Researching a transition to an organized chaos in enterprise system architectures

Master's Thesis, June 2011

Ronald van Etten

<<Page left blank intentionally>>

Researching a transition to an organized chaos in enterprise system architectures

THESIS

Submitted in the partial fulfillment of
The requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Ronald Hendrik Maria van Etten
Born in Purmerend, the Netherlands

Web Information Systems
Department of Software Technology
Faculty, EEMCS, Delft University of Technology
Delft, the Netherlands
Eecms.tudelft.nl

TAM TAM B.V.
Patrijsweg 80
Rijswijk, the Netherlands
www.tamtam.nl

Researching a transition to an organized chaos in enterprise system architectures

Author: Ronald Hendrik Maria van Etten
Student ID: 1262947
Email: R.H.M.vanEtten@student.TUDelft.NL

Abstract

After its founding 15 years ago, Tam Tam has encountered the limits of legacy systems used for the internal processes in recent years. To overcome these limits, multiple efforts have been made to enforce a transition to specific Enterprise Application Integration styles. Although the efforts resulted in working application interactions, the desire for an organized whole sustained. The motivation for this thesis project was to make one final effort towards a new architecture by providing a carte blanche for all steps to be taken as well as their outcomes. The main goal of this thesis project was to research the most logical 'next step' for Tam Tam to take with the internal systems architecture and to derive which improvements this step brings into the picture. To reach this goal, a crossroad of three key elements is identified.

Context, the first element is to figure out what the current architectural landscape looks like and what can be learned from the previous attempts at integrating the systems.

Theory, the second element is to figure out how architectures can be compared, what the possible solutions are and which of those fits Tam Tam best.

Practice, the third element is to link the theoretical design to the practice to assess its feasibility, to evaluate which improvements for Tam Tam are introduced and, finally, to evaluate the chosen methodology.

The choice for sub-steps in the chosen methodology is based on best practices and guidelines from available (relevant) literature and is adapted to be used in this specific context. The focus of this adaption is that the chosen steps must be adaptable to other contexts too. After the complete methodology has been carried out, the key deliverables can be divided into two different categories: science and Tam Tam. For science the main deliverables are:

1) An aggregated list of re-usable insights

2) A case study for an Enterprise Service Bus transition

3) A re-usable step-by-step approach from legacy systems architecture towards the 'most logical' next architecture.

The key deliverables for Tam Tam entail:

a) A conclusion of the continuous effort towards Enterprise Application Integration

b) Sellable BizTalk know-how

c) An improved internal architecture with regard to flexibility, maintainability and adaptability

Concluding, the given carte blanche allowed for a successful path towards the best fitting Enterprise Application Integration style for Tam Tam. Furthermore, the methodology used is assembled in such a way it can withstand usage in other contexts and can be seen as a first step towards a formal definition of said methodology. This re-usability is introduced by the context-free focus during the definition of said methodology.

Keywords: business processes, Enterprise Application Integration, EAI, Service Oriented Architecture, SOA, Enterprise Service Bus, ESB, BizTalk, Legacy Systems, architecture transition plan, next generation architecture

Graduation Committee:
Prof. dr. ir. G.J. Houben, Faculty EEMCS[1], TU Delft
Dr. ir. A.J.H. Hidders, Faculty EEMCS, TU Delft
Dr. ir. J. van den Berg, Faculty TPM[2], TU Delft
Ir. B.A. Manuel, Partner, Tam Tam B.V.

[1] Electrical Engineering, Mathematics and Computer Science
[2] Technology, Policy and Management

Preface

Inherent to working a prolonged time on a project is the urge to share the most precisely selected in-depth knowledge with the world. This urge is not lacking in my case either, but a step backwards is taken to present the fields within which this project is executed.

Before starting this project, I was looking for a real-world problem to tackle in which a lot of improvements could be achieved. Once the back office of Tam Tam came into the picture I took that opportunity with both hands and have never let my grip loosen. Because of my great interest in the usage of IT in business scenarios it presented a perfect fit. Although it was difficult to formulate the scientific aspects of the project at first sight, I am very grateful that I was given a 'go-ahead' for this particular project.

At the start of my project, when I 'moved into' Tam Tam, I was immediately seen, and treated, as 'one of them' by other Tammo's. This was evident from the amount of indignation if I were not planning to join Tam Tam parties and leisure activities. Furthermore, I was given freedom in every aspect thinkable, from deciding the direction of the results to the ability to extend my vision by joining external (unrelated) events.

When looking at the course of my project, some people played key roles for which I would like to explicitly mention my gratefulness. First of all, professor Geert-Jan Houben for allowing me to join the Web Information Systems group and for asking the right questions at key points during my project. Secondly, Bart Manuel for expressing his confidence in the achievements I wanted to reach by including me as a Tammo and his seemingly inexhaustible technical interests and wishes. Thirdly I would like to thank my daily supervisors Jan Hidders and Jasper van der Sterren for pointing me in the right directions and having to listen to my every story regarding both project and non-project related matters. Fourthly I would like to thank Jan van den Berg for being my external committee member and for his valuable insights in how to bridge the gap between technology and business.

Due to my reasoning that Delft was not only a place to study but a place to live, I have been surrounded with supporting people. Thanks to my companions in my year of 'free time', as a board member of both CH and DDB, I have learned the most invaluable soft skills. Furthermore, fellow members of several committees persuaded me to take a step backwards from working on my thesis and studying in general. The last groups of people, who have pulled me through both ups and downs during my study, are my friends and family; please continue to allow me to take my own drift through life.

I hope you, as a reader, will enjoy reading the results of my research the way I enjoyed carrying out the research myself.

Ronald HM van Etten

Contents

List of Figures

List of Tables

Part I:

Preliminaries, introducing the playground

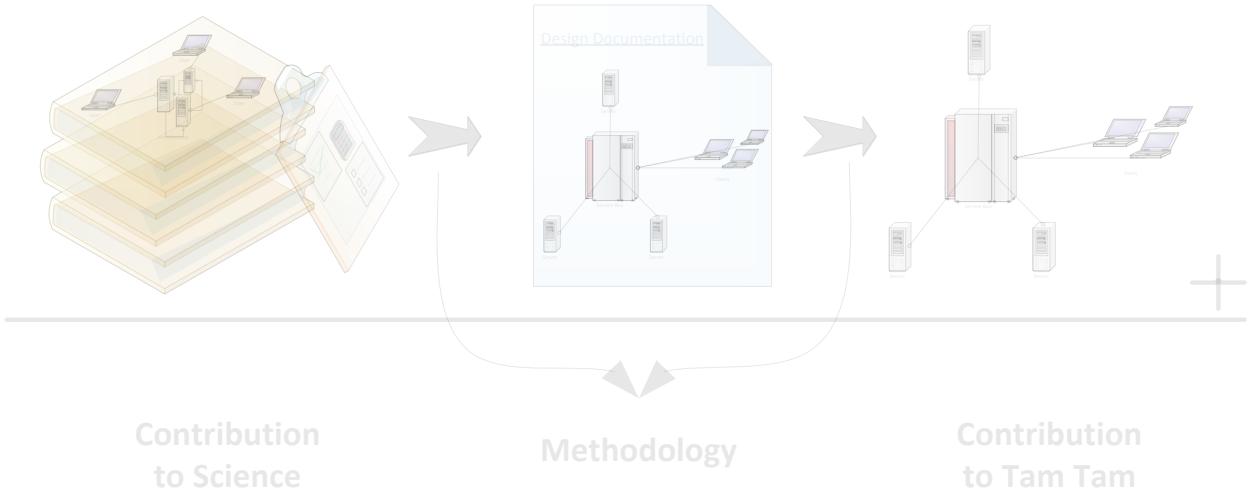

Figure 1: Empty, to be filled, playground

Chapter 1: Introduction

Right before diving into the research, this first introductory chapter strives to provide a crash course into the environment in which this project is carried out. The content of this chapter provides all information about the playground of the project necessary to gain an insight into the rationale behind this project and the environment surrounding the project. The five W's and one H –Where, When, Why, What, Who and How– are used as a starting point for creating said insights.

1.1 Background

Since the establishment of Tam Tam in February 1996, the company kept growing in several fields at a rapid pace. The amount and size of customers kept increasing over time. With this increase in customer base, the number of necessary employees increased as well. At the moment around 85 Full-time equivalent (FTE) are employed at a company competing among the top-rated full service Internet bureaus.

These increases put an enormous burden on the Information Technology systems and applications that are used in processes which keep Tam Tam up and running. The founders of Tam Tam started working with a central database to fulfill their storage needs. This central database contained every piece of data imaginable, from customer details through employee information to specific project implementations. Over time, the flexibility of the 'one-database-holds-all' architecture was questioned more and more and a decision was made to split the database into separate components. These components were identified in such a way that the functions they provide are very cohesive. Examples of such transitions are dedicated Human Resource Management and Customer Relationship Management systems.

Although the separation of concerns in more and more components does increase the ability to overview the whole architecture, each separated component carries out their own pieces of the processes. The next step from the separation of components was to integrate the components in such a way a synergy could be created by introducing a queuing mechanism for certain actions. Furthermore, several web services were introduced which created an abstraction layer.

1.2 Situation

The earlier mentioned effort to create synergy by integrating the different components was carried out by implementing different key elements from several different architectural styles. Although the chosen solution functions perfectly at the moment, it disappoints at other fields such as maintainability, flexibility and understandability. This implemented solution was subject to several attempts to convert operations to an Enterprise Service Bus in the past, of which no attempt pushed through.

The idea to design a complete revision of the architecture for the complete internal processes infrastructure continued to be relevant. Especially the introduction of services

was very appealing due to the increased flexibility by creating an abstraction layer. Due to these events, a new approach was longed for that is driven mainly by theory to provide a well-supported new future-ready architecture.

1.3 Research objectives

To overcome the need for a theoretically backed-up design, one question is identified to pose as the central research objective. Not only are the results of answering said question important but, when the question is looked at from a helicopter view, the chosen way to answer the question is of equal importance.

"How can Tam Tam benefit from the theoretically 'logical' next step for the internal systems architecture?"

To be able to formulate an answer to this question several research parts have been identified which, when the results are aggregated, form a precise reflection of both theory and practice. Each of the identified research elements is elaborated on in a more detailed fashion.

A. **What does the current situation look like?**

The current situation needs to be analyzed to be used as a foundation for further work to be based upon. The goal of this question is to get a complete picture of the systems architecture currently in place at Tam Tam from the available documentation, interviews, documentation and other available sources.

B. **What lessons can be learned from previous attempts at creating an integrated architecture?**

Before diving into the design process, a step backwards is taken to find out what previous attempts were carried out and what were the limitations encountered in those attempts. This question enforces the identification of key stumbling points that should be evaded and key lessons to be kept in mind during the design process.

C. **How can different architectures be compared?**

Once the current situation is completely clarified, a follow-up step is to design an improved architecture. The only thing missing between both steps is the definition of a way to evaluate and compare the different architectures. In answering this question, means should be identified that allow a thorough evaluation to identify the effects a new architecture has on Tam Tam.

D. **What is the theoretically 'logical' next step for Tam Tam?**

The subsequent step is to find out what theoretical next step is 'logical' to be taken in the Tam Tam context. This entails researching alternatives for the currently in place architecture and choosing a variant which offers the best fit for Tam Tam. The identified best fit architectural variant is tailored to be implemented directly for Tam Tam.

E. Is the designed next step feasible in practice?

The fact that the designed future architecture should be the best fit for Tam Tam is one of the key points kept in mind during the development. To make the designed architecture a success however, the practical feasibility should be assessed. During this feasibility study, all aspects of the design should be made tangible by implementing a proof of concept version of the envisioned future architecture.

F. What advantages and disadvantages does the new architecture offer?

After the key elements of the theoretical design have been linked to practice, by implementing a proof of concept version, the newly designed and implemented architecture is put to the ultimate test. In a full-blown evaluation session, the resulting architecture is assessed by using the means selected in the third research part.

G. How can pieces of the chosen process be used in other contexts?

This question comes from a helicopter view on the main research question. The answer to this question consists of a reflection on the chosen process in terms of key assumptions, reusability and limitations. Furthermore, this question forces some thought about the key elements present during the project that can be seen as driving causes of the achieved results.

1.4 Project organization

Due to the nature of this thesis project, two endpoints can be identified: science, represented by the Delft University of Technology; and practice, represented by Tam Tam. This means every action performed should be screened as to which goal it serves. By doing so, a balance was sought between theory and practice.

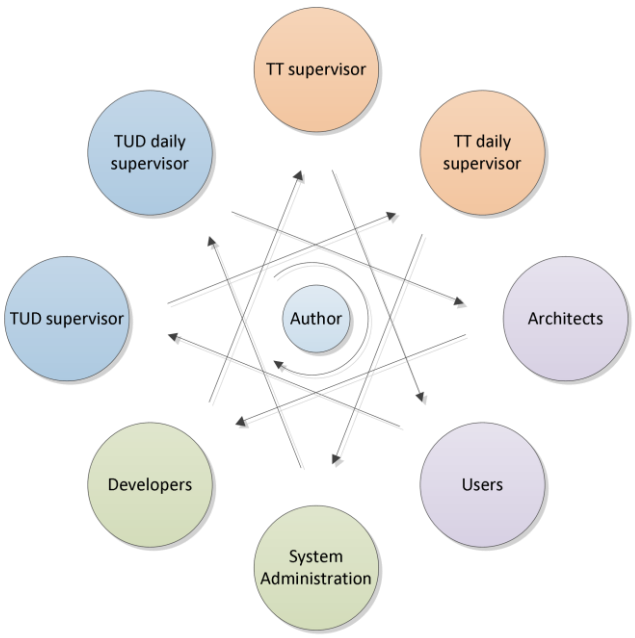

Figure 2: Involved actors

In performing this balancing act, several actors were present to support all activities as shown in Figure 2. These actors are categorized into four different categories that are created based on the phase or main activities they were particularly involved with. The first category of actors represent the university, their involvement is to supervise the addition of knowledge to science and the process throughout the entire project. The second category is the supervision body from the Tam Tam perspective, the only characteristic that differentiates them from the university actors is their attention to practice instead of science. The third category of actors involved

are the architects and users whose knowledge and expertise is used in discovering what the current situation looks like. The last category of actors consists of the developers and system administrators whose help is used during the implementation and evaluation phases.

1.5 Activities and Thesis Structure

#1 Key learning points throughout this thesis are indicated in these blue sticky notes

The main content of this thesis is divided into five different parts. The basic idea of the identified parts is to walk through the complete performed process by elaborating on activities in key categories. This section elaborates on the contents of the different parts by identifying the chapters, key activities and goals. At the end of this section, Figure 3 gives a summary of the different parts, their chapters and the research element treated in those chapters is indicated between brackets. Furthermore, throughout the thesis, key learning points are identified about the work performed that can be re-used when similar work has to be carried out (#1).

