Service Granularity in SOA Projects: A Trade-off Analysis

Claudia Steghuis

Service Granularity in SOA Projects: A Trade-off Analysis

Claudia Steghuis

MSc Business Information Technology, University of Twente
June 28, 2006

Examination Committee:
Dr. Maya Daneva, University of Twente
Prof. Dr. Jos van Hillegersberg, University of Twente
Ir. Lucas Osse, Principal Architect, Capgemini Netherlands
Ir. Piet Adriaanse, Vice President, Capgemini Netherlands

SUMMARY

Service granularity, the scope of functionality that is exposed by a service, is a crucial issue in designing service oriented architectures (SOA). Every SOA needs to have well designed services in order to gain the predicted benefits, like flexible business processes and low development costs. However, while many literature sources suggest architects to choose the right level of granularity of services, none of these studies goes into detail about how to do this. In this master thesis we explore service granularity in SOA projects to develop patterns to support service granularity decision-making. Although experience seems to be the only good resource to improve service granularity decision-making, this research examines the service granularity issues to provide help to the inexperienced architect.

This research is conducted by order of Capgemini, a large consultancy company who runs many SOA projects. Service granularity receives much attention in these projects, however, the results are not evaluated and no company wide methods exist to improve service granularity decision-making. They re-invent the wheel at every SOA-project and are therefore interested in a study to define service granularity in more detail.

First of all, a research in the literature revealed the essential aspects and service types in relation to service granularity. The aspects that were found are reusability, flexibility, complexity, functionality, composability, reusability of legacy, sourcing, genericity, transaction consistency, context-independence, and performance. Furthermore, research to service types in SOA showed three different service types: business services that provide value on business level, information system services which automate parts of business processes and software services which fulfill tasks on software infrastructure level. Secondly, a granularity framework is designed which illustrates the initial relationships between the aspects and the service types in reference to service granularity. All relations are explained by hypotheses, which make the relation more explicit. Expert interviews helped to indicate these relations. The framework exists of the aspects and their relationships, within the three different service types.

Four case studies are carried out using the positivist case study research method and the structured case approach (Carroll et al. 2000). Results from interviews with project members showed the influence of the service granularity aspects in these case studies. This new information refined the granularity framework and its hypotheses. resulted in three important areas for service granularity. Architects have to make trade-offs about service granularity in and between these areas. The areas are:

- **Flexibility in business processes**: to give the desired amount of flexibility to the services. Important aspects are functionality, context-independence, transaction consistency and sourcing. Decisions about this area have to be made mostly at the business services and IS services level.
- **Reusability**: to keep both maintenance and development costs low. Important aspects are genericity, reusability of legacy, composability, flexibility in business processes, testability, maintainability, structural complexity, cognitive complexity, functionality, and context-independence. Decisions about this area have to be made mostly at the IS service and software service level.
- **Performance**: To keep performance demands manageable in the service landscape. Important aspects are functionality and structural complexity. Decisions about this area have to be made mostly at the IS service and software service level.

To help architects in their trade-off decisions, four patterns are derived from the case studies which improve the service granularity decision making process. Every pattern represents a part of the granularity framework and offers general guidelines about how to choose service granularity in combination with the aspect. The four patterns are:

1. **Design services for flexibility:** It states that one should first think about the amount of flexibility required in the process. Then, the aspects that require attention in this area are explained.

2. **Design reusable services:** It explains the different forms of reusability and gives organizational and technical guidelines about how to generate the most profit out of reuse.
3. **Designing generic services:** It shows the aspects to consider in designing generic services.
4. **Performance and SOA:** It shows how to deal with performance demands in SOA.

Our preliminary evaluation study of validation indicated that the framework and the patterns are useful in understanding the aspects concerned in service granularity decision-making. However, the patterns are still too abstract and theoretical to use in actual practice. Therefore, future research should be conducted to define more detailed guidelines regarding the influence of the aspects in our framework on service granularity. An other future research area that has to be explored is to develop measurement methods for service granularity, in order to express service granularity in a number. A last area for future research is to compare custom build SOA with ERP SOA (for example SAP's Enterprise Service Architecture) to indicate the differences in granularity between these SOA types.

Our key conclusions that resulted from this project include the following: First, we learnt why granularity decision making is a complex process. Second, we provided a multifaceted definition of granularity by (i) proposing a framework for understanding granularity aspects in SOA projects and (ii) deriving four patterns regarding the trade-offs that architects should anticipate in SOA projects. Early evaluation of validation concerns about both the framework and the patterns indicate that the results of this research project were a good starting point in a long-term process of consolidating the knowledge of SOA literature and the architecture practice. As with all evaluation methods, we cannot claim with certainty the external and internal validity of our results. What we did was to indicate threats to validity, to address them systematically, and to make a systematic plan for future validation research.

SAMENVATTING (DUTCH)

Service granulariteit, de mate van detaillering en omvang van de functionaliteit van een service, is een essentiële kwestie in het ontwerpen van service geörienteerde architecturen (SOA). Elke SOA moet zijn services goed definiëren om de voorspelde voordelen, zoals flexibele bedrijfsprocessen en lage ontwikkelingskosten, te bereiken. Hoewel vele literatuurbronnen architecten voorstellen om het juiste niveau van granulariteit van de services te kiezen, gaat geen van deze studies in detail over hoe dit te doen. In deze master thesis wordt de service granulariteit in SOA projecten onderzocht om patterns te ontwikkelen die het service granulariteit besluitvormingsproces kunnen ondersteunen. Omdat ervaring op dit moment het enige goede middel schijnt te zijn om betere besluiten omtrent service granulariteit te nemen, analyseert dit onderzoek de kwesties betreffende service granulariteit om hulp aan de onervaren architect te bieden.

Dit onderzoek wordt gedaan in opdracht van Capgemini, een groot consultancy bedrijf dat vele SOA projecten leidt. Service granulariteit krijgt veel aandacht in deze projecten, echter, de resultaten op dit gebied worden niet geëvalueerd en er bestaan geen bedrijfsbrede methodes om de besluitvorming omtrent service granulariteit te verbeteren. Bij elk SOA project wordt het wiel opnieuw uitgevonden en daarom is dit bedrijf geïnteresseerd in een studie om service granulariteit in meer detail te onderzoeken.

Allereerst openbaarde een onderzoek naar de literatuur de essentiële aspecten en de servicetypes die betrekking hebben op service granulariteit. De aspecten die werden gevonden zijn hergebruik, flexibiliteit, complexiteit, functionaliteit, composabiliteit, hergebruik van legacy, sourcing, genericiteit, transactieconsistentie, context-onafhankelijkheid en performance. Voorts toonde het onderzoek aan dat in SOA drie verschillende servicetypes kunnen worden onderscheiden: de business services die waarde op bedrijfsniveau hebben, de informatiesysteem services die delen van bedrijfsprocessen automatiseren en de software services die taken op het niveau van de software infrastructuur vervullen. Ten tweede, wordt een granulariteitsframework ontworpen dat het aanvankelijke verband tussen de aspecten en de servicetypes in combinatie met service granulariteit illustreert. Alle relaties worden verklaard door hypothesen, die de relaties explicieter maken. De gesprekken met deskundigen hielpen om deze relaties helder te krijgen. Het framework bestaat uit de aspecten en hun relaties, afgebeeld op de drie verschillende servicetypes.

Vier case studies zijn uitgevoerd, volgens de positivist methode en de structured case study method (Carroll et al 2000). De resultaten van interviews met projectleden toonden de invloed van de aspecten op de service granulariteit in deze case studies. Deze nieuwe informatie detailleerde het granulariteitsframework en zijn hypothesen. Dit resulteerde in de afbakening van drie belangrijke gebieden omtrent service granulariteit. De architecten moeten afwegingen over de service granulariteit op en tussen deze gebieden maken. De gebieden zijn:

- **Flexibiliteit in bedrijfsprocessen**: Om de gewenste hoeveelheid flexibiliteit in het bedrijfsproces te bereiken. De belangrijke aspecten zijn functionaliteit, context-onafhankelijkheid, transactieconsistentie en sourcing. De besluiten over dit gebied moeten meestal op het niveau van business services en IS services worden genomen
- **Hergebruik**: om zowel onderhoudskosten als ontwikkelingskosten laag te houden. De belangrijke aspecten zijn genericiteit, hergebruik van legacy, composabiliteit, flexibiliteit in bedrijfsprocessen, testbaarheid, onderhoudbaarheid, structurele complexiteit, cognitieve complexiteit, functionaliteit, en context-onafhankelijkheid. De besluiten over dit gebied moeten meestal worden genomen op het niveau van IS services en software services.
- **Performance**: Om prestatie-eisen in het de servicelandschap beheersbaar te houden. De belangrijke aspecten zijn functionaliteit en structurele complexiteit. De besluiten over dit gebied moeten meestal worden genomen op het niveau van IS services en software services.

Om architecten in hun afwegingen te helpen, zijn vier patterns afgeleid uit de casestudies. Deze patterns helpen om het service granulariteit besluitvormingsproces te verbeteren. Elk pattern vertegenwoordigt een

deel van het granulariteitsframework en biedt algemene richtlijnen over hoe de service granulariteit onder invloed van de aspecten te kiezen. De vier patterns zijn:

1. **Ontwerpen van services voor flexibiliteit**: Het pattern legt allereerst uit dat men eerst moet nadenken over de hoeveelheid flexibiliteit die in de bedrijfsprocessen wordt vereist. Daarna worden de aspecten die aandacht op dit gebied nodig hebben verklaard.
2. **Ontwerpen van herbruikbare services**: Het pattern verklaart de verschillende vormen van hergebruik en geeft organisatorische en technische richtlijnen over hoe de meeste winst uit hergebruik gehaald kan worden.
3. **Ontwerpen van generieke services**: Het pattern toont de aspecten die in overweging genomen moet worden bij het ontwerp van generieke services
4. **Performance en SOA**: Het pattern toont hoe prestatieeisen in SOA te behandelen.

Onze eerste evaluatie van validatiebelangen toonde het nut van het framework en de patterns in het kader van service granulariteit besluitvorming aan. Op dit moment zijn de patterns nog te abstract en te theoretisch om in de daadwerkelijke praktijk te gebruiken. Daarom zou toekomstig onderzoek moeten worden uitgevoerd om meer gedetailleerde richtlijnen met betrekking tot de invloed van de aspecten op het granulariteitsframework te definiëren. Een ander toekomstig onderzoekgebied dat moet worden onderzocht is de ontwikkeling van een meetmethode voor service granulariteit, om de granulariteit van een service daadwerkelijk in een getal te kunnen uitdrukken. Een laatste gebied voor toekomstig onderzoek is het vergelijken van op maat gemaakte SOA met SOA ontwikkeld door commerciële bedrijven (bijvoorbeeld SAP’s Enterprise Service Architectuur) om goed op de verschillen in granulariteit tussen deze types SOA te anticiperen.

Onze meest belangrijke conclusies die uit dit project voortvloeiden omvatten het volgende: Eerst leerden wij waarom de besluitvorming omtrent service granulariteit een complex proces is. Ten tweede introduceerden wij een veelzijdige definitie van granulariteit door (i) een framework te presenteren om granulariteit aspecten in SOA projecten te verklaren en te begrijpen en (ii) vier patterns te ontwikkelen over de afwegingen die de architecten in SOA projecten zouden moeten maken. De vroege evaluatie van validatiebelangen over zowel het framework als de patterns wijst erop dat de resultaten van dit onderzoeksproject een goed uitgangspunt zijn om de kennis over service granulariteit op de lange termijn te consolideren. Zoals met alle evaluatiemethodes, kunnen wij niet met zekerheid de externe en interne validiteit van onze resultaten bewijzen. Wat we hebben gedaan, is het indiceren van bedreigingen voor de validiteit, deze systematisch te behandelen, en een systematisch plan voor toekomstig validatie onderzoek maken.

PREFACE

Nine months ago I started my master's project at the architecture and infrastructure services practice at Capgemini. My goal at that moment was to do something with ERP and SOA because ERP and architecture were the subjects I developed a deep interest in during my study. In conversations with my supervisors and by reading the literature in the SOA field, I discovered the lack of research in the field of service granularity. SAP, one of the world biggest ERP-vendors was also moving towards SOA, so this would be a nice topic for my thesis.

However, while doing my research, I found out that it was not that simple as it seemed to be. SAP was still in its infancies with their SOA migration, there was a lack of literature on granularity and many definitions existed for SOA. I tried to solve the lack of literature by doing expert interviews, to make myself more and more familiar with the subject and the SAP SOA became more and more a side-track.

I found four suitable SOA projects and interviewed between one and four people per project. While doing that, I learned that it is very useful to use a structured method for analysing the results. Reading the case reports over an over again seems very boring, but gives detailed insight into the process of granularity decision-making and it feels very good to discover new relations and connections.

Those nine months have passed with ups and downs. I was very happy when I finally finished parts, or found interesting relations, but I felt totally lost when my inspiration ran off and I did not succeed in writing anything useful. During these days, the time spent sometimes felt like an eternity, but when I look back now, time past very quickly. I learned a lot in this last phase of my study, not only about work-related subjects, but also about myself and in relations with others. And, although I'm very happy I finished it now, it is a time I will never forget.

Of course, I would like to thank some people. Firstly, I would like to thank Maya Daneva, my first supervisor, for her detailed reviews, living conversations and all other help during the writing of this thesis. Also thanks for Jos van Hillegersberg, my second supervisor, for sharing his ideas and visions with me. Lucas Osse, my first supervisor from Capgemini, has given me the support I needed from inside Capgemini and kept me updated about granularity issues in his projects. I will thank him for his dedication to my project. I would also like to thank Piet Adriaanse, my second supervisor from Capgemini, who guided me into the subject of granularity and provided me from time-to-time with feedback.

The experts that helped me during the literature study and in the validation process were great and I would like to thank Hans Goedvolk, Martin Op 't Land, Raymond Slot, Har Gootzen, and Tobias Titus for their time and support. I also want to thank the project members who have given me an inside look in their projects. While the projects are anonymous, I cannot state their names here, but I thank them all for their time and participation in this project. Also, a big thanks goes to my colleagues from P30 and P20, who showed their interest in my project and were nice sparring-partners.

At last, but certainly not at least, I would like to thank Maarten, my boyfriend, who provided me with the support I needed, both emotionally and in contents. He gave me the confidence I needed at times and brought also the welcome distraction from the thesis. Furthermore, I would like to thank my parents, brother, other family and friends for their support during my master thesis, the nice distractions from the project and also showing their interest in my project.

IJsselstein, June 23rd 2006

Claudia Steghuis

Contents

1 Introduction

1.1 Problem Statement

Service oriented architecture (SOA) is the new paradigm that is now recognized as the best solution for systems integration (Delphi Group 2005). Companies need to be both flexible and integrated. This means that it should be possible to adapt business processes realtime to the changing business circumstances (Lee et al. 2003). Nowadays, the responsiveness of the IT is as important as the integration. SOAs are today's promise to the companies' needs to get integrated internally and to partners and to be responsive at the same time. An SOA is an architecture based on services, which create business functionality.

Capgemini is a global leader in consulting, technology, outsourcing and local professional services and has almost 60,000 employees worldwide (Capgemini 2006). Capgemini NL has approximately 5,000 employees in the Netherlands and has many big customers, like ABN Amro, Ahold, ING, and Shell. Before the hype Capgemini already worked service-oriented and started ten years ago with their services architecture, supported by their own framework (IAF). The service-oriented paradigm is accelerated by the developments in the web service world, which made distributed computing much easier. SOA can take advantage of the standards developed for web services and this truly enables possibilities for the service oriented world.

Capgemini has already developed their own SOA-methodologies and has gained experience in service-oriented projects. One issue that rises every time is how to choose service granularity, i.e. the scope of functionality a service provides. In this research we concentrate on the identification and design of services to develop a method which improves the service design process. Primary concerns in this process are managing service granularity, designing for service reuse, and designing for service composability (Papazoglou et al. 2006).

Service granularity defines how much functionality is exposed by one single service. Services can be fine-grained or coarse-grained (Papazoglou et al. 2006). Service granularity seems to be an interesting field of study, while it needs attention at every service oriented project and it includes multiple dimensions. According to Foody, there are three key area's to consider when looking at granularity: performance and size, transactionality and state, and business suitability (Foody 2005). He defines some rules of thumb merely based upon technical aspects of web services (like combine operations that execute in less than 5 ms) and mentions simplicity and generality as guidelines. Colan of IBM says, for example, that the use of coarse-grained services for external consumption is recommended, while fine-grained should be considered inside the company (Colan 2004).

The granularity problem is addressed in many papers and at many Internet sources, for example (Foody 2005; Papazoglou et al. 2006; Sims 2005; Zimmerman et al. 2004), but in none of these sources the problem is addressed fully, and most of them only state that one should choose the right level of granularity, without giving further guidelines. The assumptions behind the granularity guidelines these authors define are not scientifically justified, and this was our motivation to carry out a more detailed research into this area. While SOA is getting more mature, the granularity problem seems to get more attention. Since the field of study is complex because of the many factors involved in service granularity issues, we do not intend to solve all the granularity issues. However, we try to clarify the ambiguities around service granularity and define some patterns to give support to the service granularity decision-making process.

1.2 Research Questions

The main goal of our research is to formulate patterns to support service granularity decision-making in a service oriented architecture project. To be able to meet this goal, we formulate several research questions. We start with a literature study in which the topics of SOA and SOA design are discussed and the concepts of granularity are explained. Then, all of this is matched to certain service types. This leads to a granularity framework and the first research question:

1. What granularity framework can be identified based on state-of-the art literature sources and experts opinions regarding specific service types?

Our granularity framework is a hypothesized model that we put to the test in the second part of this research project, by evaluating case studies to reveal granularity issues in practice. We look at the service granularity decision-making process in five cases and assess the hypotheses from the framework. This leads to research question 2:

2. What trade-off decisions are made in the case study projects about the granularity aspects?
    a. How do architects decide about service granularity given the service types used in the projects?
    b. To which extent do the case studies support the hypotheses from the granularity framework?

From the analysis of the case studies we can disentangle some general patterns for service granularity, which can be used in the service granularity decision-making process. These patterns are evaluated by architects on validity and usefulness.

3. What service design patterns can be derived from the analysis results of the case studies ?
    a. What aspects characterize the various levels of granularity in the case study projects?
4. How valid and useful are the granularity framework and these patterns?

1.3 Research Method

Our research method is presented in Figure 1.1. It includes the following steps:

First, an extensive review of XX literature sources on service design is used to identify typical service granularity aspects and types of services. While there is a lack of literature in this field, we also ask experts in this field about their opinions and experiences. From these sources, a granularity framework is created in which hypotheses about the relations between the aspects and service granularity are formulated.

Second, to get more insight into how granularity is used in practice, a comparative case study is conducted, to find out how and why these aspects are used. Four cases of SOA implementations are examined according to the Structured Case method (Carroll et al. 2000). These cases are SOA implementations which are completed by architects of Capgemini or other companies. More about the Structured Case method can be found in the case-study chapter. The hypotheses from the literature study are examined in detail regarding the results from the case studies. Taking into account the findings from the case studies, the granularity framework is adapted to a new version.

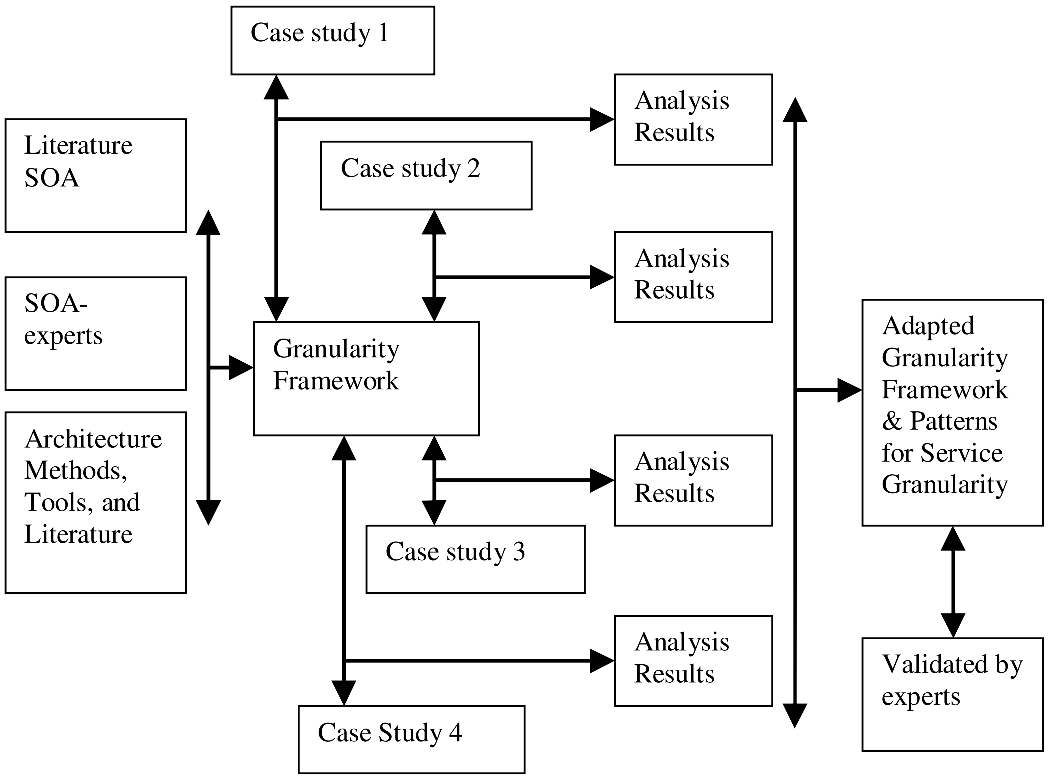


Figure 1.1 - Research framework

By reviewing the deliverables from the hypotheses, patterns can be recognized to improve the decision-making process about service granularity. These patterns are evaluated by using opinions of ICT architects who reviewed their validity and usefulness. This evaluation is carried out in individual sessions with two to three architects from Capgemini to give a first assessment of the patterns.

1.4 Structure of the thesis

The literature study is discussed in chapters 2 and 3. Chapter 2 discusses the impact of SOA with its advantages and disadvantages, different layers and service types. Also, diverse SOA design processes are explained and service design in general is considered. Chapter 3 focuses on service granularity and first gives a definition of this concept, after which the related aspects are discussed. Then, these aspects are mapped upon the service types. The result of the literature study is a granularity framework, which gives hypotheses about the aspects related to granularity.

Chapter 4 introduces four case studies of SOA implementations. First, the used method is described, after which the analysis and reflection to the granularity framework of every case is given. After that, an analysis and reflection over all cases is carried out and the adapted granularity framework is presented. Patterns are introduced in chapter 5. These patterns intent to simplify the decision-making process about granularity and are derived from the case study results.

Chapter 6 presents the validation of the research, in which the coherence of the research is described and the consensus and instrumental utility are reviewed by three reviewers. At last, the conclusions and future work are discussed in chapter seven.

2 Design of Service Oriented Architecture

Service granularity is part of service design and service design is part of SOA design and therefore, we start the literature study with the description of SOA design in general and service design in particular. Because of the limited availability of literature about service design, we interviewed several experts on this subject from Capgemini and other companies. Table 2.1 shows a list of participating experts and their roles at Capgemini. All experts experienced one or more service oriented projects in the role of architect. Their views are represented in Chapters 2 and 3 of this thesis. The interviews are referenced to in the reference list.

Date	Name	Role	Organization
9-1-2006	Raymond Slot	Certified Enterprise Architect and PhD student in architecture & business-IT alignment	Capgemini
20-1-2006	Martin Op ‘t Land	Certified Enterprise Architect and PhD student in enterprise engineering	Capgemini
24-1-2006	Har Gootzen	Managing Consultant	Capgemini
10-02-2006	Hans Goedvolk	Certified Enterprise Architect	Capgemini

Table 2.1 - Experts

In this chapter, we present a description of what an SOA actually is and what the impact of an SOA is. Then, we describe the service layers in an organization and analyze the layers which an SOA has effects on. Next, a description is given of the different types of services used by different authors and we map them on the service layers identified. Finally, we describe different methods to design SOAs and focus on the service design aspect.

2.1 Impact of Service Oriented Architecture

SOA is a concept, a process, and not a ready made solution which can be bought, installed and works. A definition of SOA is given by Papazoglou: “SOA is a way of reorganizing a portfolio of previously siloed software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols” (Papazoglou 2003). Introducing SOA into an organization makes it possible to change the organization’s IT and business processes on a low-cost base, using the applications available and only adding a new layer of services into the architecture. This makes the IT landscape responsive and integrated at the same time (Endrei et al. 2004).

The idea of dividing the architecture into components is not new. Parnas already discussed modularization and its advantages for systems (flexibility, comprehensibility) in 1972 (Parnas 1972). Since then, , many developments have been made in this field, e.g. distributed computing and enterprise application integration. However, these solutions still did not provide all wishes companies have about IT flexibility, low costs, etc. Companies are always looking for solutions with lower IT costs, more process integration, and a greater ROI (Channabasavaiah et al. 2004). SOA provides a unique chance for the first time in IT history to create artefacts that have an enduring value for both the business as well as the technology side (Krafzig et al. 2005). With SOA it may become possible to let components communicate, independent of their location and technology. This means that services should be technology neutral, support location transparency, and loosely coupled (Papazoglou 2003).

