP, Wikipedia, 2007, http://en.wikipedia.org/wiki/Rup

Balance stakeholder priorities
Another issue is that stakeholders have other priorities then improving IT, this includes business goals and stakeholder needs. Often they compete and conflict and need to be compromised.

Collaborate across teams
With a broad variety of stakeholders a good collaboration is needed to perform well on projects. This collaboration is enabled through modern communication tools and makes globalization less complicated. The collaboration is not limited to requirements, but includes exchange of metrics, test results, release management and project plans. That is especially true for RUP projects which are executed in an iterative-incremental approach.

Demonstrate value iteratively
RUP defines project as iterative processes. This means projects are delivered, even though often only internally, in increments. Each increment can be used to measure the progress of the project and encourage feedback from stakeholders about the direction of the project. This allows projects to adjust to changed situations based on the feedback. The stakeholders on the other side can influence the shape of the development effort while the project is executed. Through the focus on risk in RUP the iterative process can eliminate all possible problems during the execution of the project.

Elevate the level of abstraction
This key principle motivates the use of reusable assets such as software pattern, 4GL or Framework to name a few. This prevents software engineers going directly from the requirements to custom-made software code. A higher level of abstraction also allows discussions on different architectural levels and supports the loosely coupled relationship and can contribute to a better parallel distributed system. These can be accompanied by visual representations of the architecture, for example using UML.

Focus continuously on quality
Built-in quality checks are needed to guarantee the high quality standards applied today. This is a continuous ongoing activity in the software engineering project, often performed in a daily rhythm supported by the entire team. Automating test scenarios (scripts) helps in dealing with the increasing amount of tests accuracy.

4.2.2 RUP lifecycle (phases)

After describing the principles of RUP an implementation model is needed for the development of the software. The RUP lifecycle has the spiral model[32] as its implementation model. This lifecycle is available as a work breakdown structure,

[32] Wikipedia, 2007, http://en.wikipedia.org/wiki/Spiral_model

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[33], but only the first two are described:

- Inception phase
- Elaboration phase
- Construction phase
- Transition phase

Inception phase
In this phase the business case, the end-product vision and the scope of the project are defined. The business case includes business context, success factors (expected revenue, market recognition, etc). The end-product vision includes the financial forecast and the Enterprise Architecture principles. In this phase a basic use case model, a project plan, a initial risk assessment and a project description (the core project requirements, constraints and key features) are made.

Elaboration phase
The elaboration phase plans the necessary activities and required resources, specifying the features and designing the system architecture. This includes:

- a use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed.
- A description of the software architecture in a software system development process.
- A development plan for the overall project.
- Prototypes that demonstrably mitigate each identified technical risk.
- Finalizing this phase the project transitions are going into a high-risk operation, where changes are much more difficult and detrimental when made.

4.2.3 Disciplines

In this section the nine disciplines are described, which are a subset of the two main disciplines. Only the first three disciplines will be described in detail:

Engineering Disciplines:

- Business Modeling Disciplines
- Requirements Disciplines

[33] The Rational Unifed Process An Introduction, Philippe Kruchten, 2004, p62

- Analysis and Design Disciplines
- Implementation Disciplines
- Test Disciplines
- Deployment Disciplines

Supporting Disciplines:

- Configuration and Change Management Disciplines
- Project Management Disciplines
- Environment Disciplines

Business modeling
The purpose of the business modeling is to understand the structure and the dynamics of the organization in which a system is to be deployed. Visualize current problems in the target organization and identify improvement potentials. Ensure that customer, end users, and developers have a common understanding of the target organization and derive the system requirements to support the target organization

Requirements
The purpose of the requirements discipline is to describe what the system should do and allows the developers and the customer to agree on that description. It further describes a planning for technical contents of iterations and an estimated cost time analysis.

Analysis and design
In the analysis and design disciplines the transformation of the requirements into a specification takes place. Additionally the ‘use cases’ are transformed into specifications. During this discipline the system should be kept flexible (make changes possible).

4.3 RUP SOMA

The SOMA methodology has an analysis and design method that is used for the design and construction of a SOA solution by enabling target business processes. This method is an extension of the RUP methodology. SOMA accomplishes this through the identification, specification, and realization of services, components, and flows.

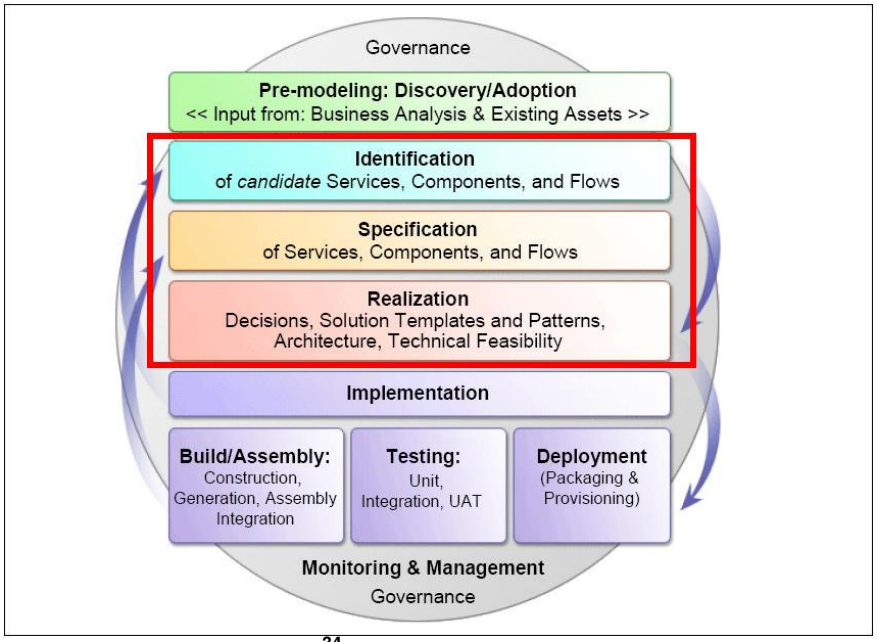

Figure 5 RUP SOMA model[34]

The difference with the traditional RUP is visualized in the red box. In this box the service identification and specification and realization are specified and change the design discipline of traditional RUP. These disciplines are described in the next sections.

4.3.1 Service identification

The goal of service identification is to create candidate services that are meaningful to the business. There are three basic techniques to identify the services[35]. Identify services through iteratively creating KPI(key performance indicators) and goals for the business. Identify services by decomposing the system into major functional areas and subsystems. At last analyze existing asset

[34] Case study: SOA design scenario , IBM red books, 2008, pag 8, http://www.redbooks.ibm.com/redpapers/pdfs/redp4379.pdf
[35] Case study: SOA design scenario , IBM red books, 2008, pag 10, http://www.redbooks.ibm.com/redpapers/pdfs/redp4379.pdf

through identifying services by using a bottom-up strategy for maximizing the reuse of existent assets.

4.3.2 Service specification

The service specification defines the dependencies, composition, exposure, decisions, messages, quality of service constraints, and decisions regarding the management of state of a service. The services specifications include:

- applying the service litmus test to make exposure decisions.
- Identifying services dependencies, like functional and processing dependencies.
- Identifying services compositions and flow, describing the choreography of the services.
- Identifying non-functional requirements, to specify the quality of services (QoS).
- Specifying service messages, to specify data types.
- Documenting state management decisions for performance reasons
- and at last creating a service component specification.

4.3.3 Service realization

In this phase decisions are made about how each component realizes its functionality. This is divided into three subtasks:

- Allocate service to components in an iterative fashion.
- Identify constrains to be sure that service realization is possible.
- Allocate the components to a SOA reference architecture.

4.4 AIM

This section describes the AIM methodology. AIM is methodology developed by Oracle to analyze, design and implement package application. All AIM tasks are organized into processes that group related deliverables together. Project team members are assigned to these groupings according to their specialization and background.

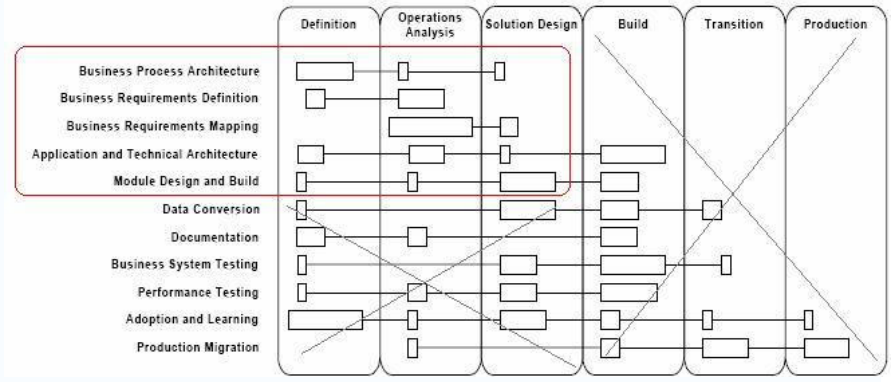

Figure 6 The AIM model. The rows are processes, the columns are phases, and the sizes of the blocks define the duration of a process.

4.4.1 Processes

The following sections describe the processes of AIM which are in the scope of the thesis. But first I give a definition of an AIM process: *A discipline or sub-project that defines a set of tasks related by subject matter, required skills and common dependencies. A process usually spans several phases in an approach.*

Business Process Architecture

Business Process Architecture addresses the need to understand organization business processes and alignment with business requirements and target applications. The team analyzes business processes to determine the degree of change required to bring them into alignment with organization business objectives, and designs new or improved business processes. The result is a set of optimized high-level designs that balance application and organization change.

Business Requirements Definition

The Business Requirements Definition process defines the business needs that must be met for the successful implementation of the application and technical suite. The project team documents business processes by identifying business events and describing the steps the organization takes to respond to those events. The team then organizes these processes into business scenarios that capture the organization's business requirements.

Business Requirements Mapping
The Business Requirements Mapping process produces and documents an acceptable, feasible solution to business requirements. As gaps between requirements and functionality emerge, the team resolves the gaps by documenting alternative solutions, designing application extensions, or changing the underlying business process.

Application and Technical Architecture
During the Application and Technical Architecture process, the project team designs an information systems architecture around the organization's business vision. Included are Oracle, third-party and custom applications; computing hardware; and networks and data communications infrastructure.

Module Design and Build
The Module Design and Build process produces custom application extensions to fill gaps in functionality identified during Business Requirements Mapping. Custom systems include program modules (forms, reports, alerts, database triggers, and so on) that must be designed, built, and tested before they can be incorporated into the new system. Module Design and Build addresses the design and development of the custom modules; the Business System Testing process supports testing of custom modules.

4.4.2 Phases

This section describes the phases of AIM, which are in the scope of the thesis. A phase can be seen as an element in time where different related task can be delivered. A next phase can only be started as the previous phase is finished.

Definition
During Definition, the project management team plans the implementation project. The goals are to identify business and system requirements, to propose the future business model, and to propose the application and information technology architecture. The project team is organized and oriented. The team develops a learning plan to ensure team members receive training and support necessary to perform their roles on the project.

Operations Analysis
During the Operations Analysis phase, the project team collects management, technical, and end-user business process information and requirements. The project team develops business requirements scenarios used to assess the level of fit between the detailed business requirements and standard application functionality. The project team also creates a model for the application structure and suggests an overall technical architecture. Prototyping of business processes may begin in this phase to aid in analysis and demonstrate feasibility of design options.