Part I: Preliminaries, introducing the playground

The first part is all about getting to know the environment in which this project is situated. This chapter, Chapter 1: 'Introduction', is started by elaborating on the history of Tam Tam and its internal systems architecture. After that, the motivation for starting this project is laid out together with the main research goals. The last step is to indicate what actors are involved in the project.

Part II: Context, gaining an understanding

The second part serves the goal of presenting the foundation on which further work in this thesis continues. The first chapter of this part, Chapter 2: 'Current Situation', elaborates on the current situation of the architecture under focus. After that, Chapter 3: 'Approach' presents the unique approach taken in this thesis project by investigating what can be learned from previous attempts to reach a similar goal.

Part III: Theory, designing a high-level architecture

After the foundation for this thesis has been laid out, the next step is to dive into the literature to investigate the theoretically possible next steps. Chapter 4: 'Evaluation of architectures' starts by defining a methodology to be used to compare different architectures. After that, in Chapter 5: 'Choice for architectural style', a decision is made for an architectural style which suits Tam Tam best. Chapter 6: 'Future architecture design specifics' concludes the theoretical part of this thesis by using common practices and guidelines from the literature for defining the design specifics.

Part IV: Practice, linking theory to the real world

Once the next step for Tam Tam is defined, it should be tested for feasibility by linking it to practice. Furthermore, the process of implementing this next step is to be assessed.

Chapter 7: 'Implementation' starts by indicating how the theoretical design was linked to practice. In Chapter 8: 'Evaluation of results' the defined architectural comparison means are used for assessing which achievements have been reached with the new architecture. The last chapter of this part, Chapter 9: 'Evaluation of process', discusses the process used in this thesis by taking a helicopter view and checking what can and cannot be re-used in other contexts.

Part V: Conclusion, wrapping up and contributions

The final part of this thesis is all about wrapping up and conclusions. Chapter 10: 'Contributions' starts with elaborating on the contributions this thesis makes to both science as well as Tam Tam. After that, Chapter 11: 'Conclusion' is the final chapter of this thesis and revisits the posed research elements and matches the derived answers to them.

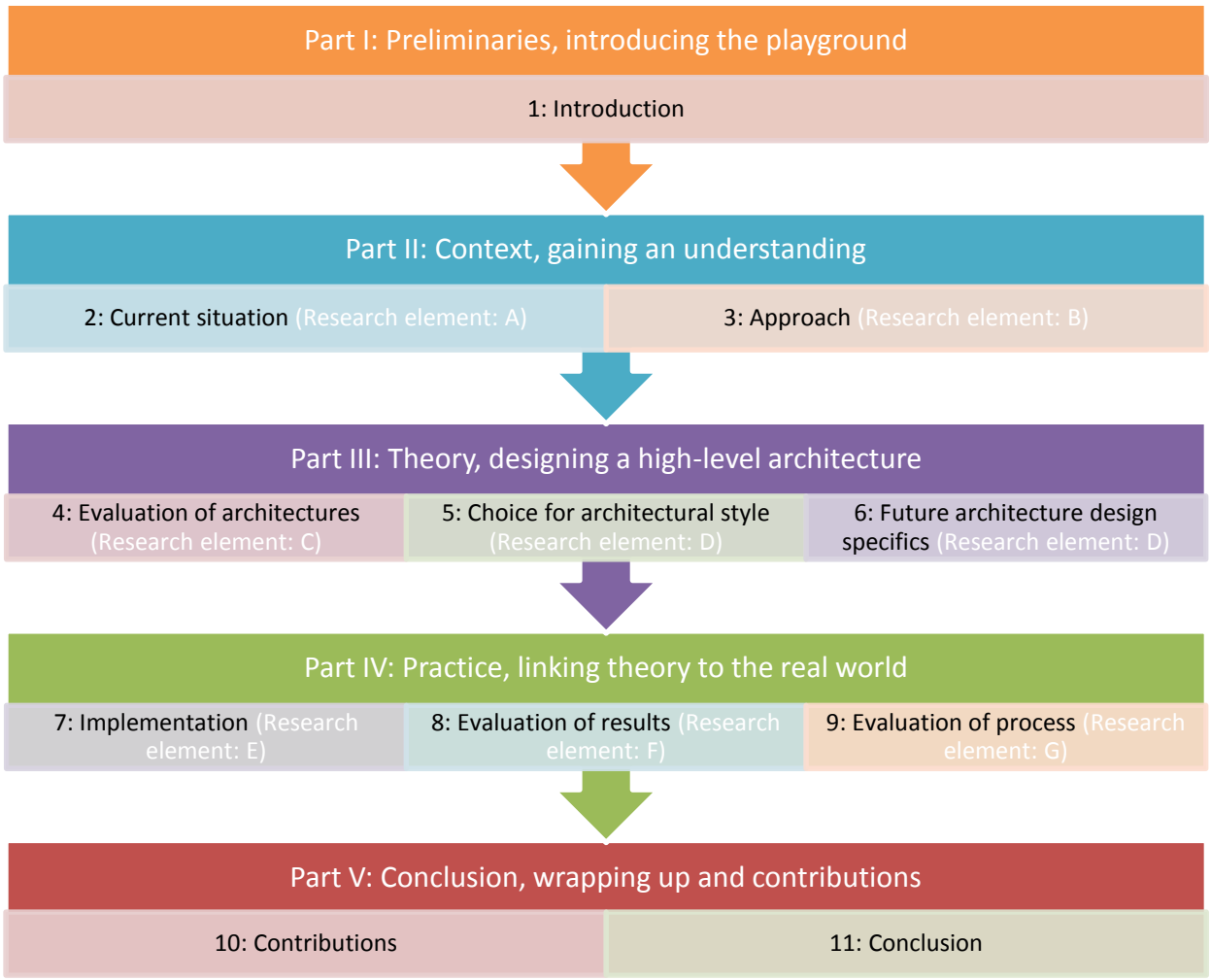

Figure 3: Thesis structure

Part II:

Context, gaining an understanding

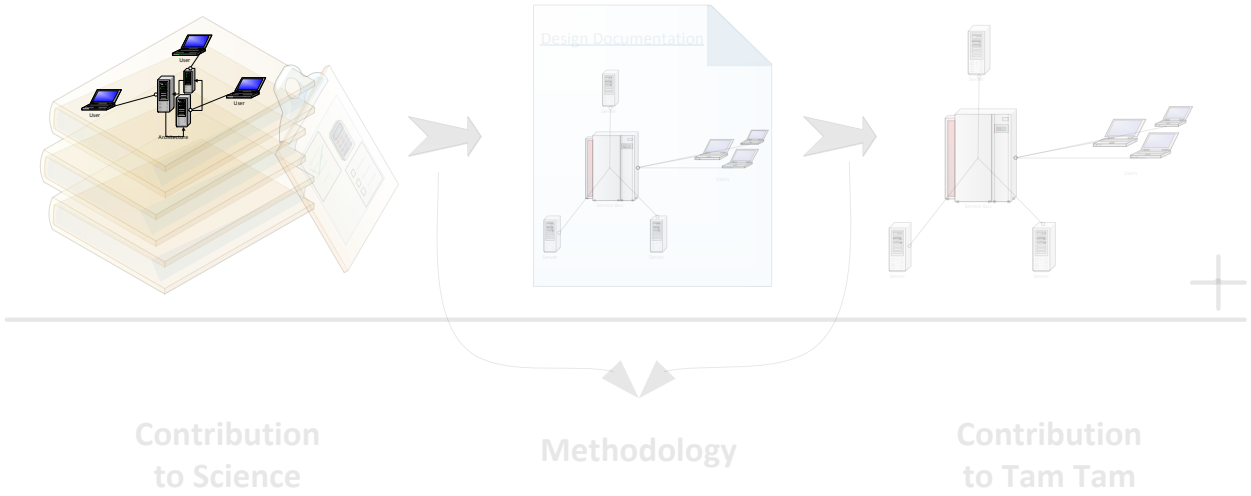

Figure 4: Understanding the current architecture

Chapter 2: Current Situation

Helping Tam Tam continue to be profitable, two different categories of processes can be identified; the first category consists of processes executed directly in the benefit of customers and the second category consists of internal processes to keep Tam Tam running. To generate the most profit for Tam Tam, the employees should be spending as little time as possible executing processes from the second category. The systems architecture within Tam Tam serves this goal but, as can be seen with almost any software architecture, the documentation of the architecture is not always compatible with the real world. The reason for this incompatibility can be attributed to either of two options: there is no documentation or the documentation became outdated due to architectural erosion and architectural drift as described in (Perry & Wolf, 1992).

Before being able to study the performance of the systems architecture currently in place, it has to be documented. The first section of this chapter describes a chosen methodology to be used for documenting the architecture by visualizing it. As stated in (Petre, Blackwell, & Green, 1998), (Perry & Wolf, 1992) & (Clements & Northrop, 1996), this visualization:

1) Presents a means for communication between stakeholders. It should bridge the gap between technical and non-technical stakeholders
2) Presents a means for identifying requirements of stakeholders. Both functional and non-functional requirements can be included
3) Serves the goal to make technical decisions early on in the design process.
4) Creates possibilities for reuse, the abstraction presented in the visualization can be transferred to other situations
5) Creates the possibility to perform an analysis of the dependency and consistency with regard to architecture, requirements and design

After the methodology has been defined, the second section elaborates on carrying out the chosen steps to describe the architecture currently in place within Tam Tam. The third section elaborates on some lessons that can be learned when the current architecture is analyzed and, finally, the fourth section describes the requirements that are posed to future architectures.

2.1 Architectural description methodology

To reconstruct the architecture from the currently in place systems, practices from two methodologies have been selected. Both the methodologies *Dali* (Kazman & Carrière, 1999) and *Symphony* (Deursen, Hofmeister, Koschke, Moonen, & Riva, 2004) assume very little is documented or known to stakeholders and thus focus on retrieving most of the data from the source code. The situation at Tam Tam however, is that some of the architects are still available to help documenting the current architecture. So instead of using only the source code, these architects are used to help visualize a course-grained architectural description. The source code is used after that to figure out more detailed information about the architecture in place. The process executed to describe the current

architecture is derived from (Deursen, Hofmeister, Koschke, Moonen, & Riva, 2004) and consists of iteratively executing three steps. The first is extracting information; the second is to take a step backwards from this information to get an abstract overview; and the third step is presenting the information. In the coming three paragraphs the aggregated result of the three steps is elaborated on.

2.1.1 Extract

The information extraction step is based on an iterative series of interviews with the architects and stakeholders. A mock-up of the architectural layout served as a starting point for discussion. This mock-up was created in collaboration with two of the technical architects who created the architecture. A white-board was used to create an overview of all systems, databases, applications and processes running within Tam Tam. This mock-up on the white-board provided a means to communicate about the architecture with other architects and evolved into a classical form of spaghetti architecture, vouching for the need to organize the components in a different way.

The result of performing the first iteration of this extract->abstract->present cycle meant the end of the mock-up on the white-board's lifetime. This first iteration resulted in an initial documented version used as a means for communication, verification and validation. With this initial version available, several interviews were held with stakeholders and users of components within the architecture. From these interviews the requirements and high-level process details are gained. This information provides a good starting point for further analysis on a lower level, the internal workings of the applications, by looking at the source code.

The last step of the extraction is the verification of the created visualization; this is performed by communicating with the stakeholders about the created visuals in several iterations. This ensures that the final visualization is reached in a step-by-step manner.

2.1.2 Abstract

The next step in this three-part iteration-cycle is the abstraction from the acquired information. This paragraph first of all presents the key characteristics of the architecture that is currently in place and after that presents insights into what to present.

Key characteristics:

1) *No need for flexibility*; due to the focus on the internal processes, no possibility for rapidly changing the architecture is necessary. This stems from the fact that the internal processes are not core business processes that should be able to change rapidly as the market changes
2) *Coupling*; a lot of applications are linked together with a lot of different interfaces. For understandability purposes the focus should be put at creating independent systems and technical harmonization in the communication between processes

3) *Structure and orchestration of application interactions*; another key focal point should be the business process orchestration to bridge the gap between business knowledge and technical knowledge

What to present:

1) *High-level & Low-level*; both high-level business process orchestration and low-level technical implementation details should be presented to give a complete overview of the architecture. The high-level business processes are used to communicate the use of certain components to the business people and the low-level technical details are used to communicate with the technical people
2) *Descriptive & Prescriptive*; when selecting ways to visualize the architecture, possibilities should be present for the description of architectures as well as prescribing what future architectures should look like
3) *Coupling*; both application and system coupling should be represented by the interfaces used, information transferred and process dependency
4) *Location of components*; it should become clear which applications run on which servers and where the source code of the applications is located

2.1.3 Present

The third step is to present the acquired information. To do so a way of visualization is selected in this paragraph. This is done by performing literature research and matching the retrieved possibilities to the requirements posed and the listings presented in the last section.

Ways of visualization

A lot of research has been performed to describe the architecture of software from certain perspectives called viewpoints; most of this research focuses on stand-alone software applications and can be divided in two forms. First of all there exist frameworks prescribing what set of viewpoints to use. Secondly, architecture description languages exist giving the user the choice regarding which viewpoints to be included in the description.

Frameworks

A lot of frameworks for visualizing architectures have been crafted, examples of these for the particular sub domain 'software architecture' are the '4+1' model (Kruchten, 1995), the Zachman framework (Zachman, 1999) and the SoftArch environment (Grundy, 2001), (Grundy & Hosking, 2003) & (Grundy & Hosking, 2000).

Each of these frameworks presents a means to describe architectures from the vision of the authors of that framework with their own ideology and requirements to the visualization of the architecture. All frameworks consist of a given set of viewpoints that –according to the authors– ought to be enough to describe most software architectures. The '4+1' model consists of four different viewpoints supplemented with scenarios illustrating the architecture forming the '+1' part. The Zachman framework defines the need for different viewpoints depending on the stakeholders involved and was created

with the mindset that there is no single 'software architecture' but a 'set of architectures' describing architectures. The SoftArch environment is a complete package for constructing and analyzing software architecture models. Next to that, SoftArch allows for visualizing dynamics to understand higher-level system behavior to be used as a debugging tool.

Architecture description languages

To gain more freedom in visualizing the architecture, several architecture description languages exist. The most prominent languages are Archimate (Lankhorst, 2004), Unified Modeling Language (Kruchten, Selic, & Kozaczynski, 2001) and the Business Process Modeling Notation (White, 2004).

The Archimate language divides software into several layers where each layer contains a specific category of elements (processes, information, applications, etc.). Next to using the advised hierarchical layered approach, one can use the notational elements fairly easily in other places. UML also defines a set of possible diagrams that can be created next to elements that can be used in other contexts. The last is the BPMN; this notation is specifically focused to communicate business processes with business people, while preserving the possibility to actually execute derivatives of the processes that are visualized.