Because of these advantages, SOAs seem to be a promising solution for the IT problems companies cope with. It claims to have a number of business benefits for organizations, which are explained in the following section. To take fully advantage of an SOA, the architectural concept needs to be designed well. To make a first start in assessing the potential impact of SOA on an organization, we look into the business drivers for SOA and we address the layers of the organization which can be influenced by SOA. Once these facets are clarified, we present the different service types that are common in SOA.

2.1.1 Business Drivers for SOA

SOA is developed to address a business need to make IT more responsive and adaptive to constantly changing business conditions. It promises advantages like leveraging existing assets, simplifying integration and managing complexity, being more responsive, having a faster time-to-market, reducing costs, and increasing reuse (Dunn 2003; Endrei et al. 2004).

Capgemini identified five major reasons why SOA is better in reducing costs and delivers better IT solutions than all the former types of solutions (Curley et al. 2003):

- SOA implies a services mindset and this anticipates shared use of services
- SOA leads to monitoring and management of quality of the services so that experiences of the shared services can be exchanged
- Services encapsulate complexity and isolate changes by only exposing their interface to the outside world
- Standardisation of interfaces and the way they are exposed reduces cost and promotes reuse
- Services architecture focuses on the end-to-end lifecycle instead on single lifecycle stages.

Besides the earlier mentioned benefits, Capgemini also sees new business opportunities like an increased value of IT and improved sourcing options (Curley et al. 2003). However, why is SOA not implemented by everybody yet, while it has so many benefits? It is a huge project to switch to an SOA environment, which takes a couple of years. Furthermore, there are some technical issues that are not yet optimally taken care of, e.g. security standards are still evolving, quality of service can be improved, and transaction coordination and manageability are not yet working properly (Dunn 2003).

To choose for an SOA means dividing the business functions or processes in terms of services. This indicates that we have to decide on how to split the organization into smaller parts. Coupling and cohesion play an important role in identifying organizational parts (Papazoglou et al. 2006). Coupling is the degree of interdependence between two business processes and cohesion is the degree of the strength of functional relatedness of operations within a service. The services need to have a low coupling and a high cohesion (Papazoglou et al. 2006). But how to decide on the granularity of services? How much functionality should a service expose? These questions are not answered yet and are addressed in this thesis.

Business drivers like flexibility and costs are keys at the basis of SOA development. Flexibility is essential in the core processes of a business, while the supporting processes should be as inexpensive as possible. Most business processes are invented as core processes but on some point they will become supporting, in stead of core (SAP AG 2005a). This process is explained in Figure 2.1. For core processes it is important to be flexible in modification, while context processes should have low costs. From this we can learn that on different periods of time during the existence of a business process, it needs different levels of granularity. This has consequences for the granularity of services, which should be small (to create flexibility) when it concerns core processes, while context processes can be of a more coarse grained granularity because they are more stable.

Besides core and context processes, the strategy of an organization does also have an effect on the desired degree of flexibility in business processes. For example a low-cost leadership strategy from Porter (Daft 2001), which has an orientation on efficiency, focuses always on standardization, so the processes of these kinds of organizations are more in the right half of the business process life cycle and are less flexible. In contrast to the differentiation strategy from Porter (Daft 2001), which has a learning orientation, and focuses on flexibility, so its processes will be more on the left side of the business process life cycle.

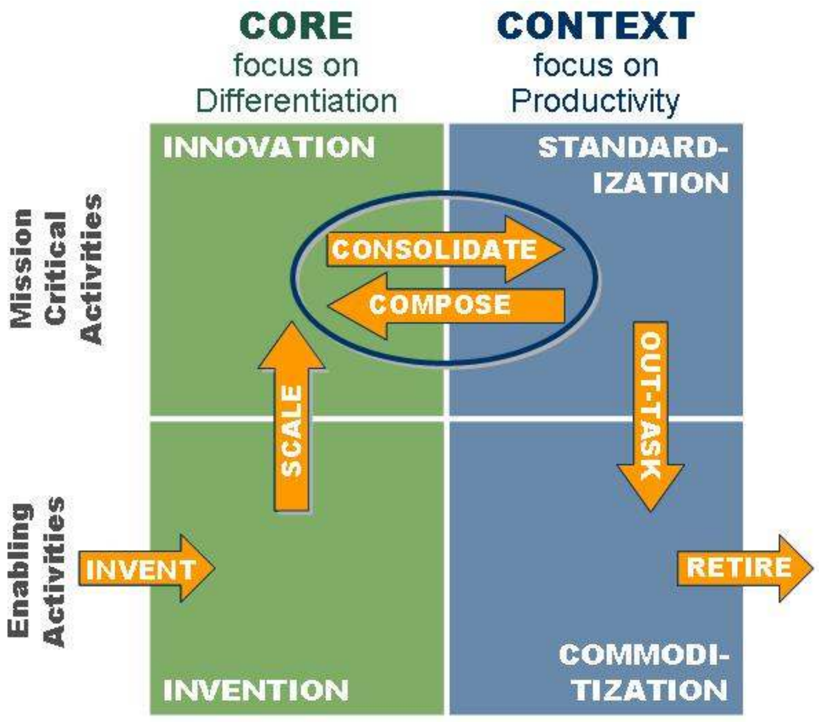


Figure 2.1 – Business Process Lifecycle (SAP AG 2005a)
Courtesy to G. Moore's "Living on the fault line"

There are many drivers behind the reasons to choose for SOA, while in the meantime there are some issues and problems with service orientation that need to be solved. However, while the potential of SOA is large, these issues and problems will probably be solved in the time that comes.

2.1.2 SOA Layers

Business and IT can be divided into layers and we indicate which layers are affected by SOA. We do this by giving two different layer classifications, those from Capgemini and the information systems department of the University of Twente, the two commissioners of this project.

The objective of SOA design is not only to find the best fit for the IT, but services also need to fit to the business processes of the organization. That is, business/information systems alignment plays an important role in SOA design. Van Eck et al. call this phenomenon application architecture alignment and use the following definition:

> "Application architecture alignment is the process that determines the optimal fit between a software application and the existing software and business environment." (Van Eck et al. 2002)

The authors present the alignment problem by using the GRAAL architecture framework (Figure 2.2). It presents the business reality in a service-provisioning hierarchy. The authors define five service layers, which exchange information through services. The business environment is the value chain in which the business operates. Application systems support or perform the business processes. The implementation platform is the software infrastructure which is needed for the application systems, and the physical network is the hardware infrastructure.

Capgemini distinguishes in their Integrated Architecture Framework (IAF; Figure 2.3) four aspect areas of architecture, namely business, information, information systems, and technical infrastructure (Van 't Wout et al. 2003). When we compare these to the GRAAL layers, we identify a number of similarities. The business area of the IAF covers the business services and processes that are required to support the mission of the business, which corresponds to the business environment and parts of the business processes layer of the GRAAL. The information area is about what information and knowledge is needed to support the business services and processes, in the GRAAL this is integrated in the business processes layer. The information systems (IS) area (in IAF) corresponds with the application systems layer (in GRAAL), and in technical infrastructure (in IAF), the software infrastructure and hardware infrastructure (both in GRAAL) are captured.

Besides these two frameworks, there are many more, like the Zachman Framework for Architecture (The Zachman Institute for Framework Advancement 2006) and TOGAF Enterprise Architecture Framework (The Open Group 2003). These also have a similar concept of layering like GRAAL and IAF, but for the purposes of our research, we focus on GRAAL and IAF. To find more information about how the architectural methods compare to each other, see (Greefhorst et al. 2003; The Open Group 2001). We use the frameworks to give an indication about the layers on which services are needed. Because the layers of the GRAAL fit more precisely to the service types we are going to use, we deploy the GRAAL to identify which business layers are influenced by the service types in question.

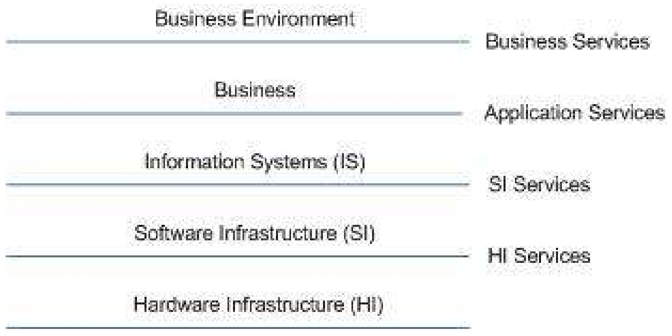


Figure 2.2 – Service layers in the GRAAL framework (van Eck et al., 2004)

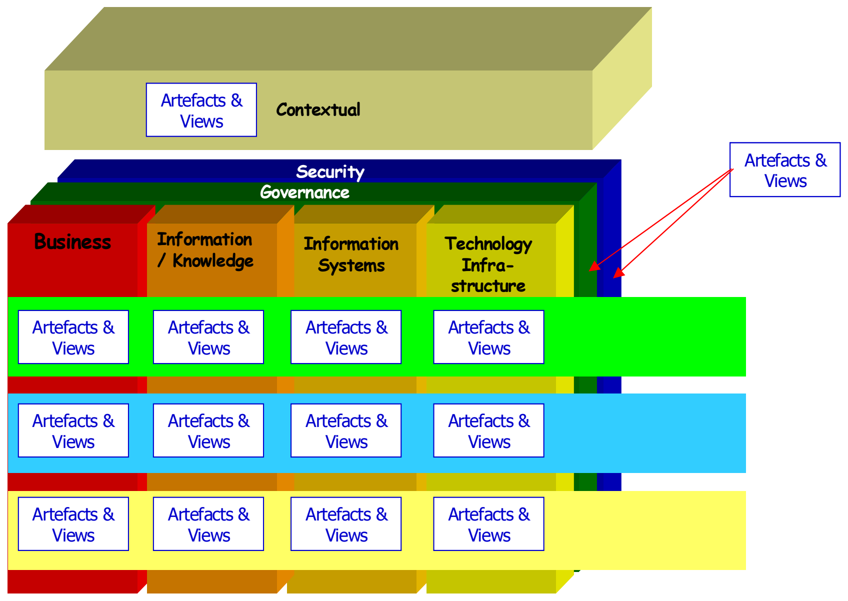


Figure 2.3 - IAFv3 (Cap Gemini Ernst & Young 2001)

2.1.3 Service Types

Before going into detail about service types, we first have to define what a service really is. There are many service definitions in the literature and some examples of these can be found in Figure 2.4. These are all IT service definitions. Capgemini identified seven basic principles for services, namely services must: have a clear identity, an understandable function, be trustable, be shared, have a safe and reliable interface, be of the right quality, and have business value (Van 't Wout et al. 2003). We follow these principles, instead of other service definitions because these seem to be useful for all types of services and show the most important characteristics of services.

Service Definitions:

A service is a software component of distinctive functional meaning that typically encapsulates a high-level business concept. (Krafzig et al. 2005)

A service is generally implemented as a coarse grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model. (Brown et al. 2005)

Services are self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications. Services perform functions, which can be anything from simple requests to complicated business processes. (Papazoglou 2003)

Enterprises services are Web Services that have an enterprise level business meaning. These are typically an aggregation of application (or web) services of lower granularity combined with simple business logic to support a step of a business process. (SAP Group 2004)

Figure 2.4 - Service Definitions

In an SOA, services are placed in a hierarchy. Business processes are translated to business services, which are converted to infrastructure services (Papazoglou et al. 2006). This is represented in the web services development life cycle hierarchy (Figure 2.5). In this hierarchy, only two types of services are distinguished, business services and infrastructure services. Business services are in this view automated business processes and infrastructure services are small software components. This hierarchy of services shows that different types of services have an influence on different layers. In the remainder of this chapter, we first introduce several sources with different types of services, and after that we map these services onto the GRAAL layers.

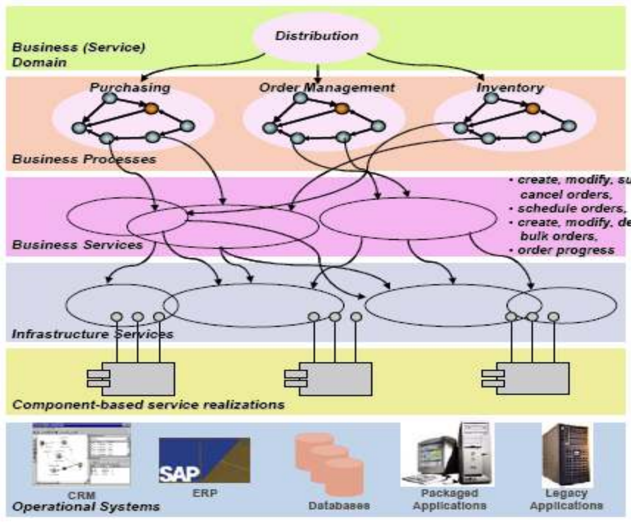


Figure 2.5 - Web Services Development Life Cycle hierarchy (Papazoglou and van den Heuvel, 2006)

There are many literature sources that mention services and give different types. In this research, we used, besides Papazoglou and Van den Heuvel, three sources. The first is Krafzig et al., who have written a bestseller book with SOA best practices. Krafzig et al. differentiate four classes of services, which are (Krafzig et al. 2005):

- Basic services that represent the basic elements of the vertical domain and are simple logic-centric or data-centric services. Data-centric services handle persistent data and logic-centric services encapsulate algorithms for complex calculations or business rules.
- Intermediary services that are stateless services that bridge technical inconsistencies or design gaps in an architecture. They are both client and server of an SOA. Four different kinds of technology gateways (bridge technology gaps), adapters (maps signatures and message formats), façades (provides a high-level view of one or more basic services), and functionality adding services (adds required functionality to a service) are represented in this type of services.
- Process-centric services that encapsulate the knowledge of the organization's business processes. They maintain a process state and are both client and server of an SOA. These services encapsulate process complexity, enable load balancing, leverage multi-channel applications, and separate process logic.
- Public enterprise services that are services that the enterprise offers to their customers and partners.

Secondly, the service types we discuss are from SAP. SAP is the first ERP vendor who is migrating to SOA, with their enterprise service architecture (ESA). SAP formulated for ESA two levels of services, namely enterprise services and below them enterprise operations (SAP AG 2005c). Enterprise services are preconfigured steps of business processes, based on web service technology (Woods et al. 2006). Every enterprise service should normally have between the 4 to 6 operations (SAP AG 2005b).

Capgemini, one of the market leaders in implementing SOA at customer sites (Hedin et al. 2005), defines three types of services: business services, which provide business value, information systems (IS) services, which provide value to business services, and technology infrastructure services, which provide value to IS services (Van 't Wout et al. 2003). These types of services fit in with three of the four areas identified in the IAF.

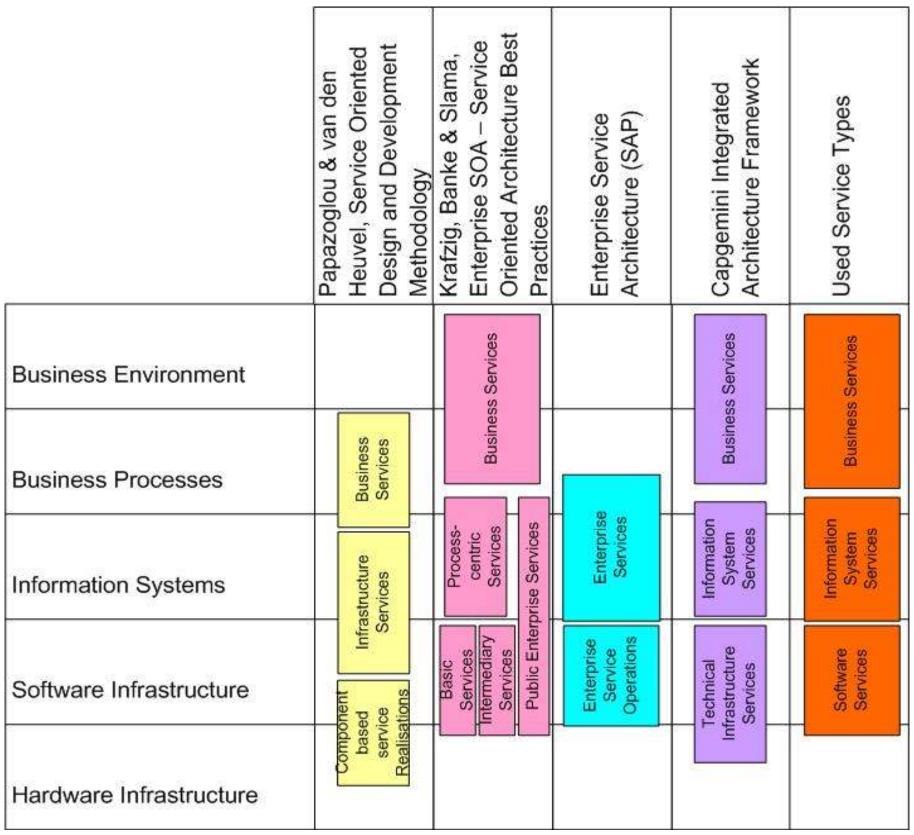

Papazoglou & van den Heuvel, Service Oriented Design and Development Methodology
Krafzig, Banke & Slama, Enterprise SOA – Service Oriented Architecture Best Practices
Enterprise Service Architecture (SAP)
Capgemini Integrated Architecture Framework
Used Service Types
Business Environment
Business Processes
Information Systems
Software Infrastructure
Hardware Infrastructure
Business Services
Infrastructure Services
Component based service Realisations
Business Services
Process-centric Services
Public Enterprise Services
Basic Services
Intermediary Services
Enterprise Services
Enterprise Service Operations
Business Services
Information System Services
Technical Infrastructure Services
Business Services
Information System Services
Software Services

Figure 2.6 - Service types mapped on service layers

To analyze the influence of the types of services on the different business layers, all services that we identified in our literature sources are mapped on the GRAAL. This mapping allowed us to compare the different services types with each other. Figure 2.6 shows the result of our mapping. The mapping leads to three distinct groups of services: business services, services at the business process and information systems layers, and software services. The services at the business process and information systems layers have all different names; we call them information system services because these services are pieces of automated business processes and are the services that have the control over the software services.

In all sources, there seems to be a fundamental distinction between business services and IT services. While business services themselves sometimes are seen as parts of automated business processes, are these in our view, services defined on business level which represents (parts of) a business process or business function. Information system services are pieces automated business process, which invoke other information system services or software services. We can conclude from this, that there is always a distinction between business and IT and that we can make classifications within these distinctions. For these types of services different information is needed to design them. Business services play a role in the business process layer and to design these, no knowledge is needed about information technology. The IT services are designed with the business services as input; so to design these, business processes should not be taken into consideration. This means that designing business services and designing IT services (information system and software services) can be seen as two different tasks. However, for making the right decisions, there should be at least one person who is involved in both tasks. Designing services is one part of the SOA design process. In what follows we describe this process in more detail and focus on the issues that are important for our further research.

2.2 SOA Design Process

There are many approaches for designing an SOA (Endrei et al. 2004; Hedin et al. 2005; Krafzig et al. 2005; Papazoglou et al. 2006; SAP AG 2005b). In this research, we describe the approaches provided by three consulting businesses: Capgemini, IBM, SAP and by the book Enterprise SOA. Capgemini and IBM are two of the four market makers in SOA (Hedin et al. 2005). First of all, we picked Capgemini because Capgemini also commissioned this research assignment, and secondly IBM because it has many sources about SOA on the web. The third SOA design approach we describe is from SAP. They are the first ERP software company that began shifting to SOA. Therefore, it is interesting to look at their approach as delivering an SOA methodology based upon existing technology and making it a commercial-of-the-shelf. The fourth and last method is from Krafzig, Banke and Slama, three authors who have many years of experience in enterprise IT in different companies and have written a book about their successful best practices in SOA since 1998, which is one of the bestselling books about SOA at the moment.

2.2.1 Capgemini's SOA Approach

Capgemini has designed SOA-like architectures for almost 10 years, although these early service architectures were not web-based (Cantara 2005). Service architecture is for Capgemini a design and

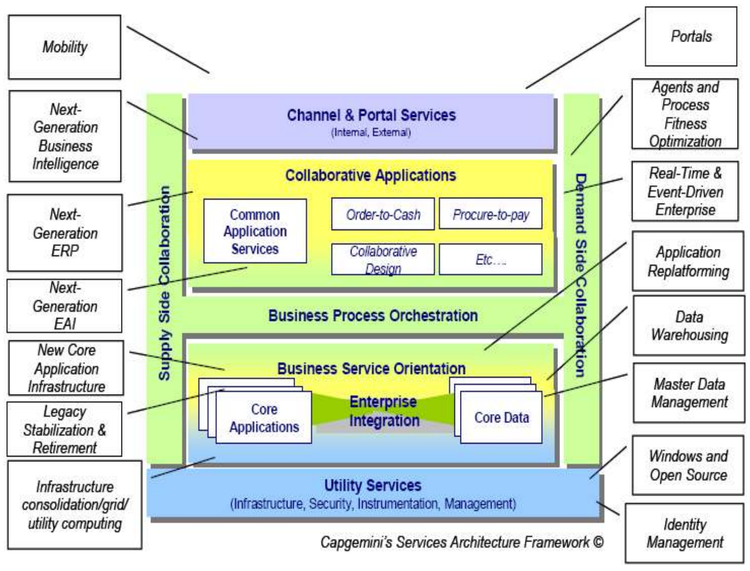

Mobility
Next-Generation Business Intelligence
Next-Generation ERP
Next-Generation EAI
New Core Application Infrastructure
Legacy Stabilization & Retirement
Infrastructure consolidation/grid/utility computing
Portals
Agents and Process Fitness Optimization
Real-Time & Event-Driven Enterprise
Application Replatforming
Data Warehousing
Master Data Management
Windows and Open Source
Identity Management
Channel & Portal Services
(Internal, External)
Collaborative Applications
Common Application Services
Order-to-Cash
Procure-to-pay
Collaborative Design
Etc….
Business Process Orchestration
Business Service Orientation
Core Applications
Enterprise Integration
Core Data
Utility Services
(Infrastructure, Security, Instrumentation, Management)
Supply Side Collaboration
Demand Side Collaboration
Capgemini's Services Architecture Framework ©

Figure 2.7 - Capgemini Services Architecture Framework

planning approach to corporate IT that delivers maximum flexibility and value (Curley et al. 2003). Capgemini uses its IAF for designing SOAs. The main goal of Capgemini Services Architecture Approach is to support continual and managed improvement. Capgemini uses a standard framework for governing the design of Services Architectures (Figure 2.7). Besides this framework, Capgemini uses reference architectures from earlier projects. A service architecture needs to start with identifying the business drivers and decide how to approach from this (Curley et al. 2003). Furthermore, the Capgemini Services Architecture Framework is used to make sure that every layer in the framework is addressed.

Capgemini uses its SOA Roadmap (Figure 2.7 - Capgemini Services Architecture Framework) to set out the path to develop an SOA. In this approach a tight linkage between business and IT is provided and the IAF always lies at the core of every step.

2.2.2 IBM´s SOA Approach

The SOA approach of IBM (Endrei et al. 2004) consists of seven steps and takes a top-down (discovering services from business processes) as well as bottom-up (leveraging services from legacy) approach. In Figure 2.8 these steps are represented. According to this approach, starting with SOA begins always with a business problem. Then the top down approach is started, identifying services from a business perspective. In the mean-time, the existing assets are analyzed to get an impression of what can be reused from there. The other steps include the allocation of services to components, their specification, and making a technology mapping. This completes the design process of an SOA.

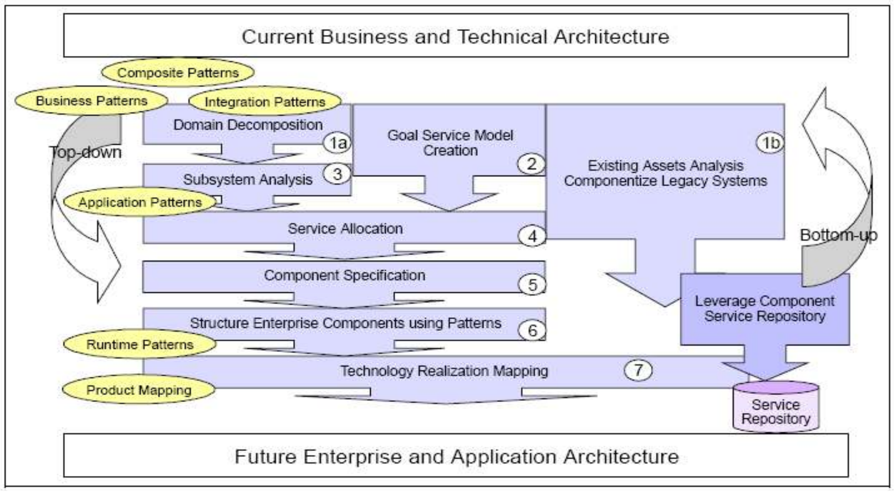

Current Business and Technical Architecture
Composite Patterns
Business Patterns
Integration Patterns
Domain Decomposition
1a
Goal Service Model Creation
2
1b
Top-down
Existing Assets Analysis Componentize Legacy Systems
Subsystem Analysis
3
Application Patterns
Service Allocation
4
Bottom-up
Component Specification
5
Leverage Component Service Repository
Structure Enterprise Components using Patterns
6
Runtime Patterns
Technology Realization Mapping
7
Product Mapping
Service Repository
Future Enterprise and Application Architecture

Figure 2.8 - Seven main steps of the SOA approach (Endrei et al. 2004)

2.2.3 SAP's Implementation Process for ESA

SAP, one of the biggest ERP vendors, is developing its own SOA, called Enterprise Service Architecture (ESA) based upon their latest ERP version. SAP's ESA is still in development, but SAP is presenting it as the solution which can better adapt to business processes and is rapidly deployable (SAP Group 2004). ERP alone is not solving the business problems anymore and, therefore, ERP systems have to evolve to survive.