Solution Design
The goal of Solution Design is to create the optimal business process solution to meet your future business requirements. During Solution Design, project team members design application configuration options and detailed business procedure documentation. Detailed design of any custom extensions interfaces and data conversions occur during this phase. The team also identifies process and organizational changes required for implementation.

5 Framework IAF

This chapter describes the IAF framework. I have chosen this framework, because this framework gives relatively straightforwardly a good view on how SOA architectures can be designed and has made high success rates in previous architectural work.

Capgemini started developing its architectural approach in 1993 and has steadily evolved a framework around it, called the Integrated Architecture Framework (IAF). This goes far beyond Technical, Software or Systems Architecture. Capgemini views architecture as providing a comprehensive and coherent view across Business, Information, Systems and Technology; not just to guide the design of IT systems but to deliver business change supported and enabled by IT. Architecture is therefore a great help for transforming an organization. The Integrated Architecture Framework is used to structure and define the architecture content, provides a model for architecture development and usage and describes format and content of elements in the architecture. The elements are specified in relation to each other.

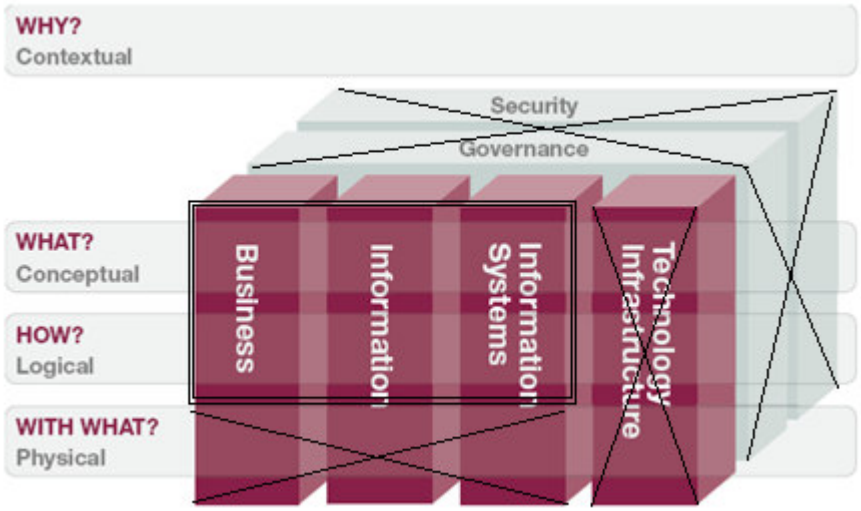

Figure 7 IAF model

Figure 7 IAF model shows the basic structure of IAF. The model is broken down into aspect areas and abstraction levels. Each “cell” in this model has a defined set artifact. Views then allow the architects to bring together and visualize the artifacts to help modeling the architecture and to communicate the architecture with the various stakeholders. Of this model I describe only the contextual layer and part of the conceptual and logical layer, because only those fit in the scope of the thesis.

5.1 Abstraction Levels

Abstraction allows a consistent level of definition and understanding to be achieved in each area of the architecture, and is especially useful when dealing with large and complex architectures, as it allows for all relevant issues to be identified before further detailing is attempted. This approach is found in most

architectural approaches including The Open Group Architecture Framework (TOGAF) and the Zachman Framework for Enterprise Architecture.

IAF defines four abstract levels:

- The contextual level
- The concept level
- The logical level
- The physical level (out of scope)

The abstraction levels should not be interpreted as series of process steps to be followed one after the other but as a mechanism to support the categorization of the artifacts and how they should be interpreted. In practice the boundaries between the Abstraction Levels is not rigid and refinement between abstractions levels is normal and appropriate.

5.1.1 The Contextual Level

The contextual level is characterized by the "Why?" question. It is not about understanding what the new architecture will be, but identifying the boundaries for the new architecture and its context. Specifically this level focuses on the business aspirations and drivers and captures the Principles upon which the architecture will be based. Unlike some approaches, these Principles are formally described to include their rationale, implications and priority, so that they reflect the often conflicting requirements of stakeholders in a business. Key artifacts include Architecture Scope, Architecture Objectives, Assumptions, Constraints, and the Architecture Principles.

5.1.2 The Conceptual Level

The Conceptual Level is characterized by the "What?" question. The requirements and objectives are decomposed, ensuring that all aspects of the scope are explored, that relevant issues identified and these issues are resolved without concern about how the architecture will be realized. The Conceptual Level outcomes are representations of the requirements in architectural terms of descriptions of "elements of behavior" and their inter-relationships.

5.1.3 The Logical Level

The Logical Level is characterized by the "How?" question. The Logical Level is about finding the ideal solution in an implementation independent manner. From this, several "solution alternatives" can be developed that either provide the same outcome, or alternatively "test" different priorities and scenarios to understand the implication of different potential outcomes. The outcome of the Logical Abstraction Level is an agreed "desired state to be achieved" representation of the architecture that is implementation dependent.

5.2 Aspect Areas

To break down the complexity of the Architecture, IAF recognizes six "Aspect Areas" four of which focus exclusively on the core aspects of the overall architecture; Business, Information, Information Systems and Technology Infrastructure. The Business aspect area is especially important, as this not only

allows modeling of the business to deliver IT solutions, but also provides a way to Architect the Business. The remaining two aspect areas specifically address the disciplines of Security and Governance. These are deemed special because they:

- Represent a set of requirements driven across all core aspect areas
- Apply to all the other aspect areas but may be applied stand-alone
- Use artifacts from the other aspect areas to describe themselves
- May significantly change the architecture structure across one or more core aspect areas.

5.2.1 The Business Aspect Area

The Business Aspect Area describes the business architecture in terms of business sub aspects representing business goals, business activities, roles and resources. The outcome of the Business Aspect Area is typically a series of business architecture components that describe process, organization, people and resources.

5.2.2 The Information Aspect Area

The Information Aspect Area describes the information the business uses, the information structure and relationships. The outcome from the Information Aspect Area is typically an information architecture and a series of business information components that describe what and how information is used and flows around the business. Key artifacts include the Information Object, Business Information Service and Logical Information Component.

5.2.3 The Information System Aspect Area

The Information System Aspect Area describes the information systems (packaged or bespoke) that will automate and support the processing of the information used by the business. The outcome from the Information System Aspect Area is typically a series of Information System architecture components that describe how the information systems will be used to support the automated aspects of the information architecture and business information architecture components. Key artifacts include IS Service, Logical IS Component and Physical IS Component.

5.3 Views

Views are used to present the architecture to the various stakeholders by defining the architecture from different viewpoints. Views can help the architects in the development of the architecture by allowing them to inspect and validate the whole architecture from one perspective (viewpoint) to bulk a complete, composite and consistent picture of the solution across all areas. Views are highly context dependent, as their applicability and use will depend on the client, the stakeholders and the engagement type. The views defined in IAF are:

5.3.1 Models and Cross-Reference Views

There are two techniques for linking services/components in IAF. Interaction Models capture links between artifacts of the same type, helping to identify interfaces and the collaboration contracts needed. Cross-reference views capture the connections between different types of artifact (e.g. Business to IS) providing the traceability back to the business needs.

5.3.2 Information Ownership View

This view provides the information objects related to their ownership (by roles).

5.3.3 Major Information System Interfaces Model

This view visualizes on a high definition level the Information System Service and their interaction between them.

5.3.4 Integration View

This view visualizes the logical components from the perspective of collaboration contracts, highlighting the different integration approaches needed.

5.3.5 Distribution View

This view visualizes the distribution of components of the solution, typically across topographical or geographic areas such as data centers and office locations.

5.4 Solution Alternatives

In determining the most appropriate architecture solution, it is critical that different options are considered—and the rationale for specific design decisions captured. Solution Alternatives are the way in which this is achieved within IAF.

Solution Alternatives are used in the Logical and Physical levels to work through the implications of different options, either architecture-wide or at the major component (or sub-system) level. The Solution Alternatives are often used to demonstrate the impact of conflicting needs expressed by the business (e.g. 24x7, secure and very agile).

The selection of the various candidate Solution Alternatives and the identification of the “best fit” solution is driven from the Principles originally defined and priorities in the Contextual level and refined throughout the rest of the architecture process. This ensures that the solution, and the design decisions, reflect and are traceable back to the business requirements.

It is expected that at least two Solution Alternatives will be considered in both the Logical and Physical levels.

5.5 Service Artifacts

In IAF artifacts are defined as the work products of the framework. This can be divided into groups where the aspect areas intersect with the disciplines. In this

section the important artifacts are described in the scope of the research. First a definition of what a service is will be given to understand the basics of the artifacts.

A Service describes an "element of behavior" in the IAF Aspect Area it is applicable to. Services are the fundamental building blocks of the architecture in IAF. Services interact with one or more other like services and the interaction relationship is described through a Service Collaboration Contract. The following list describes the artifacts within IAF on the contextual level[36]:

- Business Service - A Business Service characterizes a unique "element of business behavior" in terms of a business activity, undertaken by a specific Role that together support a specific Business Goal.
- Business Information Service - *A* Business Information Service describes the communication behavior of a business Service.
- Information System Service - An Information System Service describes an element of behavior of information automation required to support automated Business Information Services.
- Technology Infrastructure Services - Technology Infrastructure Services describe the behavior of services that primarily support the Information System Services and may also directly support generic business objectives (for example Office Automation type services).

The following list describes the artifacts within IAF on the logical level[36]:

- Logical Business Information Component - A Logical Business Information Component is a grouping of Business Information Services that represent the structure of the communication aspects of a business.
- Logical Information Component - A Logical Information Component describe the structure of Information Objects that support the architecture solution.
- Logical Information System - A Logical Information System Component is the basic element of an "ideal" or "To Be" Information system structure created by the grouping of one or more Information System Services.
- Logical Technology infrastructure Component - A set of Technology Infrastructure Services that describe an implementation independent technology component (a logical type of 'box' or 'wire') that conforms to the characteristics of its associated Technology Infrastructure Services in accordance with pre-determined grouping criteria.

[36] Integrated Architecture Framework Version 4, reference Manual, Colin Metcalfe, October 2006

6 Maturity model

This chapter describes the maturity model of the Sonic Company. The maturity model is designed to show the increasingly positive impact of SOA adoption from a business perspective. It provides IT decision makers with a framework for benchmarking the strategic value of their SOA implementation, and a model for visualizing future success.

The SOA MM derives from three key points of inspirations[37]:

- Feedback provided to 2,000 architects and developers, who over the past year have attended one of Sonic's Architect Forum events in North America, Europe and Japan.
- Industry analyst reports, such as those by Randy Heffner of Forrester Research[38], showing the various successful paths taken by companies introducing SOA.
- The successes of the Capability Maturity Model®[39] (CMM) and newer CMM Integration (CMMIsm) from the Software Engineering Institute (SEI) in providing a common framework for defining and assessing process improvement in software and other engineering endeavors.