Defining a way of visualization

As (Leist & Zellner, 2006) and (Gallagher, Hatch, & Munro, 2008) indicate, no single framework or architecture description language meets all posed requirements. Furthermore, it is very hard to find a framework that does everything necessary so there most probably is no one-size-fits-all toolkit for the purposes defined in this project. Because of this hardness, the most applicable step to be taken next is to combine the advantageous elements of all the listed possibilities.

The first step in selecting the best of all worlds is to select a set of viewpoints that are to be used. After that, the blanks have to be filled in with regard to the manner of creating the views from the perspective of the selected viewpoints.

Selecting viewpoints

According to (Smolander, Hoikka, Isokallio, Kataikko, Mäkelä, & Kälviäinen, 2001), the architect is dependent on both the organization in which one is operating and the set of requirements when selecting viewpoints. The requirements posed to this visualization can be categorized into two groups: static and dynamic. The static requirements state that the architectural visualization should provide a clear insight into what components are in place within Tam Tam and how these components interact with each other. The dynamic requirements state that it should be clear what business process uses what components and how this interaction is choreographed. When this identification is extended, three different viewpoints are exposed: 1) the static overview of all components and their connections; 2) the dynamic business process orchestrations; and

3) the link between the firsts two, describing the choreography of interfaces between components.

Defining means to create views

After the three viewpoints have been defined, this section describes how the views are constructed.

Viewpoint 1: Overview of components and connections

The creation of views from this viewpoint is based upon the logical viewpoint of both the '4+1' model and the Zachman framework. The specific way to represent components is done through the combination of the components present in the Archimate model. The way of connecting the components is derived from the component model of UML. Each connection is given a unique identifier that poses as a reference point from the other views. This identifier is composed of the following triple: <from component, to component, process>. On top of that, several colorings will be used to enable quick grasping of specific information. Typical categories like 'Information Sources', 'Applications' and 'Service Providers' are each given a unique color. The result of one such coloring can be seen in Figure 5.

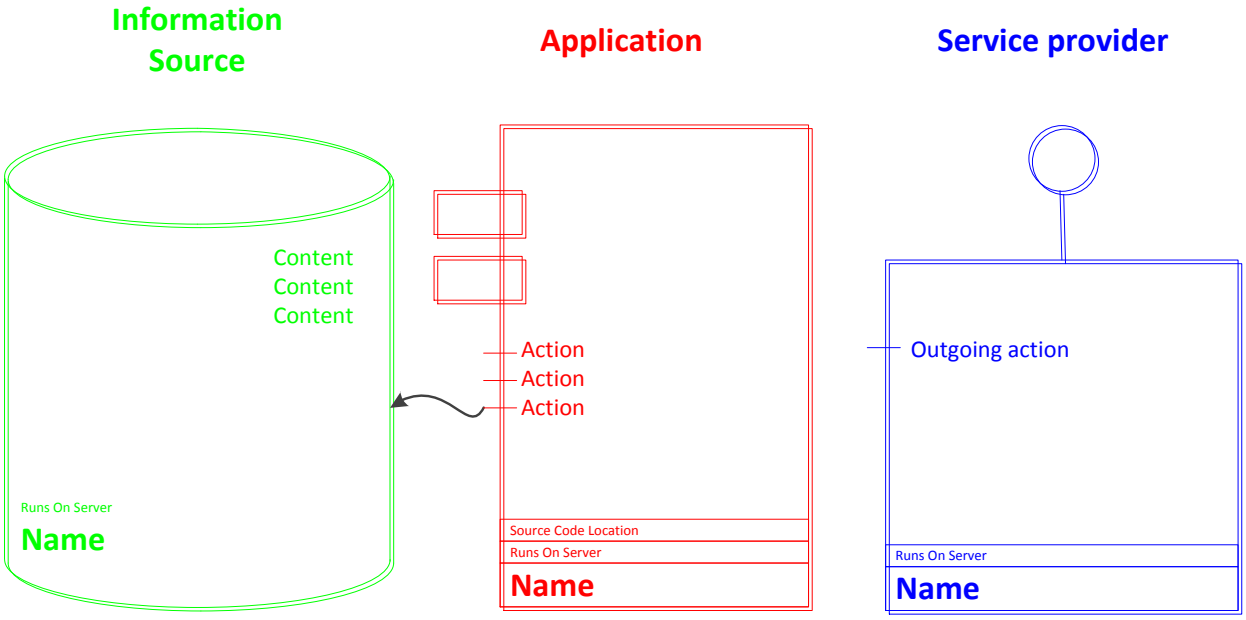

Figure 5: Architecture description viewpoint 1

Viewpoint 2: Business process orchestration

Of the options listed earlier for representing software architecture, several have the capability to present business processes. The BPMN however, is especially designed for this purpose so this approach is chosen. In modeling the business processes, the unique connection identifiers can be derived from these BPMN models by tracing what connections are used in which process. These identifiers are used as reference points for the models in the third viewpoint.

Viewpoint 3: Choreography of interfaces between components

To precisely describe the choreography of the interfaces, the UML sequence diagram approach is taken in which each interface is described as a sequence of performed actions. One of the advantages of this is that the sequential approach forms a basis for re-engineering the connections in the architecture by presenting an elaborate description of the inner workings of the connections.

2.2 Current architecture description

This section presents a summary of visualizing the current systems architecture in place, the complete description of the process is presented in Appendix A: 'Description of the current situation'.

Starting point of the architectural description are life cycles, identified high-level workflows representing a concatenation of different processes. These life cycles help identify the processes which are carried out within Tam Tam and are from the following categories:

1) Employee life cycle: representing the whole contact period an employee has with Tam Tam, from the first contact to becoming a Tam Tam alumnus
2) Customer life cycle: representing every contact with customers, from acquiring of new customers to offering and carrying out projects for them
3) Service life cycle: after projects are carried out for customers, the resulting products need to be supported. This life cycle entails the steps from handing over the project to fixing issues
4) Administrative life cycle: this life cycle has to do with all processes involved in managing the employees and customers, it includes sending invoices, booking hours and paying salaries

With these life cycles in hand, different processes were identified and used to identify individual components. All components are listed with their main roles and their relation with processes. After all data is extracted, the next three steps are visualizing the data in the form of the three defined viewpoints by creating models:

1) Viewpoint 1 'Global overview': a map with locations and connections between components
2) Viewpoint 2 'Visualization of component usage': 14 different models representing the actions taken to complete processes.
3) Viewpoint 3 'Component interactions': 45 different models representing the interactions between two different components

2.3 Lessons learned from current architecture

Now the systems architecture currently in place is described, a step backwards can be taken to figure out which characteristics stand out. A critical view is necessary to derive what can be learned from the current architecture. The approach taken here is to

aggregate feedback from architects supplemented with an analysis with respect to prior knowledge. Each of the identified lessons learned is elaborated on in this section.

2.3.1 Databases

The amount of databases present in the current situation is limited but they serve a wide variety of purposes. A positive tendency can be identified because a clear transition is visible from a situation with one central database to a more by divide and conquer ruled division of databases. Although this helps diminishing the risk of a single point of failure, an overall design is missing.

2.3.2 Management

Another characteristic that becomes apparent is the lack of a responsible entity that has the decision rights and knowledge necessary for initiating and declining changes. Furthermore, the lack of knowledge management causes an unclear vision of what has to be changed where when some functionality needs to be adapted or added (#2).

> ***#2 Knowledge Management** is very important, but not limited to sharing knowledge. Sharing what knowledge lies where can save maintenance time as well*

2.3.3 Coupling

A high amount of coupling is present in the current architecture, meaning that a lot of dependencies can be identified. These dependencies cause a ripple effect when changing some functionality in one location. This can happen directly or only become noticeable after some time has gone by since the initial change was performed.

2.3.4 Development and production environment differences

From the source code and live code of different applications it appears that the configuration files are very different and the live configuration files are only available at the live locations. This means it is very hard to trace what happens exactly with what components because the correct information cannot be retrieved from source control systems.

2.3.5 Complexity

The complexity of the current situation is still reasonably manageable but when, in the course of time, more components are introduced and knowledge is lost, this manageability will diminish. This causes the current architecture to be reluctant to change on a technological field instead of the normal human resistance for change.

2.3.6 Hardcoded connection strings

A lot of hardcoded connection configurations are present. The problem with these is that they are not used in a uniform way so it is very hard to list all of them. Furthermore, when a referenced component changes it is hard to find out where to change the references to it.

2.4 Requirements for the new architecture

The requirements to be reached can mostly be derived from the motivation for performing this thesis project. Most motivation elements stem from several issues, ranging from the way the systems architecture is currently documented to the eagerness for certain characteristics which are not present in the current architecture. This section firstly describes the different motives posed before starting the project and secondly indicates the requirements that are to be reached by the new architecture.

2.4.1 A priori justification

Documentation deficiencies

Almost every software engineer who just starts crafting computer programs comes across the urge to document his design decisions and program details. Within Tam Tam the amount of documentation is limited, and the documentation that is present deals with details of specific applications only. There is a clear lack of a documented global overview of software components in place within the company. This lack should be turned around in such a way that every Tam Tam employee has the possibility to find out what components are in place where and for what reason.

Knowledge about process orchestration

Although stored documentation is missing, knowledge about the systems architecture is present in the minds of the developers and architects. The problem with this way of organizing the knowledge is that very few people know what is happening behind the curtains. Of those who do have knowledge about the architecture, the architects know the most but do not have a complete picture either because several aspects have been changed over the course of time by application developers. These application developers however, have a complete picture of how certain applications work and with what other applications they communicate.

This statement makes clear that there exists a knowledge gap between how business processes work and how the technical components work together. The low level interaction can be retrieved easily, but what their place is in the bigger picture should be made clear.

Legacy systems

Since the establishment of Tam Tam in 1996 more and more applications were introduced to create more efficient workflows. Once in a while new requirements for applications became apparent and were implemented in the existing systems architecture. It becomes clear that a lot of applications work together in reaching a common goal. The way in which they are coupled is not optimal and is not very well documented either. To move away from the practice ‘do not touch applications when they are not broken’ to a better maintainable situation, means should be available to replace or expand legacy applications.

2.4.2 A posteriori requirements

Less coupling

At the moment a lot of applications are connected in a higgledy-piggledy way. This can mainly be attributed to direct hardcoded service requests and hidden function calls. What is desired is an architecture design with far less connections and, on top of that, far less coupled applications.

Create more insight

As was described in previous parts, there is little complete insight into the systems architecture. More insight should be created into what systems and applications compose the current architecture and the inner working of the interactions between these components.

Create an independent situation

Some processes that are performed within Tam Tam span over a variety of components. The case is that once one component stops responding for a while, it is very hard to trace where the problem originates. This creates situations in which problems arise at 'random' places due to a defect in another 'seemingly unrelated' application. The new situation should provide a burden to create components that are directly dependent on each other.

Increase maintainability

Due to the lack of documented knowledge about the systems, it is hard for application developers to maintain applications they have not created. The process of maintaining starts with searching for information like the location of the source code or where the application is running. When this information is retrieved, the maintainer can start figuring out how the application works and how functionalities are invoked of other applications. To increase the efficiency of this process, clarity should be crafted by creating one starting point and clearly defining external interfaces for applications from which maintainers can start performing their job.

Harmonization of technology

Although a lot of applications already work in comparable ways, the situation improves when clearly documented common practices regarding used technology are used. This increases the maintainability and extensibility through the use of standards by encouraging everyone to use the same practices.

Chapter 3: Approach

Multiple goals that are to be reached with this thesis project are far from new for Tam Tam. A key element in the IT strategy of Tam Tam is to regain control over the internal systems architecture. In the past, several attempts have been executed with the goal of reaching a Service Oriented Architecture or an Enterprise Service Bus. The main motivation for Tam Tam to renew said tendency to carry out such a transition is the fact they sell similar solutions to clients and want to adopt the motto 'practice what you preach'. This chapter elaborates on the previous attempts and sets the mentality for the approach used in this thesis based on findings about previous attempts. The first section provides details on several of the previous attempts and what can be learned from those. The second section lists the key principles used throughout the remainder of this thesis.

3.1 Lessons learned from previous attempts

In the past, several attempts were made to formalize the internal systems architecture of Tam Tam. For each attempt the key characteristics and objectives are listed, the reached goals are elaborated on and the reason for the (partial) failure is identified.

New internal portal (2006/2007)

In 2006, the realization that not everything could be stored in one central database caused the division of concerns into separate applications and databases. Furthermore, new functionality was added to the intranet by creating a new internal portal, adding a customer project environment and starting to use SharePoint for the diverse portals. The ultimate goal of this project was to create a single point of access for employees. The failed aspect in this project is that the focus was mainly on adding new functionality in a quick and effective way, which succeeded gracefully (#3).

*#3 **Focus on addition of components** limits the possibility to carry out a thorough architectural revision*

Project for integrating the CRM system with the central database and customer access point (2008)

With this new portal, more and more components were introduced. The next step after this introduction is to couple the components in such a way they can cooperate. This was performed in an outsourced project with the main goals being: 1) creating a flow from the Customer Relationship Management system to the central database to synchronize the information and 2) enabling customers to log into and use the projects SharePoint environment. During this project, an implicit choice was made to use known technologies to implement the interconnections between components. Although an

*#4 **Limited knowledge and time** prevents a complete architectural overhaul*

Enterprise Service Bus such as BizTalk was an option, limited knowledge and time decreased the feasibility of such an attempt (#4). This attempt resulted in the current architecture that, as is explained in section 2.2, contains a minimal Enterprise Service Bus version and a first step of data abstraction in the form of several web services.

Redesign portal architecture (2009)

The next step was to rethink the locations where applications are hosted. The goal of this project was to clarify which application was running where and to straighten these locations. This could be reached by carrying out the created design, which indicates what (category of) applications should be running on what server. As can be seen from the current architecture, this design was not carried out; applications are still freely distributed over available servers. The reason for this was plausibly the complexity of the architecture or the limited resources available for internal projects to correctly relocate applications with all their runtime environment requirements (#5).

*#5 **Complexity and limited resources** cause the focus on projects that do not directly influence income to fade away*

BizTalk attempts (BizTalk 2000, 2002, 2004 & 2006)

Several projects were focused on introducing Microsoft BizTalk for the internal systems architecture. A lot of different versions were attempted to get up and running. Each of these versions provided new and improved functionality, but the main reason that BizTalk kept coming back was the partnership of Tam Tam with Microsoft. Several of these attempts failed at the implementation stage, others while configuring and still others did get simple file-handling processes working but that was their final stage. The attempt that was most successful was carried out in 2007 and got as far as to have implemented several processes in BizTalk 2004. These processes never found their way into the production environment however, because the developers kept running into problems (#6).