ESA is a set of principles and guidelines for creating a flexible IT architecture with the high business value at the lowest possible cost (Woods 2003; Woods et al. 2006). The platform that makes it possible to realise ESA architectures is called NetWeaver. NetWeaver is SAP's open application and integration platform that enables the development, deployment and administration of services (Woods et al. 2006). ESA describes

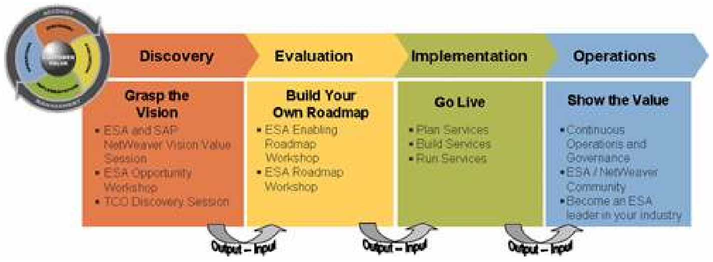


Figure 2.9 - ESA Adoption Program

enterprise services which provide value for the enterprise and which exist of web services. Composite applications extend or enhance business functionality by building a new application on top of the existing services.

SAP developed the ESA Adoption Program, which is their implementation process of ESA and consists of four phases. In the first phase the alignment of business and IT is important and the value of ESA should be recognized. This phase is called "Discovery". In the second phase, "Evaluation", the roadmap for the change to ESA is build, i.e. the business services are discovered and mapped upon the enterprise services. In the "Implementation" phase a plan is made about when to lay out which service. Then these services are build and run. The last phase is an ongoing phase in which possibilities for continuous improvement are explored.

2.2.4 SOA Roadmap (Krafzig et al.)

Krafzig et al. developed an architectural roadmap to implement an SOA. Therefore, they identified three expansion stages that indicate different maturity levels of an SOA (Krafzig et al. 2005). The stages specify the allocation of responsibility between the application front-ends and the services. The first stage is the fundamental SOA, which is the base for an enterprise application landscape. It is a simple SOA, in which two or more applications share live data, and it has still complex application front-ends. The next stage is the networked SOA. This SOA adds a layer with intermediary services (façades, technology gateways, etc) and deals with backend complexity. The intermediary services bridge technical and conceptual gaps and application front-ends are less complex. This type of SOA enables the organization to integrate applications independently of underlying technology. The third expansion stage is the process-enabled SOA. This adds another layer to the SOA, the process layer. Process-centric services are stateful, and can, therefore, be used for encapsulating the complexity of processes, sharing state between multiple clients, and handling long-living processes. Front-ends in this type of SOA only have to deal with user interaction and complexity of processes and backend is encapsulated in process-centric and intermediary services.

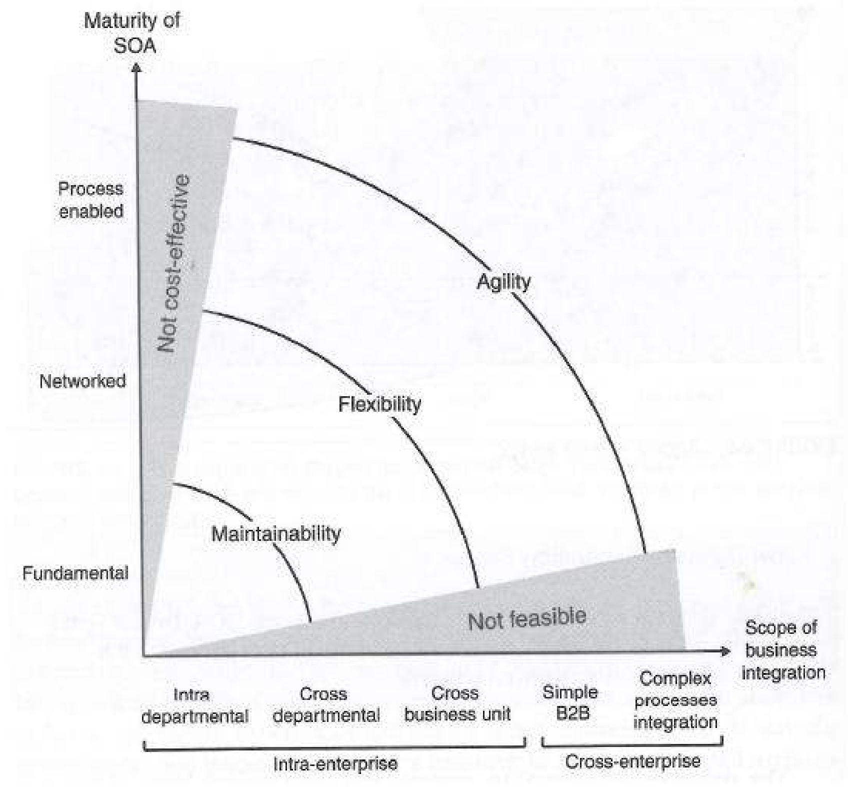


Figure 2.10 - SOA Expansion Stages plotted against the scope of the business integration (Krafzig et al. 2005)

2.2.5 Evaluation of SOA Approaches

We evaluated the four approaches by searching for similarities and remarkable differences in their approach. While the four approaches vary widely in structure (from step-by-step plans to frameworks), it is still possible to compare them on the similarities in terms of process aspects they include and the parts they have effect on.

The Capgemini method as well as the IBM method start with designing the business architecture. So, starting with a top-down approach seems to be normal. However, when there is legacy present in the organization, this also needs to be regarded. Therefore, we need a bottom-up approach as well. We call this a meet-in-the-middle approach (Zimmerman et al. 2004) and it needs to solve the gap that occurs between what the business wants and what the current information systems can provide.

Both SAP and Krafzig et al. recommend organizations to start small and slowly expand SOA. Creating an SOA is a process that takes years, and to gain enough in the early phases, it is important to innovate incrementally (Dunn 2003). Choosing projects which are small, but create gain (the so-called quick wins) has two advantages: the knowledge of the organization about SOA can grow and the organization gets used to SOA.

Designing an SOA can be scoped as an enterprise-wide project, as a project referring to one business domain only, or as a single projects, according to principal consultant Slot of Capgemini (Slot 2006). The smaller the scope is, the higher the level of detail. These scopes can also be placed in a hierarchy: an enterprise SOA project exists of a couple of domain projects, which exists of many small projects. The roadmaps presented here can be used for all these scopes, but are most relevant for the larger projects. When going into detail, more in depth roadmaps are necessary. Designing implementable services is done in the projects and not in enterprise or domain designs. Granularity does play a role in these scopes, but it is more in an indicative way. Therefore, we focus on designing services for single projects.

2.3 Service Design

Designing services is a small part of the complete SOA design process. We have to design different types of services, namely business services, information system services, and software services, and, as we have seen in the previous paragraph, we have to do this in two ways: bottom-up and top-down. A top-down approach is preferred, to start from the business drivers and build an architecture which completely supports the business. However, often legacy applications have to be taken into account in which a compromise has to be reached about the best fit for business and the possibilities of the current IT. In the following some methods about how to derive services are explained.

2.3.1 Service Design Methods

For designing the business service architecture, Jones and Morris from Capgemini developed the following method (Jones et al. 2005). This method starts with creating the big picture of the architecture, which determines the scope of the services and identifies which services talk to another and to which external actors. The method starts at the top of the domain and uses the organizational functions to create services in stead of the high level business process because business processes are about to change and services need to be able to step in to that. The big picture is decomposed into finer grained services until the desired level of detail is achieved. After that, support services and shared services are identified and assimilated in the service structure (Jones et al. 2005). They state that a top down method is used to identify a business architecture, not starting by identifying services from business processes, but from tasks. Flexibility of these business processes is one of the main benefits of an SOA, but is enough flexibility achieved when designing services from the current business processes? Therefore, identifying tasks is an important process that cannot be neglected.

The design process of SAP for its enterprise services exists of three levels (SAP AG 2005b). On the lowest level single enterprise services are discovered and designed with the help of some indicators and design guidelines. Level 2 designs systems of related services to support more complex business processes. We can call these workflow services, an important subclass of services with operational behaviour described as a workflow (Massuthe et al. 2005). Level 3 is the service enabling of many different enterprise applications. This means that more and more applications are considered for making them SOA compliant.

In (Fatolahi et al. 2005) a method for service discovery based on business processes is provided. For that purpose, they couple business process granularity to service component granularity. These authors state that a business process is a collection of services and that, by drawing use cases, services can be derived from the business processes. A business component is a collection of services and can be represented by a business use case. This business use case is based on elementary business processes, but no guidelines are given to discover these elementary business process parts.

Stojanovic et al. also use UML use cases to design services. They distinguish two basic service component types: business service component (BSC) and application service component (ASC) (Stojanovic et al. 2004). Because both types are automated services, we can compare the BSCs with the information system services mentioned in the previous chapter and the ASC with the software services. BSCs have to be identified with domain object analysis as well as use case analysis (Stojanovic et al. 2004). They give some criteria about how to design services from the use cases, like "use cases that handle the same domain information objects by creating, updating or deleting them should be under the responsibility of a single BSC" (Stojanovic et al. 2004).

Perrey and Lycett give some higher-level criteria about how to identify business and technical services (Perrey et al. 2003). According to them, in identifying business services it is important to look at the business value of the service and not at technology. Business services are defined by context not by size. Technical services are small parts of functionality, which are abstracted from their context. Technical services should be efficient, reusable, and context-independent. Services need to describe "the boundary between a consumer understandable value proposition and a technical implementation" (Perrey et al. 2003).

2.3.2 Evaluation of the Literature Survey

Our comparison of the methods from existing literature sources indicates that services are mainly derived from business processes or business functions. Indeed, most of the methods derive business services from business processes (Endrei et al. 2004; Fatolahi et al. 2005; SAP AG 2005b; Stojanovic et al. 2004), often with the help of use cases. However, Jones and Morris do have a point in mentioning that starting with the business processes tends to "drill-down" to early and leads to process "silos" of services (Jones et al. 2005). Also Op 't Land mentions this in the interview (Op 't Land 2006): services need to be defined on base of output and not on outcome. The business goal of a service must not be taken into consideration in designing the service, only the output of the service, i.e. the business function.

General guidelines in designing services are that business services need to have business value and should therefore, be chosen more coarse-grained. These services are decomposed into smaller services, where other aspects play a role, like reusability and complexity. In all these methods are some general guidelines given about how to design services, but every method gives different guidelines and there is no general framework to work with in deciding about the granularity of services. The granularity of services is the most difficult, but in the same time one of the very important aspects of service design.

2.4 Review

In this chapter we looked at the impact of SOA on businesses, the SOA design process in general and the service design in particular. SOA has an impact on both the business and IT side of the organization, while at the IT side this is reduced to the information systems layer and the software infrastructure layer. SOA seems to have good potential for the future, but still has to overcome some issues.

Many approaches are developed to design SOA, four of them are explained in this chapter. The most important lessons we have learnt from them are to choose a meet-in-the-middle approach when working with legacy systems and to start small. After that, we zoomed into the service design process. Services are often derived from business processes or business functions and use cases are mostly used to help with that. How to choose the size of services is a question that is not answered completely. In the next chapter we give more insight into the aspects that influence service granularity and develop a framework to support the decision-making process about granularity.

3 Service Granularity

Granularity is a term that reflects the degree of modularity of a system. Fine granularity implies more flexibility in customizing a system, because there are more, smaller increments (the so called granules) for clients to choose from (The Computer Language Company Inc. 2005). Subsequently, service granularity is the degree of modularity of services, or the scope of functionality exposed by a service (Papazoglou et al. 2006).

In what follows, we first dive into the granularity of a service and explain some more about that. Then, we sum up the aspects that have an influence on choosing granularity and we give definitions of the aspects and their effect on granularity. As an evaluation, we give the interrelations between the aspects. Building on this, we go into detail about what happens to service granularity in the different service types (business, information systems, and software services) and what aspects are important for these service types. Finally, our service granularity framework is presented.

3.1 The Granularity of a Service

Before defining the granularity of a service, we first have to define what a service is made of. A service exists of a contract, an implementation, and an interface. The contract contains the informal specification of the service, i.e. its purpose, functionality, constraints and usage. One element of this is a formal interface definition in for example WSDL (web services description language) or IDL (interface description language). The description of the interface is specified in the service contract. The implementation of the service physically provides the required business logic and appropriate data (Krafzig et al. 2005).

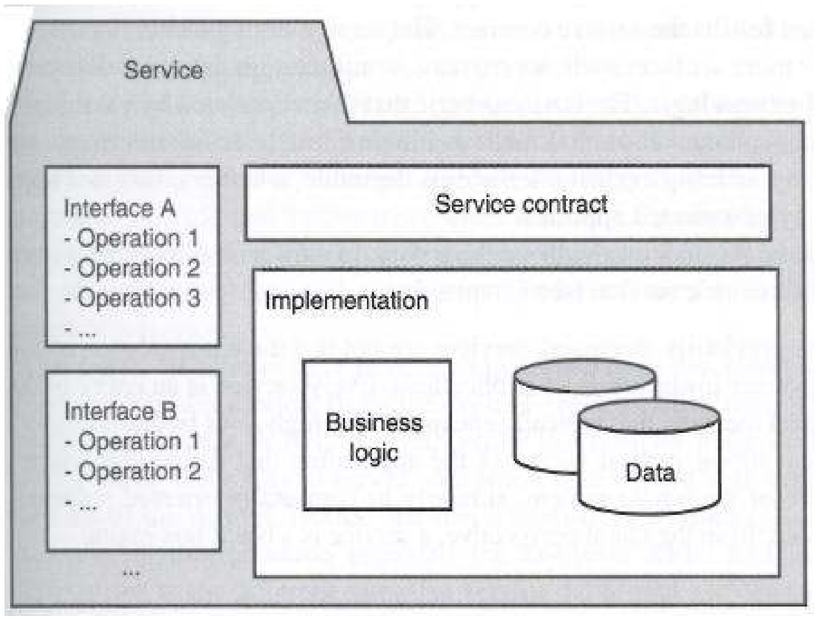


Figure 3.1 – Service (Krafzig et al. 2005)

The granularity of a service is influenced by all these elements. The description of the interface gives clear guidance in how the service is implemented and reduces complexity in the further implementation. Therefore, it adds to granularity. So, to talk about granularity, not only the size and complexity of the implementation play a role, but also the size and complexity of the interface.

3.2 Service Granularity Aspects

Dividing both the organizations processes and IT into services is a difficult task, which relies on many factors. Choosing which level of granularity is needed, depends on aspects like how flexible the organization wants to be and whether they are focusing on service reuse (Arsanjani et al. 2004). There are many literature sources (Arsanjani et al. 2004; Channabasavaiah et al. 2004; Fatolahi et al. 2005; Krafzig et al. 2005; McGovern et al. 2003; Papazoglou 2003; Papazoglou et al. 2005; Papazoglou et al. 2006; Perrey et al. 2003; SAP AG 2005b; Sims 2005; Stojanovic et al. 2004) about service oriented architecture and about how to define and design services. Many of these sources mention the concept of granularity for a certain type of service, but often it only states that one has to choose the right level of granularity for the services. Furthermore, there are no methods or guidelines about how to choose this level. In this paragraph we identify aspects that influence the level of granularity chosen, and give definitions for these aspects. We do not distinguish between the effects of these aspects on the different service types (while some are more businesslike, others are more IT-based), but we give more insight into this in a later paragraph.

3.2.1 Functionality

Functionality "captures an intuitive notion of the amount of function contained in a delivered product or in a description of how the product is supposed to be" (Fenton et al. 1997). The functionality of a service says something about its granularity. Functionality can be measured by function points. Function points are able to express the complexity and size of software applications in a number (Furey 1997).

Business functionality is about the useful tasks on business level, i.e. the business functions that provide value to the organization. Business functionality has an influence on the granularity of business services in the way that it determines the degree of granularity of a function. IT functionality, on the other hand, is the functionality provided by an application. IT Services also have a measurable degree of functionality and this defines the degree of granularity of the IT function.

3.2.2 Flexibility

One of the key benefits of an SOA is that it makes the IT landscape flexible and adaptable to changing business situations (Papazoglou et al. 2005). Therefore, flexibility is also recognized as one of the most important criteria for choosing granularity. However, only designing for flexibility gives a large amount of small services, which causes problems with maintainability, retracing services, and higher development costs. For this reason, it is important to design services in such a way that both business processes are flexible, the underlying design is well-organized, and development and maintenance costs are low.

Being flexible is being "able to change or be changed easily according to the situation"(Cambridge University Press 2006). Being flexible in business and IT will mean being able to change business processes or IT easily. Flexibility has an influence on the level of alignment between business and IT. Knoll and Jarvenpaa defined flexibility as the ability of software to change or fit the changing turbulent environment (Knoll et al. 1994). Knoll and Jarvenpaa identify three types of flexibility: flexibility in functionality, in use, and in modification (Knoll et al. 1994).

- Flexibility in functionality is the ability to withstand variability in input conditions. This will result in a service that operates well in many different environments.
- Flexibility in use is related to the outcome of a function. It is the strategic flexibility that is useful when decisions are made toward unknown outcomes. It represents the number of different tasks for which a service can be used and has a strong link with reuse.

- Flexibility in modification is the ease and variability with which the technology can be modified. Being able to change the business processes of an organization quickly means that they can be easily modified.

According to Nelson et al. (Nelson et al. 1997), technology flexibility includes the characteristics of technology that allow or enable adjustments and other changes to the business process. They define technology flexibility as the characteristics of technology that allow or enable adjustments or other changes to the business process. They proposed a measurement framework for technology flexibility that include such factors as modularity, change acceptance, and consistency in the structural flexibility dimension, and rate of response, expertise, and coordination of actions in the process flexibility dimension (Nelson et al. 1997).

Looking at flexibility and its influence on granularity in services, we actually see two types of flexibility that influence service granularity. The first is the flexibility to change business processes quickly and the second is the flexibility to use services. The first has the influence on granularity that it reduces the service functionality to be able to make other compositions of services quickly, while the other is the reusability of services and will therefore be handled at reusability.

3.2.3 Reuse

"Reuse is the process of adapting a generalized component to various contexts of use" (Basset 1997). That means that reusing services is using services again for other purposes. This lowers the development costs for the services as well as the maintenance costs, while the development time is shortened and a higher level of quality can be achieved (Poulin et al. 1993). Reuse is important, also to prevent redundancy in the system.

Two of the concepts of modularity of McGovern et al. are related to granularity and reuse (McGovern et al. 2003): 'modular decomposability' and 'modular composability'. The first divides an application into many smaller modules. Each module should be responsible for one function within an application, so that it can be reused. The second criterion refers to the degree in which services can be combined to one bigger service.

The aspect 'reusability' often results in smaller services because all the function-specific parts are left out. However, it is also possible to reuse parts of business processes, which means that those parts can be fixed in one service. Composability is the process of combining services to create a more powerful service (Ferguson et al. 2003). For example, through combining functions from different legacy systems, composite services can be created which have new functionality. Composability is a special form of reusability.

Besides reuse of services, underlying functionality from applications designed earlier can also be reused in services. We call this reuse of legacy. An SOA is often built upon legacy systems. When this is the case, the functionality of this legacy is reused in the services. These systems are integrated in the SOA so that the important functions can be reused in the services.

3.2.4 Complexity

Complexity is closely linked to some of the other aspects, like genericity and reusability. Complexity is something to take into account when overdoing one of the other aspects. For example: when making a large service very generic (that is, lots of parameters to fill in), it becomes too complex to understand, or when putting to much functionality in a service, it becomes complex to execute. Complexity is a main aspect to consider because it lies at the core of many of the other aspects.

According to schools of thought in software measurement (Fenton et al. 1997), there are four types of software complexity measures, namely:

- Problem complexity, which measures the complexity of the underlying problem

- Efficiency which measures the performance of the software by reflecting on the algorithm
- Structural complexity measures the structure of the software used to implement the algorithm
- Cognitive complexity measures the effort required to understand the software

We must distinguish between the complexity of the problem and the complexity of the implemented solution. When the complexity of a certain problem is high, the problem should be divided into smaller parts all of which can be solved individually. Efficiency and structural complexity are both internal complexity aspects of the solution and therefore measurable in the solution. Efficiency refers to performance and is therefore handled by the aspect performance. Structural complexity refers to the number of parts and dependencies between them. Cognitive complexity is a measure about the understandability of the software and addresses the cognitive abilities of human beings.

3.2.5 Context-Independence

Context-independence means that a service does not need to have knowledge about its surroundings. This is also called "loosely coupled" (Papazoglou 2003). It means that no problem exists for services to get input from users, but they must not need to know states of other services or maintain an own state.

Foody mentioned in his web-log the aspects transactionality and state, which means that services should be self-contained and data changes should be done in one transaction (Foody 2005). Services themselves should be stateless and do not need to know anything about their surroundings. This is called context-independence. To be context independent, services need to include more information themselves, which can make them more coarse grained.

3.2.6 Performance

Performance is a quality attribute which specifies how fast the product shall be, or "how well a person, machine, etc. does a piece of work or an activity" (Cambridge University Press 2006). Performance requirements state how quick a software product should respond on a command.

Foody also gives some guidelines for performance and size of web services in his weblog (Foody 2005). Operations should be executed between 5ms and 5s, and messages should not be larger than 1MB. Performance is a constraint for granularity that cause services not to be too coarse grained or fine grained and can have an influence on both business as IT services.

3.2.7 Genericity

Genericity (also called generality) is the concept of making a service able to use in many different ways. Dependent on the level of functionality in a generic service, it can be more reusable or more complex. Small generic services are often more reusable, while services with much functionality and many parameters are highly complex.

One more area mentioned by Foody is business suitability, which says: to understand the business, one should design services from that perspective for simplicity and generality. Making services simpler, makes them more general (Foody 2005). Genericity has something to do with reusability and composability of services because making a service more general or generic, makes it more suitable for reuse. However, the goal of making a generic service is to create a service with much functionality, but without unnecessary functionality (keep the service simple).

3.2.8 Sourcing

Sourcing is the decision-making process about having part of its work done by other companies (outsource) or keeping it inside (insource). To identify sourceable units, the coupling between service components should be low, while the cohesion should be high. Only secondary processes are eligible for outsourcing.

Because outsourcing of business activities is something increasingly common in today's business environment, it may make good business sense to keep business processes prepared for out- or insourcing. To simplify outsourcing, the business services have to fit in with the IT services, to make it possible to lift a part of the business process as a whole out of the organization. Thus, the sourcing principle can be a reason to choose granularity in a way to support this (Arnold et al. 2002).

3.3 Granularity of Service Types

Different service types can have different granularities. Because services are placed in a hierarchy, it often applies, that the higher a service is located in the hierarchy, the more coarse-grained it is. Besides that, granularity aspects can have an influence on services at some layers, in stead of all layers. In this paragraph we summarize what we found in the literature about granularity and service types to get a more profound look on this. Besides the literature we also used the interviews for these purposes.

Business Service	**Information system service**	**Software Service**
Functionality	Functionality	Functionality
Flexibility in business processes	Flexibility in business processes	
Problem Complexity	Cognitive and Structural Complexity	Cognitive and Structural Complexity
Reusability	Reusability	Reusability
Composability	Composability	Composability
	Reusability of Legacy	
Sourcing	Sourcing	
Genericity	Genericity	Genericity
Context-independence	Context-independence	Context-independence
	Performance	Performance

Table 3.1 - Classification of Aspects in Service Types

Perrey and Lycett name, for example, performance, reusability, and context-independence as aspects for the software services (Perrey et al. 2003), and as we consider the other aspects, we can make classification about what aspects in what types of service exist. Table 3.1 shows our classification. We only discuss the aspects that are not important for all types of service. In other words: we discuss the empty cells of the table.

Business processes should be flexible, so flexibility has a main influence on business services, which have to be built in a way in which they can easily be rearranged. This also counts for information system services, because they control parts of the automated business process. While software services are called in those information system services, they themselves need to be stable. Only the orchestration of services needs to be flexible and this can be done in the information system services.