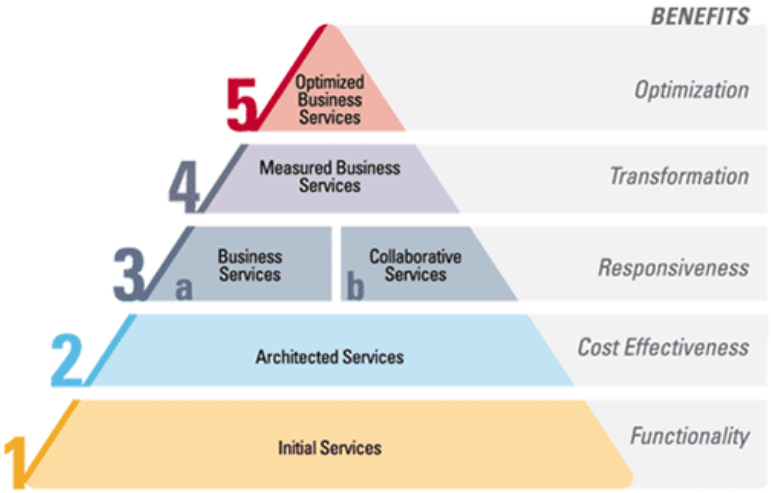

Figure 8 Sonic Maturity Model[37]

[37] Movin' SOA On Up, Jon Bachman, September 2005, http://www.sonicsoftware.com/solutions/learning_center/soa_insights/movin_soa_on_up/index.ssp

[38] Heffner, Randy, "Your Paths to Service-Oriented Architecture", Forrester Research, Dec. 2004.

[39] Software Engineering Institute, Capability Maturity Model® Integration, August 2006, http://www.sei.cmu.edu/cmmi

6.1 SOA Maturity Model Level 1

SOA Maturity Level 1 is Initial. Initial Services represent the initial learning and initial project phase of SOA adoption. Projects here are typically done to simultaneously meet a specific need to implement functionality while trying out specific technologies and an approach to SOA. This maturity level also includes initial R&D activities testing the SOA technologies in a laboratory environment. Usually, the initial introduction of SOA is driven by the application development organization - often as part of an application integration project. New development skills are learned and initial attempts at quantification of ROI are created.

6.2 SOA Maturity Model Level 2

SOA Maturity Level 2 is Architected Services. It is at this level that standards are set as to the technical governance of SOA implementation, typically under leadership of the architecture organization. The key business benefit of this level is development and deployment cost reductions through the use of SOA standard infrastructure and components as compared to using older technologies or costs accumulated through multiple unique one-time projects. These benefits are greater in the heterogeneous environments typical of most enterprises.

6.3 SOA Maturity Model Level 3

The focus of SOA Maturity Level 3 is on the partnership between technology and business organizations in order to assure that the use of SOA provides clear business responsiveness. Core to the value of SOA is the linkage between business process and digital processes. SOA Maturity Level 3 is defined with two complementary paths to attaining these goals - one, Business Services, focused on the improvement of internal business processes, and one, Collaborative Services, focused on the improvement of collaborative processes with external partners.

6.4 SOA Maturity Model Level 4

While SOA Maturity Level 3 focuses on the implementation of internal and/or external business processes, SOA Maturity Level 4 focuses on measuring and presenting these processes at the business level so as to provide continuous feedback on the performance and business impact of the processes implemented at Level 3. This level includes business activity monitoring to allow business users to transform the way they respond to business events.

6.5 SOA Maturity Model Level 5

SOA Maturity Level 5, Optimized Business Processes SOA, adds automatic response to the measurements and displays of Level 4. In this way, the SOA information systems becomes the "enterprise nervous system" and takes action automatically according to events occurring at the business level according to rules optimizing business goals.

This SOA Maturity Model provides a framework for discussion between IT and business users about the applicability and benefits of SOA in an organization across five levels of adoption maturity. The goal of its authors is not only to provide a means for organizations to benchmark current implementations, but

also to provide a guideline for current IT leaders visualizing a path to successfully advance the value of SOA for their organizations.

7 SOA principles mapped on the methodologies

This chapter describes how the SOA principles of Thomas Erl can be mapped on the RUP and the AIM methodology for creating a SOA solution. In this chapter guidelines are described for these methodologies to design a SOA solution with extra emphasis on the reusable and composable aspects of services.

7.1 RUP[40] and SOA principles

This section describes how the SOA principles from Thomas Erl can contribute to the RUP methodology. There are two version of RUP considered in this thesis. The first traditional version of RUP is a software engineering process, which provides a disciplined approach to assigning tasks and responsibilities within a development organization. The second version of RUP, named RUP SOMA, is specially made for SOA solutions as shown in figure 9. This chapter describes a mix of both methods, because nowadays enterprises over the world change their architectures from a non-SOA infrastructure to a SOA infrastructure. The guidelines identified for the RUP methodology are based on the RUP methodology literature and other scientific literature and the interviews I had with a RUP expert, G. Kuijpers and a SOA expert J. van der Vlies.

[40] Rational Unifed Process V7, 2007, http://deliver-rendering.capgemini.com/components/RUP_V7_Large_Projects/index.htm

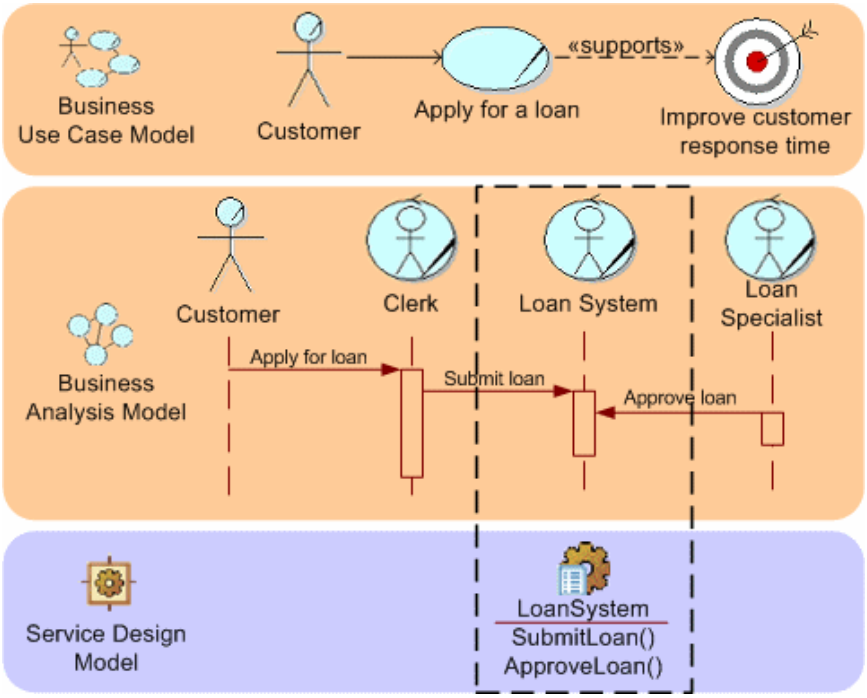

Figure 9 RUP service model[41]

In the RUP methodology a clear distinction between business and IT is made to first identify the business goals and get the essence of the business and then translate them into its assets. For the alignment of business and IT the same modeling techniques are used. The sections in this chapter describe only the most important diagrams and models, where a clear view is shown on how the SOA principles can be mapped on the RUP methodology. An impact analysis on which models are most affected is done by architects I interviewed for the selection of the models. First the business use case model is described, because this model is the basis model of the RUP methodology and describes the basic functions for an enterprise. The next model, the analysis model, describes the analysis for the communication between the different basic functions. The next model the use case design model finalizes this discipline with a design for information objects supporting these basis functions. Finally the design component model in combination with the service use cases model form the basis for the composition of the basic functions together to get a coherent service network. For each of these models we try to identify how the SOA principles change them and describe guidelines on how to apply these principles in these models.

[41] RUP 2007, Capgemini, http://deliver-rendering.capgemini.com/components/RUP_V7_Large_Projects/rup/capabilitypatterns/identify_services,_WnOpAEocEdqrjq4i3fchvA.html

7.1.1 Business Use case model

The business use case model describes roles and their actions and serves as a contract between the customer and the developers. The business use cases define for each role their objectives in the system. Figure 10 shows an example of a business use case model. This model gives a first impression about how the new system will look like and creates the benefits to identify, from a user central point, the functions of the system. The user can be seen as the service consumers. This model visualizes the enterprise on the highest level of abstraction and will only display high level processes. The following guidelines on how to apply the SOA principles are determined by interviewing experts on the field of RUP and the literature described by Thomas Erl and for designing business use case models:

- Service reusability – Overlapping functionality should be detected during the creation of the use case model. To create as much reusability as possible the reusability principle of SOA must be considered, which includes the grouping of the use cases according to shared goals and the creation of a decomposition of the system[42].
- Service Composability – In this model the first impressions about where services should be composed is necessary by decomposition of the system. Important for this decomposition is to design service as single tasks for the business.

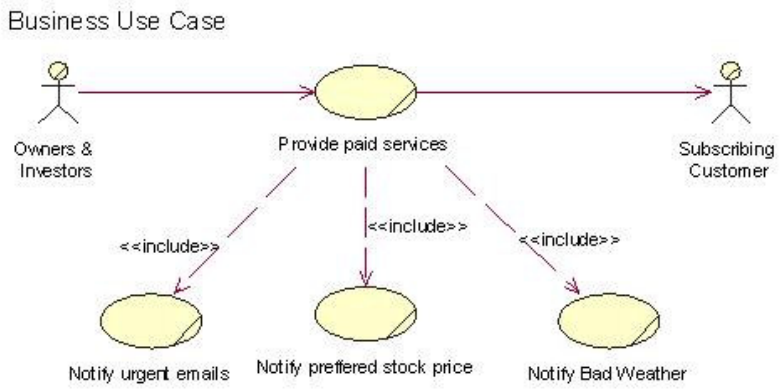

Figure 10 business use case diagram[43]

7.1.2 Business analysis model

The next model the business analysis model describes the following items:

[42] Case Study: SOA Design Scenario, John Ganci, IBM, 2008
[43] Business Use Cases versus System Use Cases, Process Wave, 2006, http://www.processwave.net/Essays/BusinessUCVsSystemUC/BusinessUC_versus_SystemUC.htm

- Business Event
- Business Use-Case Realization
- Business Worker
- Business Entity
- Business System

In this thesis only the details of the business use-case realization are described, because in that item the models are major influenced through the SOA principles. The business use-case realization will detail the business use case. This model is divided into the following models where SOA principles can be applied:

- Sequence diagrams
- Activity diagrams
- Analysis classes

Business sequence diagram
A sequence diagram graphically depicts the details of the interaction among business workers, business actors, and how business entities are accessed, during the performance of a business use case. A sequence diagram briefly describes what the participating business workers do, and how the business entities are manipulated, in terms of activations, and how they communicate by sending messages to one another. Figure 11 shows an example of sequence diagram.

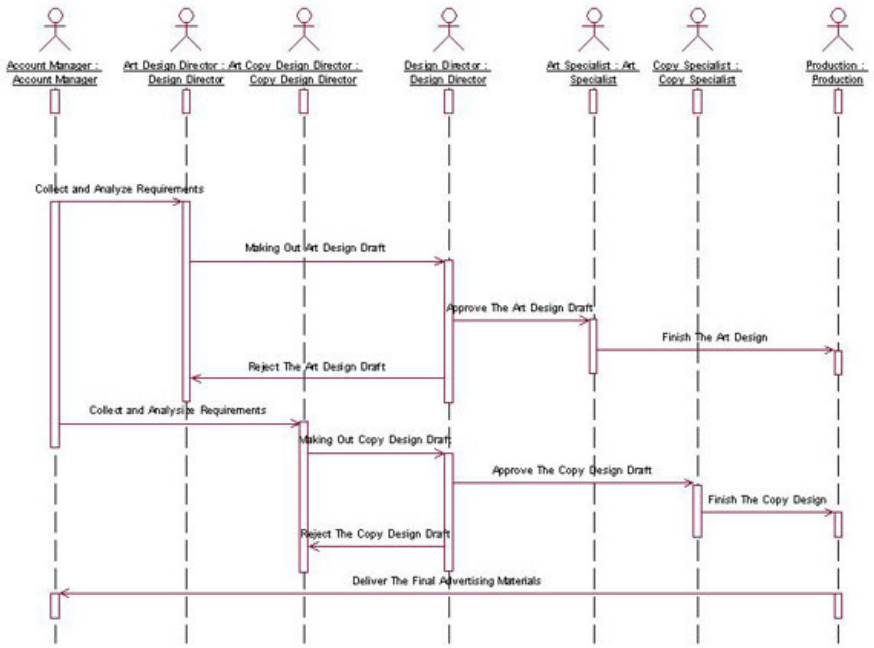

Figure 11 sequence diagram[44]

From this figure and the definition of the sequence diagram we can identify the following important guidelines when applying the SOA principles on the RUP methodology:

- Standardized service contracts – These diagrams are a detailed version of a process and make the service contracts visible from the inside and outside of the application. The following points should be considered:
    - Use standard market reference models for the creation of the sequence diagram for standard processes where possible.
    - Create documentation about which processes are the added value processes of the application and determine for them how they can be designed according to current best practices or with the aid of design patterns.
    - The basic idea behind the sequence diagram is to setup the construction for service contracts. The visualization of a line between two actors represent a service contract, where in this model the service consumer and service provider are determined.