*#6 **Technology push** limits the possibility to use a different product, which might form a better fit, to be used*

3.2 Key principles

After having listed what the previous attempts were and what posed as the key factors causing the limited success rate, the next step is to learn from those to make the common goals more easily reachable. The idea of this section is to derive principles from the listed insights of previous attempts and to supplement those with other defined key values that should force the approach to be successful.

Focus on understanding the environment

Before being able to improve a situation, the complexity of the design process should be reduced by a fair amount by making every aspect of the current situation clear. Due to this, the amount of effort put into gaining an understanding of the current situation was

a major part in the beginning of the project, with the details discussed in Chapter 2:'Current Situation' as a result.

Focus on organizing the current architecture, not adding functionality per se

The key idea behind starting this project was to take the systems architecture to the next level. Part of this idea entails that the creation of insight into what is in place is far more important than the addition of extra functionality at the operations level. At a higher level, extra functionality should be added in the form of flexibility, maintainability and manageability as stated in section 2.4.

Carte blanche

Before the project started, a carte blanche was given which reduces the consequences of a limited choice for a specific vendor of technology or a specific way of working in pursuing the objectives. This freedom of choice adds the possibility to find the 'best-fit' architecture for Tam Tam without being forced to focus on one specific area.

Exit strategy

To create the biggest possible support from all stakeholders involved with the architecture, much attention is given to exit strategies (#7). Resulting from this attention is that the transition from the current to the future architecture should be possible to perform at the last possible moment while implementing. While performing the transition in such fashion, the chances of the end users noticing any change should slim to none.

*#7 **Fallback policies** should be available at all time, hereby limiting the amount of points of no return and conversely introducing exit strategies*

Hands-on driven design

The nature of creating a theoretically backed up design may cause incompatibilities between theory and practice. To overcome this risk, an approach is adopted to design the future architecture in a hands-on driven fashion. By continuously creating proof of concepts for key decision points, the feasibility to link theory to practice should never decrease (#8).

*#8 **Proof of concept driven design** helps making feasible designs by continuously implementing and testing the designed elements*

Evaluation in iterations

The nature of this project is to reach an end state in which several improvements have been reached. Due to the fact that the Tam Tam way of working is based on Agile Scrum[3], such an approach is used here too. It can be envisioned as rowing a boat

[3] http://en.wikipedia.org/wiki/Scrum_(development)

towards a lighthouse, the goal is clear but the way of reaching there should continuously be adapted to the wind and water flow. By iterating with evaluation sessions, the project keeps focused on the set goals.

Design for change

Key objective for this project is to move to an architectural situation that is much less rigid. This entails that the architecture maintainers should be capable of achieving changes to the architecture in very limited time. To enforce this, the first step is to focus the design on the ability to change easily.

Part III:

Theory,
designing a high-level architecture

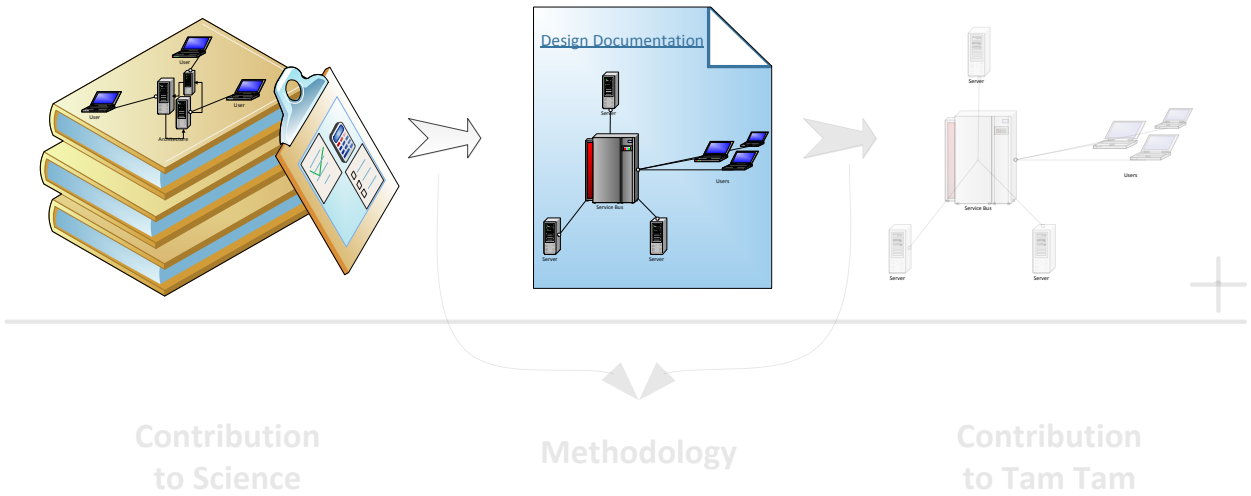

Figure 6: What does the theory say about the architecture

Chapter 4: Evaluation of architectures

Verbalization of the current architecture presents the foundation for further work in this thesis; the next step is to take said architecture to the 'next level'. Before being able to do so, a crisp definition of how the 'next level' can be assessed should be defined. That definition is performed in this chapter by firstly defining a methodology for architectural evaluation. Secondly, the methodology is adapted to fit in the Tam Tam context. Finally, the first evaluation iteration is executed on the current architecture to provide a reference point for comparing the advantages and disadvantages of possible future architectures. This first evaluation iteration is followed by several other iterations at key decision points in the subsequent chapters, the results of which can be found in Appendix E: 'Evaluation iterations'.

*#9 **After decisions have been made** it is still necessary to re-assess whether or not the made decisions were correct*

4.1 Architectural Evaluation Methodology

This section elaborates on the methodology of selecting means and metrics to be used for comparing the current architecture to possible future architectures. The goal is to define several means and metrics to evaluate at several instants throughout the project. The first occurrence of an evaluation instance is before designing the new architecture. The underlying argumentation for this is that the way of evaluating architectures can also pose as 'design goals' that need to be kept in mind. As (Assmann & Engels, 2008) put it, a model should be crafted that allows the architect to evaluate whether the evolution of the architecture still conforms to the desired style or not. The second occurrence of an evaluation instance is between the selection of an architectural style and the design of the new architecture. The third occurrence of an evaluation instance is between designing and implementing the architecture. This is because early in the process, changes are still relatively cheap to make in contrast to changes to be made after the implementation of the architecture is started (#9). The fourth occurrence is after the first few implementation iterations by checking if the implementation is still on track to achieving the desired improvements. Furthermore, this last iteration is to thoroughly evaluate whether or not the implementation should proceed.

4.1.1 Methodology

The main goal of this paragraph is to derive a methodology for selecting means and metrics for evaluating architectures based on the available literature. As stated in (Lindvall, Tvedt, & Costa, 2003), the first thing that becomes apparent is that means and metrics are only effective when they are selected based on the context in which they have to be used. This means that the means and metrics will have to be defined for the Tam Tam context specifically. With this perspective as a starting point the next step is to define practices to be performed in the evaluation steps.

In the literature two main classes of software architecture evaluation techniques can be found: 1) questioning, based on qualitative questions and 2) measuring, based on quantitative measurements (Dobrica & Niemel, 2002) (Abowd, Bass, Clements, Kazman, & Northrop, 1997). The questioning part is present a lot in the literature which can be deduced from the amount of research into scenario-based evaluation methodologies (Dobrica & Niemel, 2002) and (Babar, Zhu, & Jeffery, 2004). The reason behind this, as stated in (Kazman, Abowd, Bass, & Clements, 1996), can be found in the same reasoning as the need to define a specific perspective: it is very hard to find a 'universal' approach for evaluation, the only approaches that give a good image of architectural quality are derived from the specific context in which they have to be used.

Following this reasoning, the approach presented in (Lung & Kalaichelvan, 2000) is adopted. This approach stems from the 'goal->question->metric paradigm' presented in (Basili, Caldiera, & Rombach, 1994) and allows one to identify evaluation means in three steps by moving from objectives through scenarios to individual metrics. The first step to be taken is to derive the objectives that are in place from the defined requirements. These posed requirements can be attributed to two main sources because the transition to a Service Oriented Architecture bridges the gap between the technology layer and business layer (Assmann & Engels, 2008). This means that opinions and requirements from those two fields are represented in the list of objectives.

From the described objectives several scenarios are to be defined, these scenarios represent plausible future situations and are explicitly not desirable future situations. Defined scenarios can fall into three categories: use-case, growth and exploratory (Kazman, Klein, & Clements, 2000). These are typical instances based on: standard use of the system, anticipated maintenance and extreme changes to the system. The approach used here is that scenarios should be chosen without looking at the possibility of occurring, thus presenting a more complete overview and including the black-swan events (Lassing, Rijsenbrij, & van Vliet, 1999).

The last step of the leading triple is to identify metrics from the separate scenarios. This is done by deriving what the success or failure factors are for the specific scenarios. Key idea to keep in mind while performing this is that values for the metrics should be easily retrievable. This retrieval can be performed by either using quantitative metrics, or by using more qualitative metrics that can be valued by expert-based assessment. This expert-based assessment serves another goal by gaining expert opinions of the different architectures. Although the experts will most likely use the same perspective, the fact that they are not deeply informed in the matter allows them to look at the architectures from a larger distance.

4.2 Architectural Evaluation Means and Metrics

This section carries out the architecture evaluation methodology to define the scenarios and metrics to be used in the evaluation iterations more precisely. The following paragraphs describe the output of all four steps, 1) defining a perspective, 2) defining objectives, 3) deriving scenarios, and 4) selecting metrics.

4.2.1 Defining a perspective

While defining the perspective to be used in the evaluation, the starting point is Tam Tam and their use of the architecture being evaluated. The first thing that comes forward is that the architectures in the scope of this project are composed of the back-end parts of a lot of components. From this focus is derived that use-case scenarios of users using the components are of very little use here. The focal point should be the application developers and system administrators perspective because these are the users whose interest is at stake here. Typical interests are thus the addition and removal of components, updating functionality of components, managing the server locations these applications run on and maintaining the interactions in place between diverse components.

4.2.2 Defining objectives

The second step that needs to be carried out is to identify the relevant objectives from the defined viewpoint. The main source of these objectives is found in the requirements and statements of project necessity listed in section 2.4. Table 1 states all objectives to be used in the evaluation.

Table 1: Evaluation objectives

Name	Identifier
Decrease documentation deficiencies	A
Increase knowledge about process orchestration	B
Increase manageability of legacy systems	C
Decrease amount of coupling	D
Create more insight	E
Create an independent situation	F
Increase maintainability	G
Harmonize the use of technology	H

4.2.3 Deriving scenarios

From the defined objectives, scenarios are derived by investigating what actions are influenced by a change in each of the objectives. An example of this is the 'Decrease documentation deficiencies' objective. When a new application needs to be added to the architecture in place, the developer should investigate how the new application should interact with the components currently in place. This can be a very cumbersome task when little documentation is available, but in the case the documentation deficiency is diminished it will become a task that can be executed more easily. The derived scenarios are elaborated on one-by-one, indicating the objectives they stem from and providing a short description about what happens in the specific scenarios.

Scenario 1: Addition of an extra component (objective: A)

Every once in a while a new functionality needs to be added to the architecture in place. This can be the result of either completely new functionalities or the branching of component functionalities to more dedicated solutions. This scenario deals specifically with the part of adding new functionality, the part of removing functionality from other components is represented in another scenario. Key stumbling point is that the developer should know how the new component should interact with existing components and what processes it is involved with.

Scenario 2: Moving a component to another location (objectives: A & D)

The location of components is fairly static in the current architecture, but it might occur that a component should be moved to another location. When this occurs, all coupled components should be changed to reconnect to the component on the new location.

Scenario 3: Extension / withdrawing functionality of a component (objective: A)

A very common activity in the history of the architecture is splitting components into separate dedicated components. When this happens, several elements need to be adapted. First, the incoming interactions should be retrieved. Secondly, it should be made clear what component fulfills what functionality. These tasks can be completed more easily when the documentation is up-to-date.

Scenario 4: Carrying out maintenance on a component (objectives: E & G)

As with every man-crafted element, the running components need to be maintained. The main problem with this is that if another developer than the creator performs the maintenance work, a lot of effort is put into understanding the component. Questions that need to be asked here are for example 'where is the source code?', 'on what place does the live component run?' and 'how does the component work (together with other components)?'

Scenario 5: Change the functionality of a component (objectives: D, F & H)

When the functionality of a certain component needs to be changed, other components might have to be changed as well due to the ripple effect the first change initiates. The focal point of this scenario is this ripple effect, where scenarios 3 and 4 focus on adaptions to single component in focus.

Scenario 6: A process cannot be executed anymore (objectives: B & F)

Although servers usually have a very long uptime, it might occur that a server goes offline for a while. This can have the effect that a certain process cannot be carried out anymore. A way to fix this is by finding out which step of the process stopped functioning correctly and fix that part. To be able to do this, there should be knowledge about which components are used by what processes. Another key element that should be in place is that no process transactions get lost due to the server crash.

Scenario 7: Replace a component (objectives: C, E & H)

When a component is replaced by another implementation, it should be clear what other components are dependent on the old component that should be changed as well. Furthermore, the exposed functionality the new component provides should match that of the old version.

Scenario 8: A certain process needs to be extended (objectives: A & E)

When the functionality of a certain process needs to be extended, one (or more) components should inherit the new functionality. To do so, the developer should have a

clear insight into what components perform which functions. Next to the amount of cohesion within components, the available documentation presents a key source of information.

Scenario 9: A server is going offline (objective: E)

Servers do not have an eternal lifetime so they have to be replaced at some point in time. Another event that can occur is a cut in the costs of the server park. Both actions can only be executed when some level of certainty is achieved that no critical components are running on those servers. In other words, a server can only be shut down when all components that ran on it are restarted at other locations.

4.2.4 Selecting metrics

The fourth and last step is to identify the metrics to be used from the identified scenarios. Table 2 presents all metrics to be used, combined with the scenarios that lie at the foundation for specifying them. The last column in the table represents the manner these metrics are measured while evaluating. The first four metrics are mainly calculated by counting and the last six are retrieved by measuring the time needed to perform the separate actions (#10). Values for most of the metrics for the current architecture can be measured directly, values for possible future architectures are a bit harder to attain. The values for the future architectures are attained by carrying out careful desk research.