At the business level it is important to hide complexity and to understand the problem complexity to be able to translate it into IT solutions. At the IT level, it is more about making the complexity manageable.

Reusability is important in all types. It should be possible to reuse parts of business processes, as well as smaller software services, as well as the control parts of information system services. Composability is essential in all types. On all levels composing services should be regarded. Making services simpler and more general has advantages for both business and IT services, while it makes them easier to reuse and it makes the SOA more conveniently arranged.

Reusability of legacy is only important for the information system services. They have to include legacy functionality into their automated business process. This is often done by building wrappers, which act as interfaces between the legacy application and the information system service.

Sourcing is an aspect that is important for both business and information system services. To make sourcing a success, business services and IT services have to fit neatly into each other. Choosing the granularity for these service types has, therefore, a big impact on the success rate of the sourcing projects. Information system services are the pieces automated business process and these should fit in to the pieces business process that the business services represent.

Context-independency is important for all services types in different ways. Business services have to be placed within a domain (context), and within that domain they need to be as context-independent as possible. The most information system services can never be totally context-independent, because they represent a flow of services and need to know the state. A software service does not need to know anything about its surroundings or about the service by which it is called, it only has to do its action and give the result back.

Performance, a term specifically used in IT, is also important for business, according to Op ‘t Land from Capgemini (Op 't Land 2006). Knowing how many times a service will be used, or how many errors can be made is important for the design of the service. For example a service that shows the amount of traffic on the roads can be a business service. However, to do this for one particular road or getting statistics of the whole country are two completely different things. The performance requirement in the first situation is very high (being on the road, you have to know immediately whether there is a jam in front of you), while the researcher who needs the statistics can wait for some time on the data (Op 't Land 2006).

3.4 Granularity Framework

The previous subsections presented the aspects that can influence the service granularity, as far as we are able to see them at the time of writing. Furthermore, the difference in aspects for business and IS services is also shown. In Figure 3.2 a granularity framework is presented, in which all the aspects are placed, their relations to each other and, as can be seen, all aspects have a relation with granularity.

All the aspects have influence on each other. For example, a small generic service is very reusable and a large generic service can be very complex to understand (cognitive complexity). In Figure 3.2 these relationships are represented in the granularity framework. Each of the aspects has at least one relation with another aspect and all are related with granularity. Although the aspects’ relationships with granularity are not yet clear, we already examined how the aspects relate to each other with granularity in the back of the mind. The figure is probably not complete; more relations can be distinguished between the aspects. However, the figure gives an overview of how deeply the aspects are interconnected in relation to granularity and these are the results from the literature study and the expert interviews.

To go further, two business drivers have an influence on the level of granularity, namely flexibility and costs. Organizations want to have both flexible business processes as low IT costs, and this means that a compromise has to be found between those two. Low IT costs translated in SOA means that development costs and maintenance costs have to be reduced and many of the granularity aspects contribute to this (reuse, complexity, etc). Reusability is a key element in SOA; some people even say that a service which is not reusable should not be a service. Flexibility is obtained by (re)composing services. But then these services should have the right size, so that recomposing the business process means that only the service composition has to be recomposed and that the services themselves are not changed. The aspect performance is a quality attribute that has its own constraints in every project. While it is always present, the focus can lie more or less on them. Context independence and transaction consistency are both applicable to all types of services and are always important to consider. We always have to strive for generic services but how far we have to go in that depends on aspects like complexity and reusability. Reusability has two special forms, namely composability and reusability of legacy. Composability, reusability of legacy, and sourcing are aspects that dependent of the situation can have influence on granularity.

The granularity framework shows the relations between the aspects, service types, and granularity. This gives more insight into the complex world of designing services and choosing the right granularity of services. By using this framework in granularity decision-making it will be easier to think about side-effects of focusing on a single aspect to choose granularity on. Although, more research is needed to get more clarity on the granularity relations. In Table 3.2 these relationships are explained in detail and hypotheses are given about each relationship. These hypotheses are tested in the next chapter, discussing the case studies. Also, the granularity relationships are explored to a larger extent in combination with the case study. After that, directives are given about how to choose service granularity.

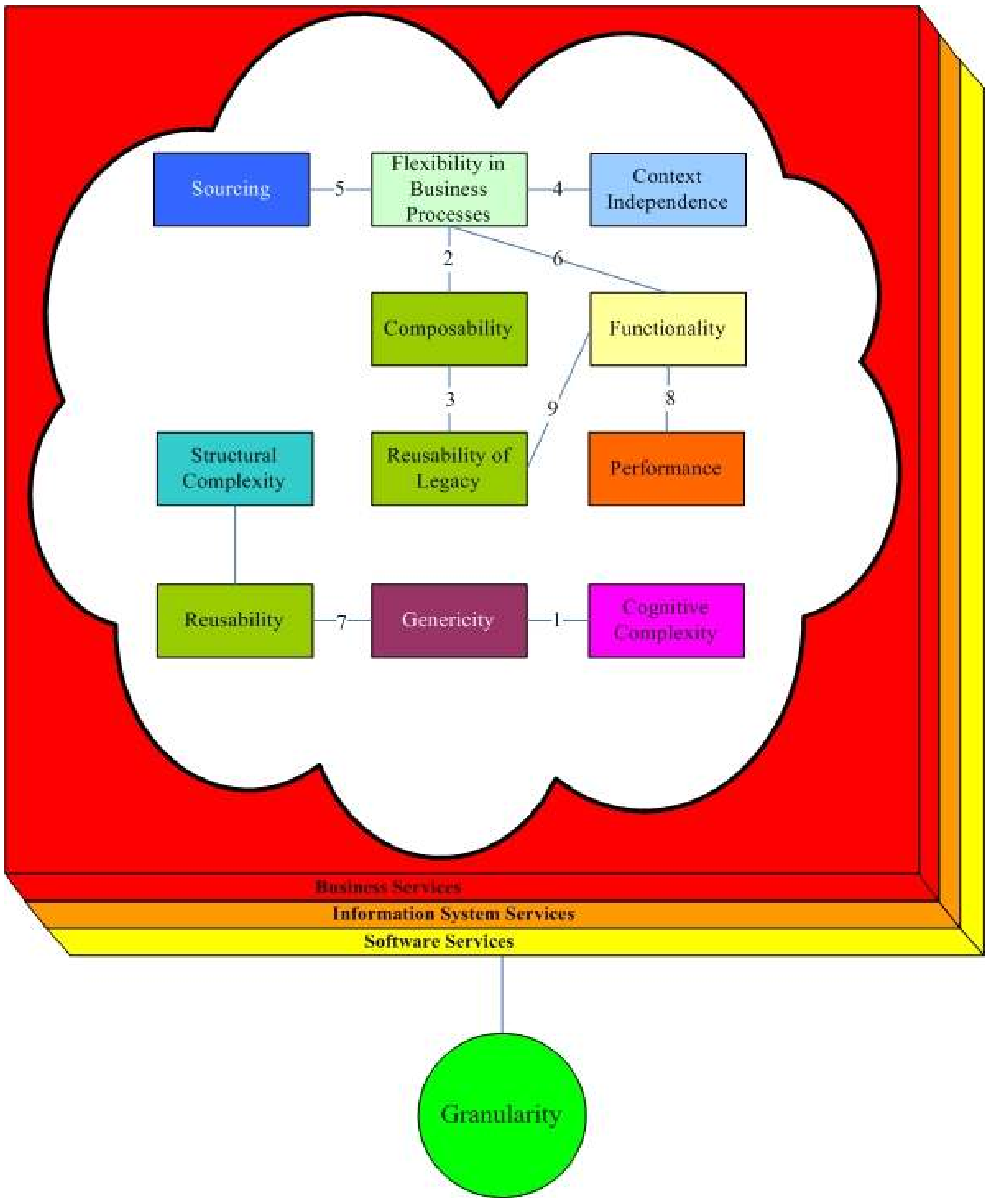


Figure 3.2 - Granularity Framework

Nr	Aspect	Relation with Aspect	Hypothesis	Literature References	Interviews
1	Cognitive Complexity	Genericity	Large generic services can be more complex to understand	(Foody 2005)	
2	Composability	Flexiblity in BP	Composing services makes the business process more flexible		(Goedvolk 2006; Op 't Land 2006)
3	Composability	Reusability of Legacy	New functionality is created by composing services from legacy systems	(Papazoglou et al. 2005)	(Goedvolk 2006; Op 't Land 2006)
4	Context-Independence	Flexibility in Business Processes	Services which are independent of their context are easier to rearrange, i.e. more flexible	(McGovern et al. 2003)	(Goedvolk 2006; Gootzen 2006; Op 't Land 2006)
5	Flexibility in BP	Sourcing	Designing for outsourcing makes business processes more flexible	(Foody 2005)	(Goedvolk 2006; Op 't Land 2006)
6	Flexibility in BP	Functionality	When business processes are flexible, it is easier to change their functionality	(Arnold et al. 2005)	(Op 't Land 2006)
7	Reusability	Genericity	Generic services are more flexible in use	(Foody 2005)	(Goedvolk 2006; Gootzen 2006)
8	Functionality	Performance	Performance requirements limit functionality options		(Goedvolk 2006; Op 't Land 2006)
9	Functionality	Reusability of Legacy	Legacy applications can be used to reuse existing functionality and to create new functionality		(Goedvolk 2006; Gootzen 2006; Op 't Land 2006)
10	Reusability	Structural Complexity	Reusability makes the structure of the landscape more complex		(Goedvolk 2006; Op 't Land 2006)

Table 3.2 - Hypotheses Aspect Relationships with Granularity

4 Service Granularity in Practice

To extend our literature findings, we examine case studies to investigate how granularity is used in practice. We look at the differences and similarities in granularity in two different characteristics, namely service type (business, application or software service) and software type (customer-build or commercial-off-the-shelf). First of all, we observe several service oriented cases, in which we compare the granularity aspects of the business and IT services with each other. These cases exist of one or more SOA projects within one company. Secondly, we analyze two customer-build cases implemented by Capgemini NL, one case in a big financial organization, and one case which is still in its infancy. Because these cases are confidential, we only mention their product category, and not the names of the businesses involved.

4.1 Method

Given the lack of research on service granularity and the fact that the boundaries between service granularity and its context are not evident, it seems appropriate to apply a qualitative approach to our research question. Specifically, we choose to use an approach based on the structured case method (Carroll et al. 2000; Yin et al. 2003). This method is used for several reasons: (i) it was found particularly well-suited to IS research situations in which an in-depth investigation is needed, but in which the phenomenon in question can not be studied outside the context in which it occurs, (ii) it offers a great deal of flexibility in terms of research perspectives to be adopted and qualitative data collection methods, and (iii) case studies open up opportunities to get the subtle data we need to increase our understanding of complex IS phenomena such as service granularity.

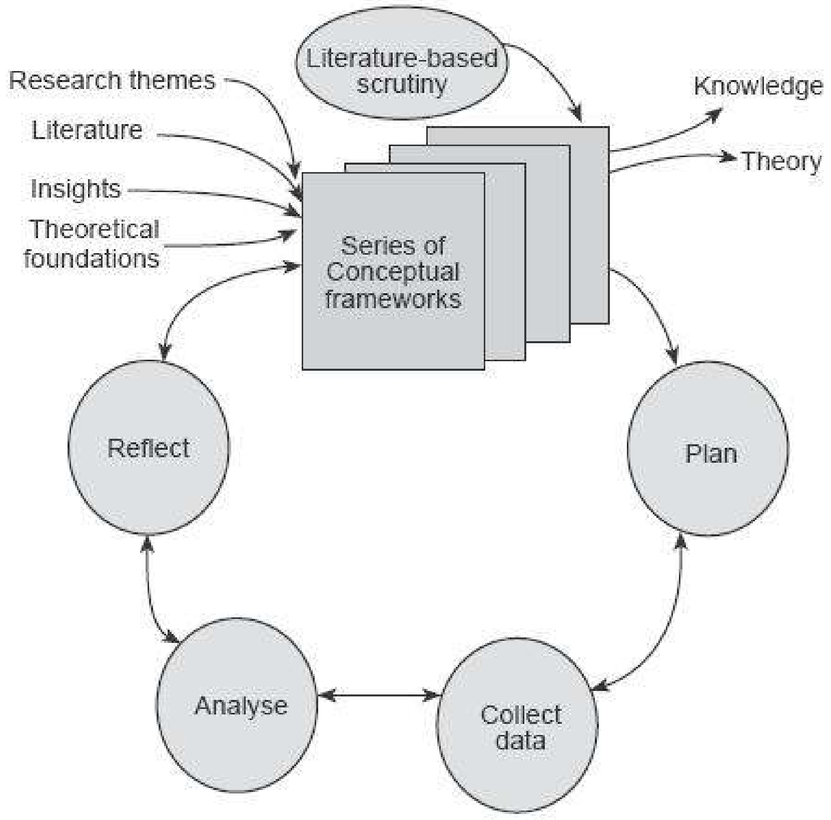


Figure 4.1 - The Structured-Case Research Method (Carroll et al. 2000)

The structured case method provides a framework that includes five phases, which are thoroughly described and aims to build theory in a rigorous manner (Carroll et al. 2000). Although, the structured case method focuses on the interpretivist field research, it is also suitable for positivist case study research as used by us. Positivist case study research is used because it is based on quantifiable measures, hypothesis testing, etc.(Klein et al. 1999). Choosing granularity is a human decision making process which is very complex, however we take a hypothesis testing approach, which is more based upon facts than upon subjective thoughts.

The granularity framework (Figure 3.2) provides the conceptual framework for the case study research. It explains the aspects that influence granularity and how these aspects are related to each other. The hypotheses (Table 3.2) are to be tested by looking at the results of the case study research. The research process is an iterative process (Figure 4.2) in which the conceptual framework is extended by including the results of each single step. In this project, the first cycle of the process is passed through. We start with the granularity framework described in chapter 3 and we adapt this according to the case study results.

4.1.1 Conceptual Framework

The input for the conceptual framework is the Granularity Framework (Figure 3.2) presented in chapter 3. The corresponding aspects and hypotheses defined in the granularity framework are the key concepts and relationships for the research.

4.1.2 Plan

We execute a hierarchical case study method, which means that we examine several cases independently from each other. This method is used because four different cases are studied, each of which defined by different drivers and offering different things that can be learnt.

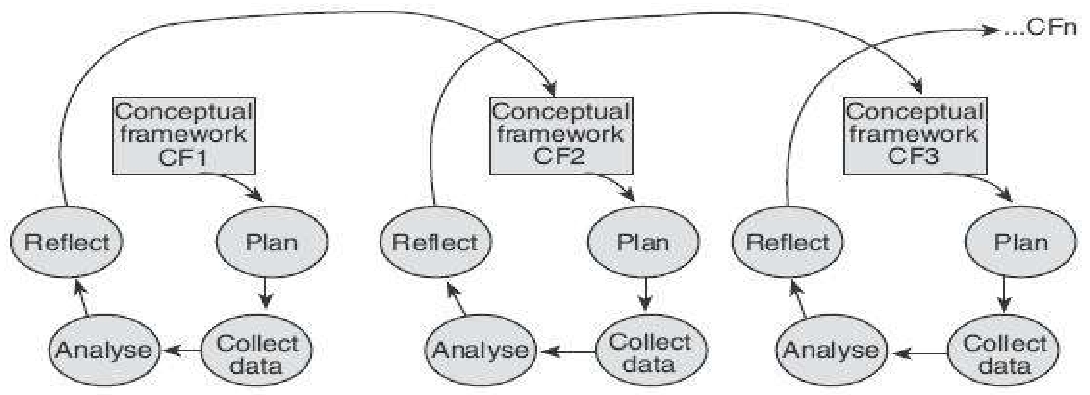


Figure 4.2 - A Spiral towards Understanding (Carroll et al. 2000)

Cases are selected by the criteria: project status, access to resources, and principles for service orientation. We use these three criteria because it is important to have some projects that are in an advanced state and have obtained much experience. However, it is at the same time interesting to look at projects that are still struggling with the granularity problems because in these cases architects can easily recall the details of the examples. All projects have to be service oriented, but it is interesting to notice differences between projects that use other principles for service orientation. We distinguish between between greenfield systems (also called new system development projects) and building upon legacy systems (also called system replacement projects). The criteria for the selected case studies can be found in Table 4.1.

4.1.3 Data Collection

We gather the data for the first four cases by face-to-face interviews with architects of the project teams. In two cases we interviewed project managers, which were suitable information sources because of their active involvement in the architecture process and they were easily accessible. Appendix B shows the global interview scheme for case Alpha, Beta, Gamma, and Delta. Interviews were conducted because of the lack of information about granularity in written reports and the complexity of the subject. The interviews had an open character in which we first asked the architects about their experiences with service granularity and, after that, they were introduced to our literature study and granularity aspects. Three of the four interviews are recorded; the fourth one is directly worked out on paper. The interview reports were written and, after that, we have created first drafts of the case study reports. These reports are discussed with the interviewees over email.

Project	Product Category	Kind of project	Project Status at the time of the interview	SOA principles	Interviewees
Alpha	Governance	New to build system to integrate 40 existing systems	Two weeks before completion	Customer-build Greenfield	2 software architects and 2 business architects of Capgemini
Beta	Finance	New to build sales system with thight links to legacy	Completed	Customer-build Build upon legacy	Project manager of Capgemini
Gamma	Finance	Complete backend system of which parts are service oriented	Several projects of which some are still running and others are completed	Customer-build Build upon legacy	2 enterprise architects of that company
Delta	Utilities	Making HRM processes and access and identity management service oriented	Design and small start with execution	Customer-build and parts are COTS Build upon legacy	Project manager of a consulting company

Table 4.1 - Case Studies

For each case study, we first give a project description of the cases, after which we present the observations of the interviewed architect about the granularity aspects.

4.1.4 Analysis

The analysis of the cases is performed by using coding (Carroll et al. 2000). The concepts of the conceptual framework are used as initial codes to structure the data upon. We do this by using the aspects to describe and arrange the experiences in the project into comparable units. In the case analysis we start with the business services and the effects of the aspects on these services in the light of granularity. After that, we do the same for the IS and software services together. In Appendix C the results of the coding can be found and an example description of how we arrived at the analysis results. Every case is analyzed independently looking at its influence on the granularity. By analyzing the table in the appendix, the results of the analysis were derived.

After the analysis of each of our cases, we analyze cross-case patterns using the case study approach from (Eisenhardt 1989). We do this by comparing the analysis results of the single cases with each other.

4.1.5 Reflection

In the final phase, we reflect on the findings of the case with respect to the granularity framework. In tables we present the hypotheses from the granularity framework and give possible adaptations to the granularity framework. After doing this independently for every case, we implement the proposed changes in the granularity framework by which we test every change proposed by one case to the other cases.

4.2 Case Alpha – Government

Case Alpha is a system replacement project for a national government organization. Forty existing systems had to be integrated into one new system by designing a new service oriented system. The two basic principles of this project were:

- Emphasize the generic parts of the architecture
- Separation of logistics (process flow) and content (dossiers)

The descriptions of Case Alpha can be found in Appendix D.

4.2.1 Analysis

Business service granularity is analyzed in this project. Granularity on business level is given attention by looking at the concepts functionality and flexibility. Business services are logical units of work which fall within one domain. Flexibility is created by separating the logistics (process flow) from the content so that it is easier to adapt the business processes.

At IS and software service level, generic parts in the design were utilized to reuse services. This reuse had the advantage that the landscape was kept maintainable, and complexity in the landscape was reduced, while the single services were (a bit) more complex. Reusing services was an aim in itself in this project. Services were as much as possible reused on all levels (from business service to software service).

The interviewed architect indicated that performance was a problem and that the asynchronous calls from services did not live up to the expectations of the organization. Therefore, small services were created, which could fulfil the performance wishes, but did not accommodate the design wishes mentioned above. We can learn from this, that performance demands can be that strong to overrule other demands. Also, we can see that it might sometimes be better to think about other solutions, to make sure performance demands are met.

In conclusion, we can say that reusability and flexibility were the two main drivers for the business architecture. Emphasizing the generic parts of the architecture was a main goal, while flexibility in the business process was created by separating logistics and content. These were also the two main goals of the design. The main aspects in the software architecture were reuse and performance. Reuse was an aspect involving the minimization of the number of services that were used. The performance demands of this project were very high, which sometimes caused things to be solved purely in order to satisfy these goals.

4.2.2 Reflection

Alpha supports some of the hypotheses in the granularity framework. In Table 4.2, we discuss every hypothesis in relation to the service types of Alpha. A '+' means the hypothesis is supported with the findings in the case and a '++' means that it is supported and it is even a remarkable point of this case, a '-' suggests that the case findings undermine the hypotheses, and the '--' indicates a remarkable point in the undermining. 'n/a' means that there was no information available in the case about this hypothesis, while the gray cells mean that the hypothesis is not applicable to that service type. The explanation describes the support or undermining of the hypothesis and in the last column the adaptations to the granularity framework are depicted.

In general, we can see that case Alpha supports many of the hypotheses, while some are not applicable to this case study. There is one hypothesis that particularly stands out positively in this case study, namely hypothesis 1. Hypothesis 1 mentions the inclusion of specific parameters into generic services. The architects in this project perceived this to be possible, under the condition that it should be understandable for the project team: i.e. the cognitive complexity of the service should not be too high. Relation 9 was not fully supported in this project. The generic services on software level were reusable for many different purposes, while the IS services provided more functionality, but were not reusable.

Nr	**Hypothesis**	**Alpha**			**Explanation**	**Adaptations to the Granularity Framework**
		Business Services	**IS Services**	**Software Services**		
1	Large generic services can be more complex to understand		++	+	To limit the number of services, generic services had to be made which provide much functionality, but were complex to understand	The amount of functionality a service provides also plays a role in this hypothesis; add functionality
2	Composing services makes the business process more flexible		n/a	n/a	There were no composite services	
3	New functionality is created by composing services from legacy systems	n/a	n/a	n/a	There was no reusability of legacy	
4	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible	+	+		By separating logistics from content, it is easier to change only the process flow or implement changes in the content	Making a division between different function types is important for this hypothesis; add functionality
5	Designing for outsourcing makes business processes more flexible	n/a			Outsourcing was not an option	
6	When business processes are flexible, it is easier to change their functionality	+	+		See 4; because services do not have a context it is easier to change them	Goes to relationship 4
7	Generic services are more flexible in use		+/-	+	The large generic services were not reusable, while the small generic software services were reusable in many different occasions	
8	Performance requirements limit functionality options		+	+	Physical implementation of services had to be adapted to meet the performance demands	

Nr	Hypothesis	Alpha			Explanation	Adaptations to the Granularity Framework
		Business Services	IS Services	Software Services		
9	Legacy applications can be used to reuse existing functionality and to create new functionality	▨	n/a	n/a	There was no reusability of legacy	
10	Reusability makes the structure of the landscape more complex	▨	+	+	There has to be a balance between the complexity of the service landscape and the complexity of the service	

Table 4.2 - Reflection to the Granularity Framework of Case Alpha

4.3 Case Beta – Finance

Case Beta is the design and implementation of a system for business products, from sales to administration. The project had two goals: from business perspective, the project objectives were to support workflow and implement a multichannel approach in the front-end to create flexibility. From IT standpoint, the project goal was to build a system that is easy to maintain and to extend.

The descriptions of case Beta can be found in Appendix D.

4.3.1 Analysis

To create flexibility, elementary task steps were transformed into services, while a multichannel approach took care of separating the channel from the content. Transaction consistency is important in this perspective, every service had to leave a consistent state.

To support the goal maintainability, a very generic and reusable design was made which in the end resulted in project failure. Defining generic services caused too much complexity in this project. The overview of the subject matters, the new system and the old system were too complex. No project member had taken this into account. Besides that, there were too many connections with the legacy system, which gave performance problems and difficulties during the testing phase.

In conclusion, we can say that the bottlenecks in this project were the focus on reusability and genericity, while flexibility and testability were underrepresented. Although granularity was not the only reason for failure (there were also errors in the design), the granularity choices did contribute to this.

4.3.2 Reflections

This project failed and many of the hypotheses are, therefore, not supported. Genericity was a main concern in this project and this leads to an adaptation of the first hypothesis. Furthermore, we add two new aspects to the framework, namely testability and transaction consistency. These aspects seem important to take into consideration while deciding about service granularity and are, therefore, added.