[44] A guided tour to business process modeling and implementation in DB2 Content Manager for AD Company case, Xiao Li, 03 August 2006, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608li/7.jpg

- Define standards about the information within a communication line.

- Service reusability – During the design of the business sequence diagram reusable parts can be identified through coupling the sequence diagram to the class diagram, where different actors can use the same methods.
- Service composability – In the sequence diagram the progress of the process is determined. Create this progress as flexible as possible, so that change can be easily implemented.

Business Activity diagram

An activity diagram of a business use case realization explores the ordering of tasks that accomplish business goals, and that satisfy commitments between external business actors and internal business workers. A task may be manual automated and completes a unit of work.

Activity diagrams help:

- Provide a rationale for and understanding of the introduction of information systems into the business.
- Establish objectives for system development projects to implement business transformation initiatives.
- Justify automation investment based on detailed business process metrics.

An activity diagram with swim lanes and object flows focuses on how you divide responsibilities into classes. Activity diagrams are often used as an elaboration of a sequence diagram and zooms in on the activities within one process.

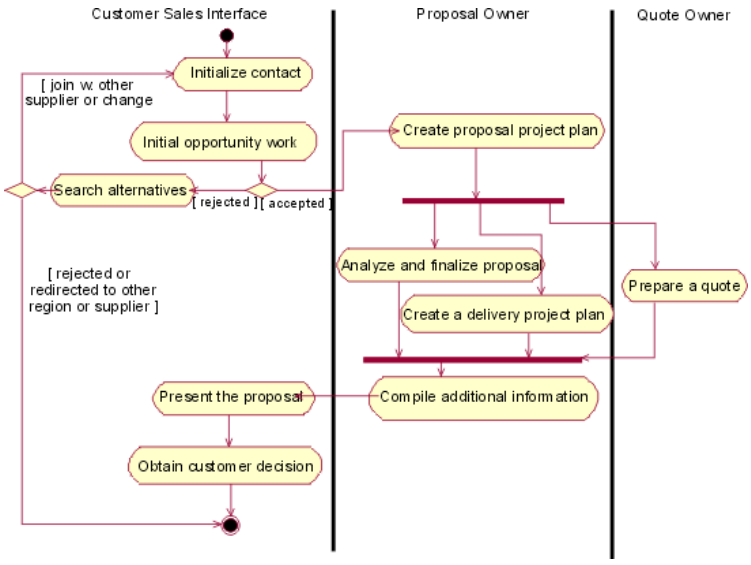

Figure 12 an activity diagram [40]

The following guidelines are identified on how the SOA principles are mapped on the RUP methodology for defining activity diagrams:

- Service reusability – During the creation of the activity model overlapping functionality should be detected. Define the activity diagram for one functional context and describe activities, which can be used by multiple actors.
- Service composability – Create the activity diagram in such a way that it can be used as a mapping with the support of business services and create the activities as flexible as possible.
- Service autonomy- Define a functional boundary for the activities that should not overlap with other activities.

Analysis classes
Analysis classes are used to capture the major responsibilities in the system. They represent the prototypical classes of the system, and are a first view on the major abstractions. Analysis classes include the design classes and subsystems of the system.

The following guidelines are identified for the mapping of the SOA principles on the RUP methodology for defining the analysis classes:

- Standardized service contracts – The first impressions about the service contracts are constructed in this model. Standardizing these service contracts in this phase is needed for a fundamental basis. Define standard stereotypes for the classes to categorize them to allow change within one category.
- Service reusability –This model creates the objects as generic as possible and define them in a functional context.
- Service composability – Keep the object classes as generic as possible to guarantee the system flexibility by allowing different data types exchange in the service contracts.

7.1.3 Use case design model
The design model is an abstraction of the implementation of the system. It is used to conceive as well as document the design of the software system. It is a comprehensive, composite work product encompassing all design classes, subsystems, packages, collaborations, and the relationships between them. The following models are described in this model where the SOA principles can be applied with:

- System use case diagram
- Activity diagram
- Sequence diagram
- Class diagram
- Service design model

System use case diagram
The system use case diagram has the same functionality as the business use case diagram, it only describes the process from an IT viewpoint. The actors in this model can be seen as the service consumers.

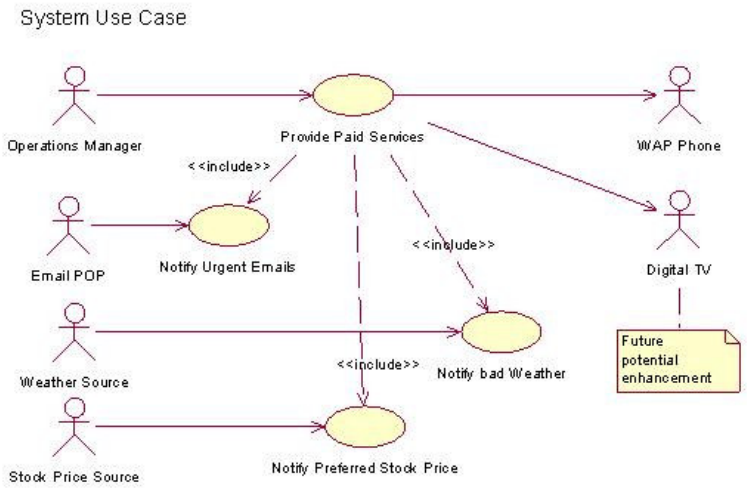

Figure 13 system use case model[43]

The following SOA principles should be kept in mind when defining the system use cases:

- Service reusability – During the creation of the use case model overlapping functionality should be detected. Group the use cases according to shared goals and create a decomposition of the system.
- Service Composability – In this model the first impressions about where service should be composed is necessary. Grouping the use cases for decomposition and shared goals is used to gain control over the composability.

System activity diagram

The system activity diagram has the same functionality as the business activity diagram, it only describes the process from an IT viewpoint. The following SOA principles should be kept in mind when defining activity diagrams:

- Standard service contracts - The following guidelines should be considered when making this diagram:
    - Create documentation for standard business processes, which can be based on reference market models or design patterns.
- Service reusability – Identify processes with shared goals to look for reusable parts.
- Service composability – During the creation of the activity model documentation should be set up on how service should be orchestrated. For creating this document, you can consider the BPEL specifications. This includes documentation about how services compositions can be reused. Documentation should be set up about the clustering of the services, where autonomous groups of services should be created. This

can lead to outsource possibilities, one of the advantages of using an SOA structure.

System sequence diagram

A sequence diagram graphically depicts the details of the interaction among business workers, business actors, and how business entities are accessed, during the performance of a business use case. A sequence diagram briefly describes what the participating business workers do, and how the business entities are manipulated, in terms of activations, and how they communicate by sending messages to one another. Figure 10 shows an example of a sequence diagram.

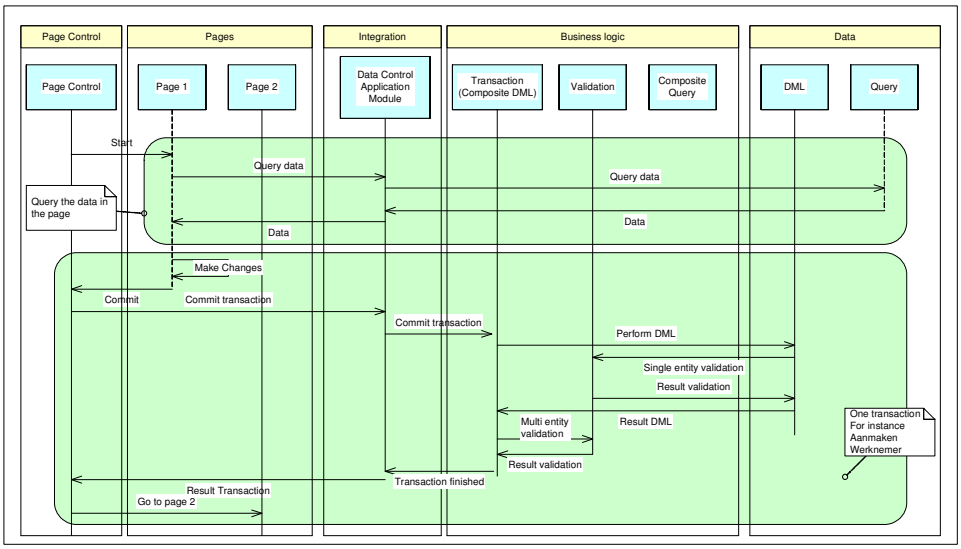

Figure 14 sequence diagram[45]

From this figure and the definition of the sequence diagram we can identify the following important SOA principles:

- Standardized service contracts – These diagrams are a detailed version of a use case diagram and make the service contracts visible from the inside and outside of the application. The following guidelines should be considered:
    - Create documentation about which processes can be seen as standard processes, where market reference models can be used for implementation.

[45] A guided tour to business process modeling and implementation in DB2 Content Manager for AD Company case, Xiao Li, 03 August 2006, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608li/7.jpg

  - Create documentation about which processes are the added value processes of the application and determine for them how they can be designed according to current best practices or design patterns.
  - Create documentation about the input and output variables between the different services, where data types should be standardized across the current enterprise.
  - Define standards about communication objects between different roles.

- Service loosely coupling – In these diagrams separation of different levels can be imported for creating loosely coupled services. The separation of these layers can best be based on reference architectures, like the OASIS reference architecture[46]. This increases the ability to maintain and govern the system.
- Service composability – Services should be made as generic as possible to create flexibility in the system. This means the creation of services which allow generic input and output data to be more adaptable to other services and the creation of an integration layer to make services connectable to all kinds of services through this layer. This layer can be based on the canonical data model[47].
- Service reusability – Create reusable services to connect the design of the sequence diagram to a service repository or to a class diagram. Apply design patterns for promoting reuse.
- Service autonomy – Define functional boundary that should not overlap with other services by layering the sequence diagram. This creates services with a high level of control over their own functions.

During the interviews I notice that the sequence diagram is a very important diagram. Through the introduction of a layered approach the sequence diagram can be structured and every layer can be separately designed. The reuse of services can be improved by reusing the class model into the sequence diagram. The creation of a utility layer helps to identify reuse of service across domains as well. Issues in the creation of the sequence diagram are the determination of the granularity of the services and the determination of where exactly the control of the integration layer should be applied. Guidelines about this are to create as much flexibility in the system as possible, and keep the integration layer as separate as possible.