*#10 **Increased values for metrics** might also be advantageous although this change might feel unnatural*

Table 2: Evaluation metrics

Metric	Used for Scenarios	Manner of measuring
Number of interaction schemes	Sc1;Sc7;	Source-code inspection & design guideline
Number of hardcoded connection strings	Sc2;	Source-code inspection & design guideline
Number of incoming method calls per component	Sc3;Sc5;Sc7;	Source-code inspection & interaction contracts
Number of components with related functionality (duplication)	Sc3;Sc8;	Architectural inspection
Speed of source code retrieval	Sc4;	Usage of documentation
Speed of live location retrieval	Sc4;Sc9;	Usage of documentation
Speed of understanding application interactions	Sc4;	Usage of component choreography
Speed of understanding process orchestrations	Sc6;Sc8;	Usage of process orchestration
Speed of checking component status	Sc6;Sc7;	Manual process & design guideline
Speed of understanding component purposes	Sc8;	Usage of documentation & usage of process orchestration

Chapter 5: Choice for architectural style

After the current architecture and way of comparing architectures have been laid out, the first step towards the design is to figure out the surroundings within which the design can be formed. This is done by deciding on the architectural style that fits Tam Tam best. The first step in this process is to define a framework for comparing the different possible styles. The second section uses the listing of all available options in Appendix B: 'EAI styles' as an input for defining means for comparing the different styles by selecting key differentiation continua. The third step, after the continua are selected, is to retrieve values for all different possible architectural styles from the literature. These values are used to make a well-founded decision. The last step is to evaluate and verify the chosen theoretical style in the second evaluation iteration presented in Appendix E: 'Evaluation iterations'.

5.1 Architectural style comparison framework

The decision for an architectural style lays the foundation for the design by providing a domain within which some more specific decisions are to be made. Despite the fact that all architectural styles can be used to simulate every other style, a decision for one has to be made to provide a starting point that bridges the largest possible gap between the current and a possible future situation. To make a well-founded decision, a comparison framework was used on which this section elaborates.

Although this project started from a wish to go 'service oriented' in the form of creating a Service Oriented Architecture, the given carte blanche facilitated a helicopter view with regard to finding out which architectural style suits Tam Tam best. To make this decision, the first step is to investigate all possible architectural styles and after that diminish the possibilities by creating trade-offs. This trade-offs represent continua onto which all architectural styles can be mapped. A value is attached to each architectural style so that style can be mapped on the continuum. When a specific architectural style is mapped on all identified continua, the result is a representing vector for that specific architectural style.

After a representing vector is identified for all architectural styles and the current situation, a complete picture is sketched of what possible changes can be reached by choosing a specific style. With this complete picture, a decision can be made for a certain style that fits Tam Tam best. This is done by interviewing the stakeholders to ask their opinions regarding the place the future architecture should have on the different trade-off continua (#11). The opinions of the different stakeholders are averaged into one vector defining the desired situation.

*#11 **Obscurity of relations** between trade-offs and possible future architectural styles prevents bias*

With the sets of vectors available, the distance between the desired vector and all possible architectural styles can be calculated. The result of these calculations identifies the architectural style that presents the best fit for Tam Tam. The best architectural style is verified by checking whether the differences between that style and the desired situation are manageable.

5.2 Decision trade-offs

To make a choice for one of the specific forms of future architectures listed in Appendix B: 'EAI styles', several decision trade-offs have been crafted that aid in comparing the possible architectural styles. Each identified trade-off is discussed briefly and an indication is given how this trade-off was defined.

5.2.1 Centralized versus decentralized process knowledge

There are basically two types of organizing knowledge about process orchestrations. The first is storing all knowledge in one place by making the entity in which this information is stored responsible for carrying out the processes. In this type, the application logic is separated from the integration logic (de Leusse, Periorellis, & Watson, 2007). The Message Oriented Middleware and Enterprise Service Bus approaches both incorporate a similar mechanism. The other type is by storing the knowledge in a decentralized fashion by letting components coordinate the process orchestrations themselves. The Service Oriented Architecture and point-to-point topology are examples of this second type.

Rationale: This trade-off is included because the requirements state that insight should be created into the process orchestration and the consequence of a decision on this trade-off facilitates this insight for a large part.

Current situation: Currently the process knowledge is decentralized; components in which processes originate mostly have the orchestration knowledge hidden within them.

5.2.2 Direct connections versus Hub and spoke technique

This twofold is centered on the way of connecting components together. The first is to make ad-hoc connections like those done in the point-to-point and Service Oriented Architecture styles. The second is to create a central entity that handles all connections in a 'man in the middle' fashion.

Rationale: Two of the main requirements state that the amount of coupling should be decreased and the use of technology should be harmonized. The decision in this trade-off presents a starting point to think about how the interactions should work exactly.

Current situation: In the current architecture a hybrid form is adopted. Most connections are direct connections, but the ones through the CustomerPortal Web Service are based on the hub and spoke technique.

5.2.3 Active versus passive component integration

To inform other components that some functions are requested two different forms are supported. The first is the passive approach where the components just wait until they are needed and some function is called on them. The second is the active approach, in which the component monitors a certain location to see if something has to be performed or not. The first approach can be used in any topology, but the second approach has the need for a service bus on which the component can hook in to.

Rationale: From the requirement of creating an independent situation this trade-off comes forward. To reduce the knowledge about other components an active component integration form could be used.

Current situation: In the current architecture, the passive approach is used almost everywhere. The only exception is the queuing of the SharePoint operations in which the Queue Runner polls the database whether new operations need to be executed.

5.2.4 Synchronous versus Asynchronous communication

The trade-off that is presented here are synchronous communication where the results are directly returned to the calling component versus asynchronous communication in which the calling component does not know when the desired functionality is performed. Event-based architectures and Enterprise Service Busses are per definition asynchronous because with both a message is send to a central entity, after which the calling component should just wait for results. Architectures without a message queue use a synchronous form of communication because otherwise method calls will be lost.

Rationale: Asynchronous communication can always be implemented by adding a message queue to a certain component, but when there is a need for synchronous communication the choice of architectures is limited.

Current situation: A lot of components are interacting in a synchronous way, the only exception is with the CustomerPortal SharePoint operations in which the action is added to the queue in a synchronous way, but the execution thereof is performed asynchronously.

5.2.5 Centralized versus decentralized component location knowledge

This trade-off deals with the awareness of component locations. The first form is that all knowledge is centralized. Calling components can query this centralized knowledge base to find out about other components' locations similar to the Service Oriented Architecture registry. The second form is to decentralize the location knowledge by allowing all components to have the knowledge about the locations of components to which call-outs are performed.

Rationale: From the viewpoint of the requirements to create an independent situation and decrease coupling this trade-off was created. By centralizing the location knowledge, the location coupling is decreased by a large amount.

Current situation: Currently, each component contains a hardcoded connection string for all called components.

5.2.6 Open formats versus Limited to standards

The last trade-off is based on the restriction to what programming languages are used in the architecture. The first option is to restrict the components to use the same implementation language, as is done in the middleware approach. The second option is to leave the choice for future components open. This is done in the Service Oriented Architecture and Enterprise Service Bus approaches, due to the use of a standard interface description language the implementation form does not matter.

Rationale: Although Tam Tam is a Microsoft licensed company, the given carte blanche allows other formats if that creates better possibilities.

Current situation: Currently, components are created using different programming languages

5.3 Architectural style decision

To make a decision between the several different forms of Enterprise Application Integration, a score is given to each form and trade-off combination. These scores range from 1 to 5 where a 1 represents the left side of the trade-off continuum and a 5 represents the right side. Values 2 and 4 denote that in principle one side is desired but exceptions may occur and value 3 means that it does not matter, both are equally likely.

The values for the different types are filled in through literature research and the values for the current situation are retrieved by analyzing the current architecture description. Table 3 shows the completely filled in table to be used to form a decision. Once all these values are known, several architects were interviewed to derive the desired functionalities in the future architecture with respect to the individual trade-offs. The values of all architects are merged by averaging the values into a single vector of desired values. This merging was possible because the values for the last five trade-offs did not conflict at a single point. The value for the first trade-off was harder to average because there were two groups of opinions; the first was in favor of keeping the process logic decentralized and the second was strongly in favor of creating a centralized option. This issue was settled by illustrating the advantages and disadvantages of each choice more sharply resulting in a consensus that 'it depends on the requirements'. After consulting the requirements again the most fitting solution for Tam Tam is to centralize the process logic.

Table 3: Trade-off/EAI style valuation

trade-off	Point-to-Point	Middleware - Transactional	Middleware - Message oriented	Middleware - Procedural	Middleware - Object and component	Middleware - Data-access	Event-driven	Service Oriented Architecture	Enterprise Service Bus	Current	Desired
centralized versus decentralized process knowledge	5	1	5	5	5	3	5	5	1	5	**1.67**
direct connections versus hub and spoke technique	1	5	5	1	1	4	5	1	5	2	**4.00**
active versus passive component integration	5	5	4	5	5	5	1	5	3	4	**4.33**
synchronous versus Asynchronous communication	3	1	5	1	1	3	5	3	5	2	**3.67**
centralized versus decentralized component location knowledge	5	1	3	5	1	3	3	1	1	5	**1.33**
open formats versus limited to standards	5	5	1	5	1	4	5	1	1	3	**1.67**

The next step is to match an architectural type to the identified desired values. Table 4 shows a similar table as the one in Table 3 but this table indicates for each type and trade-off pair the absolute distance from the desired values. From the table can be deducted that the Enterprise Service Bus is most similar to the desired situation –with respect to the selected trade-offs- due to the smallest sum of differences. Although it is possible to choose for one certain architectural type, another possibility is to combine ideas of several types (Microsoft Patterns & Practices Team, 2009). Due to the nature of an ESB solution, a combination of ideas is inherent, but careful thought should be given to the possibility of allowing support for transactions.

Table 4: Trade-off/EAI style difference vectors

trade-off	Point-to-Point	Middleware - Transactional	Middleware - Message oriented	Middleware - Procedural	Middleware - Object and component	Middleware - Data-access	Event-driven	Service Oriented Architecture	Enterprise Service Bus	Current
centralized versus decentralized process knowledge	3.33	0.67	3.33	3.33	3.33	1.33	3.33	3.33	0.67	3.33
direct connections versus hub and spoke technique	3.00	1.00	1.00	3.00	3.00	0.00	1.00	3.00	1.00	2.00
active versus passive component integration	0.67	0.67	0.33	0.67	0.67	0.67	3.33	0.67	1.33	0.33
synchronous versus Asynchronous communication	0.67	2.67	1.33	2.67	2.67	0.67	1.33	0.67	1.33	1.67
centralized versus decentralized component location knowledge	3.67	0.33	1.67	3.67	0.33	1.67	1.67	0.33	0.33	3.67
open formats versus limited to standards	3.33	3.33	0.67	3.33	0.67	2.33	3.33	0.67	0.67	1.33
difference desired and EAI type	**14.67**	**8.67**	**8.33**	**16.67**	**10.67**	**6.67**	**14.00**	**8.67**	**5.33**	**12.33**

Although an Enterprise Service Bus fits Tam Tam best, there are quite some differences between the desired situation and a theoretical ESB causing the difference to be unequal to zero (#12). The differences between the desired situation and an ESB are indicated in Table 5, which is reflected upon to derive if an ESB is the true best fit.

#12 One centralized entity brings advantages, but is not free of disadvantages because the desires might differ (a little) from the possibilities

1. Centralized versus decentralized process knowledge does not differ significantly

Table 5: Difference between ESB and desired situation

Trade-off	Enterprise Service Bus	Desired
centralized versus decentralized process knowledge	1	1.67
direct connections versus hub and spoke technique	5	4.00
active versus passive component integration	3	4.33
synchronous versus asynchronous communication	5	3.67
centralized versus decentralized component location knowledge	1	1.33
open formats versus limited to standards	1	1.67

2. Direct connections versus hub and spoke technique is very important because an ESB only offers hub and spoke mechanisms. For this reason the difference should be analyzed with respect to the Tam Tam situation. After indicating this difference to the architects the desire for a central point outweighed this difference.
3. Active versus passive component integration is not significant either because the ESB can fit all desires on that trade-off
4. Synchronous versus asynchronous communication presents a highly significant difference. Although an ESB can offer synchronous communication, it does not offer that functionality in real time because time-outs are introduced when elements fail. This means that very close attention should be given to implementing processes.
5. Centralized versus decentralized component location knowledge does not present a significant difference.
6. Open formats versus limited to standards also does not present a significant difference.

Chapter 6: Future architecture design specifics

Now the surroundings within which the design should be created is in place, in the form of an Enterprise Service Bus, the next step is to decide on specific design decisions. This is performed by first of all retrieving design guidelines and principles from the literature in addition to the general approach defined earlier in Chapter 3:'Approach'. With these rules in mind, the next step is to elaborate on the general idea of the design, which provides a reference for further decisions on the technical part of the design. Not only the technique in place should change, but the organization should also adapt to be able to use the new architecture optimally. This is performed by scrutinizing the necessary IT governance rules. The penultimate step elaborates on the transition towards the designed architecture because it is infeasible to just replace the existing architecture with a newly implemented version. The step following these activities is the third evaluation iteration presented in Appendix E: 'Evaluation iterations' which indicates to what extent the designed architecture improves the current situation.

6.1 Design guidelines and principles

Before rushing into the design, some principles and lessons learned are retrieved from the literature about designing new architectures. These guidelines and principles are used to guide the design process in addition to the elements from the approach elaborated on in section 3.2.

Much literature is devoted to a Meta level of architectural design, (Janssen, Gortmaker, & Wagenaar, 2006), (Ross, 2003), (Hazra, 2002) & (Janssen & van Veenstra, 2005) all state that it is not an option to just design a new architecture and replace the old one with it. The insight these authors share is that an architecture designer should design a roadmap for the transition towards the new architecture too. This transition can take the architecture through several stages, where each author defines another set. A decision is made here to follow the four staged approach presented in (Ross, 2003): 1) application silos, 2) standardized technology, 3) rationalized data and 4) modular. This designed roadmap should satisfy several requirements. (Deepview case study, 2010) states the roadmap should be possible to implement incrementally, should clearly state milestones for developing attributes and should facilitate the co-existence plus non-interference of new attributes with legacy attributes.

Next to guidelines for designing the way of implementing the new architecture, some principles and lessons learned are retrieved from the literature. (Microsoft Patterns & Practices Team, 2009) states that the new architecture should be designed for change instead of designed to last, models and visualizations should be used as communication and collaboration tools and key engineering decisions should be identified to share with stakeholders. (Ross, 2003) adds several more guidelines to this list by stating that the

focus should be on key business processes, no developing stage should be skipped, the architecture should be kept in the improvement loop and the architecture capability should be kept in-house to decrease the costs of maintenance. Throughout the design of services, focus should be on decreased coupling, increased cohesion and a fitting granularity with regard to how much functionality is exposed (Papazoglou & van den Heuvel, 2006b).

6.2 General idea of design

With the stated different approach and the identified guidelines and principles from the literature in mind, the next step is to come up with a suitable design for the internal systems architecture for future use within Tam Tam. The first step in this process is to scaffold the future architecture by sketching the general idea of the design. To perform this, first of all several questions are answered and after that the answers to those are aggregated into design decisions.