Nr	**Hypothesis**	**Beta**			**Explanation**	**Adaptations to the Granularity Framework**
		Business Services	IS Services	Software Services		
1	Large generic services can be more complex to understand	▨	++	++	Trying to make one generic solution made this project too complex	Adaptation to the hypothesis: to prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts
2	Composing services makes the business process more flexible	▨	-	-	The services which were composed from the legacy functionality prevented redundancy in the system, but included much complexity	
3	New functionality is created by composing services from legacy systems	▨	+	+	To prevent redundant information, the necessary functionality from the legacy system was reused in the new system	Add functionality
4	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible	▨	+	▨	Services were too dependent upon their context and were, therefore, not flexible	
5	Designing for outsourcing makes business processes more flexible	n/a	▨	▨	Outsourcing was not an issue	
6	When business processes are flexible, it is easier to change their functionality	+	▨	▨	Services were elementary task steps; transactions that had to be completed	Add new aspect: transaction consistency and add to the hypothesis: however, transaction consistency should be protected
7	Generic services are more flexible in use	▨	-	-	The generic services in this project were large generic services which included many parameters to satisfy the specific elements	
8	Performance requirements limit functionality options	▨	+	+	There were many service calls to the legacy system, which gave performance problems in this case	New hypothesis: Many service calls between systems in the SOA cause performance problems (aspects: Performance & Structural Complexity)

Nr	Hypothesis	Beta			Explanation	Adaptations to the Granularity Framework
		Business Services	IS Services	Software Services		
9	Legacy applications can be used to reuse existing functionality and to create new functionality	▨	-	-	This was done, however, in this project, services were too context-independent Services were too dependent upon their context, which gave problems during the testing phase	Hypothesis says the same as 3, so can be combined with 3 New hypothesis: When working with legacy, context-independence is even more important to keep the landscape testable (aspects: reusability of legacy, testability, context-independence)
10	Reusability makes the structure of the landscape more complex	▨	+	+	The reuse of the legacy made the structure too complex which caused problems during testing& maintaining	Add testability and maintainability

Table 4.3 - Reflection to the Granularity Framework of Case Beta

4.4 Case Gamma – Finance

This case study is a combination of service oriented projects in the back office of a large financial organization. The reasons to choose for services are to reduce and structure the complexity of the IT architecture and develop reusable services. At the business side, it is very important to be flexible in product types.

The description of this case can be found in Appendix D.

4.4.1 Analysis

The flexibility in product types is obtained by developing context-independent services which represent functional units of work. These services combined with the multichannel approach give the desired flexibility in the business processes.

Services in this project have to be reusable, and because of this reason there is a focus on building reusable and generic services. Maintainability of the landscape and testability of the services play an important role in this process, in order to keep the complexity level manageable. However, it seems difficult to reuse services over different projects. Documentation of services is a problem, even as the financial structure because it focuses on speed and low costs. In order to improve the reusability of services, it seems necessary to use some organizational guidelines.

Performance can be at odds with SOA because of the asynchronous nature of SOA. High performance demands seem to be a reason to look at other solution options in stead of SOA.

As concluding thoughts, we can say that the three most important aspects of this project are flexibility in business processes, complexity and reusability. Services are only built when they are reusable and they are built to reduce and structure the complexity of the landscape. Furthermore, the need to introduce new products to the market, asks for a flexible business process.

4.4.2 Reflections

This project made stable generic services, which did not lead to problems with complexity. It seems that generic services should include only generic parts and no specific parts to work properly. Reusability was a goal for every service, but at the same time the reusability over projects was not very high (because of bad documentation and a financial model that was focused on speed and low costs). Maintainability and testability are important aspects for this organization.

Nr	Hypothesis	Gamma			Explanation	Adaptations to the Granularity Framework
		Business Services	IS Services	Software Services		
1	Large generic services can be more complex to understand	▨	-	-	Generic services are small and stable.	
2	Composing services makes the business process more flexible	▨	+	+	Composing services creates flexibility in the business process	
3	New functionality is created by composing services from legacy systems	▨	+	+	legacy systems are opened up by wrapping techniques and new functionality can be created from this	Combine with 9
4	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible	▨	+	▨	Concepts are dependent of their context and it is difficult to get services (within one domain) as context-independent as possible	
5	Designing for outsourcing makes business processes more flexible	n/a	▨	▨	Outsourcing was not an issue	
6	When business processes are flexible, it is easier to change their functionality	+	+	▨	The multichannel approach in this project and the partition of the processes in functional units of work created the flexibility in this project	
7	Generic services are more flexible in use	▨	+/-	+/-	It was a demand for services to be reusable. However, because of a lack of documentation and a financial model that was focused on speed and being cheap, the reusability does not always get off the ground	Add to hypothesis: provided that services are documented well and the financial structure supports reuse
8	Performance requirements limit functionality options	▨	+	+	Performance is sometimes at odds with SOA	
9	Legacy applications can be used to reuse existing	▨	+	+	See 3	Combine with 3

Nr	Hypothesis	Gamma			Explanation	Adaptations to the Granularity Framework
		Business Services	IS Services	Software Services		
	functionality and to create new functionality	▨				
10	Reusability makes the structure of the landscape more complex	▨	+	+	The structure of the landscape needs to be testable and maintainable	Add aspects: Maintainability and Testability and add to hypothesis: and more difficult to test and maintain

Table 4.4 - Reflection to the Granularity Framework of Case Gamma

4.5 Case Delta – Utilities

This project focuses on service oriented projects in two processes of a utilities company: HRM processes and Identity Management. At the moment of the interview, this project is still in its infancy. The goals of this project are to improve processes, to reduce IT costs, to get improved access to the systems and to improve cooperation between the systems.

More about this project can be found in Appendix D.

4.5.1 Analysis

While this project has just been started with making services, they did not yet address many granularity problems. The goals of the project are to improve processes and to reduce costs. One important finding of this project was that it is possible to change the granularity of a service during the project period in order to improve the efficiency of the processes.

For the business services, the granularity is set by two guidelines: a process step is defined by a state change and these are divided into services by business ownership. The granularity of the more complex IT services is harder to define and, unfortunately, not addressed properly yet. At the moment, creating reusable and generic services is the most important finding.

In conclusion, we can say that at the moment in this project reusability and genericity have played the largest roles at the moment in this project at the IT side. At the business side, business ownership and the division into process steps created the flexibility the company wanted to have.

4.5.2 Reflection

While this project is not in a very far state yet, there are no direct adaptations to the granularity framework. Two things to keep in mind are business ownership and financial motives. Services do not have to be more flexible than needed and every service needs to have an identifiable business owner.

Nr	Hypothesis	Delta			Explanation	Adaptations to the Granularity Framework
		Business Services	IS Services	Software Services		

Nr	**Hypothesis**	**Delta**			**Explanation**	**Adaptations to the Granularity Framework**
		Business Services	IS Services	Software Services		
1	Large generic services can be more complex to understand	▨	n/a	n/a	Not seen yet	
2	Composing services makes the business process more flexible	▨	+	+		
3	New functionality is created by composing services from legacy systems	+	+	+	This is an option, however, sometimes other alternatives are better options, also taking into account financial motives	
4	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible	+	+	▨	Business ownership also plays a role in deciding about granularity. A service must not be owned by more than one business owner and of course no business knowledge should be provided in a service	
5	Designing for outsourcing makes business processes more flexible	n/a	▨	▨	Outsourcing was not an issue	
6	When business processes are flexible, it is easier to change their functionality	+	+	▨	Flexibility was created by dividing the processes into processteps with one business owner and a multichannel approach	
7	Generic services are more flexible in use	▨	+	+	Reuse is a demand in this project and when services are more generic it is easier to reuse them	
8	Performance requirements limit functionality options	▨	n/a	n/a	Not seen yet	
9	Legacy applications can be used to reuse existing functionality and to create new functionality	▨	+	+	See 3	Combine with 3
10	Reusability makes the structure of the landscape more complex	▨	n/a	n/a	Not seen yet	

Table 4.5 Reflection to the Granularity Framework of Case Delta

4.6 Analysis & Reflection over all Cases

We apply cross-case analysis to the four customer-build cases. We do this by selecting categories or dimensions to compare the findings in these categories between the cases (Eisenhardt 1989). Our dimensions are the codes from the single case analysis and these are the service types and the service granularity aspects.

We carried out a service type comparison and our results are presented in Table 4.6. The business services are comparable over three projects, only Gamma does not define business services. Between the IS and Software services the border is not that clear. The Beta project has identified three types of services that can play a role in both service types (although, most of the time, calculation services will be software services), and the Gamma project does not have many real IS services, but handles this with applications. However, some of their software services are moving more and more towards IS services. In the Delta project there are first level services at the top and system services and application interfaces at the bottom. In the middle are some service layers which are not totally defined at the moment (the project is still in development).

	Alpha	**Beta**	**Gamma**	**Delta**
Business Services	Business Services	Business Services	-	First level services
IS Services	IS Services	Calculation Services; External Services; Object services	Applications; IS Services	Some Middle layer services
Software Services	Software services		Data services & Transactional services	System Services; Application Interfaces

Table 4.6 - Service Type Comparison

In Appendix C the statements about the granularity aspects of the cases are compared. What strikes is that functionality and flexibility are decided upon in the business services in three of the four cases (the fourth case does not provide information in this respect). Functionality sets the borders for granularity by dividing the business services into logical units of work. Flexibility is created by the way of working and making decisions about separating process flow from content or channel from content.

Reusability takes a very important role in every project. Two of the four projects even say that services have to be reusable; otherwise they are not worth the effort. However, it is important to keep in mind that it should not be a goal in itself.

Performance can be important, depending on the client's priorities. Because services in an SOA are largely asynchronous, it is not the best performing solution. Therefore, when performance demands are very high, it should be considered whether it is feasible to take an SOA approach.

Our analysis indicates that genericity and complexity are closely linked. The functionality provided in one service should be limited to keep the service understandable, but this has the consequence that many services are designed, which leads to more service calls. Therefore, services have to be made as generic as possible. This can have huge impacts and should be done very carefully because adding parameters to a service makes it more complex and results in services that cannot be understood or reused. Therefore, there has to be a balance between generic services and a maintainable landscape.

For these four case studies, the most important aspects to decide upon granularity can be found in Table 4.7. Each case addresses reusability as a very important aspect. The aspects flexibility, genericity and complexity share a second place.

Alpha	**Beta**	**Gamma**	**Delta**
Reusability	Reusability of Legacy	Reusability	Reusability
Flexibility	Maintainability	Complexity	Genericity
Performance	Genericity	Flexibility	
Complexity			

Table 4.7 - Most Important Granularity Aspects for Cases

4.7 Granularity Framework

The new version of the granularity framework is based upon the adaptations proposed by every case study. All adaptations are verified to the other case studies by reflecting the materials available on the case studies. To be accepted, the adaptation should not contradict the findings in the other cases and it should add something to the current hypothesis. This process can be found in Table 4.8. In this table, the aspects that concern the hypothesis can be found, the new hypotheses and the acceptation of the cases for this hypotheses. A green cell with a √ in it, means that the case supports the hypothesis. A yellow cell with a ≈ in it, means that the hypothesis is not directly supported by the case results, although, it is not contradicted either. The grey cells show that these hypotheses are not addressed in the cases.

Nr	Aspect	Relations	Hypothesis	Alpha	Beta	Gamma	Delta
1	Flexibility in BP	Sourcing	Designing for outsourcing makes business processes more flexible				
2	Flexibility in BP	Context-Independence; Functionality	Context-independent services are easier to rearrange functionality, i.e. more flexible	√	√	√	√
3	Flexibility in BP	Functionality; Transaction Consistency	When business processes are flexible, it is easy to change functionality; however, business transaction consistency should be protected	√	√	√	√
4	Reusability	Structural Complexity; Maintainability; Testability	Reusing services makes the structure of the service landscape more complex and more difficult to test and maintain	≈	√	√	
5	Reusability	Context-independence; Genericity	Context-independent, generic services can be easily reused, provided that services are documented well and the financial structure supports reuse	√	√	√	√
6	Reusability of Legacy	Context-independence; Testability	When working with legacy, technical context-independence is even more important to keep the landscape testable		√	≈	
7	Composability	Flexibility in Business Process	Composing services makes the business processes more flexible		√	√	√
8	Composability	Reusability of Legacy; Functionality	New functionality can be created by composing services from legacy systems		√	√	≈
9	Genericity	Cognitive Complexity; Functionality	To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts	√	√	√	
10	Performance	Functionality	Performance requirements limit the amount of functionality in single services	√	≈	√	
11	Performance	Structural Complexity	Many service calls between systems in the SOA could cause performance problems		√	√	

Table 4.8 - Hypotheses Adapted Granularity Framework

These adaptations are also reflected back in the granularity framework (Figure 4.3) and this leads to three related blocks of aspects. The first block has its influence on the business and information systems services and handles hypotheses 1, 2, and 3. Hypotheses 2 and 3 are concerned with the flexibility of the business process and each aspect that influences that. Functionality, flexibility, context-independence, and transaction consistency are the most important aspects to think of when designing flexible services. Hypothesis 1 is about the influence of granularity in outsourcing or insourcing of processes. However, none of our cases handled with these matters, and, therefore, we cannot give statements about this hypothesis.

The second block handles the reusability and genericity of services and has influence on the IS services and software services. This block is identified by hypotheses 4 to 9. Hypotheses 5, 7, and 9 are supported by all the applicable cases, while hypothesis 8 is not completely supported by case Delta. While it is possible to create new functionality by composing services, it is not always the most desirable solution. Due to financial reasons, it is sometimes better to develop a cheaper solution, which gives the same end result, but is a bit less perfect. Hypothesis 4 is supported by Beta and Gamma, while case Alpha did not support or undermine this hypothesis. It seems that maintainability and testability become more important when projects are larger and legacy applications are concerned. Hypothesis 6 is about legacy and, therefore, only concerns the cases Beta and Gamma

The third block looks at the performance and complexity of IS services and software services. It deals with hypothesis 10 and 11. Both hypothesis are supported by the cases. Performance was a problem in three of the four cases, but in two different ways. Single services can be too complex that they take too much processing time. Also, many service calls can give performance problems. Therefore, it is important to consider structural complexity and performance to get a maintainable SOA landscape.

In conclusion, we can say that most granularity problems occur on one of the three areas flexibility in business processes, reusability, and performance. For these three areas, we develop four patterns to support the granularity decision-making process. These patterns are presented in the next chapter.

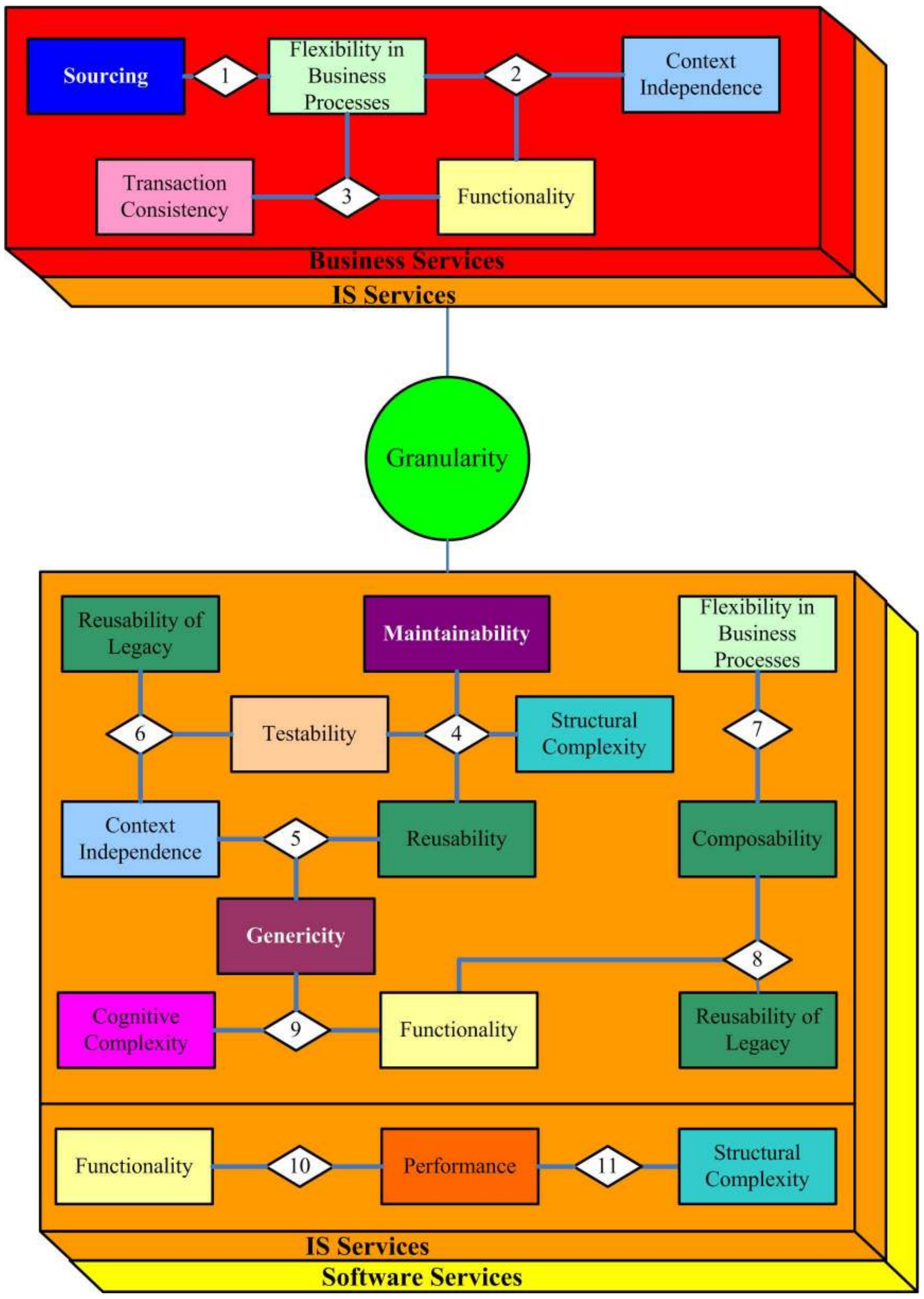


Figure 4.3 - Adapted Granularity Framework

5 Service Granularity Patterns

5.1 Patterns

Patterns were first introduced by Christopher Alexander in the building architecture world. He defined a pattern as a description of “a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use that solution a million times over, without ever doing it the same way twice” (Alexander et al. 1977). Since then, patterns have been used in many different areas including software engineering and software architecture (Buschmann et al. 1996; Fowler 2003)

A definition for a software architecture pattern is given by Buschmann et al.: “A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.” (Buschmann et al. 1996)

Patterns differ from guidelines in the way that guidelines give some kind of rules that are considered to be valid in almost every situation. Patterns, on the other hand, give some kind of context of a problem and give insight into the forces and constraints that can work on a problem. Because service design is a new field that is not crystallized yet, guidelines cannot be derived based on contributions from previously done research. However, some pre-patterns (patterns that are to be proven) can be formulated on the basis of case studies.

In this thesis, we adopt the pattern approach and use what we learnt in our literature review and our four case studies to formulate (pre-)patterns for service granularity. We call them pre-patterns because they are still under development and not in common use yet in the service oriented area.

5.2 Pre-Patterns for Service Granularity

We identified four pre-patterns which are all related to service granularity. Because service granularity is not only a technical aspect, but also has links to the organizational side of projects, the patterns we formulate are not only technological, but also organizational patterns . To present our patterns, we combine two methods of describing patterns (Buschmann et al. 1996; Fowler 2003):

- *Name:* a pattern name is meant to provide a common vocabulary (Buschmann et al. 1996; Fowler 2003)
- *Context:* short description of the background of the pattern (Buschmann et al. 1996)
- *Problem*: one sentence in which the most important problem the pattern solves is explained
- *Forces:* the forces that work on the problem; in this research: the granularity aspects
- *How it Works:* a description of the solution (Fowler 2003)
- *When to use it*: a description of the circumstances in which to use this pattern (Fowler 2003)
- *Related Patterns*: the relation with the other patterns explained (Buschmann et al. 1996)
- *Further Reading*: more information about the subjects of the pattern ((Fowler 2003)

As indicated earlier, the patterns we identify are pre-patterns which means that they are not in use yet, therefore, the elements normally seen in patterns, like ‘Examples’ and ‘Known Uses’ are left out in this research.

In Figure 5.2 patterns are mapped upon the granularity framework. The four patterns each cover their own part of the framework, although they have some overlapping aspects.

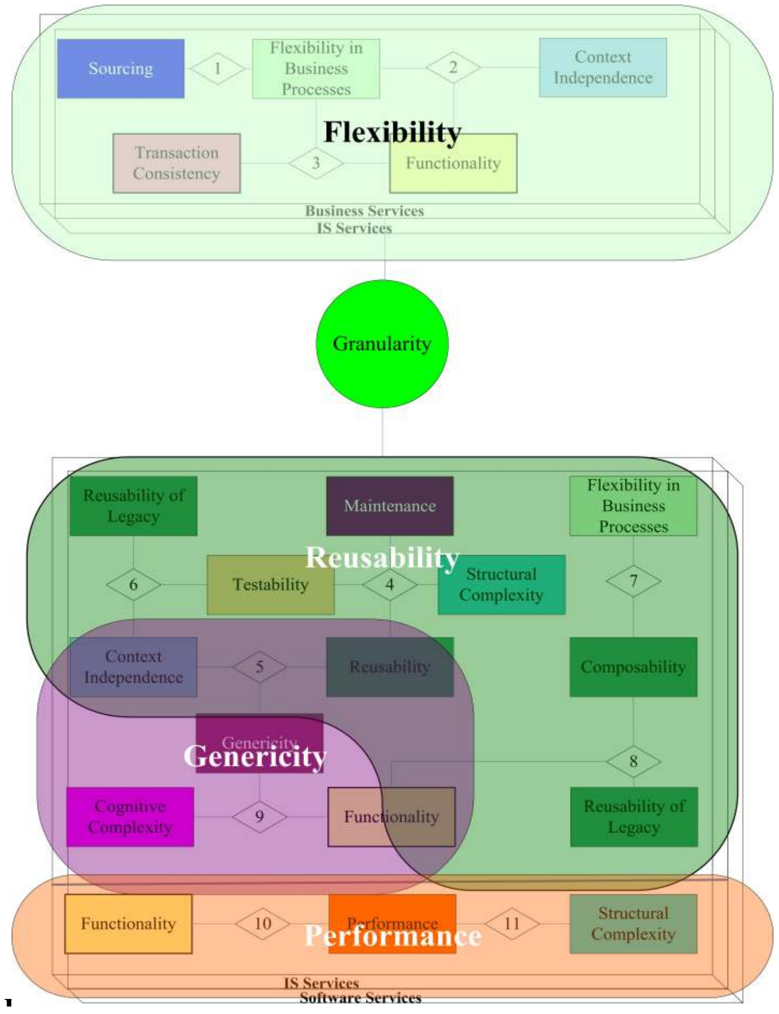


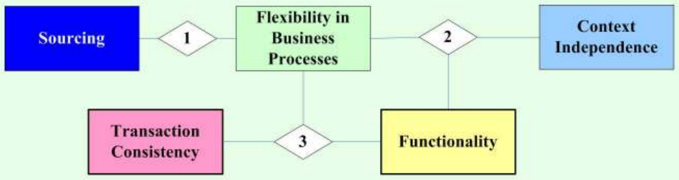
Figure 5.1 - Patterns positioned in Granularity Framework

Pattern 1: Design services for flexibility

Context
Organizations need flexible business processes which can be adapted to the current wishes of the market. SOA meets this requirement by dividing the adopting organisation into flexible services. Creating flexibility starts with the definition of the business services and has an effect on all underlying service layers.

Problem
How to choose service granularity to make both the architecture and the business as flexible as they want to be?

Forces
1. Designing for outsourcing makes business processes more flexible
2. Context-independent services are easier to rearrange functionality, i.e. more flexible
3. When business processes are flexible, it is easy to change functionality; however, business transaction consistency should be protected



How it Works?
Before designing services, the amount of flexibility required in the business processes should be considered. Not every business process needs the same amount of flexibility and because creating smaller services costs more money, it is important to determine the amount of flexibility needed. To decide upon which processes need flexibility and which processes do not, a division should be made between core processes and context processes. Core processes are the organization-unique processes and these have to be made more fine-grained than the remaining context processes.

Logical units of work set the borders of business services. Services have to operate within these borders. Guidelines to use when defining these logical units concern the aspects transaction consistency, business ownership, and context-independence. A service needs to leave a consistent state in the system, so a service needs to include a complete transaction. To make sure service ownership is taken care of, services should have one identifiable owner.

Services have to be as independent of technology as possible and should include as little business knowledge as possible. Business knowledge should be defined clearly and only the business knowledge that the service needs to fulfil its task should be in the service. All the knowledge that is unnecessary to complete its goal, should be left out. The necessary knowledge should be documented.

Two guidelines to create flexibility in the business process are to use a multi-channel approach and to separate process flow from content. These methods will make the business process more flexible.

When designing for in- or outsourcing, flexibility needs special attention. Further research should bring more clarity in the relationship between sourcing and service granularity.

When to use it?
The amount of flexibility required in the business process should be determined during the design of the business services.. This flexibility rate is then used throughout the next steps in the process when deciding on the granularity of the services.

Related Patterns
Designing Generic Services: By making more generic solutions, it will be easier to change business processes because one does not need to change these generic parts. Therefore, this pattern is related.

Further Reading
About Core/Context:
Khan, S. Weblog with explanation about SAP Core & Context processes. https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/2087

About Outsourcing:
Arnold, B., Op 't Land, M., and Dietz, J."Effects of an Architectural Approach to the Implementation of Shared Service Centers," in: *Second International Workshop on Enterprise, Applications and Services in the Finance Industry (FinanceCom05), to be held in co-location with the 13th European Conference on Information Systems (ECIS 2005)*, Regensburg, Germany, 2005.