[46] OASIS reference model, Duane Nickull, 2008, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
[47] Canonical data model , Gregor Hohpe , 2003, http://www.enterpriseintegrationpatterns.com/CanonicalDataModel.html

System Class diagram
Class diagrams show associations, aggregations and generalizations between business workers and business entities. The following kinds of class diagrams might be of interest:

- Inheritance hierarchies.
- Aggregates of business workers and business entities.
- Diagrams explaining how business workers and entities are related by means of associations.

Class diagrams show generic structures in the business domain model, but can also be part of the documentation of a business use case realization by showing it participating business workers and business entities. Figure 10 shows an example of a class diagram.

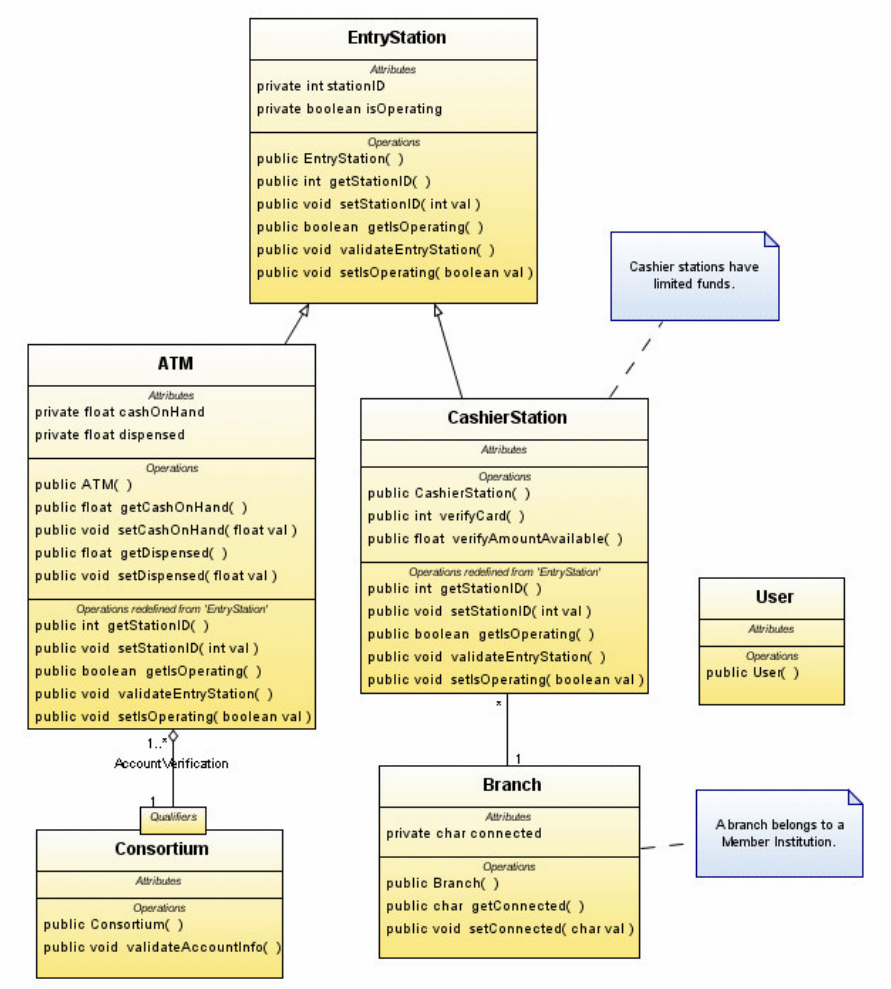

Figure 15 a class diagram[48]

From this figure and the definition of the class diagram we can identify the following important SOA principles:

- Standardized service contract – During the creation of the class diagrams the service contract are becoming more and more important through capturing the essence of the system. The following guidelines should be considered from a SOA viewpoint[49]:
    - Define standard service contracts using interfaces.
    - Make standard variables name for the input and output of services across the enterprise.

[48] UML modeling: Creating a class diagram, SUN Developer Network, 2008, http://developers.sun.com/jsenterprise/learning/tutorials/jse8/uml_class_diagram.html
[49] Interview Ruben Spekle, Capgemini architect Oracle

  - Define standard data types across the enterprise.
  - Define policies between the different services which are documented and described in protocols, like WS-policy and WS-reliable messaging.

- Service reusability - Through the construct of a service repository the management for services can be simplified. To promote reuse between classes the following concepts can be used[50]:
  - Super Class and Sub Class
  - Inheritance
  - Class Library
  - Multiple Inheritance
  - Abstract Class
  - Class Inheritance and Interface Inheritance
  - Delegation and Object Composition
- Service composability – Keep the object classes as generic as possible to guarantee the system flexibility by allowing different types of data exchange in the service contracts. Design the class for extensibility and for multiple actor roles. Tools which generated automated meta data can be used to create composability information during the creation of the class diagram to standardize and increase the use of composability.
- Service autonomy – The class diagram is modeled as a pre for the service implementation. For services to carry out their capabilities consistently and reliably, their underlying solution logic needs to have a significant degree of control over its environment and resources. The principle of service autonomy helps to discover isolation levels and normalize services to achieve a suitable measure of autonomy.

SOMA Service design Model

In this phase services are identified and specified and realized as described in chapter 6. For each of these phases I identify guidelines for applying the SOA principles:[42]

Service identification

- Service Reusable – In this phase the existing service is evaluated and maximization of occurrences is attempted. Grouping services with the similar goals can also increase the amount of reusable services.

[50] Basic concepts on reuse, Koichiro Ochimizu, 2005 (presentation)

- Standardized service contracts – This phase evaluates services contracts in the existing architecture. If there is a defined standard they can be reused for the new created services in the new architecture.
- Service discoverability – In a SOA architecture the services should be registered with a register to increase the amount of discoverability. This register should be maintained through the whole enterprise and discovering the needed service must be a very time-intensive task.

Service specification

- Standardized service contracts – In this phase the specification of the contracts are made. This should be done with standard procedures and a lot of care to create an aligned service contracts across the whole enterprise.
- Service Autonomy - Through the decomposition of the system the autonomy of the services are more secured.
- Service Composability – During the service specification the messaging and coupling languages are determined. This creates the possibility to optimize the composability of services according to the best practices. Services should be specified as a single task for increased composability.
- Service loosely coupling – In the specification the granularity of the services are determined. There is still much discussion about which granularity is the best, but to create services as much loosely coupled as possible is one of the guidelines in the granularity discussions.
- Service stateless – When documenting the state management of services, design as much stateless services. This includes design of services with interpretive programming routines and the definition of highly business process logic services.

Service realization

- Service composability – In this phase the services should be clustered to maximize the subsystem connectivity in iterative fashion. Using a reference architecture can be very helpful here.

7.1.4 Design component model

The design component model describes how the modules of enterprise software are connected. In this model the interfaces of different components are described and information about which communication is needed between the subsystems of an enterprise system is created. In this model layering techniques can be used to maximize the separation of concern principle and create service autonomy.

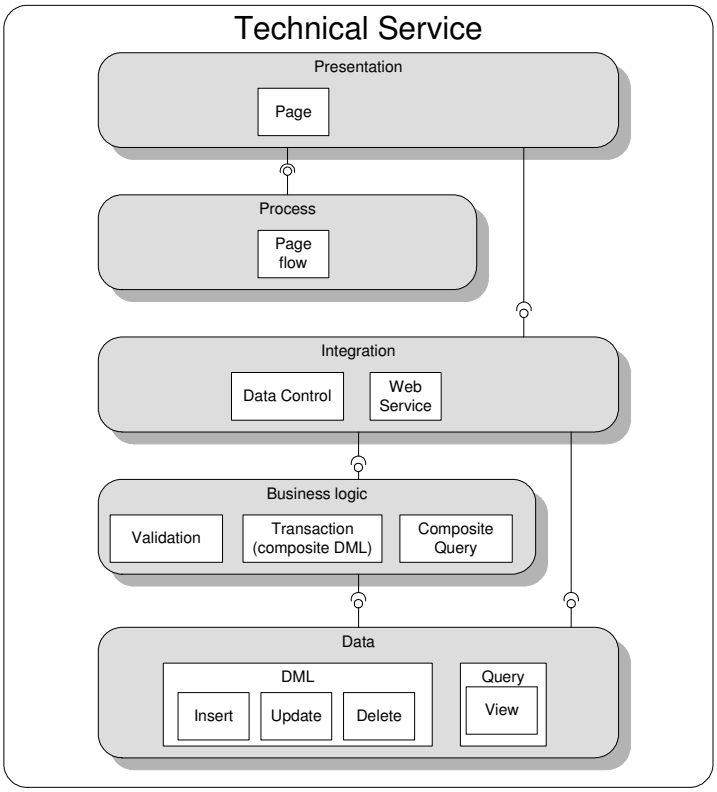

Figure 16 a component model [44]

From this figure and the definition of the model the following SOA principles can be marked as important for this model:

- Standardized service contract – In this model the service contract between different subsystems is described. Thoughts should be made about which technology can be used for the different layers. Standards for each layer can be the basis for the implementation.
- Service Composability – This model describes in different layers how services should be composed. The composability principle should be kept in mind to make the service as composable as possible, to make the system adaptable. In this model the services are clustered for each layer and governance and security issues can be monitored easily in this way.

7.2 Mapping RUP onto the IAF framework, considering the SOA principles

This section describes how the IAF framework can influence the RUP methodology for building a SOA solution. This is an example of how an architecture framework can help to model the analysis and the design of a SOA solution. The most important idea behind this framework is to separate the 'what' from the how. This means that it is important to first look at what a business should be doing in a functional way and finalize your design with how you are going to do this. IAF can be of great help to support a design methodology to sharpen their view on the problem. Figure 14 describes the mapping of the IAF model on the RUP methodology. This figure describes how the IAF framework can be used as input for some of the models used in the RUP methodology. The

following list describes per RUP model, which artifacts from IAF can help to support a SOA solution in the RUP methodology.