6.2.1 Questions to be answered

To initiate the design process of the future Enterprise Service Bus architecture at a rapid pace, several key questions were posed that help make decisions. These questions were derived from the differences between the current architecture and the Enterprise Service Bus architectural style. By answering the stated questions a scaffold for the design is created, causing the remainder of the work to be 'filling in the gaps' and trying to create prototypes to link the theoretical design to feasible implementations. The origins of the answers lie partly in the architectural style and partly in existing implementations of Enterprise Service Busses.

How are functionalities requested?

The first question is focused around the way of requesting functionalities from other components. The chosen architectural style gives a direction for the answer because of the service orientation. Resulting from this is that functionalities should lie in functionally cohesive entities. The requests of these functionalities are to be executed by the service bus. Several existing implementations offer an orchestration engine (Wikipedia Community, Orchestration (Computing), 2010) where the main technology used is BPEL (Alves, et al., 2007)

Which entity knows what components exist where?

In the current architecture, the information regarding the location of a specific component is stored in all locations in which some functionality of said component is requested. In the new situation the locations are to be stored in the service bus. Because the orchestrations are the only place in which direct connections should be used, a logical place to store the locations would be either in the orchestration engine or in a separate registry (i.e. an UDDI server (Wikipedia Community, Universal Description Discovery and Integration, 2010)) attached to the service bus.

How do components know their functionality is needed?

The third question is about how components know their functionality is longed for. From the architects' requests is learned that passive services are the way to go within Tam Tam. To allow both synchronous and asynchronous messaging, instances of the process orchestrations are to be created when they are called. This creates an implicit queuing mechanism when combined with quality of service rules.

Where lays the process logic knowledge?

The answer to this question is already implicitly given in the answer for the first question, with the choice for an orchestration engine. The process orchestrations should contain the logic currently present in the separate components. This logic is to be invoked by publishing the created orchestrations as individual services.

Which entity manages the component statuses?

To be sure the orchestrations work correctly, all called components must be functioning. For the calling applications, the handling of broken components or services should not be an issue. The first mechanism that is designed into the service bus is a 'ping' orchestration which tests all known services for their status. This does not remove all risks of failing services, but does start sounding alarm bells when something is wrong. To overcome all the risks of failing services, quality of service settings are to be defined that –for example– retry calling some functionality when things fail. In the event the set number of retries has passed, a Tam Tam employee needs to check the log files to carry out the requested operation manually.

How are processes monitored?

Monitoring the processes means indicating which processes can be carried out and which cannot. This can be performed by using the management console of a chosen service bus implementation. This console should indicate what instances of orchestrations are currently pending or have failed, giving the administrator the insight into which services or components failed and the ability to restart the orchestrations.

6.2.2 Division of areas of focus

From the answers to the posed questions, several areas of focus have been defined. Figure 7 shows the interactions in the current situation at a conceptual level. In this situation, the process logic and presentation are merged into the applications and the data on which they act is reachable via the network.

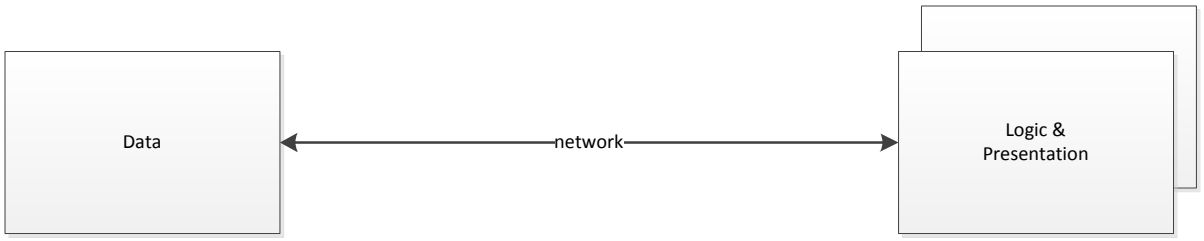

Figure 7: Current situation areas of focus

The idea for the new situation is depicted in Figure 8, the main elements that are changed are: 1) the relocation of the process logic to the service bus, 2) the manner in which functionality is invoked and 3) the way data is exposed to the presentation layer. This new division of areas of focus allows a 'two level programming' approach in which programmers create the offered functionality and non-programmers connect the offered functionality together by creating the process logic. The numbers in the figure represent the sequence of activities that usually take place in processes. The first step (1) is for the presentation layer to request data to be presented to the user. This data is adapted by the user and pushed to the logic in the service bus (2). This logic takes care of updating the data in the right places with or without a delay (3).

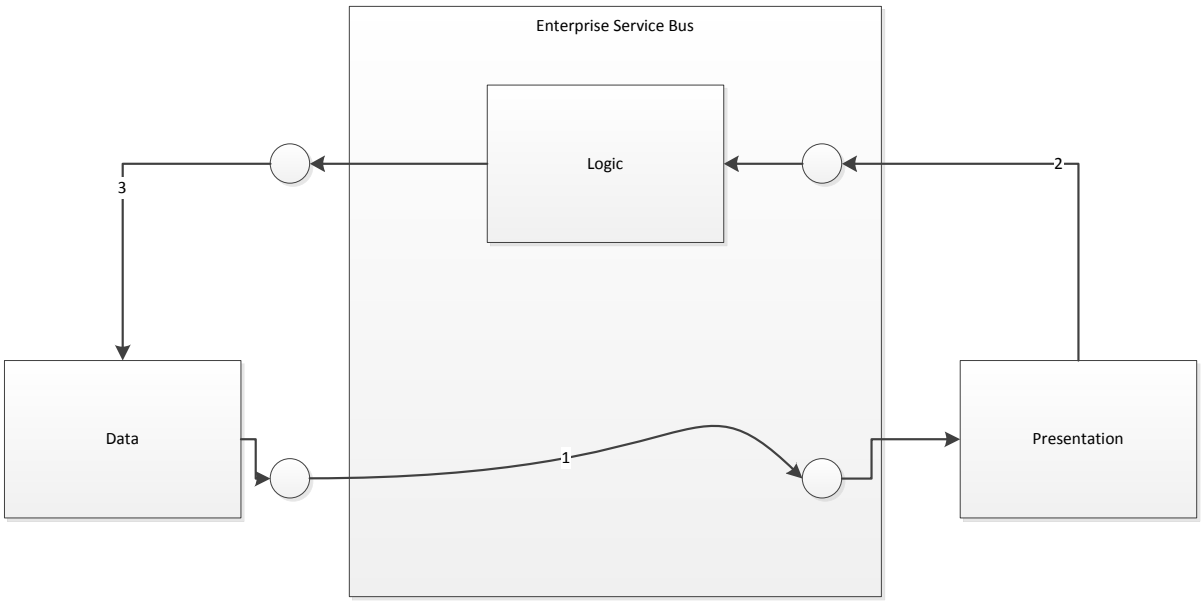

Figure 8: Designed situation areas of focus

6.2.3 Abstraction layers

The designed architecture can be explained by elaborating on several levels of abstraction. The key design decisions of all layers shown in Figure 9 are elaborated on in this paragraph from the bottom to the top.

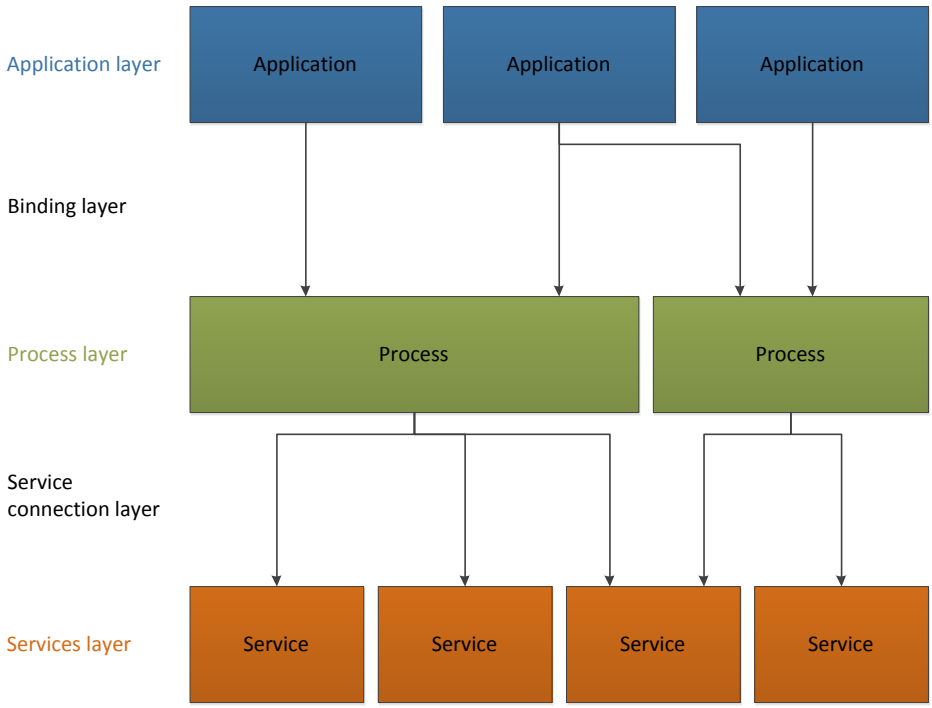

Figure 9: Designed architecture abstraction layers

Services layer

The bottom layer is composed of services that offer cohesively aggregated functionalities. The design for this layer is made by creating a logical division of functionalities into categories (#13). These categories are translated into separate services and offer highly cohesive functions. During implementation, the specific code that should be brought together can be gathered from the functionality requesting applications and web services already in place.

> ***#13 Cohesive services*** *allow splitting up big databases in smaller dedicated versions in an easier fashion*

Service connection layer

To be able to create an abstraction from the provided services, the interfaces of the methods that the services offer are expressed in the form of WSDL documents (Chinnici, Moreau, Ryman, & Weerawarana, 2007). The use of this standard for defining interfaces creates a standardized way for component interaction while, at the same time, not jeopardizing the choice for a specific form of implementation. Furthermore, access management is centralized too by the introduction of the abstraction layer (#14).

> ***#14 Security*** *for databases is improved when dedicated services are introduced as the only place where a connection to databases is possible*

Process layer

The process logic is moved from the specific components into re-usable process orchestrations in this layer. Two main functionalities are found in this layer: the

sequential (or parallel in some cases) execution of steps that form the processes and the knowledge about what services provide which functionality at what locations. Another functionality of this layer is the quality of service aspect on two fronts. The first front is towards the bottom of Figure 9 by setting values for what to do in the case services fail (to respond). The second front is towards the top of the figure to indicate what should happen when multiple applications are invoking the same orchestration at the same time. Decisions for both fronts are usually linked to each other. For example, if a synchronous process needs to be performed, the incoming quality of service values can regulate the instantiation where in other cases the outgoing quality of values are to regulate the sequence of executing.

Binding layer

This is the second layer that acts as an implementation abstraction layer. The orchestrations in the process layer are published to applications as WSDL documents again. This allows the details of the implementation to be changed and enables the use of versioning of processes by keeping the old orchestrations in place (#15).

*#15 **Binding layers** allow components to be easily replaced, as long as the new implementation adheres to the published contract*

Application layer

The top layer of the figure represents the (stripped down versions of) applications in place within Tam Tam for users to interact with. The interactions these applications have with the service bus are twofold. First of all, the data is retrieved from databases through the service bus and secondly the updated or new data is stored by calling the orchestrations in place within the service bus.

6.3 Technical design

After the coarse grained idea of the future architecture has been defined, the next step is to fill in the specifics. The approach followed here is threefold, firstly all processes are categorized to identify which (parts of) processes should be designed how. Secondly the manner of binding processes is laid out and the third step was to decide on key design elements for the future architecture listed in Appendix C: 'Key design elements'.

6.3.1 Categorizing processes

This paragraph categorizes all (parts of) processes in three different categories. First of all are the atomic processes, processes that can execute without user intervention. Secondly, the remaining processes are split into two parts, the data flow part towards the applications and an atomic part. Thirdly all one-way data access parts are identified.

Atomic processes

This category of processes is filled by internal Tam Tam processes that do not require any user intervention to be executed. An example of this is the process of adding a new employee to Tam Tam. After it is initiated, no action from the user is necessary because all data of the new employee is already entered in the HRM application. Table 6 lists all processes in this category. All these processes are modeled as BPEL processes to enable additional steps to be added easily and fault handling functionality to be regulated. Due to the characteristic of BPEL models to be executable, modeling the processes is the first step taken towards an implementation (#16).

> ***#16 Designing with BPEL*** *is concurrently designing and implementing, which prevents infeasibilities during the actual implementation*

Table 6: Atomic processes

Process name	Rationale
1 Generate invoices	
2 New employee	
4 Employee quits	
5 New customer	
6 Rename customer	The data necessary in this process is stored locally in the CRM application
7 New customer contact	
8 Update customer contact	The data necessary in this process is stored locally in the CRM application
11 New opportunity	
12 Update opportunity	The data necessary in this process is stored locally in the CRM application
13 New Project	Can be called in multiple instances; processes 13{a&b} deal with atomic project toolkit instances parts
13a Save order regel mutatie	
13b+16 Save new budget	

Non-Atomic processes

The second step was to add all atomic parts of two-way processes. The processes in this category and their atomic parts are shown in Table 7. These atomic parts are also modeled as BPEL processes just like the previous category.

Table 7: Atomic parts of non-atomic processes

Process name	Atomic part
3 Change employee	Update the employee data in several systems
14 Update project	Update the project details
15 Close project	None, its internals are the same as updating a project
17 Book hours : hours application	Store hours

18 Book hours : Issue Tracker	Store hours
19 book lunch and declarations	Store lunch participation; store declaration
20 Update budget	Store updated budget

One-way data access parts

The non-atomic parts of the non-atomic processes together with all other data request operations fall in the second category that needs to be modeled. The key idea here is that data is just passed through the Enterprise Service Bus. This might seem cumbersome, but it has several advantages. Firstly, using the service bus as a proxy for retrieving data creates a data abstraction. Secondly, triggers can be added to information requests to monitor and audit data access (#17). And thirdly, by using process orchestrations a standardized manner of implementation is forced.

*#17 **Use of orchestrations** for passing function invokes allows triggers to be attached for monitoring purposes*

6.3.2 Binding the processes

The next step of the design is binding and publishing the created BPEL orchestrations in such a way they can be used by the different applications. This step is carried out by loading the BPEL documents into the BPEL-engine of a service bus that takes care of instantiating the orchestrations and the quality of service attributes.

One part of the binding process is specifying the locations of the invoked web services; the other part is defining on what URL the created orchestrations can be reached. Usually this address is specified in the WSDL documents for the orchestrations, but the BPEL-engine should allow connections to the specified URL to be made.