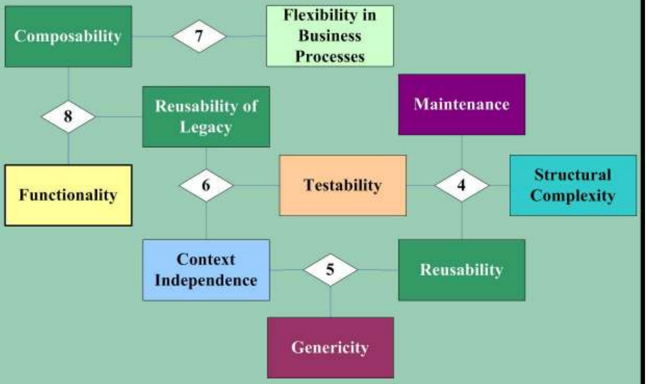
Pattern 2: Design Reusable Services

Context
Reuse is seen as the number one aspect to choose SOA and service granularity. Although reusable services are thought to provide the advantages SOA promises (e.g. lower costs, a faster time to market and simplified integration), SOA project realities indicate that services made for reuse are not reused in new projects. This is due to two reasons: the reusable service cannot be found and the functionality provided in this service is only a partial match with the new service.

Problem
How to design reusable services and organize the business in a way that these reusable services really are reused?

Forces

4. Reusing services makes the structure of the service landscape more complex and more difficult to test and maintain
5. Context-independent, generic services can be easily reused, provided that services are documented well and the financial structure supports reuse
6. When working with legacy, technical context-independence is even more important to keep the landscape testable
7. Composing services makes the business processes more flexible
8. New functionality can be created by composing services from legacy systems



How it Works?
Reusability has two special forms, namely composability and reusability of legacy applications. Composability is the creation of (new) functionality by combining multiple services. Reusability of legacy applications needs to get even more attention, because when legacy is not decoupled in a context-independent way, it will make the landscape more complex. To keep services reusable some guidelines are identified to fulfil the needs of reusable services:

Organizational guidelines:
1. Provide better documentation upon existing services, by giving more insights into the business and IS architecture.
2. Promote and reward the reuse of services (for both the user and the original designer).
3. Define guidelines about how to cope with reusable services (for example: about the use of multiple instances of a service and how to handle changes to one instance, etc)

Technical guidelines:
1. Limit the complexity of the service design to keep the landscape testable and maintainable.
2. Make services stabile and follow the guidelines about them.
3. Keep services technology independent and minimize the amount of business knowledge in the service

When to use it?
This pattern should be used at the beginning of the project when more than one project is involved as reuse and documentation becomes a bigger problem in a multi-project environment.

Related Patterns
Designing Generic Services: Generic Services are services that are reusable in many ways. So to make good reusable services, one should also follow the tips and tricks in this pattern.

Further Reading
Artus, DJN. SOA Realization: Service design principles. IBM. 17-02-2006
Bloomberg, J. Should All Services Be Reusable? Zapthink. 31-5-2006. http://www.zapthink.com/report.html?id=ZAPFLASH-2006531
Schmelzer, R. Solving the Service Granularity Challenge Zapflash. 9-3-2006. http://www.zapthink.com/report.html?id=ZAPFLASH-200639
Sims, O. Developing the Architectural Framework for SOA – part 2 – Service Granularity and Dependency Management. CBDi Journal 2005 / June: Best Practice Report.

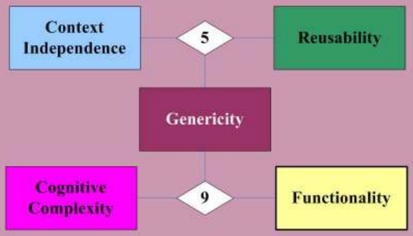
Pattern 3: Designing Generic Services

Context
To be able to maintain the SOA landscape, the total number of services has to be limited. However, to maintain services, the complexity of the service should also be limited. Genericity deals with these two constraints. When the balance tips to one of these two extremes, two forms of generic services arise, namely:
- small generic services that provide little functionality and are reusable
- large generic services that provide much functionality and are mostly not reusable

Problem
How to design generic services that are reusable and provide enough functionality, while both the complexity of the service itself and the complexity of the service construction are maintained?

Forces

5. Context-independent, generic services can be easily reused, provided that services are documented well and the financial structure supports reuse
9. To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts



How it Works?
1. Search for the genericity in processes and make one service for these generic parts, keep the process-specific parts out of this service and build new services for them (i.e. limit the number of parameters).
2. Watch for cognitive complexity when a service has to deal with much and complex functionality as this aspect limits the amount of functionality a service can realize. When a service needs to realize such complex functionality that it is not understandable by the human designer, the service is too large.
3. To achieve reusable generic services, make the services context-independent both on a technical way (platform-independent, limited knowledge about other services etc) and in an organizational way (as little knowledge as possible about the domain).

When to use it?
When dealing with cognitive complex functions on the IS domain and when reusability of services is important.

Related Patterns
Reusable services: generic services have to be reusable, therefore these two patterns are linked strongly.

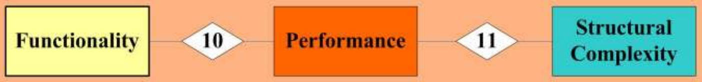
Further Reading
No relevant sources at the moment.

Pattern 4: Performance and SOA

Context
An SOA exists of many small parts and the communication of these parts can get in conflict with performance requirements. Both complexity of the landscape as complexity of the service can have influence on the performance.

Problem
How to make an SOA perform well?

Forces
10. Performance requirements limit the amount of functionality in single services
11. Many service calls between systems in the SOA could cause performance problems



How it Works?
When performance demands are high, two aspects should be kept in mind during the development of the services. The first is the performance of the service itself and the second is the performance of the service landscape. To keep the performance of the service under control, the complexity and the amount of functionality the service provides should be limited. By testing the performance of the service in an early stage, the performance problems can be cleared away.

To keep the service landscape performable, the number of service calls in the landscape should be limited. When legacy is concerned, it is even more important that services should be context-independent and that the service calls perform well.

The two performance requirements are mutually exclusive and a balance in this conflict in goals should be found to deal with these problems. When performance demands are very high and the functionality is more complex, other solutions (in stead of SOA) should be considered.

When to use it?
This pattern should be used when performance demands are high during the development of an SOA.

Related Patterns
Generic Services: Generic services try to solve the dilemma between a complex SOA landscape and complex services, both which can have performance problems.

Further Reading
Linthicum, D. Designing SOA Web Services Services for Performance. 02-08-2005. http://webservices.sys-con.com/read/114127.htm

6 Validation

6.1 Method

Validating qualitative research is different from quantitative research. Quantitative research can be validated by heuristics techniques that have measurable outcomes, while validation in qualitative research is about quality, rigor, and trustworthiness (Golafshani 2003). Three aspects should be considered in validating qualitative research, namely coherence, consensus, and instrumental utility (Eisner 1991).

- Coherence reflects the consistency of the research. Questions regarding this are: How are the conclusions supported? What data sources are used?
- Consensus shows the consistency of the findings, whether the findings are consistent with the experiences of the reviewers of the documents.
- Instrumental utility proves the usefulness of the study

The coherence of our research is out of the scope of this thesis. We asked experts to validate the consensus and instrumental utility of the granularity framework and the patterns. We used structured interviews for this validation, of which the scheme can be found in Appendix E. Our experts are three experienced architects from Capgemini, two of which were already involved in the literature study.

Name	Role	Experience in Architecture	Number of large SO-projects involved in
Reviewer 1	Certified Enterprise Architect	14 years	7
Reviewer 2	Certified Enterprise Architect	10 years	3
Reviewer 3	IT Architect	10 years	1

Table 6.1 - Reviewers

We implemented improvements and corrections to the framework and patterns introduced by the reviewers which improved the readability and comprehensibility of them. The new versions are implemented in the chapters 4 and 5. The validation document for the reviewers can be found in Appendix F.

6.2 Evaluation of Validation Concerns about the Granularity Framework

The granularity framework is evaluated on the accuracy of the hypotheses and the usability of the framework. Hypotheses 2, 6, 8, 10 and 11 are directly accepted by the reviewers, the others are discussed in more detail.

Nr	Hypothesis	Accepted by number of reviewers?	Explanation
1	Designing for outsourcing makes business processes more flexible	1	Accepted by one of the reviewers, while the other two think that outsourcing is made easier as an effect of service oriented thinking. Designing services makes the business process more flexible, and that makes it more feasible for outsourcing.
2	Context-independent services are easier to rearrange functionality, i.e. more flexible	3	

Nr	Hypothesis	Accepted by number of reviewers?	Explanation
3	When business processes are flexible, it is easy to change functionality; however, business transaction consistency should be protected	2	The third reviewer agrees partly with this hypothesis. How flexible the business processes are depends upon the flexibility of the services.
4	Reusing services makes the structure of the service landscape more complex and more difficult to test and maintain	1	Two disagree strongly with this hypothesis because reuse is needed to push back the costs of development and maintenance and the alternative (no reuse) makes it even more complex. One agrees because the interrelatedness between the services is much larger, which causes more dependencies and vulnerabilities
5	Context-independent, generic services can be easily reused, provided that services are documented well and the financial structure supports reuse	3	All three agree, however one does think that the financial structure is not always a prerequisite for reusing services. It can go well without a supporting financial structure
6	When working with legacy, technical context-independence is even more important to keep the landscape testable	3	
7	Composing services makes the business processes more flexible	2	According to the third reviewer, the ability to compose services makes the landscape more maintainable and manageable, but not more flexible
8	New functionality can be created by composing services from legacy systems	3	
9	To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts	3	Accepted, with the remark of two of the three interviewers that generic services should never include specific parts
10	Performance requirements limit the amount of functionality in single services	3	
11	Many service calls between systems in the SOA could cause performance problems	3	

The hypotheses that were not directly accepted by all three the interviewers are mostly a definition problem. That shows how difficult and ambiguous the aspects are and that a correct phrasing is very important. The reviewers helped to adapt the hypotheses to make them clearer. We implemented their remarks in the context of clarity to the hypotheses in the foregoing chapters.

All reviewers expressed the opinion that the framework seemed most helpful to an inexperienced architect. They agreed that such an architect can use the framework to develop an initial understanding of the trade-off decisions the SOA project team would have to deal with in the service design stage. Furthermore, to extend the use of the hypotheses, all of the three reviewers thought that our claims should be made more specific. We agree with them, though, we are aware of a common threat to any pattern design activity. For example, if a pattern is specified in too concrete terms, this may detract from its usability and if the pattern is in too general terms, it will be relevant to any project and will loose its meaning as a kind of a guideline to the service architect.

Finally, the reviewers checked the completeness of the framework. They verified that to the best of their knowledge, there were no missing elements that were deemed essential for a study on granularity.

6.3 Evaluation of Validation Concerns about the Patterns

Our reviewers confirmed that, in their SOA project practice, they consistently observed the four patterns we formulated in Chapter 5. Their remarks and clarifications are given per pattern:

- **Pattern 1:** More clarity should be provided to the paragraph about core and context processes. The first impression of the reviewers about core processes are the essential or organization-critical processes and not the distinguishing ones. An addition to the two guidelines is given by reviewer 1: Not only the process-flow should be separated from the content, but the content should also be divided in a production part and a coordination part. Reviewer 2 deemed that the extent of uncertainty is also important in defining the service granularity because uncertain (parts of) business processes are more likely to change and should therefore be separated from certain (parts of) business processes.
- **Pattern 2:** In the context we state that reuse is the number one aspect to choose SOA and granularity. According to reviewer 3 not reuse, but flexibility should be this aspect. Reviewer 3 would also like to see more about costs, while this is a very important aspect in reuse issues and about governance. Reviewer 1 approves the added value of the organizational guidelines, while he thinks that the technical guidelines are too vague and trivial.
- **Pattern 3:** All reviewers have no comments on this pattern. Reviewer 3 indicates that the use of layers in the services landscape can help to bring more genericity into the services.
- **Pattern 4:** All reviewers indicate that the last paragraph about the two performance problems should be rephrased because now it is not clear what is meant. Reviewer 2 states that this pattern also depends on the underlying hardware.

As can be seen, most of the comments are clarifications or extensions to make the patterns more useful in practice. The clarifications are implemented in the patterns in chapter 5, while the extensions have to be considered in future research in the field.

The reviewers agreed that the patterns help to see what the important aspects in service granularity issues are, but, at the same time, they all agreed that there should be more clarity to the guidelines and tips to make them more suitable for practical use. The implications of the aspects on service granularity have to be made more concrete, to give more meaning to the direct outcomes on granularity.

The patterns can be used at the beginning of an SOA project and in developing the services landscape, in both occasions for obtaining more clarity around service granularity issues. Again, the reviewers checked the completeness of the patterns. Although, they indicated some aspects that need more attention (governance, costs, extent of insecurity), these all fall within the areas of the current patterns.

The added value of this research is that it defines the subject in a clear way and demarcates the areas for future research. According to the reviewers, we indicate interesting relations concerning service granularity which need more detailed exploration. Our research findings can act as a foundation for further research on service granularity.

6.4 Generalizability

This research has omitted a number of elements, which affect the applicability of our approach in comparable settings:

- We acknowledge that we only use customer-build projects in our research. Neither the influence of granularity on services developed for commercial applications is considered, nor what it means to use commercial services in a customer-build SOA.

- We were unable to do method triangulation or researcher triangulation, this means that we were only able to use one resource for the input (interviews with project members) and that these interviews were carried out by the same person.
- The granularity framework and patterns are largely based on the experiences of Capgemini architects or from architects of other companies that worked together with Capgemini employees. Therefore, more case studies are needed to explore the external validity of our results.

Despite these three remarks, our first evaluation of validation concerns of the framework and the patterns gives an early indication that the framework and patterns can be effectively employed in every SOA project to improve the understanding of service granularity decision-making.

Finally, we note that our results can be used as a foundation for future service granularity research. Our approach provides insight in the concept of service granularity and in the trade-offs that are necessary to choose service granularity. We demarcated the subject to make it easier for future research to go into detail about parts of service granularity.

7 Conclusions & Future Work

7.1 Conclusions

In chapter one, we started with three main research questions for this research. Now, we look back at these questions, to see whether we answered them completely. The first research question is:

1. What granularity framework can be identified based on state-of-the art literature sources and experts opinions regarding specific service types?

Our main result from the literature study was the granularity framework. It is based upon the literature sources and expert opinions, it reflects a first glance at service granularity, and it establishes nine hypotheses regarding service granularity. This framework allows us to conclude that granularity is a complex subject, which is interrelated with many aspects. Flexibility and reusability seem to play important roles in deciding about service granularity.

The second research question is:

2. What trade-off decisions are made in the case study projects about the granularity aspects?

Four different case studies about SOA implementations were regarded in our investigation on service granularity decision-making. Our hypotheses from the granularity framework are refined by the analysis of the case studies and the granularity framework is adapted. The case studies identified three main areas in which architects should make trade-off decisions in the context of service granularity. The first area is the creation of business process flexibility. Flexibility is created by identifying logical units of work, while keeping an eye on the transaction consistency and the context-independence of the services. The creation of a maintainable and low-cost landscape reflects the second area. Such a landscape is achieved by designing reusable and generic services, to keep both the costs of development and maintenance low. The last area involves the performance of SOAs and keeps an eye on both the complexity of a service and the complexity of the SOA landscape. Each area needs requirements about how to handle the aspects concerned. For example, processes need to be indicated as core or context and performance requirements need to state the performance of the SOA. Furthermore, not all aspects can be regarded with the same importance. Therefore, the focus should be given to those areas that need priority and this focus can differ for different parts of the SOA landscape.

The third research question is:

3. What service design patterns can be derived from the analysis results of the case studies ?

In the three areas four patterns emerged, which reflect guidelines and tips to meet the goals of each area. The first and the third area are each covered by one pattern, while the second area is divided into two patterns: one about reusable services and one about generic services.

To create flexible services, functionality, transaction consistency, and context-independence play important roles. Logical units of services are identified, which are transaction consistent and as context-independent as possible. Context-independence is needed both on a technical and on an organizational level. Services need to be technology independent and need to include only the necessary business knowledge to complete its goal.

For reusable services it is important to keep both the technical as well as the organizational guidelines in mind. The organization has to be equiped with reusability guidelines (about documentation and financial structure), while on the technical side the weight has to be placed to maintenance, testability, and context-independence to build reusable services.

Generic services are services that provide a generic solution. These services are designed to limit the total number of services in a landscape. Two aspects are important in the design of these services, namely context-independence and cognitive complexity. Context-independent services are easier to reuse, while cognitive complexity needs to be kept in mind by designing services with multifaceted functionality. When the service is too complex to comprehend, the complexity should be limited by designing multiple services. One way to do this, is to eliminate all specific parts from the service and to maintain only the shared functionality in the generic service.

Sometimes, performance and SOA are not compatible. To keep performance under control, both the performance of the service as well as the performance of the service landscape need to be maintained. The performance of the service has to deal with the complexity of the functionality, while the performance of the landscape has to cope with the number of service calls.

The fourth research question is:

4. How valid and useful are the granularity framework and these patterns?

The granularity framework and the patterns are evaluated by three reviewers. The hypotheses and patterns reflected the experiences of the reviewers. They indicated the value of the thesis both as a foundation for future service granularity research and to develop initial understanding of service granularity trade-offs.

7.2 Discussion

In this thesis we discussed the concept of service granularity from an architectural perspective. We learnt why service granularity decision-making is such a complex process: it not only depends on organizational goals, but also on technical possibilities. Other literature sources about service granularity indicate similar aspects as we discuss (Artus 2006; Papazoglou et al. 2006; Schmeizer 2006; Zimmerman et al. 2004), however, these sources discuss granularity not in as much detail as we do.

We provided a multifaceted definition of granularity by (i) proposing a framework for understanding granularity aspects in SOA projects and (ii) deriving four patterns about the trade-offs that architects should anticipate in SOA projects. First, we discuss the results of the case studies, after which we go into detail about the patterns, and, at last, we debate different options for future validation research.

In the case studies, project members sketched a picture of granularity decision-making in their project. However, this picture is limited because it only reflects their own experiences. Because of time constraints and limited availability of resources, it was not possible to talk to more people involved in the same projects. Nevertheless, the pictures which resulted from the different case studies were converging. We chose to examine very different cases to be able to generalize our findings. Looking back, this gave relatively better results for cases Alpha, Beta, and Gamma. Case Delta was found too immature to be able to provide enough additional value to our results. One lesson we have learnt from case Delta is that the granularity of a service can change throughout the SOA project. We observed that incorporating improvements into business processes leads to changed services which have a different granularity than before. Also, one of the reviewers said that the uncertainty about the stability of a business process has to be considered when defining services. When we combine the ability of changing service granularity as a response to process uncertainty, it seems important to distinguish between uncertain (parts of) business processes and stable (parts of) business processes to prevent (or at least minimize) the implementation of unnecessary changes later on in the project.

Based on our results from the case studies, we gave a first indication of the influence of service types on service granularity. This showed that trade-offs around flexibility were mainly considered at the business and IS service level, while trade-offs in the reusability and performance areas were addressed at the IS and software level. This is only a preliminary and high-level conclusion and more research on the service granularity aspects and service types is needed to support this conclusion and to be able to refine it.

The patterns are based on the adapted granularity framework and the findings from the cases and literature. We chose to design the patterns based on the three important areas of the granularity framework. Because we saw that the genericity was a problem in all our four cases, we decided to formulate an additional pattern of it to address the reusability area. In future research, each of the patterns can be broken down into a set of more specific patterns. For example, the flexibility pattern can be subdivided in (i) patterns about choosing the business processes that need flexibility, and (ii) patterns suggesting how to make a business context-independent service. In this way the patterns will be extended with more detailed information and will be more suitable for practical use. Of course, we should take into account the key characteristics of patterns, to be applicable in many situations, and, therefore, the patterns should not be made too specific.

Furthermore, we acknowledge the fact that definitions are very important in SOA research because many aspects have ambiguous meanings and it is essential to know the correct meaning in this fairly unexplored and diverse context. This definition problem made it difficult to implement the opinions of different interviewees because we had to continue asking questions until we were sure the interviewees were on the same page. It was not possible to understand all hypotheses and patterns without some explanation from our side. We, therefore, recommend for future researchers to be very precise in the phrasings about service granularity.

Last but not least, more research is needed into evaluating validation problems and their solutions. In this project, we were only able to do an early evaluation of validation concerns and we justified why more validation research is needed to confirm the strength of our research results. One way to validate the research results is to let both inexperienced and experienced architects use the patterns in a project setting to see whether they make the same trade-offs regarding the service granularity. Other validation research direction can be undertaken by caring out more in-depth case studies that confirm our results. Quantitative research to support our findings can be performed by measuring service granularity in services. However, in this case, a measurement method has to be developed first.

7.3 Future Work

As indicated in the previous paragraph, there is enough to do in the service granularity research area. Three side-tracks of our research are given extra attention and are discussed as future research themes in the paragraphs below.

7.3.1 Design and Diagnosis tool

By providing a deeper understanding about service granularity and by going into detail about how this granularity can be achieved, the current framework and patterns can get much more quality and value. At the moment, the guidelines are still a black box. Giving details about how to follow up the guidelines can be very helpful for architects. With these guidelines, architects can both design and diagnose services on granularity.

The granularity framework can be expanded to function as a diagnosis tool for services. Such a tool can be used to judge the chosen service granularity in the design as well as in audits of SOAs. By doing more research to what the influence of the aspects on granularity are, checklists can be developed with statements which services have to meet to be of good granularity. The current research does give many functional statements, but does not indicate how to work these out in detail. In the follow-up research, these statements have to be examined in more detail.

7.3.2 SOA and ERP

SAP's Enterprise Service Architecture is a business-oriented approach to SOA. ESA is a set of principles and guidelines for creating a flexible IT architecture with the high business value at the lowest possible cost (Woods 2003; Woods et al. 2006). The platform that makes it possible to realize ESA architectures is called NetWeaver. NetWeaver is SAP's open application and integration platform that enables the development, deployment and administration of services (Woods et al. 2006). ESA describes enterprise services which provide value for the enterprise and which exists of web services. Composite applications extend or enhance business functionality by building a new application on top of the existing services.

According to Thomas Mattern, solution marketing manager of SAP, ESA is introduced for flexibility reasons. Flexibility in business processes is obtained by using the process lifecycle (Figure 2.1) in which core and context processes are situated.

SAP has four core values of ESA and those are managing complexity, simplifying development, promoting reuse, and lowering TCO. As we can map those on the granularity aspects then we see that complexity and reuse will probably be core concepts for service granularity in ESA. SAP designed the enterprise services in ESA also in terms of functionality. SAP defined the business drivers and divided the services into four functional categories: component services, process services, entity or engine services, and utility services (SAP AG 2005b). Most services have one business driver and one functional category only. When a service can be categorized into two functional roles, the granularity might be too high, and splitting up should be considered (SAP AG 2005b).

It will be very interesting to look into the differences between enterprise services from ESA and customer-build services, and in the motives behind these services. Knowing these differences tells more about the possibilities of the different systems and about what a right service granularity is.

7.3.3 Measuring Service Granularity

We can express granularity in a number by measuring certain aspects of it. Because of this, terms like fine-grained and coarse-grained will have the same meaning for everybody (Sims 2005). It should be possible to couple certain levels of granularity to some of the aspects of granularity. To measure the granularity of a service, it is essential to know what the motives behind that special service are. Every service can have different underlying motives, and only knowing the general motives behind the service design is not enough.

At the moment, there is no available method to measure granularity. One idea comes by Sims, who says that service granularity can be measured by (Sims 2005):

- Dependency to lower-level services: this is expressed through the number of services that are invoked through a given operation on a service interface
- Functionality: this can be determined by counting the function points for each component
- Data processing: this can be measured by calculating the number of database tables updated, the number of update operations invoked on resource-domain components, or the number of types in a service information model

While functionality says something about the service itself, the other two say more about the relation of the service with other services. In addition to these aspects, the number of times a service is called does for example also say something about the granularity. Therefore, it is not said that this is a good method for measuring granularity and more research should be performed to find a satisfying solution. As defined in paragraph 3.1 the granularity of a service is connected to the functionality and complexity of the interface and implementation, and a measurement method should at least be able to measure these aspects.

The functionality can be measured by function points (Fenton et al. 1997; Garmus et al. 2001), which is a proven method for measuring the functionality of a software component. However, a counting method for services is not yet been developed.

Part of the granularity of services is the dependencies to other services. There are modularity metrics, which measure these dependencies. These are called inter-modular measures (Fenton et al. 1997). This type of measuring is in particular important for the information system services which often are compositions of services. We can measure the structural complexity of a service with the Shepperd complexity method (Fenton et al. 1997). For this we need to have an overview of the application and software services with their dependencies. This method rests on a counting mechanic for the incoming information flows (fan-in) and the outgoing information flows (fan-out). The formula is:

Shepperd complexity (M) = ((fan-in(M) * (fan-out(M))^2

Where M = the module

Combining function point analysis with Shepperd complexity results in a measure for service granularity. This measure can be used to express service granularity in a number and with these numbers, quantitative research can be performed to combine granularity aspects with different amounts of service granularity. Having this, tells us more about what the right level of granularity is for different combinations of aspects.