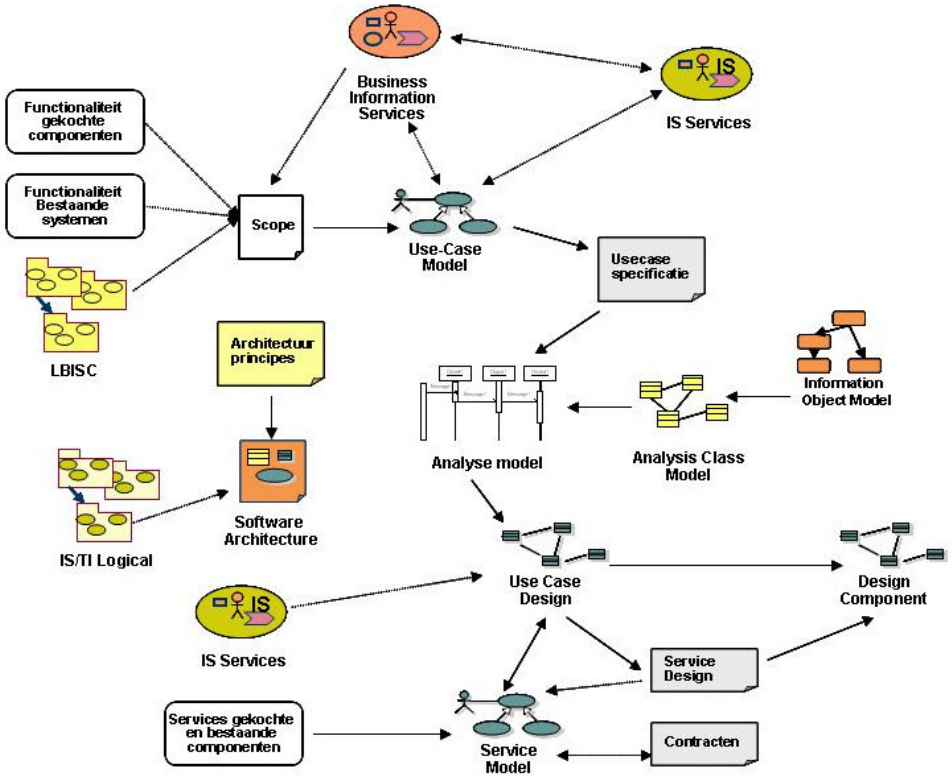

Figure 17 IAF dependency on RUP[51]

Use case model

The business information services in collaboration with the information system services can be used as input for the business use case through the identification of the business goals, business roles, business activities, business interaction models. The advantages of using the business information services for the input for the use case models are[52]:

- They are specified initially by attributes (Quality of service, Output and Throughput characteristics/metrics) which form the basis for the use case specifications
- The set of services contributes to the scope (mission) completely so they are infinitely denumerable

Concerning this from a SOA viewpoint, higher service autonomy is reached by specifying the attributes. The clustering of the business services and the

[51] Modellencyclus IAF/RUP for SOA, Joost van der Vlies, Capgemini, 2007
[52] Software engineering and architecture, Robert de Boer, Capgemini, 2004 (internal presentation)

information services can add value to the reusability and the composability principle.

Analysis model
The information object models are the basis for the analysis model by describing the external relations between different services.

The advantages of using the information object models for the input for the analysis models are[52]:

- They are related to functions and therefore needed as long as the mission is stable.
- They consist only of existential relations and are therefore future proof.

Concerning this from a SOA viewpoint, the information object models contribute to the standard service contract principle by contributing to the definition of the functions and communications of a service on a standard matter. The information model describes only the existential relations which are supported by the service abstraction principle.

Use case design model
The logical information system component in collaboration with the technical infrastructure component can be used as input for the use case design model. The advantages of using the logical information system component in collaboration with the technical infrastructure component for the input for the Use case design model are[52]:

- They cluster the logical services into logical groups
- They determine the starting point of the components of the Software Component Architecture and the corresponding scope.

Concerning this from a SOA viewpoint, the logical information system component in collaboration with the technical infrastructure component contributes to the composable principle and the autonomy principle by grouping information objects and grouping the technical services. This decomposition of the system will create the advantage of understanding where to put which services and how integration connections will be arranged.

Design component model
The logical information system component in collaboration with the technical infrastructure component can be used as input for the design component model by the description of the software architecture.

The advantages of using the logical information system component in collaboration with the technical infrastructure component for the input for the Use design component model are[52]:

- They cluster the logical services into logical groups
- They determine the starting point of the components of the Software Component Architecture and the corresponding scope.

Considering this from a SOA viewpoint, the logical information system component in collaboration with the technical infrastructure component

contributes to the layering of the components, which increase the composability of the system.

7.3 AIM and the SOA principles

This section describes how the SOA principles described by Thomas Erl can be used in the AIM methodology for creating a SOA solution. The guidelines identified for the AIM methodology are based on the AIM methodology literature and other scientific literature and the interviews I had with AIM experts, H. Vermeulen and F. Dingemans and R. Spekle. The following list of documents, which is compounded by the AIM experts I interviewed, describes which documents of the methodology are the most important used for the analysis of the SOA principles in the AIM methodology:[53]

- BP.030 - Determine Data Gathering Requirements
- BP.080 - Develop Future Process Model
- BR.010 – Analyze high level gaps
- BR.060 - Create Information Model
- MD.030 - Define Design Standards
- MD.050 - Create Application Extensions Functional Design

7.3.1 BP.030 Determine Data Gathering Requirements

The determination of the data gathering requirement includes the documentation of the current relevant process data, as well as the process data which are collected during the remainder of the project. The following SOA principles with the added guidelines can be used for this document:

- Service reusability – This document creates the first insight in reusable parts. These can be found by clustering shared goals requirements.
- Standardized service contract – This document describes the non-functional requirements, which are included in the service contracts. Tune the different non-functional requirements of the service contracts with each other to create design standards.

7.3.2 BP.080 Develop Future Process Model

The future process model includes the process flow diagrams of the events and business processes that the applications and the associated functions of the business area. The following SOA principles with the added guidelines can be used for this document:

[53] Oracle Method Application Implementation Process and Task Reference, Steve Buchan, November 2005,

- Standardized service contracts – Describing this document the following guidelines should be considered:
    - Create documentation about which processes can be seen as standard processes, where reference models can be used for implementation.
    - Create documentation about which processes are the added value processes of the application and determine for them how they can be designed according to current best practices or design patterns.
    - Define design standards for the input and output variable of the business flows. This can be seen as input for the design standards of the service contracts.
- Service composability – Describe the processes in the flow diagram as flexible as possible. This includes that processes should be maximized on accepting multiple actor inputs and outputs but also adaptable to new processes.
- Service reusability – During the design of the flow diagram it is very important to understand that reusability should be applied when possible. Looking at reusable parts within the enterprise must be part of the design during a project.
- Service autonomy – In this document the first step to the grouping of processes is made. This grouping is done on process level. Keep in mind that for building a good SOA solution, the grouping of the data should not only be applied at process level, but also on reusability aspect.

7.3.3 BR.010 Analyze high level gaps

In this task, you compare the process as envisioned in the Future Process Model (BP.80) with the processes supported by Oracle Applications.
The differences (gaps) revealed by this analysis need to be resolved by producing alternatives that balance change in the application against change in processes and organization. The following SOA principles with the added guidelines can be used for this document:

- Service composability – When high level gaps are determined it's very important to understand the newest integration techniques. This will have influence on how gaps will be solved. Most of these techniques make it possible to separate different technology layers and to create a more flexible governance model.

7.3.4 BR.060 Create Information Model

The information model creates a view of the future business by specifying the future information usage and flow requirements across business functions, business organizations, applications, and data centers. The information model visually depicts information and access flows in the business.

The following SOA principles with the added guidelines can be used for this document:

- Standardized service contracts – During the creation of the information model, data types should be standardized across the enterprise. Standardization of exchange data formats should be made. These

exchange data formats should be made as generic as possible for maximal adaptability and interoperability possibilities.

- Service interoperability – Through the use of standardized data types service interoperability is increased. While creating the information model techniques like BPEL and ESB should be kept in mind for grouping the services.
- Service composability – While creating the information model techniques like orchestration tools and services buses should be kept in mind for grouping of the services, the information model should be optimized to create as less duplicate information in the databases and maximize reusable data. The data should be defined as generic as possible.

7.3.5 MD.030 Define Design Standards

This task describes the standards that designers will follow when designing customizations. Clear and detailed design standards help make sure that all designs are in a consistent format and include the appropriate level of detail. Standards enforce a high level of quality.
The following SOA principles with the added guidelines can be used for this document:

- Standardized service contracts – When design standards for a package application are created, most of these standards are already prescribed by the application. This has the advantage that most of the design standards are already in place and will be documented for the customization software. Minimize the data conversion by defining as less different standards over different domains.
- Service composability – To define a standard for composition, two things should be considered. One is which techniques are going to be used, like BPEL or ESB. Secondly, services for further upgrades or changes of the system should be designed and a governance subsystem should be kept in the system for control of these changes.
- System autonomy – Design standards determine the control of a service when built. Services should have a high level of control over their own functions. Considering a package application, it means that the dependency between the package application and the customization software should be minimized to create a maximal flexibility in the system.

7.3.6 MD.050 Create Application Extensions Functional Design

This task describes the functional features, use, and behavior of required customizations. The Application Extensions Functional Design describes user requirements, and allows users to evaluate and approve the resulting features that the new modules will provide. This task will also deliver the specification for the implementation.
The following SOA principles with the added guidelines can be used for this document:

- Standardized service contracts – Describing this document the following guidelines should be considered:

  - Create documentation about which processes can be seen as standard processes, using the design standards for its specification.
  - Create documentation about which processes are the added value processes of the application and determine for them how they can be designed according to current best practices or design patterns.
  - Create documentation about the input and output variables between the different services, where data types and data format should be standardized across the enterprise.

- Service Composability – The functional design contains a flow diagram for execution of the business processes. This execution is influenced by how services are designed. Keep in mind to make the processes execution as independent as possible and leave room for extensions, because these are the strong advantages of the service model.
- Service reusability – For designing these diagrams the principles of reusability should be kept in mind. Group shared goals service together, because of the high reuse potential. Designing the flow diagram and the creation of iterations to optimize them for reusability can be used as well. This task should be done for the whole enterprise and is very time-intensive.
- Service Autonomy – In this document the first step to the grouping processes is made. This grouping is done on process level. Keep in mind that to build a good SOA solution, the grouping of the data should not only be applied at process level, but also on the reusability aspect. Design processes in such a way that they can easily be replaced or put in other order to increase autonomy in the system.

7.4 Similarities

This section describes the similarities between AIM and RUP considering the SOA principles.

From a principle viewpoint it seems clear that both the approaches have the standardized service contract as their starting point for implementing a SOA solution. They are becoming more visible in current SOA design and are the key to make the application flexible, adaptable to change in the market. By making these contract standardized within one enterprise reuse of service is promoted and the services are made easily connectable to each other, which generates flexibility in the composition of the system.

Service composability is also of major importance. The approaches in this paper describe how to make systems flexible through composability. One of the importance aspects within this field is the creation of services which can be easily connectable to other services, by using standard communication protocols.

7.5 Differences

This section describes the differences between AIM and RUP considering the SOA principles. The clear different between RUP and AIM is that AIM is created for the implementation of a package implementation and RUP is made for customization. The main difference between the RUP method and AIM is that

RUP is methodology where the user is positioned central, where the AIM methodology is used for configuring a package application. AIM is a so called technology push method, where RUP is a demand driven approach.

The creation standardization of the service contracts is made from a different viewpoint. In AIM the service contracts are matched with the enterprise application software package, while RUP has defined the standards of the service contracts during the design phase. This implies that by using AIM it's easy to maintain the standard service contract, because they are prescribed, while when RUP is used for customization software, standardization across the whole enterprise isn't guaranteed.

When we look at the composability of services AIM will take the technology of the package application and use those to connect services to each other. It depends on the package application what for type of metadata is used in the service contracts for composability purposes. This means that not project specific meta data can be applied and also not state of the art technology can be used to create the best composability for the services. On the other hand the contracts are already standardized, so connection between services should be easy to maintain.

7.6 Mapping SOA Maturity Model on the SOA principles

This section describes how the Maturity Model (MM) of Sonic creates more value in constructing a SOA solution based on the SOA principles of Thomas Erl. For each layer in the maturity model the corresponding SOA principles are described and explained why those principles have influence on the MM and how it is based on the literature of Sonic Maturity Model, the book SOA design principles of Thomas Erl and the interview with T. Elzinga, senior SAP architect.

7.6.1 Level 1

This level creates new services on a standard matter, with techniques like XML (eXtended Markup Language), WSDL (Web-Service Description Language) and SOAP (Simple Object Access Protocol). This level contributes to the standardized service contract principle and to the composability principle. The input and output are predefined and how service should be connected by using the WSDL format to find other service are clear examples of this. The creation of service contracts induces the loosely coupled principle through the creation of atomic services.