Another part of the binding process is specifying the quality of service values since some steps of processes may only be executed one at a time (#18). This causes the necessity to limit the number of instances in some cases and limit the number of parallel outgoing calls in other cases. Examples of these are the division of synchronous and asynchronous process orchestrations. In the first case, a limit needs to be put on the incoming orchestration calls where in the second case limits need to be put on the outgoing service requests to guarantee the prevention of concurrency.

*#18 **Quality of Service** for orchestrations is important for regulating concurrency and performance penalties*

6.4 Organizational design

Next to a change in the technology field, for the new architecture to be successful, the organization needs to adapt as well to protect the investment by aligning the IT to the business or the other way around (Leenslag, 2006) (#19). In many companies this change is achieved by implementing some sort of IT governance mechanisms. A

correlation was identified between the amount of IT governance awareness and the successfulness of companies (Educause Centre for Applied Research, 2004).

IT governance is a term for activities about which a lot of discussion is going on (Simonsson & Johnson, 2006) & (Brisebois, Boyd, & Shadid, 2007) but an ISO standard has emerged stating what actions company management should perform expressed as 6 key principles (ISO/IEC 28500, 2008).

> ***#19 Technical change*** *on itself is not enough to reach the next level. Changes on various other fields have to be pursued as well*

Some research is devoted to the use of IT governance in large companies (Hardy, 2003) & (Weill, 2004), but the main mechanisms discussed are committees which are given decision rights, input rights and some accountability. To port these approaches to a smaller company like Tam Tam, it might seem overkill to create separate committees like big companies do. But since a good approach is to implement IT governance step-by-step (Rau, 2004), the transition towards a full-blown IT governance structure is started with assigning and charting responsibilities and accountabilities. These initial responsibilities and accountabilities are firstly assigned to individuals, but these individuals may be replaced by committees in the case Tam Tam increases in size.

In essence, IT governance can help make the new architecture a success by enforcing a mentality for the support of the transition and use of the designed future situation. Next to that, when clear accountability is laid out, it can be used to mitigate risks. The key question that will be answered which defines the effectiveness (de Haes & van Grembergen, 2005), and is often missing (PricewaterhouseCoopers, 2006), is 'which entity is allowed to perform what action to collaborate in compliance with the designed future architecture?' This question is posed more delicately by (Weill & Ross, 2004b) through the use of three separate questions: 1) 'What decisions must be made to ensure effective management and use of IT?' 2) 'Who should make these decisions?' and 3) 'How will these decisions be made and monitored?'

6.4.1 Decision domains and roles specification

To find answers to the stated questions, an approach used fairly often in the literature is adopted. This approach entails creating a 'one-page-IT-governance'-matrix (Weill & Woodham, 2002), (Weill & Ross, 2004a), (Weill, 2004) & (Weill & Ross, 2005). Said matrix is constructed by indicating what division of accountability and responsibility are in place in what decision domain.

Because an excess amount of IT governance rules will make IT rigid and might jeopardize creativity and innovation (PricewaterhouseCoopers, 2006) an attempt is made to only centralize a few specific decisions and not go along completely with the sinusoid trend between centralized and decentralized IT governance (Sambamurthy & Zmud, 1999). Most of the elements in the decision domains do not get altered by this project so the data depicted in Table 8 shows how the roles are currently in place or are currently tried to get in place.

Table 8: Decision domains and roles

Domain Archetype	IT Principles	IT Architecture	IT infrastructure Strategies	Business Application Needs	IT Investment
Business Monarchy					
IT Monarchy	CIO			System Admins check	CIO
Federal					
IT Duopoly		Developers implement Business needs			
Feudal					
Anarchy			Developers	Developers propose	
Don't Know					

Table format: © 2003 Massachusetts Institute of Technology, Sloan Center for Information Systems Research

The rows indicate different forms of organizing the location of decisions. The first two are centralized in either business leaders or IT leaders. The federal archetype is reached by letting business representatives and operating groups decide together, and the IT duopoly is reached by a two-party decision making process with IT executives and business leaders combined. Feudal means that unit or process leaders make their own decision based on their needs. The last two are self-explanatory. The columns indicate the domains of the different decisions:

- IT Principles take care of deciding the role IT has in the company and are managed by the CIO, that role either allows or disallows certain actions
- Decisions in the IT Architecture domain are related to new business processes or extension to certain processes and are identified by its users and implemented by the developers
- Decisions based on what functionalities are offered and maintained are performed by developers. They are in charge of keeping the running components, applications and services up-to-date
- Business Application Needs is the domain related to identifying and deciding on what functionality to add to the orchestrations or services. These decisions are made –and changes are implemented– by developers. These actions are audited by the system administrators whether they conform to the current architecture and the rules and guidelines
- Decisions on IT Investments, like what projects to start, are made by the CIO

The degree of success of this division of accountabilities and responsibilities is almost fully dependent on the stakeholder participation and organizational readiness (Rau,

2004). Success factors identified by (PricewaterhouseCoopers, 2006) and (Weill, 2004) include transparency, clear communication and simplicity with clear incentives to live up to the governance rules. This indicates the next step to be taken is identifying clear IT governance rules and guidelines.

6.4.2 Concrete rules and guidelines

Several rules and guidelines have been designed in addition to the technical design to help enforcing the use of the new architecture. Most of these rules are already in place, but stating them makes them transparent and communicable.

CIO decides on projects to be started

There should be one place in which the decision is made for the (kind of) new projects to be started. The underlying reasoning for this is that started new projects all adhere to one perspective on the designed architecture as much as possible.

CIO decides on exceptions to the standard

Another category of decisions that is centralized is the ability to make decisions on exceptions to the current architecture. With these decisions centralized at the CIO in the role of architecture owner, the line of thought behind the newly designed architecture can be enforced (#20).

#20 A man in the Middle is necessary to bridge business and IT efforts

System administrators keep live components up and running and audit changes requested by developers

To enforce the decisions the CIO makes, the system administrators should only comply with a request in the case the CIO has approved said request. This is the way of working at Tam Tam at the moment, but it is listed here for awareness and to improve chance of success of the defined rules.

Developers maintain components

For maintaining the components it is preferable that the original developers are kept involved because they know the ins and outs of those components. This means they are responsible for updating the components and fixing bugs. Key rule here is that the developers are allowed to create extensions to (or new) services, but these are only published by the system administrators after the CIO has given the go-ahead. When the original developers are not available anymore, the Operational Services department can take over the responsibilities because such maintenance lies in their line of work.

First test, and then publish

Because every hour of the system administrators is reimbursed on freelance basis, these hours are very expensive. This means that the amount of work performed by the system administrators should be kept as smal as possible. This results in the fact that to be

published components or services need to be verified and checked thoroughly. This guideline should be kept in mind while performing the maintenance.

Discourage legacy interaction possibilities

To enforce the use of the newly designed manner of component interaction, the legacy interaction possibilities should be deprecated. This should be done both in a technical and non-technical way. The technical way is fairly straightforward by disallowing connections, but the non-technical way should discourage 'workarounds'. This is done by creating awareness among developers that the CIO decides on exceptions to the new architecture. Furthermore, the developers should be kept up-to-date about the current architecture in place and exceptions to it.

Change the way of working

The current way of working within Tam Tam is mostly focused on creating value for customers. Internal projects are sparsely carried out and, when carried out, the focus is to get something working instead of going through a whole waterfall model for example. Change in this mentality is started by defining these rules and guidelines but should gain some momentum by sharing the advantages of the new situation with the developers.

6.5 Transition path

Because it is almost impossible to just implement the new architecture and replace the old version with it (Armour & Kaisler, 2001), a transition plan is necessary that delineates the implementation of the new architecture in different steps (#21). Several steps have been defined that should offer Tam Tam as little downtime as possible. Due to this constraint on minimal downtime, the goal is to put as much overlapping functionality in place as possible. This should allow a flick of a switch to be enough to complete the transition. The following five steps provide the steps to be taken to convert the whole architecture at once. Because the nature of projects within Tam Tam is mainly based on the Agile Scrum methodology these five steps can also be performed in iterations for small groups of processes. The starting point for this transition path is shown in Figure 10, which indicates that there are several components calling some functionality (circles) in other components.

> *#21* ***Entirely replacing architectures*** *is far from feasible, it is important to move with little steps towards the newly designed architecture*

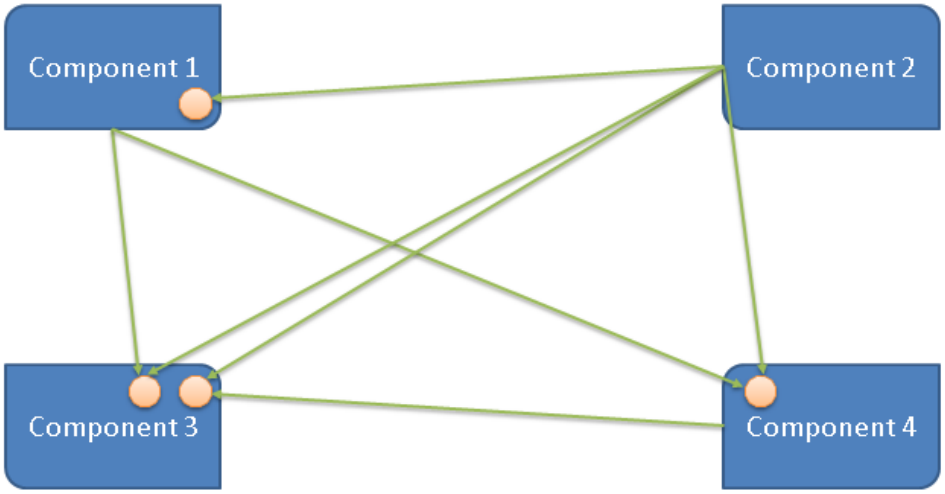

Figure 10: Transition to an ESB: starting point

6.5.1 Step 1: Implement wrapper services

In this step all worker units should be implemented with respect to the designed interfaces. These designed interfaces present contracts to which both sides of the interaction should adhere. As can be seen in Figure 11, this step can be executed without any downtime because the services are additional components in the architectural landscape and do not change existing components or interactions. The specific implementation of service methods can mainly be taken from existing components as these examples indicate:

- Existing process flow information can be split up into different service functionalities
- Database connections can be added to service functionalities by using hardcoded SQL configurations and existing stored procedures as method bodies
- Specific functionality which interacts with components can be replaced from components to the new services

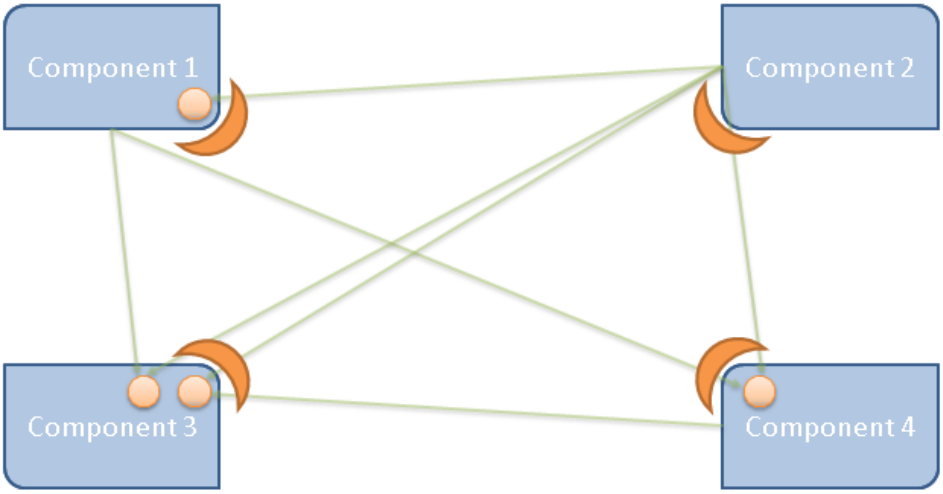

Figure 11: Transition to an ESB: step 1

6.5.2 Step 2: Get an Enterprise Service Bus up and running

The second step is to add an instance of an Enterprise Service Bbus to the architectural landscape. This service bus should contain an orchestration engine with the designed process orchestration documents loaded. Next to that, the service bus instance should have knowledge about the created services in step 1 and thus act as a broker between applications and needed services. The orchestrations in the service bus should already call the correct service functions, as shown in Figure 12. Another required functionality for this service bus is managing the statuses of process orchestrations. This step can be executed without any downtime at all. Key characteristics of this service bus should be:

- Locations of services are managed
- An orchestration engine is in place that takes care of executing the processes
- Component statuses are discoverable
- Process status is retrievable
- Executing orchestrations are made transparent

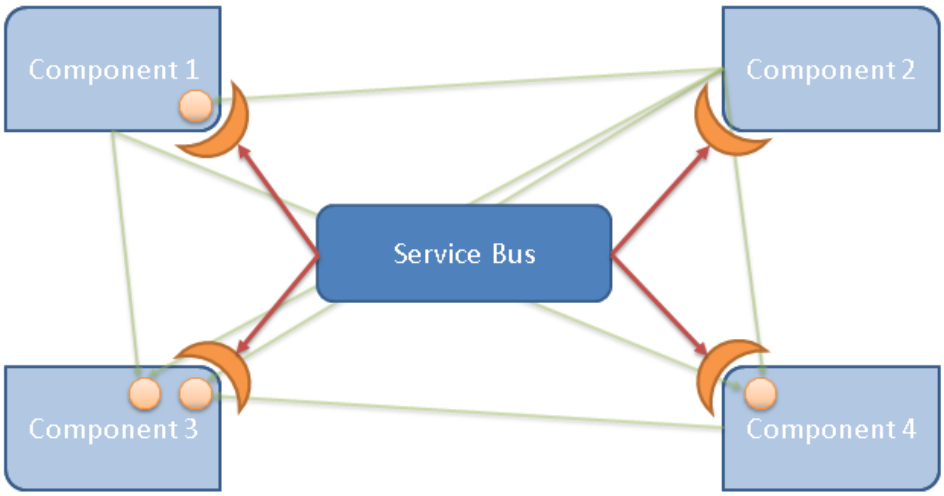

Figure 12: Transition to an ESB: step 2

6.5.3 Step 3: Perform the switch from old to new architecture

In this step the switch is made from the old situation to the new situation by changing the way functionality is called in the separate applications. Some downtime is inherent because some deprecated source code will be removed and the new way of invoking processes will be added. Figure 13 shows this process of fading out old interactions and creating new interactions with the Service bus to replace them. Characteristics of this step are:

- Process requesting applications are taken down one at a time for a short amount of time, implying some processes will stop functioning for a while
- The switch should be made preferably with the original application developer to increase the chance of success
- Legacy direct interaction methods are still possible after this step has been executed so no application stops working when a hidden request is overlooked

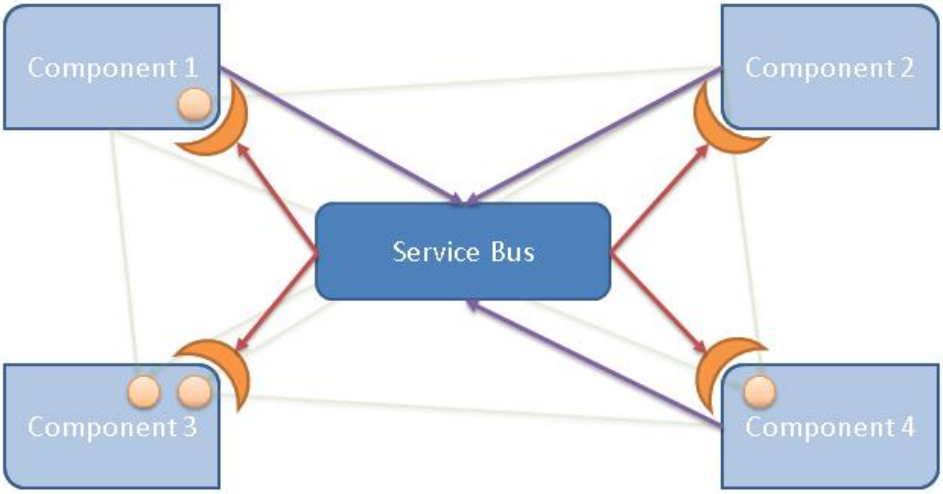

Figure 13: Transition to an ESB: step 3

6.5.4 Step 4: Enforce IT governance mechanisms

To enforce the human-side of the architecture to change as well, some IT governance mechanisms are selected (Section 6.4) to discourage the use of the legacy interaction methods as shown in Figure 14. Next to that, these mechanisms should make the use of the new architecture much easier by plotting which powers lie with what people and in what way the new architecture is allowed to be extended. This step does not bring any downtime into the equation but is one of the more important actions to execute (Webber, 2005). This step is important because the application developers should be aware of the new architecture and should especially know why and how they benefit from the new situation in order to keep using the designed mentality. Key questions are:

- Which entity can make what decisions?
- What procedures are in place to update the architecture?
- What to do with exceptions?