References

Alexander, C., Ishikawa, S., Silverstein, M., and Jacobson, M."A pattern language: towns, buildings, construction," Oxford University Press, New York, 1977, pp. XLIV, 1171.

Arnold, B., and Op 't Land, M."An Architectural Approach to the Implementation of Shared Service Centers," in: *Dutch National architecture Congress 2002 (LAC2002)*, Zeist, the Netherlands, 2002.

Arnold, B., Op 't Land, M., and Dietz, J."Effects of an Architectural Approach to the Implementation of Shared Service Centers," in: *Second International Workshop on Enterprise, Applications and Services in the Finance Industry (FinanceCom05), to be held in co-location with the 13th European Conference on Information Systems (ECIS 2005)*, Regensburg, Germany, 2005.

Arsanjani, A., Borges, B., and Holley, K."Service-Oriented Architecture," in: *DM Direct Newsletter*, 2004.

Artus, D. "SOA Realization: Service Design Principles," *IBM developerWorks*) 2006.

Basset, P. *Framing Software Reuse - Lessons from the Real World*, (1st ed.) Prentice Hall, New York, 1997.

Brown, A., Delbaere, M., Eeles, P., Johnston, S., and Weaver, R. "Realizing service-oriented solutions with the IBM Rational Software Development Platform," *IBM Systems Journal* (44:4) 2005, pp 727-752.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M."Pattern-Oriented Software Architecture - a System of Patterns," Wiley, Chichester, 1996, pp. XVI, 467.

Cambridge University Press "Cambridge Advanced Learner's Dictionary". Retrieved at 01-02-2006 from http://dictionary.cambridge.org/

Cantara, M."External Service Providers' SOA Frameworks and Offerings: Capgemini," G. Research (ed.), 2005.

Cap Gemini Ernst & Young "White Paper - CGEY Architecture - IAFv3," 2001.

Capgemini "Who We Are". Retrieved at 26-5-2006 from http://www.capgemini.com/about/

Carroll, J., and Swatman, P. "Structured-case: A methodological framework for building theory in information systems research," *European Journal of Information Systems*) 2000.

Channabasavaiah, K., Holley, K., and Tuggle, E. "The Case for Developing a Service Oriented Architecture". Retrieved at 31-10-2005 from www.webservices.org/ws/content/view/full/3777

Colan, M. "Service-Oriented Architecture expands the vision of Web services, Part 1". Retrieved at 21-04-2004 from http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro.html

Curley, S., Downey, K., van 't Wout, J., Klay, H., Metcalfe, C., and Gibbs, J."Services Architecture - Making It Real," Capgemini (ed.), 2003.

Daft, R. *Organization Theory and Design* Thomson's Learning, Ohio, USA, 2001.

Delphi Group."SOA: A Business Architecture for Managing Uncertainty," 2005.

Dunn, B. "A Manager's Guide to Web Services," *EAI Journal* (Januari 2003) 2003.

Eisenhardt, K. "Building Theories from Case Study Research," *Academy of Management Review* (14:4) 1989, pp 532-550.

Eisner, E."The enlightened eye: qualitative inquiry and the enhancement of educational practice," Macmillan, New York, 1991, pp. VIII, 264.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T. *Patterns: Service-Oriented Architecture and Web Services* IBM Redbooks, 2004.

Fatolahi, A., and Shams, F. "Service Discovery Based on Business Processes: A Practical Approach Using UML," International Conference on Services Systems and Services Management, IEEE CNF, Chongqing, China, 2005.

Fenton, N., and Pfleeger, S. *Software Metrics - a Rigorous & Practical Approach*, (Second Edition ed.) PWS Publishing Company, Boston, 1997.

Ferguson, D., Lovering, B., Storey, T., and Shewchuk, J. "Secure, Reliable, Transacted Web Services: Architecture and Composition". Retrieved at 02-02-2006 from http://www.cs.virginia.edu/~humphrey/cs551/Other/SecureReliableTransactedWSAction.pdf

Foody, D. "Getting Web service granularity right". Retrieved at from http://www.soa-zone.com/index.php?/archives/11-Getting-web-service-granularity-right.html#extended

Fowler, M. *Patterns of Enterprise Application Architecture* Pearson Education, Inc, Boston, 2003.

Furey, S. "Why We Should Use Function Points," *IEEE Software* (14:2) 1997, pp 28-30.

Garmus, D., and Heron, D. *Function Point Analysis; Measurement Practices for Successful Software Projects* Addison-Wesley, 2001.

Goedvolk, H."Personal Interview," Certified Enterprise Architect, Capgemini NL, 2006.

Golafshani, N. "Understanding Reliability and Validity in Qualitative Research," *The Qualitative Report* (8:4) 2003, pp 597-607.

Gootzen, H."Personal Interview," IT Architect, Capgemini NL, 2006.

Greefhorst, D., Koning, H., and van Vliet, H. "De Dimensies in Architectuurbeschrijvingen (in Dutch)," *Informatie* (Nov 2003) 2003.

Hedin, M., and Mayo, S. "Worldwide SOA Professional Services Vendor Analysis: 17 Service Firms are Getting Ready to Address a Fast-Growing and Expanding Market Opportunity," IDC, 2005.

Jones, S., and Morris, M. "A Methodology for Service Architectures," Capgemini, 2005.

Klein, H.K., and Myers, M.D. "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems," *MIS Quarterly* (23:1) 1999.

Knoll, K., and Jarvenpaa, S. "Information Technology Alignment or Fit in Highly Turbulent Environments: The Concept of Flexibility," Proceedings of the 1994 Computer Personnel Research Conference on Reinventing, Alexandria, Virginia, United States, 1994, pp. 1-14.

Krafzig, D., Banke, K., and Slama, D. *Enterprise SOA - Service Oriented Architecture Best Practices* Prentice Hall, New Jersey, 2005.

Lee, J., Siau, K., and Hong, S. "Enterprise Integration with ERP and EAI," *Communications of the ACM* (46:2), February 2003 2003, pp 54-60.

Massuthe, P., Reisig, W., and Schmidt, K."An Operating Guideline Approach to the SOA," in: *Informatik-Berichte 191*, Humboldt University, Berlin, 2005.

McGovern, J., Tyagi, S., Stevens, M., and Mathew, S. *Java Web Services Architecture* Morgan Kaufmann Publishers, 2003.

Nelson, K., Nelson, H., and Ghods, M."Technology Flexibility: Conceptualization, Validation, and Measurement," in: *Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences*, Computer Society Press, Maui, Hawaii, 1997.

Op 't Land, M."Personal Interview," Certified Enterprise Architect, Capgemini NL, 2006.

Papazoglou, M."Service-Oriented Computing: Concepts, Characteristics and Directions," in: *Fourth International Conference on Web Information Systems Engineering*, IEEE, Roma, Italy, 2003.

Papazoglou, M., and Van den Heuvel, W. "Service Oriented Architectures: Approaches, Technologies and Research Issues," *VLDB Journal* ((To appear)) 2005.

Papazoglou, M., and Van den Heuvel, W. "Service-Oriented Design and Development Methodology," *International Journal of Web Engineering and Technology* ((To appear)), To appear in 2006.

Parnas, D. "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM* (15:12) 1972, pp 1053-1058.

Perrey, R., and Lycett, M. "Service-Oriented Architecture," Symposium on Applications and the Internet Workshops, IEEE Computer Society 2003, Orlando, FL, USA, 2003.

Poulin, J., Caruso, J., and Hancock, D. "The Business Case for Software Reuse," *IBM Systems Journal* (32:4) 1993.

SAP AG."Business is not a One-Way Street," in: *Presentation slide*, 2005a.

SAP AG."Enterprise Services Design Guide," 2005b.

SAP AG "ESA Preview System". Retrieved at from https://www.sdn.sap.com/irj/sdn/developerareas/esa/esapreview

SAP Group "Enterprise Services Architecture - a Glossary," 2004.

Schmeizer "Solving the Granularity Challenge". Retrieved at 13-3-2006 from http://www.zapthink.com/report.html?id=ZAPFLASH-200639

Sims, O. "Developing the Architectural Framework for SOA - Part 2 - Service Granularity and Dependency Management," *CBDi Journal* (June 2005) 2005.

Slot, R."Personal Interview," Principal Consultant and Enterprise Architect, Capgemini NL, 2006.

Stojanovic, Z., Dahanayake, A., and Sol, H. "Modeling and Design of Service-Oriented Architecture," IEEE International Conference on Systems, Man and Cybernetics: Impacts of Emergence Cybernetics and Human-Machine Systems, Omnipress, Madison, Wisconsin, USA, 2004, pp. 4147-4151.

The Computer Language Company Inc. "Granularity". Retrieved at 11-12-2005 from http://computing-dictionary.thefreedictionary.com/granularity

The Open Group "Other Architectures and Architectural Frameworks". Retrieved at 6-5-2006 from http://www.opengroup.org/architecture/togaf7-doc/arch/p4/others/others.htm

The Open Group "TOGAF "Enterprise Edition" Version 8.1". Retrieved at 10-02-06 from http://www.opengroup.org/architecture/togaf8-doc/arch/

The Zachman Institute for Framework Advancement "Zachman Framework". Retrieved at 10-02-2006 from http://www.zifa.com/

Van 't Wout, J., and Curley, S."Web Services and Services Architecture - CGEY Architecture & Technology Position Paper," Cap Gemini Ernst & Young, 2003.

Van Eck, P., Blanken, H., Fokkinga, M., Grefen, P., and Wieringa, R."A Conceptual Framework for Architecture Alignment Guidelines," in: *Project GRAAL WP1 Whitepaper*, Enschede, the Netherlands, 2002.

Woods, D. *Enterprise Service Architecture* O'Reilly and associates, Inc, Sebastopol CA (USA), 2003.

Woods, D., and Mattern, T. *Enterprise SOA - Designing IT for Business Innovation* O'Reilly Media, Inc, Sebastopol, CA (USA), 2006.

Yin, R., and Campbell, D. *Case Study Research: Design and Methods* Sage Publications, Inc., Thousand Oaks, California (USA), 2003.

Zimmerman, O., Krogdahl, P., and Gee, C. "Elements of Service-Oriented Analysis and Design - An Interdisciplinary Modeling Approach for SOA projects". Retrieved at from http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/index.html#author

Appendix A: Terminology

Definitions	**Explanation**
Business Service	A service that provides business value on business level
Cognitive Complexity	The effort required to understand the software
Composability	Composability is the process of combining services to create a more powerful service
Context-Independence	A service does not need to have knowledge about its surroundings, also called loosely-coupledness
Efficiency Complexity	The performance of a service oriented landscape
Flexibility in Business Processes	The possibilities to change business processes quickly
Flexibility in using services	Reusability of services
Functionality	Captures an intuitive notion of the amount of function contained in a delivered product or in a description of how the product is supposed to be
Genericity	Making a service able to use in many different ways
Granularity	The degree of modularity of services or the scope of functionality exposed by a service
Information System Service	These services are pieces of automated business processes and are the services that have the control over the software services
Maintainability	The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment
Performance	How quick a software product should respond on a command
Reusability of Legacy	Provide functionality of legacy systems in services or create new functionality by combining functionality of different legacy systems
Reusability of Services	Using services again for other purposes, can also be seen as flexibility in using services
Service	A service is a facility that has to adhere to seven service principles: value, identity, function, trust, shared, interface, and quality.
Service Oriented Architecture	SOA is a way of reorganizing a portfolio of previously siloed software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols {papazoglou, 2003}
Software Service	A service that fulfils tasks on software infrastructure level
Sourcing	The decision-making process about having part of its work done by other companies (outsource) or keeping it inside (insource)
Structural Complexity	The complexity of the structure of the software
Testability	The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
Transaction consistency	To leave a consistent state in the application after executing a service

Appendix B: Interview Scheme A

Cases: Alpha, Beta, Gamma, Delta

Introduction interviewees

1. Background (Work area, experience)
2. Introduction organization and project(s)
    a. Level of service orientation of projects

Service Oriented Design

3. What does the design process looks like?
4. What was the role of the interviewees in this process?
5. How are the services designed?
    a. What service types are used and why?
    b. On which base are the services designed?
    c. Where there specific thoughts about granularity and on which base is the granularity chosen?
6. Results?
    a. Is the service granularity well-chosen?
    b. Looking back, would the granularity of the services been chosen differently?

Explanation Granularity Research

7. Explanation granularity framework and aspects

Connection between granularity research and experiences interviewees

8. What aspects for granularity decision-making are aware or unaware taken care of in the project?
    a. Are there other aspects that are taken care of?
    b. What were the most important aspects?
9. Do you want to cooperate with the validation of this research?

Appendix C: Coding and Analysis Cases

Every aspect of service granularity is a code and in Table B.1 we state the findings of the cases about these aspects. The three aspects that represent reuse are taken together, because these relate closely to each other. In analyzing the cases, we examine the aspects that relate to the goals of the case and combine these aspects.

Example:

Case Alpha completes two aspects about business services, namely functionality and flexibility, so these are the main aspects to consider in deciding about granularity on business service level. And one of the goals of Alpha is the separation of logistics (process flow) and content (dossiers) to create flexibility. This leads to the following statement:

Business service granularity is analyzed in this project. Granularity on business level is given attention by looking at the concepts functionality and flexibility. Business services are logical units of work which fall within one domain. Flexibility is created by separating the logistics (process flow) from the content so that it is easier to adapt the business processes.

Conclusions about the analysis are a combination of the main aspects of the case and the goals for service orientation. We look at how the main aspects influences the granularity in this case and how this contributes to the goals of the project. For Alpha this lead to the following concluding thoughts:

In conclusion, we can say that reusability and flexibility were the two main drivers for the business architecture. Emphasizing the generic parts of the architecture was a main goal, while flexibility in the business process was created by separating logistics and content. These were also the two main goals of the design. The main aspects in the software architecture were reuse and performance. Reuse was an aspect in the sense of minimizing the number of services that were used. The performance demands of this project were very high, which caused that things sometimes were solved purely to satisfy these goals.

Aspects	**Alpha**		**Beta**		**Gamma**		**Delta**	
	Business Services	**IT services**	**Business Services**	**IT services**	**Business Services**	**IT services**	**Business Services**	**IT services**
Functionality	Logical units of work		Elementary task steps		Represent a functional unit of work; May not go beyond borders of activities		Besides functionality does business ownership also play a role in dividing the services	
Flexibility	Separating Logistics and Content		Flexibility created by using a multi-channel approach		flexibility in product types and by using a multichannel approach		flexibility is created by the multichannel approach and the division of process steps	
Reuse (Reuse of Legacy, Composabilit y)		reusability in parts that are still under development and on process level		reusability of legacy in new system		Services have to be reusable, but the financial model and the documentatio n about the services do not work with this		Services have to be reusable

Aspects	**Alpha**		**Beta**		**Gamma**		**Delta**	
	Business Services	**IT services**	**Business Services**	**IT services**	**Business Services**	**IT services**	**Business Services**	**IT services**
Complexity	Cognitive complexity was high because of complicated domain knowledge	limit number of services by making them more generic; cognitive complexity was high because of complicated domain knowledge	Cognitive complexity was too high	too complicated forms of business products; structural complexity was also too high, because of too much connections between legacy and new system		Human Standard for groups of services		
Context-independence		Stateless services		Too dependent of context, too much mutual dependencies	services need to fall within one business domain and need to have little business context			
Performance & Capacity		Demand of 2s response time		Performance was much too low because of the many dependencies with the legacy system		Dependent on business wishes; can be at odds with the SOA concept		
Genericity		limit number of services by making them more generic		services were too generic		Services have to be generic		
Sourcing								

Table B.1 - Service Granularity Aspects of Cases Compared

Appendix D: Case descriptions

Case Alpha

Project Description

The first case is a system replacement project for a national government organization which does the coordination over different divisions spread throughout the Netherlands. This organization, we call it Alpha, is introduced to gear the business processes of these different divisions to each other. First, Alpha tried to do this with an organizational change top-down through the business, but that did not work. Therefore, they now wanted to enforce an organizational change by introducing a services architecture into the organization.

Four architects that were working on this project gave the input for this case. These were two business architects and two software architects of Capgemini NL. The two business architects had both ten years experience in architecture projects. One of them developed the information architecture for this project and the other made sure this architecture was secured through the project. The two software architects both work in the Accelerated Delivery Center of Capgemini NL. This is an innovative and inspiring development facility which makes it possible to accelerate the projects. Both worked in the past three years for this project.

This project had to integrate 40 existing systems into one new system, while not changing the business processes of the different divisions. This is contradicting because all the systems had their own processes and these have to become one process. However, the differentiations to the current processes should be limited. At first, an information architecture, which had to structure and optimize the primary process, was designed. There were two basic principles for the design of the information architecture, namely:

- Emphasize the generic parts of the architecture
- Separation of logistics (process flow) and content (dossiers)

At the moment logistics and contents were closely knit, which made changes in the business processes very difficult. The objective of the project was to replace the current systems without optimizing them. The main principle for choosing SOA was that the organization wanted to be technology independent. Besides that they wanted to be able to integrate future systems easily into the existing architecture. For the future projects it is a goal to automate more, which changes processes at the customer. They expect to have advantages of the SOA approach in those projects.

The information architecture gives a higher granularity of services than the implementation project, because during implementation you always have to go deeper into details. The components defined in the information architecture are used in the implementation architecture; these components exist of groups of services. The business services were sometimes changed because of new insights. The business services are real business services which do not say anything about automation.

The objectives prescribe the level of granularity chosen. For example, for the design of a system another, more fine-grained, granularity level is chosen as for making an information plan. And when services use other systems, they are more coarse-grained then self-processing services. This system is separated in a front-end and back-end. The back-end is generic and the same for all the divisions, while there are target group specific front-ends.

In the information architecture a certain level of granularity was chosen and sometimes this sat in the way of the implementation architecture. Therefore, it is very important to think about service granularity in advance and define the right boundaries. The information architecture (with the goals reusability and flexibility) had much influence on the division of business services. Only when performance gave problems, smaller services were made.

Findings about the Granularity Aspects

The business services were comparable with units of work, i.e. what the user will see as a logical task. Functionality was the main criterion in this. To complete a business service, several process services had to be implemented. These process services were defined on the basis of sea-level use cases[1]. Combined with the separation of logistics and contents did this create the flexibility required for the solution, according to the architects.

This project was part of a bigger project which encloses the replacement of the whole application landscape, and this ultimate goal was kept at the back of the mind in designing this part. This resulted in designing services also for reuse in the future parts and future developments (like an electronic file system) impacted the size of services. These services were already designed to make it possible to incorporate these future wishes. Besides this reuse of services on business and IS level, there are many software services that are being reused.

It was important to limit the number of services by making them more generic to keep a maintainable SOA landscape, also with the eye on the future extensions. This brought more complexity to the services themselves, by adding more functionality (this is represented by parameters in the services). The most generic service had seven dimensions, in which each dimension had its own parameters, and this worked well. The architects did choose the information system services on base of the business services and this led to larger services in this project (for example: ‘change person’ in stead of ‘process change of address’). The changes that are done in such a service are accumulated and processed in one service call, which reduces the number of service calls, but led to extra efforts per service call.

There was a very strong performance requirement caused by a synchronous call of the services, which stated that the maximum processing time should be 2 seconds. Because of this demand, it was sometimes not possible to build a service to do problems which were fixed by defining smaller services. These services did not fulfil the wishes of the organisation to make reusable services that represent logical units of work.

Every service is completely stateless, besides a version number that accompanies a service to prevent overwriting. Services were kept context-independent to make them easier to reuse. The cognitive complexity of this project was particular high because the required domain knowledge was complicated. Therefore, it was difficult to design the project in such a way that the programmers did not need to have this domain knowledge. At the other side, paper was the reality in this project. What in the paper files stood, was true and not the data in the system.

The architects learned from this project that an SOA only has to be applied where it is really useful. In this project were many little CRUD services[2] to fulfil the performance demand. These are actually operations and have a too small granularity to be called services.

[1] These are user goals, which represent an elementary task of a business user (Cockburn, 2001)

[2] Services that perform one of the following commands: create, read, update, delete

Case Beta

Project Description

The Beta project included the construction of a system for business products, from sales to administration. This case is based on the experience of a project manager of Capgemini NL in a project at a large financial organisation. He was closely involved in the architecture process of this project and he answered the questions from the architectural point of view.

This system had to work together with a legacy administration and management system in which all the products were kept up. The architecture existed globally of a workflow in which the process flow was rebuilt. Services supported this workflow and had connections with the legacy system. The project had two goals: from business perspective, the project objectives were to support workflow and implement a multichannel approach in the front-end. From IT standpoint, the project goal was to build a system that is easy to maintain and to extend.

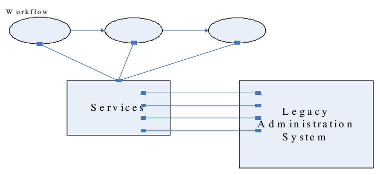


Figure C. 1 - Architecture Beta

There were a large number of products, approximately 60, which differed mainly in acceptance criteria. These acceptance criteria were represented in a matrix. All these types of products existed in the legacy system and during the workflow they were put to a test to see whether the product was allowed (satisfied all the requirements attached to that product). This caused that many services made calls to the administration system, which made the interfaces with this system very complex. The services were made as generic as possible, with the highest level of reuse. There were three types of services: object services, calculation services, and external services.

Findings about the Granularity Aspects

The system was designed with a multi-channel approach in the front-end; multiple communication channels could be used to reach this system. This created flexibility in the system and in the functionality. Services were elementary task steps in the process. Transactions had to be completed, otherwise a rollback had to take place.

One of the main drivers of this project was reusability. The design team did not want redundant information and reused all the functionality that existed in the legacy system in the new system. Besides that, the team wanted a good modular design, which was maintainable and extendable. According to the project manager, the reusable parts, even as the generic parts, were already extracted in the design.

All the access requirements of the product types were represented in a large matrix. Every product type was tested on the basis of this matrix and when it passed the test, it should be admitted at the end of the process to the legacy administration system. This matrix was very complex and included errors, which created the problem that products which were approved in the beginning of the process were not approved in the administration system. According to the manager, this was one of the reasons the project failed; the matrix was too complex to represent that many data and not even the subject matter experts did fully comprehend the structure.

During the testing phase this problem and other problems came to light, which caused, according to the project manager, the downfall of this project. The performance of the system was far too slow because of many mutual dependencies between the two systems. Besides that, the services were too dependent of their context, which caused many problems in debugging.

In the end, it is solved by rebuilding parts of the administration system and by not testing the products to the administration system. The lesson that can be learned from this is that there has to be a balance between genericity and specificity in services. Generic services can be very complex in design and development. Therefore, it is important to assess the design in complexity.

Case Gamma

Project description

The Gamma case is a combination of service-oriented projects in the back office of a large financial organisation. It is based upon the experience of two architects of this organization. They have done several projects in a service oriented way. One of the architects works 10 years for this organization and already 25 years in the IT, the other architect works 7 years for this organization. Both are enterprise architect at the moment, which means that they have a coordinating task and are responsible for the issues that rise on top of and between the projects. The organisation exists of different domains like savings and investments. All these domains have their own projects in which mainly domain architects partake.

Their architecture is globally given in **Error! Reference source not found.**. They use a multichannel concept which permeates all layers: only the presentation layer should be channel dependent, all other layers are channel independent. Below this layer are the applications, which guide all user interactions and, in the ideal situation, must be easy to change and should exist mostly out of software services combined with a little extra logic. Applications actually should only do the orchestration part for the software services and should include as little business logic as possible. An application is more specific, while services are more generic. Unfortunately, because of a financial model that is based upon speed and costs per project, not all projects are service oriented, which causes that applications become too large. Services that already exist need to be reused, but the development of services is because of financial reasons always a struggle.

The reasons to choose for services are to reduce and structure complexity of the IT architecture and develop reusable services. Because services are more expensive to develop than applications, the services need to be reusable. Other characteristics are that the service has to be significant for the business and has to fall within one domain. The heart of a service is a logical data model which gives the “semantics” of the service. An interface offers the logic to the outside world and all these things are described in the service contract, e.g. the available communication protocols. Furthermore, a service can have process and rule models, however, these do not exist very often.

There are three types of services, namely:

- Data services which are used for example for data about customers
- Transactional services, which support payments and suchlike
- IS services, which support generic parts of the business process
- Services are stateless. Services themselves have to be stable, while applications are the changing parts.

Granularity Aspects: Findings

While developing a new architecture for a project, they start with a business process domain architecture, which describes the business processes with a certain stratification. From this the application domain architecture is made, in which the applications and services are designed. A business service can support more than one activity; however it should respect the borders of an activity. Similarly, there are certain decoupling points which set straight borders which the business services may not cross.

For developing services and deciding about its granularity, certain guidelines are given in this organization, like a service has to fall within one domain and needs to be reusable. Other guidelines are the ACID properties[3], in case of transactional services, and looking at the information and data model of the service. The functionality should be in the services and not in the above lying applications. Too get a profound understanding of the business it is important to get round the table with business architects.