7.6.2 Level 2

This level creates architected services. This level has the same functionality as level 1 plus the ability to support the services for heterogeneity and distributed systems. Techniques like UDDI (Universal Description Discovery and Integration), WS-Policy and WS-Reliable Messaging are used to create such services. This creates services which are optimized for composable and discoverable usage, conform the composability and the discoverability principle. The UDDI register keeps tracks of all the services and the WS-Policy and WS-Reliable Messaging protocol are used for keeping the connection between the services on the right level.

7.6.3 Level 3

This level creates business services and collaborative services. The business services promote reuse, ease of modification, availability, composite applications. The WS-BPEL protocol is used here to create a chain of business processes. From this perspective the reusability principle and the composability principle get more shape. The reusability principle gets its shape through the promotion of reuse of business services. The composability principle gets its shape through the creation of composite application.

Collaboration services focus on the improvement of collaborative processes with external partners. ESB's can be used here especially to create standard connections to external partners. This can be done conform the principles of composability and the standardized services contracts.

7.6.4 Level 4

This level creates Measured Business Services. The business services are being monitored and dashboard and alerts are created for insight on the process performance and stability. This level creates value for the autonomy principle, by allowing the service to be monitored.

7.6.5 Level 5

This level creates Optimized Business Services. The services will be event-driven, automated and optimized. This level contributes to the autonomy level of services and the abstraction level of services. Through the use of those principles services can be self automated.

7.6.6 Conclusion

Through the use of a maturity model it can be seen that the most of the SOA principles are captured. The advantage of using this model is that SOA is not set up at once, but can be slowly introduced into the enterprise. This has the advantage that the adapting phase to a SOA environment is in controlled environment and will decrease user based faults.

7.7 Architecture patterns[54]

This section gives some examples of how to apply SOA principles in practice with the help of architecture patterns. The following sub sections are the example architecture patterns, concerning the most important principle: the standardized service contract principle. This section gives an example how the SOA principles can be applied in practice and shows the value of these principles.

[54] SOA patterns, Thomas Erl, 2007, www.soapatterns.org

7.7.1 Contract centralization)

This pattern is developed, because consumer programs can be designed to access underlying service resources, resulting in implementation dependencies that undermine strategic SOA goals associated with loose coupling. Contract centralization creates a solution for this problem by making the service logic limited to the service contract, forcing consumers to avoid implementation coupling. This means that developers can only use the a service by using the contract of the service and cannot connect directly to the source. This gives an example how pattern can be seen as a deviation of the principles of composability and service loosely coupling.

7.7.2 Service Façade

This pattern is developed to maintain logic-to-contract coupling when supporting multiple contracts. Services with standardized contracts will have high logic-to-contract coupling which can conflict with requirements to support multiple contracts. The underlying service implementation can contain a façade component which corresponds to a contract and enables the abstraction of a core body of service logic, allowing for additional facades and contracts to be added to the same service. It's like a specialization of two service contracts.

7.7.3 Conclusion

From these patterns it becomes clear that a SOA solution needs certain patterns to overcome certain problems. These patterns have their roots in the SOA principles. So the SOA principles help not only to create a good design for a SOA solution, but are also the baseline for design patterns to create a solution for several problems.

7.8 Responsibilities

When designing a SOA solution new problems will occur. One of the biggest problems is how to handle the alignment of all the services within one enterprise. For this problem and others new roles will be developed to create a SOA solution. In this section these new roles are identified and the issues around each role are described.

7.8.1 Enterprise Architect

The Enterprise Architect[55] deals with strategic software decisions (aligning IT with the business), typically involving many software systems within an organization, across several projects teams, typically at more than one site. The Enterprise Architect may seldom see or interact with source code.

Enterprise architects should create guidelines and policies that ensure agility and coherence which are strongly aligned to the business requirements and directions.

[55] Software Architect, Wikipedia, 2008, http://en.wikipedia.org/wiki/Chief_Software_Architect

However, since the apps and infrastructure teams plan solutions, there could be strong resentment or defensive management within those areas. Enterprise architects must ensure that architecture awareness and support is present in the enterprise. Considering the SOA perspective, the enterprise architect is responsible for aligning all IT-services and the creation of a clear vision about how, which and what every services does. He is responsible for understanding how the SOA principles should be applied on a high level for the whole enterprise.

7.8.2 Solution Architect

The solution architect may refer to the focus on pushing a particular business solution, which needs interactions between multiple applications. This creates for him the role to work together with the enterprise architect to align the services he has with the other services in the enterprise. The solution architect is responsible for applying the SOA principles within one project.

7.9 Conclusion

This chapter has described the guidelines of how RUP and AIM will develop SOA solutions. These guidelines helps to create a SOA solution and we can learn from them that the basis of SOA solution is to design services, which are standardized, easily connectable and atomic. The power of a SOA solution lies in the structure of flexibility to change and adapt to current market changes. This structure can be made by RUP for customization software and AIM can be used for the package applications with the help of the additional guidelines described in this chapter. A framework, like the IAF framework can help to structure design and creates the benefits of aligning the services throughout the enterprise. When designing a SOA solution design patterns can be of great help to make right design decisions for a SOA solution. In addition clearly defined roles, such as an enterprise architect and a solution architect will help to create clear responsibilities concerning the parts in the SOA solution and will support a good design.

7.9.1 Results proposition

This section describes the propositions which can be concluded from this thesis. These propositions describe in general how to create a SOA solution without the use of a specific methodology to design them. The following list describes these propositions:

- Services are grouped by shared goals for reusability
- A top-down approach can be used to find reusable subparts in a process.
- A bottom-up approach can be used to find reusable assets in the existed architecture.
- Meta data is required for contract negotiation when designing a service.
- SOA solution can be based on design patterns.
- SOA solution can be based on a reference architecture.

- No information about how a service works should be described internal in a service contract.
- Services in a SOA must be as generic as possible.
- SOA solutions are based on market reference models.
- Services in a SOA are made for only a single business task.
- An enterprise architect is helpful for aligning services to the current architecture.
- Maturity models can be used for making a roadmap to SOA.

From this list we can identify that a SOA solution is especially controlled by the use of standardized approaches. The help of reference models and maturity models will increase the healthiness of the design of a SOA solution. This creates the benefits of making adapting and connection between the different services easy.

8 Validation

This chapter describes the validation process of this thesis. In this process two techniques are used to validate the propositions founded in chapter 7.9.1. A survey is used to validate the reliability of the propositions. This can be done using a so called t-test. The validity of the subject is crosschecked through interviewing different experts in the field, which are in most cases multi-years experience IT architects.

8.1 Reviews by Architects

The propositions are formed from a combination of scientific literature as stated throughout chapter seven, the interviews I conducted and some logic reasoning. These propositions are crosschecked with senior architects from Capgemini to evaluate of the results are logic and useful for their practice. I interviewed ten architects from different disciplines to support how SOA principles can be mapped on the RUP and the AIM methodology. These architects all have a minimum of ten years of working experience in the field of IT-system design. These architects have good understanding of how the SOA will change the business and they have been of great help for me to understand the underlying problems in designing a SOA solution.

8.2 Survey

For validating the founded propositions a survey is made in which the propositions are stated and the participants are asked to give their opinions about the subject. The validity is checked with the use of a t-test. A t-test is any statistical hypothesis test in which the test statistic has a Student's t distribution if the null hypothesis is true. It is applied when sample sizes are small enough to use an assumption of normality.

The statements in the survey I created are based on the literature I found and the interviews I conducted (see appendix). To see if these statements are correct I proposed 20 architects with the questions based on the Likert scale. I have chosen the Likert scale here, because it creates a good evaluation of their opinions on the subject and is frequently used in scientific papers. The questions I raised were stated in such a way that they were polarized to one clear opinion, to create clear measurements.

The following statements of the survey were accepted by the architects community of Capgemini after translation of the polarized opinion to the founded propositions of the literature and the conducted interviews:

- Services are grouped by shared goals for reusability
- A top-down approach can be used to find reusable subparts in a process.
- A bottom-up approach can be used to find reusable assets in the existed architecture.
- Meta data is required for contract negotiation when designing a service.
- SOA solution can be based on design patterns.
- SOA solution can be based on a reference architecture.

- Services in a SOA must be as generic as possible.
- An enterprise architect is helpful for aligning services to the current architecture.
- Maturity models can be used to make a roadmap to SOA.
- No information about how a service works internal in a service contract should be described.

The following statements of the survey were declined by the architects community of Capgemini:

- In order to construct reusable services, services are grouped by sequential processes.
- SOA solutions are based on market reference models.
- Services in a SOA are made for only a single **business** task.
- Services in a SOA are made for only a single **IT** task.

This survey confirms that it is very important to look at the reusable aspect of the SOA service design. Top-down as well as bottom approaches can be used for finding reusable services. Good SOA architectures are based on design patterns and reference architectures. The service itself should be generic and should have appropriate contract negotiation protocols. This survey declined that service should be made for only one task.

In this survey I raised another important issues which is in the scope of this thesis. A services value is mostly determined by their services contract, but what do you have to put in the contract to make it most valuable? According to the Capgemini architects the following should be in the service contract:

- Input and output data types (i.e. int, string)
- Input and output data format
- Availability (i.e. request per minute)
- Governance (who is responsible)
- Functionality (describing the function of a service)

On the following items there are doubts if they should be in the service contract:

- Policy (who can access the services)
- Logic (business rules)

The following items should not be included into a service contract:

- Used technology (Java, C#)
- Registry information (link to a register)

Remarkable from this list is that there are doubts about whether the policy should be included into the service contract or should be placed into a different component. According to Thomas Erl (describing the SOA principles), they should not be placed into the service contract, because a policy exists of one or

more general rules in the area of security and should be set into a business rule. A policy is always applied for a domain of services.

The other doubtful item is the logic. This is explained by the fact that for each services it can be a design issue whether to put the logic into the service and so into the service contract or be extracted from the services and put into external business rules. This depends on the context of the services.

9 Conclusion

This sections describes the conclusions of this paper and identifies of the research questions are answered and the research objectives are defined. To answer the main question: "*How to safeguard the SOA principles, stated by Thomas Erl, when designing an Enterprise Architecture solution using the RUP or the AIM methodology?*" I identified guidelines for building a solution where the SOA principles are safeguarded using the RUP or AIM methodology through investigating many scientific literature and holding many interviews about this subject. From the interviews I learnt more about the practical side of the subject, where the literature describes more the general vision of the subject.

The guidelines for building a SOA solution with RUP or AIM are created by first looking at the IAF framework and the Sonic maturity model to help understanding SOA designs. First the IAF framework, a framework for identifying services within a service architecture, helps to identify the right services on a business level and on an IT level. After the identification of business and information services, the services are grouped in the different layers of the framework. The grouping of these services are determined by different design principles of the service, the service composability, service loosely coupling, and the service autonomy. The power of this framework lies in separating those layers by asking the question of what does a business do and asking how can this be implemented. This framework will eventually deliver an architecture design.

Secondly the Sonic maturity model describes when to implement what part of a SOA architecture. This includes that enterprises should implement a SOA architecture incremental. This model supports the SOA design principles and makes it clear that not every principle can be applied at once. This model will deliver a roadmap for a SOA architecture.

Through the introduction of SOA a new role is added within enterprises. This role is named enterprise architect in Capgemini and its responsibility is to maintain all the IT-assets and align them. For him/her it should be of big importance to create as much reusable services as possible conformed to the reusable principle. The other important task is to create a good overview, what, where and how every service is built and where every service is connected to.

The last tree topics prepares the reader for a better understanding of what kind of problems there can be and how to support solving them in designing and building a SOA solutions. Now we can answer the main question.

Designing a SOA solution with RUP or AIM is based on finding reusable parts in the enterprise. Similar functions are grouped by shared goals during the analysis phase for finding those parts. A top down approach helps even further to identify more reusable parts through looking at reuse of identical subparts of the identified processes. (decomposition of the system). The last method that can be applied to find reusable parts is a bottom up approach, which helps to identify more reusable parts by looking at reuse of the existence services.

The second design issue for creating a SOA solution is how to build a SOA service. In the early stages of designing a SOA service with RUP or AIM agreements should be made about standards in a service contract, like data types, data format, exchange format, policies format, and security format. The composability of the services should be safeguarded by designing the service

with enough meta data so that Enterprise Service Bus (ESB) or other integration software can easily make connections with other services. The use of reference models, design patterns, and a reference architecture model helps to create a standardized structure in the enterprise architecture.

The change from a non-SOA architecture to a SOA architecture in an enterprise architecture is enormous, but has the benefits of a fundamental and cost effective architecture. This change not only influences the IT-system, but will also influence the main ideas people have about how to create IT-systems. Concluding: using the RUP or AIM methodology can help to support building a SOA solution, but extra guidelines are needed for them to conform to such architecture. Understanding the transformation from object-oriented programming to service-oriented programming is a difficult path, but this thesis has described some steps of this transformation. But still a lot is to be discovered in this field.

10 Further research

This chapter describes which researches are needed to make this paper more valid and complete. During my research I investigated the principles of SOA. The SOA principles prescribe what an architecture model should look like to create the maximal benefits of a SOA implementation, but don't describe how to implement them. The implementation of a SOA can be done by using a service variant of the RUP model, called SOMA. In this variant special phases are added for the service identification, specification and realization. During the identification phase an important issue was raised about the level of granularity of services. The SOMA model doesn't describe what to do about this and I didn't find any known solution in the literature and the interviews I conducted about this problem. Another problem I found was that in the current methodologies little information is known about how to align services with the enterprise. This problem can be a big issue because it captures the essence of the reusability principle. The principle described in this thesis only describes the high level thoughts about how SOA can be applied. A lot of room is still available on how to create a best practices roadmap to SOA, but the first steps of this roadmap are identified in this paper. This paper clearly describes that there can be a very positive benefit of using a framework for building SOA solutions, but what exactly the benefit is became unclear.

This thesis answers a view questions, but creates a lot more questions. All these questions cannot be raised in this chapter, but I will give the important ones.

- Are RUP and AIM the best methodologies to build a SOA solution?
- Does Thomas Erl describe all the SOA principles?
- What level of granularity is needed for what services?
- Which architectural framework is best suited for a SOA architecture?

Appendix

10.1 Interviews

The interviews I conducted are Company confidential, so I cannot include those in this thesis.

10.2 Survey: SOA principles in RUP and AIM

This survey proposes propositions on how SOA principles can be applied within the RUP and AIM methodology. During this survey you will be asked to give your opinion about these propositions through marking the right box with an 'x'. The place of the mark indicates how you think about the propositions and the scale ranges from totally disagree, a little disagree, neutral, little agree, to totally agree. The answers you give will be processed anonymously and are used to validate my research about this subject.

For this survey I will give an appropriate definition about a service to prevent ambiguity.

Definition services: A service is a unit of solution logic to which service-orientation has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguishes a unit of logic as a service compared to units of logic that may exist only as objects or components.

1. In order to construct **reusable** services, services are grouped by shared goals.

Totally disagree | | | | | Totally agree

2. In order to construct **reusable** services, services are grouped by sequential processes.

Totally disagree | | | | | Totally agree

3. A **top-down** approach is necessary to find reusable subparts in a process.

Totally disagree | | | | | Totally agree

4. A **bottom-up** approach is necessary to find reusable assets in the existed architecture.

Totally disagree | | | | | Totally agree

5. Meta data is required for contract negotiation when designing a service.

Totally disagree | | | | | Totally agree

6. SOA solutions are based on market reference models.

Totally disagree | | | | | Totally agree

7. SOA solutions are based on design patterns. (i.e. www.soapattersn.org)

Totally disagree | | | | | Totally agree

8. SOA solutions are based on a reference architecture. (i.e. the OASIS reference architecture) http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf)

Totally disagree | | | | | Totally agree

9. Services in a SOA are 100% generic.

Totally disagree | | | | | Totally agree

10. Services in a SOA are made for only a single **business** task.

Totally disagree | | | | | Totally agree

11. Services in a SOA are made for only a single **IT** task.

Totally disagree | | | | | Totally agree

12. An enterprise architect is necessary for aligning services to the current architecture.

Totally disagree | | | | | Totally agree

13. Maturity models are necessary for making a roadmap to SOA.

Totally disagree | | | | | Totally agree

14. No information about how a service works internal in a service contract should be described.

Totally disagree						Totally agree

The following question will specifically describe what to put in a service **contract**.

About these questions I will ask if they are required to be used and if you use them in practice.

Attributes in a service contract	**Required**		**Used**	
	Yes	*No*	*Yes*	*No*
Input and output data types (i.e. int, string)				
Input and output data format				
Availability (i.e. request per minute)				
Policy (who can access the services)				
Governance (who is responsible)				
Functionality				
Technology (Java, C#)				
Logic (business rules)				
Registry information (link to a register)				

10.3 Result of the survey

The values in the table represent the answer values which are significant on a level of 5% done by 20 Capgemini architects. The first column has values from 1 to 5, which represent totally disagree until totally agree. In the last two columns the 1 represents a yes and 0 represents a no and 0, 5 is in between.

Scaled Questions	Required for Service Contract	Used in practice for Service Contract
1. 3	1. 1	1. 1
2. 1	2. 1	2. 1
3. 3	3. 1	3. 1
4. 3	4. 0,5	4. 0,5
5. 4	5. 1	5. 1
6. 2	6. 1	6. 1
7. 3	7. 0	7. 0

8. 3,4	8. 0,5	8. 0
9. 2,3	9. 0	9. 0
10. 2,3		
11. 2,3		
12. 2,3,4		
13. 3		
14. 4		

If both crosses are checked, then half of the surveyed people answered yes and the other half no.

Attributes in a service contract	**Required**		**Used**	
	Yes	*No*	*Yes*	*No*
Input and output data types (i.e. int, string)	X		X	
Input and output data format	X		X	
Availability (i.e. request per minute)	X		X	
Policy (who can access the services)	X	X	X	X
Governance (who is responsible)	X		X	
Functionality	X		X	
Technology (Java, C#)		X		X
Logic (business rules)	X	X		X
Registry information (link to a register)		X		X

References

1. SOA Advances, Gartner, 17 November 2006
2. E – Supply Chain Orchestration Using Web Service Technologies:A case using BPEL4WS, Simon Samwel Msanjila, 2003, http://portal.acm.org/ft_gateway.cfm?id=1089636&type=pdf&coll=GUIDE&dl=GUIDE&CFID=46560301&CFTOKEN=73758637
3. Wikipedia, 2007, http://en.wikipedia.org/wiki/Business_process
4. Service-oriented architecture (SOA) definition, Barry & Associates, Inc., 2008, http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html
5. Process computing, Wikipedia, 2008, http://en.wikipedia.org/wiki/Process_%28computing%29
6. SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service.asp
7. Enterprise, Business and IT Architecture and the Integrated Architecture Framework, Capgemini, 2007
8. SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_oriented_architecture.asp
9. Design Principle, Thomas Erl, 2007, http://www.soaglossary.com/design_principle.asp
10. Service Oriented Architecture, Joost van der Vlies, 12 February 2008 (presentation)
11. Principles of service Design, Thomas Erl, July 2007
12. Zapthink, Ronald Schmelzer, August 2005, http://www.zapthink.com/report.html?id=ZAPFLASH-2005824
13. SOA Zone, Dan Foody, August 2005, http://www.soa-zone.com/index.php?/archives/16-Defining-loose-coupling.html
14. SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_reusability.asp
15. SOA glossary, Thomas Erl, 2007, http://www.soaglossary.com/service_autonomy.asp
16. The principles of service-orientation, Thomas Erl, May 2006, http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1192369,00.html
17. Reference model for Service Oriented Architecture, C. Matthew Mackenzie, August 2006, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
18. Wikipedia August 2007, http://en.wikipedia.org/wiki/Service-oriented_architecture
19. Lego, 2007, http://factory.lego.com

20. Integrated Architecture Framework Version 4, reference Manual, Colin Metcalfe, October 2006
21. http://en.wikipedia.org/wiki/Rup
22. Wikipedia, 2007, http://en.wikipedia.org/wiki/Spiral_model
23. The Rational Unifed Process An Introduction, Philippe Kruchten, 2004, p62
24. Case study: SOA design scenario , IBM red books, 2008, pag 8, http://www.redbooks.ibm.com/redpapers/pdfs/redp4379.pdf
25. Movin' SOA On Up, Jon Bachman, September 2005, http://www.sonicsoftware.com/solutions/learning_center/soa_insights/movin_soa_on_up/index.ssp
26. Heffner, Randy, "Your Paths to Service-Oriented Architecture", Forrester Research, Dec. 2004.
27. Software Engineering Institute, Capability Maturity Model® Integration, August 2006, http://www.sei.cmu.edu/cmmi
28. Rational Unifed Process V7, 2007, http://deliver-rendering.capgemini.com/components/RUP_V7_Large_Projects/index.htm
29. RUP 2007, Capgemini, http://deliver-rendering.capgemini.com/components/RUP_V7_Large_Projects/rup/capabilitypatterns/identify_services,_WnOpAEocEdqrjq4i3fchvA.html
30. Agile modeling, S.W. Amber, 2004, http://www.agilemodeling.com/images/models/useCaseReuse.gif
31. A guided tour to business process modeling and implementation in DB2 Content Manager for AD Company case, Xiao Li, 03 August 2006, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608li/7.jpg
32. A guided tour to business process modeling and implementation in DB2 Content Manager for AD Company case, Xiao Li, 03 August 2006, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608li/7.jpg
33. OASIS reference model, Duane Nickull, 2008, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
34. UML modeling: Creating a class diagram, SUN Developer Network, 2008, http://developers.sun.com/jsenterprise/learning/tutorials/jse8/uml_class_diagram.html
35. Modellencyclus IAF/RUP for SOA, Joost van der Vlies, Capgemini, 2007
36. Software engineering and architecture, Robert de Boer Capgemini, 2004 (internal presentation)
37. Oracle Method Application Implementation Process and Task Reference, Steve Buchan, November 2005,
38. SOA patterns, Thomas Erl, 2007, www.soapatterns.org

About Capgemini

Capgemini is headquartered in Paris, France and operates in more than 36 countries. We are, above all, a people company—over 86,000 people in North America, Europe, and the Asia Pacific region. Management and support roles aside, our employees are grouped into four major disciplines, each of which is governed by its specific economic rules, and managed with its own profit.

Four fundamental objectives guide the operation of our business:

- To use our expertise to the benefit of our clients and partners through an open, collaborative approach.
- To ensure sustainable and profitable long-term growth.
- To provide a return on investment to shareholders.
- To promote employee development.

In addition to these goals, a formalized set of shared values guide our business decisions and shape our culture. The practical expression of these values and objectives can be seen in the collaborative relationships we build, the standards by which we measure our work, and the commitments we make to our people.