Services are small and sometimes even too small. There is a tendency towards making big applications for reasons of budget. Therefore, service orientation is best realized by new projects for larger systems (and hence enough budget). Apart from this, it is hard to develop services that are reusable for other projects, which lowers the enthusiasm for designing services.

Aspects that play a prominent role in deciding about granularity at this project are functionality, reusability and business context-independence. Functionality means that services need to fulfil a functional unit of work, while business context-independence has to do with the domains services fall in. Services need to have as little business context as possible, to keep them suitable for reuse. Flexibility in product types is very important for this organization. Reduce time-to-market for new products is an important argument. This can be reached with flexibility in business processes.

Cognitive complexity is an aspect they take into account for components (collections of business services). This is called the “human measure”, which represents the amount of information or complexity that can be processed by a human being.

Every service has a service contract and in this contract are some constraints secured. The business has certain demands about, for example, performance and these are captured in requirements in this contract. The same counts for capacity, testability, maintainability, and transaction consistency. During the testing phase, these contracts should be held.

Looking at the effects of the aspects on granularity, the architects saw that the more often a service was reused, the more generic it was. The more a service is reused, the more stable it is, and the harder to change.

Not the services themselves are flexible, but the compositions of the services. A service has to hide complexity as much as possible.

Case Delta

Project Description

This case is the development of a service oriented project at a utilities company. The project focuses on two parts of the organisation: HRM processes and Identity and Access Management. The input for this case comes from the business development manager of a company which creates integration solutions based on SAP ESA/Netweaver. He has a background in web and intranet development and works since 2001 with SAP Netweaver and ESA. This project for the utilities company is still in its infancy.

[3] ACID = atomicity, consistency, isolation, and durability; to leave a consistent state in the database (Alonso et al., 2004)

The goals of this project are to improve processes and reduce IT costs, to get easy access to the systems and get a better cooperation between the systems. The services for this project are still in development. At the base of service design is the process description. Each process step (a task executed by one user) is described in a written use case which both models the human interaction and makes the semantic object model. From these use cases the first level services are made, which are business services and these are of the largest granularity. Application interfaces are the smallest services and are of the lowest granularity. Between these two there can exist several layers with services, but this should be kept as limited as possible, otherwise it is not maintainable.

Granularity Aspects: Findings

The real choices for granularity are made when the systems are designed and not in the earlier plans. In the plans a base for the granularity is set, but this is further defined when really building the systems. Granularity of business services is decided by changes in the state of the process. Every state change marks a new business service and such a service exists of a task executed by one actor. This sets the borders for business services.

Services are only made when they are reusable. Services that are not reusable do not have the right to exist because of economical considerations. Reusability can exist on process level and over the different channels. Reusability is together with genericity often an argument for choosing granularity.

Flexibility is created by the way of working; the multichannel approach and the division of the processes in steps executed by one user. The main reason for now was to consolidate the processes, so the processes are not made extremely flexible, but the way of working of an SOA already gives much flexibility. Sometimes it is financially better to implement a less perfect solution on the short term because the perfect solution for the long term is not yet realistic.

Complex services are not yet implemented by this project, even as composite services. Sometimes it is better not to choose for composability, for example when it is about gathering data from different applications because it is easier to make a redundant database, then doing this with a composite service.

The management of services also have influence on the granularity. There should be one owner of the service; otherwise no one takes responsibility over it. This makes it sometimes hard to realise services, because for example a service asset management which contains all the information about a car, laptop etc of the employee can have different owners (ICT department, lease department).

Reuse has the influence that system services become larger and business services smaller. On a system level consolidation is most important to make generic services, while on business process level parts of business processes are reused, which makes the service smaller.

Two major things to take into account in building an SOA are the economical consequences. You do not have to make things prettier than they are (for example more flexible). With services you are building some kind of customer-build systems and sometimes there are better alternatives for that. The other major thing is that you are very dependent of how the current systems are implemented. This can lay many boundaries to the SOA.

Appendix E: Validation Interview Scheme

Date	
Reviewer	
Role	
Experience	

Granularity Framework

1. Do the hypotheses given below reflect the experiences of the reviewer?

Nr	Hypothesis	Accept	Explanation
1	Designing for outsourcing makes business processes more flexible		
2	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible		
3	When business processes are flexible, it is easy to change functionality; however, transaction consistency should be protected		
4	Reusing services makes the structure of the landscape more complex and more difficult to test and maintain		
5	Context-independent, generic services can be easily reused in different places, provided that services are documented well and the financial structure supports reuse		
6	When working with legacy, technical context-independence is even more important to keep the landscape testable		
7	Composing services makes the business processes more flexible		
8	New functionality can be created by composing services from legacy systems		
9	To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts		
10	Performance requirements limit functionality options		
11	Many service calls between systems in the SOA cause performance problems		

2. Can this framework help you to define the granularity of services?

3. In your opinion, what purposes this framework can be used for?

Granularity Patterns

1. Do the patterns reflect the experiences of the reviewer?

Pattern	Accept	Explanation
Pattern 1		
Pattern 2		
Pattern 3		
Pattern 4		

2. Do these patterns help you to define the granularity of services?

Pattern	Accept	Explanation
Pattern 1		
Pattern 2		
Pattern 3		
Pattern 4		

3. In your opinion, what are the purposes these patterns can be used for?

What is the added value of this research?

Appendix F: Validation Framework and Patterns

Granularity Framework

Nr	Aspect	Relation with Aspect	Hypothesis
1	Flexibility in BP	Sourcing	Designing for outsourcing makes business processes more flexible
2	Flexibility in BP	Context-Independence; Functionality	Services which are independent of their context are easier to rearrange functionality, i.e. more flexible
3	Flexibility in BP	Functionality; Transaction Consistency	When business processes are flexible, it is easy to change functionality; however, transaction consistency should be protected
4	Reusability	Structural Complexity; Maintainability; Testabiltiy	Reusing services makes the structure of the landscape more complex and more difficult to test and maintain
5	Reusability	Context-independence; Genericity	Context-independent, generic services can be easily reused in different places, provided that services are documented well and the financial structure supports reuse
6	Reusability of Legacy	Context-independence; Testability	When working with legacy, technical context-independence is even more important to keep the landscape testable
7	Composability	Flexibility in Business Process	Composing services makes the business processes more flexible
8	Composability	Reusability of Legacy; Functionality	New functionality can be created by composing services from legacy systems
9	Genericity	Cognitive Complexity; Functionality	To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts
10	Performance	Functionality	Performance requirements limit functionality options
11	Performance	Structural Complexity	Many service calls between systems in the SOA cause performance problems

Table F.1 - Hypotheses Adapted Granularity Framework

These adaptations are also reflected back in the granularity framework (Figure F.) and this leads to three related blocks of aspects. The first block has its influence on the business and information systems services and handles hypotheses 1, 2, and 3. Two and three are about the flexibility of the business process and everything that enters into that. Functionality, flexibility, context-independence, and transaction consistency are the most important aspects to think of, when designing flexible services. One is about the influence of granularity about outsourcing or insourcing of processes. However, none of our cases handled these matters, and therefore, we cannot issue statements about this hypothesis.

The second block handles the reusability and genericity of services and has influence on the IS services and software services. This block is identified by hypotheses 4 to 9. Five, seven, and nine are supported by all the applicable cases, while eight is not completely supported by case Delta. While it is possible to create new functionality by composing services, it is not always the most desirable solution. Due to financial reasons, it is sometimes better to develop a cheaper solution, which gives the same end result, but is a bit less perfect. Hypothesis four is supported by Beta and Gamma, while case Alpha did not show this finding, but also did not undermine it. It seems that maintainability and testability become more important when projects are larger and legacy is concerned. Hypothesis six is about legacy and concerns therefore only Beta and Gamma

The third block sees to the performance and complexity of IS services and software services. It deals with hypothesis 10 and 11. Both hypothesis are supported by the cases. Performance was a problem in three of the four cases, but in two different ways. Single services can be too complex that they take too much processing time or many service calls through the system can give performance problems. Therefore, it is important to consider structural and efficiency to get a maintainable SOA landscape.

In conclusion, we can say that most granularity problems occur on one of the three areas mentioned. For these three areas, we develop four patterns to support the granularity decision-making process. These patterns are presented in the next chapter.

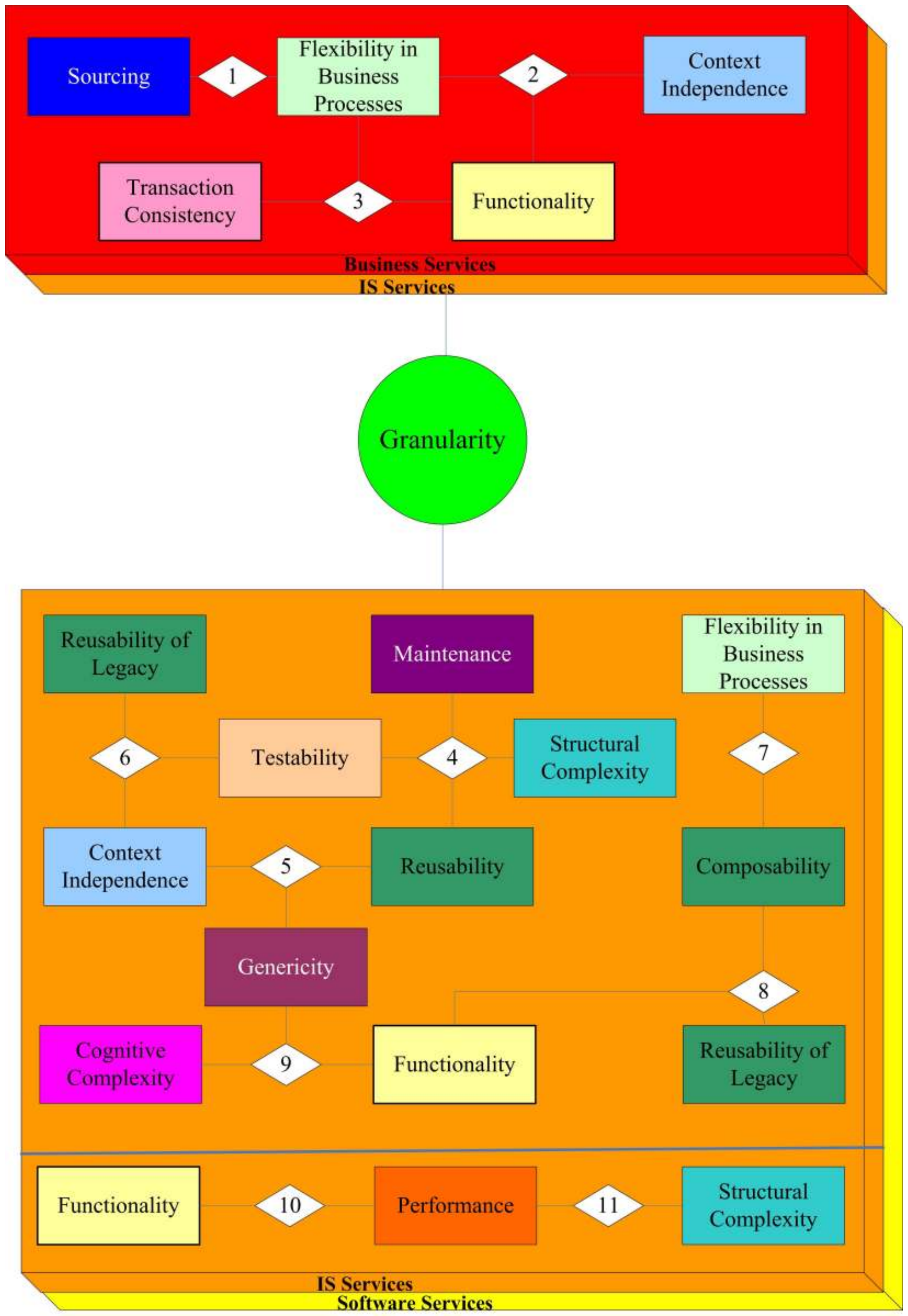


Figure F.2 - Adapted Granularity Framework

Pre-Patterns for Service Granularity

We identified four pre-patterns which are all related to service granularity. Because service granularity is not only a technical aspect, but has also links to the organizational side of projects, the patterns we formulate are not only technological, but also organizational patterns . To present our patterns, we combine two methods of describing patterns (Buschmann et al. 1996; Fowler 2003):

- *Name:* a pattern name is meant to provide a common vocabulary (Buschmann et al. 1996; Fowler 2003)
- *Context:* short description of the background of the pattern (Buschmann et al. 1996)
- *Problem*: one sentence in which the most important problem the pattern solves is explained
- *Forces:* the forces that work on the problem; in this research: the granularity aspects
- *How it Works:* a description of the solution (Fowler 2003)
- *When to use it*: a description of the circumstances in which to use this pattern (Fowler 2003)
- *Related Patterns*: the relation with the other patterns explained (Buschmann et al. 1996)
- *Further Reading*: more information about the subjects of the pattern ((Fowler 2003)

As indicated earlier, the patterns we identify are pre-patterns which means that they are not yet in use, so the elements normally seen in patterns, like ‘Examples’ and ‘Known Uses’ will be left out of this research.

In Figure 5.2 patterns are mapped upon the granularity framework. The four patterns cover each their own part of the framework, however they have overlapping aspects.

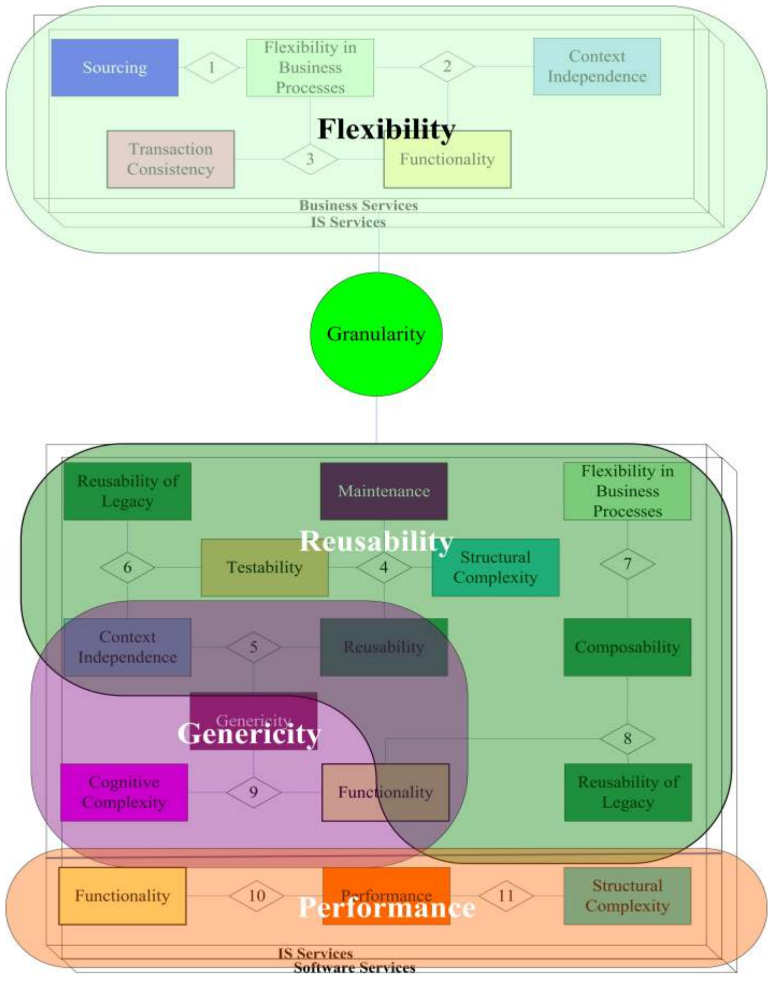


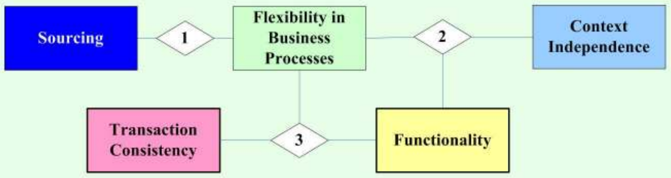
Figure F.3 - Patterns situated on Granularity Framework

Pattern 1: Design services for flexibility

Context
Organizations need flexible business processes which can be adapted to the current wishes of the market. SOA meets this requirement by dividing the adopting organisation into flexible services. Creating flexibility starts with the definition of the business services and has an effect on all underlying service layers.

Problem
How to choose service granularity to make both the architecture and the business as flexible as they want to be?

Forces
1. Designing for outsourcing makes business processes more flexible
2. Services which are independent of their context are easier to rearrange functionality, i.e. more flexible
3. When business processes are flexible, it is easy to change functionality; however, transaction consistency should be protected



How it Works?
Before designing services, thoughts should go to the amount of flexibility required in the business processes. Not every business process needs the same amount of flexibility and because creating smaller services costs more money, it is important to determine the amount of flexibility needed. To decide upon which processes need flexibility and which processes not, a division can be made between core processes and context processes. Core processes are the organization-unique processes and these have to be made more fine-grained than the remaining context processes.

Logical units of work set the borders of business services. Services have to operate within these borders. Guidelines to use when defining these logical units concern the aspects transaction consistency, business ownership, and context-independence. A service needs to leave a consistent state in the system, so a service needs to include a complete transaction. To make sure service ownership is taken care of, services should have one identifiable owner.

Services have to be independent of technology and should include as little business knowledge as possible. Business knowledge should be defined clearly and only the business knowledge that the service needs to fulfil its task should be in the service. All the knowledge that is unnecessary to complete its goal, should be left out. The necessary knowledge should be documented.

Two guidelines to create flexibility in the business process are to use a multi-channel approach and to separate process flow from content. These methods will make the business process more flexible.

When designing for in or outsourcing, flexibility needs special attention. Further research should bring more clarity in the relationship between sourcing and service granularity.

When to use it?
The amount of flexibility required in the business process should be determined during the design of the business services.. This flexibility rate is then used throughout the next steps in the process when deciding about the granularity of the services.

Related Patterns
Designing Generic Services: By making more generic solutions, it will be easier to change business processes because you do not need to change these generic parts. Therefore, this pattern is related.

Further Reading
About Core/Context:
Khan, S. Weblog with explanation about SAP Core & Context processes. https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/2087

About Outsourcing:
Arnold, B., Op 't Land, M., and Dietz, J."Effects of an Architectural Approach to the Implementation of Shared Service Centers," in: *Second International Workshop on Enterprise, Applications and Services in the Finance Industry (FinanceCom05), to be held in co-location with the 13th European Conference on Information Systems (ECIS 2005)*, Regensburg, Germany, 2005.

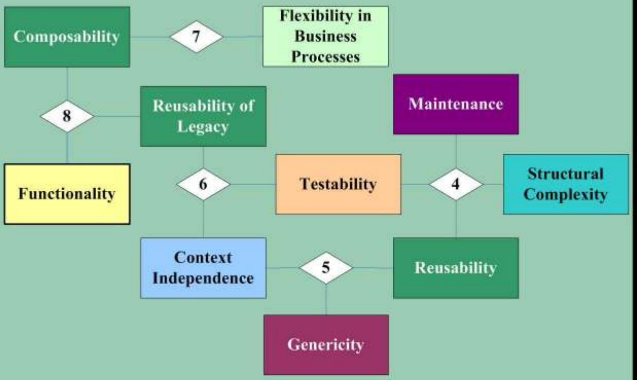
Pattern 2: Design Reusable Services

Context
Reuse is seen as number one aspect to choose SOA and service granularity on. Although reusable services are thought to provide the advantages SOA promises (e.g. lower costs, a faster time to market and simplified integration), SOA project realities indicate that services made for reuse are not reused in new projects. This is due to two reasons: the reusable service cannot be found and the functionality provided in this service is only a partial match with the new service.

Problem
How to design reusable services and organize the business in a way that these reusable services really are reused?

Forces

4. Reusing services makes the structure of the landscape more complex and more difficult to test and maintain
5. Context-independent, generic services can be easily reused in different places, provided that services are documented well and the financial structure supports reuse
6. When working with legacy, technical context-independence is even more important to keep the landscape testable
7. Composing services makes the business processes more flexible
8. New functionality can be created by composing services from legacy systems



How it Works?
Reusability has two special forms, namely composability and reusability of legacy applications. Composability is the creation of (new) functionality by combining multiple services. Reusability of legacy applications needs to get even more attention, because when legacy is not can make the landscape more complex.
To keep services reusable some guidelines are identified to fulfil the needs of reusable services:

Organizational guidelines:
1. Provide better documentation upon existing services, by giving more insights into the business and IS architecture.
2. Promote and reward the reuse of services (for both the user and the original designer).
3. Define guidelines about how to cope with reusable services (for example: about the use of multiple instances of a service and how to handle changes to one instance, etc)

Technical guidelines:
1. Limit the complexity of the service design to keep the landscape testable and maintainable.
2. Make services stabile and follow the guidelines about them.
3. Keep services technology independent and minimize the amount of business knowledge in the service

When to use it?
This pattern should be used at the beginning of the project when:
1. more than one project is involved as reuse becomes a bigger problem in a multi-project environment. .
2. documentation becomes important.

Related Patterns
Designing Generic Services: Generic Services are services that are reusable in many ways, so to make good reusable services, also follow the tips and tricks in this pattern.

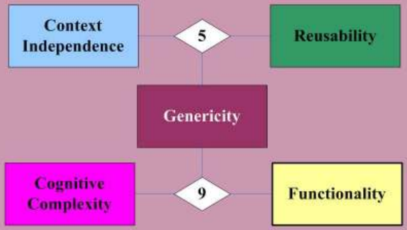
Further Reading
Artus, DJN. SOA Realization: Service design principles. IBM. 17-02-2006
Bloomberg, J. Should All Services Be Reusable? Zapthink. 31-5-2006. http://www.zapthink.com/report.html?id=ZAPFLASH-2006531
Schmelzer, R. Solving the Service Granularity Challenge Zapflash. 9-3-2006. http://www.zapthink.com/report.html?id=ZAPFLASH-200639
Sims, O. Developing the Architectural Framework for SOA – part 2 – Service Granularity and Dependency Management. CBDi Journal 2005 / June: Best Practice Report.

Pattern 3: Designing Generic Services

Context
To be able to maintain the SOA landscape, the total number of services has to be limited, but to maintain services, the complexity of the service should also be limited. Genericity deals with these two constraints. When the balance tips to one of these two extremes, two forms of generic services arise, namely:
- small generic services that provide little functionality and are reusable
- large generic services that provide much functionality and are mostly not reusable

Problem
How to design generic services that are reusable and provide enough functionality, while both the complexity of the service itself and the complexity of the service construction are maintained?

Forces
5. Context-independent, generic services can be easily reused in different places, provided that services are documented well and the financial structure supports reuse
9. To prevent too complex solutions and to stimulate reuse, generic services should include not many specific parts



How it Works?
1. Search for the genericity in processes and make one service for these generic parts, keep the process-specific parts out of this service and build new services for them (i.e. limit the number of parameters).
2. Watch for cognitive complexity when a service has to deal with much and complex functionality as this aspect limits the amount of functionality a service can expose. When a service needs to expose such complex functionality that it is not understandable by a human, the service is too large.
3. To achieve reusable generic services, make the services context-independent both on a technical way (platform-independent, no knowledge about other services etc) and in an organizational way (as little knowledge as possible about the domain).

When to use it?
When dealing with cognitive complex functions on the IS domain and when reusability of services is important.

Related Patterns
Reusable services: generic services have to be reusable, therefore these two patterns are linked strongly.

Further Reading
No relevant sources at the moment.

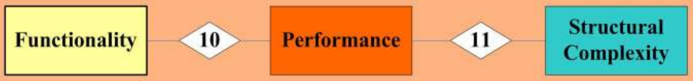
Pattern 4: Performance and SOA

Context
An SOA is based upon asynchronous communication and this can get in conflict with performance requirements. Both complexity of the landscape as complexity of the service can have influence on the performance.

Problem
How to make an SOA perform well?

Forces

10. Performance requirements limit functionality options
11. Many service calls between systems in the SOA cause performance problems



How it Works?
When performance demands are high, two aspects should be kept in mind during the development of the services. The first is the performance of the service itself and the second is the performance of the service landscape. To keep the performance of the service under control, the complexity and the amount of functionality the service provides should be limited. By testing the performance of the service in an early stage, the performance problems can be cleared away.

To keep the service landscape performable, the number of service calls in the landscape should be limited. When legacy is concerned, it is even more important that services should be context-independent and that the service calls perform well.

The two performance problems are counterproductive and a compromis should be found to deal with this. When performance demands are very high and the functionality is more complex, other solutions (in stead of SOA) should be considered.

When to use it?
This pattern should be used when performance demands are high during the development of an SOA.

Related Patterns
Generic Services: Generic services try to solve the dilemma between a complex SOA landscape and complex services, both which can have performance problems.

Further Reading
Linthicum, D. Designing SOA Web Services Services for Performance. 02-08-2005. http://webservices.sys-con.com/read/114127.htm