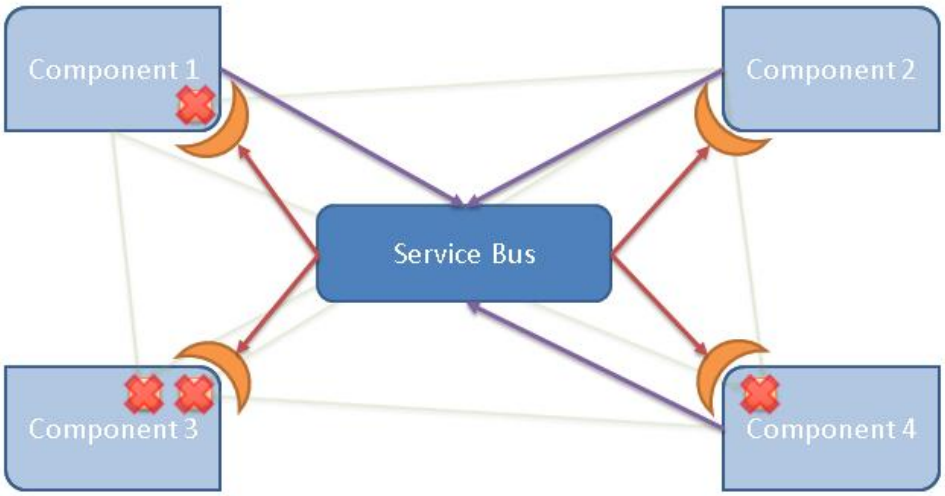

Figure 14: Transition to an ESB: step 4

6.5.5 Step 5: Deprecate legacy interaction methods

To complete the transition to the new architecture, the legacy methods of interactions should be disallowed by disabling them. Figure 15 shows disabling the interaction methods in a conceptual way. Due to the nature of this action, some overlooked hidden method calls might pop up by stopping certain (parts of) processes to work. The timeline of this, however, is very hard to get a grasp of so some activities have been laid out in this step:

- Monitor direct call-ins of components for which a 'service-wrapper' has been created. When some call-ins are still present it becomes apparent that some things have been missed
- This step is executed in a little by little approach so it should be known which action caused a failure. Side note however, is that it might take a very long time for effects to present themselves

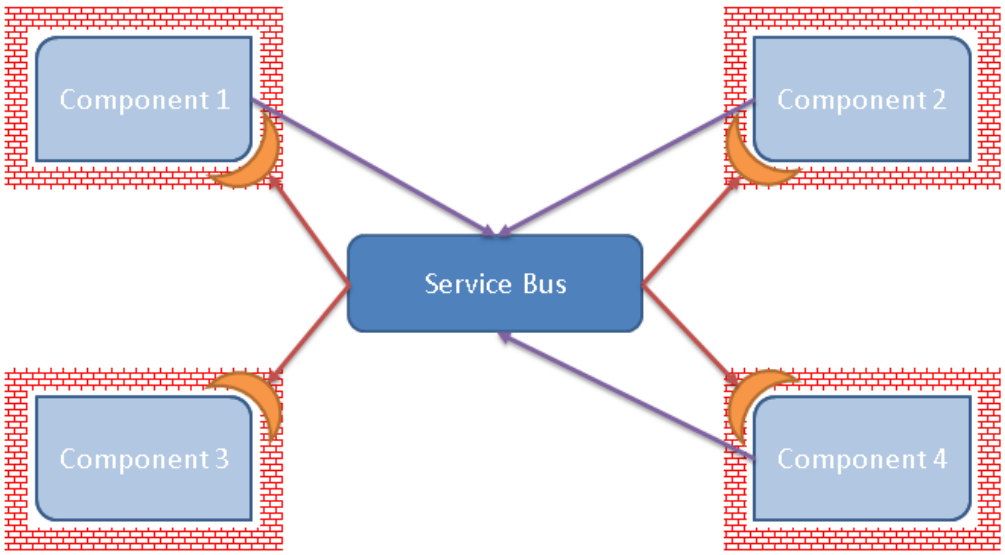

Figure 15: Transition to an ESB: step 5

Part IV:

Practice, linking theory to the real world

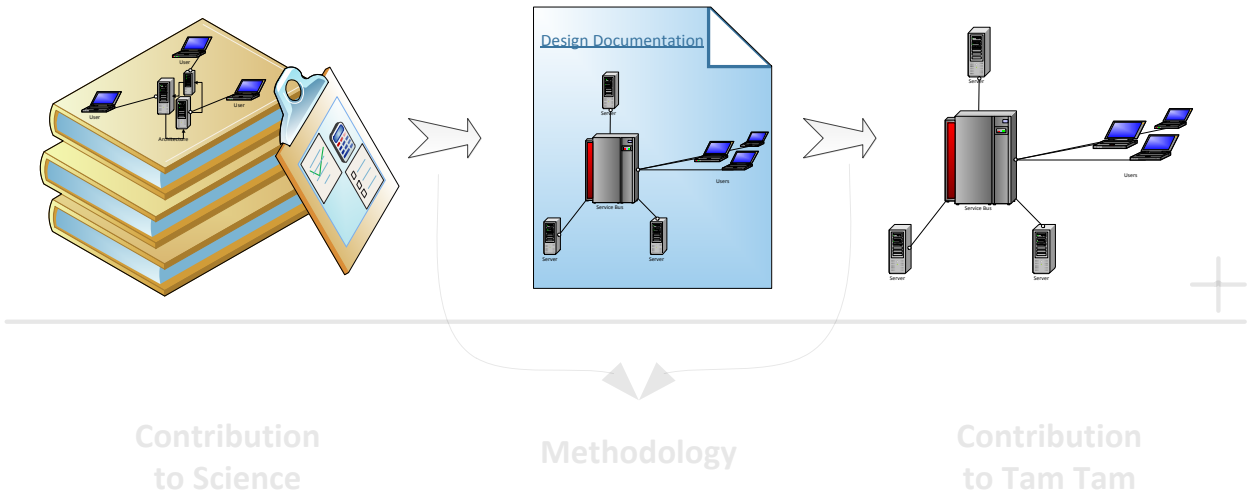

Figure 16: Transforming theoretical design into a real-world architecture

Chapter 7: Implementation

Embodied in linking theory to practice is implementing the designed architecture. This chapter elaborates on the strides taken towards the implementation of the replacement architecture. The first step is to define an implementation strategy by retrieving key insights from the literature. The second step is to make a decision for an available Commercial off-the-shelf product that diminishes the time necessary for implementation by a large sum. When a decision has been made for a specific product, a step backwards is taken to gain an overview of possible issues and to elaborate on everything that makes the decision unjustified. The last step of this chapter is to elaborate on experiences throughout the implementation process.

7.1 Implementation strategy

After the design and the transition path towards the design are created, the next step is to decide on the implementation details. Several options are available for this purpose (Richardson, Jackson, & Dickson, 1990) and are in order of preference: 1) use existing systems, 2) purchase a system, 3) develop internally, or 4) insource from other parties. Guidelines for this process are, as stated by (Hagel & Brown, 2001) and (Hazra, 2002), the focus on re-using existing systems and creating a shared terminology by using industry standards. The last insight comes from (Hagel & Brown, 2001) and states that the implementation should start at the edge where the quick wins can be reached in connecting the architecture with the 'outside-world' (#22).

> ***#22 Implementation starting point*** *should be at non user-interacting processes because quick wins can be reached, for these are in the back-end and no user testing is necessary*

From these insights a key idea for the implementation strategy to be used here is derived. The idea is to gather the necessary building blocks for a minimalistic version of the design as soon as possible. The result of this is to have several proofs of concept using different available products to allow a decision for a specific product. The proof of concept for the chosen product can be seen as a small path towards the designed architecture. Due to the nature of this narrow path, a lot of events can be triggered in the form of additional requirements to hinder the traversal of said narrow path. To overcome the possibilities for such hinder, a risk analysis is performed to identify as many additional requirements as possible and create mitigation plans for those.

Figure 17 shows a graphical overview of the key activities and sequence of activities in the used implementation strategy. The first two steps of creating proofs of concepts and choosing a specific product are elaborated on in the next two sections. The steps after those are performed on individual product basis. The first three individual steps of carrying out a risk analysis, creating mitigation plans and documenting those plans are discussed in the third section. The remaining steps are elaborated on in the fourth section by discussing implementation experiences.

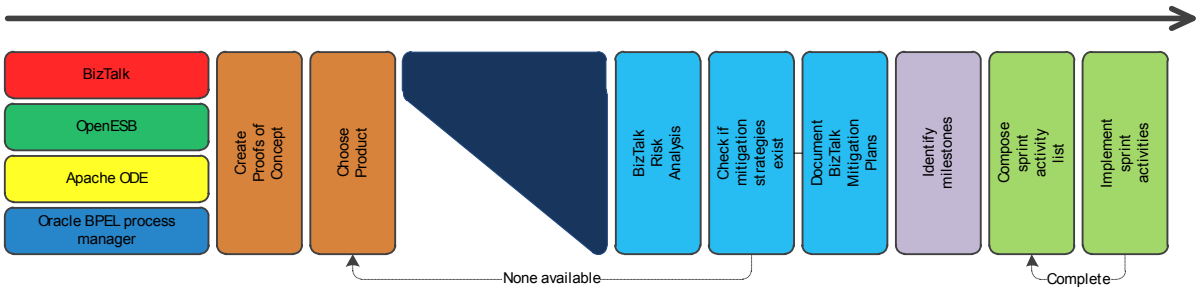

Figure 17: Implementation phases

7.2 Decision for a COTS ESB product

From the literature, the insight is taken to re-use existing systems and purchase the parts that are missing. This is the main reason why a Commercial off-the-shelf (COTS) Enterprise Service Bus product is chosen. Another reason is that using a COTS product a lot of implementation time can be saved. To research which COTS product matches the Tam Tam needs, several steps are taken. The first is to identify the necessary basic building blocks allowing the implementation of a simplistic version of the theoretical design. The second step is to experiment with all the different products to retrieve key values to be used in the actual comparison of the available products in the third step.

7.2.1 Necessary basic building blocks

The goal of experimenting with the variety of products is to check if a set of building blocks is available. Table 9 lists the identified building blocks with a short description and the reasoning behind including them in the basic building blocks list.

Table 9: Basic building blocks

Building block	Description	Reasoning
Create a sample proxy orchestration	A proxy orchestration receives a request, which is passed to a web service. The results are relayed back to the requester again	Once this building block is available, all designed orchestrations can be implemented because they are sequences of web service invoking operations
Conditional branching	Several different paths through orchestrations should be possible	The processing of some designed orchestrations proceed in different paths depending on some value
Publish orchestration as web service	Implemented orchestrations need to be published to client applications, this is preferably done by using web services	Because one of the foundations of the designed architecture is web services, the ability to publish orchestrations as web services standardizes the use of technology

Create an asynchronous orchestration	After an orchestration has send its response, it should not stop performing operations	These orchestration instances can be used to emulate a queuing mechanism for slow web service functionalities
Set binding properties	The binding properties define how the system reacts if multiple requests come in at the same time	In the designed architecture some operations should not be allowed to be executed concurrently

7.2.2 Experiment with products

This paragraph describes the process of getting the basic building blocks to work on several products in a chronological way. A start was made with one product and from that product the disadvantages were covered by moving to other products. The choice for different products was very limited due to the nature of the design. The main criteria the product should comply with are: 1) a visual workflow-like process orchestration development environment should be present, and 2) the system should maintain on-premise for security motivations.

BizTalk attempt

Despite the given carte blanche, the choice for the first product under focus was made because of Tam Tam is a Microsoft affiliated company and there exists a lot of Microsoft product knowledge within Tam Tam. The first product under focus was Microsoft BizTalk 2010[4]. After the request was made, the system administrators of Tam Tam were able to get an instance running in about one working day. Due to the fluent integration with Visual Studio 2010, IIS 7 and Windows Server 2008 R2, the time it took to get a simple file-handling process orchestration working was very limited. Another advantage of BizTalk is that a lot of tutorials are available for creating such a file-handling orchestration.

When the complication is added to invoke a web service from the created orchestrations, the steep learning curve obstructed further progress. This resulted in a failed attempt after struggling from error to error for several days. Because there was no hindsight into how many more errors needed to be overcome, the decision was made to broaden the scope.

Open source

This broadened scope resulted in the quest for open source alternatives. One product came forward as an excellent and complete package: OpenESB[5]. The installation went prosperously and getting a first sample proxy orchestration running was very easily

[4] http://www.microsoft.com/biztalk/en/us/2010launch.aspx
[5] http://open-esb.java.net/ but was taken offline by Oracle, community continued the project at http://openesb-dev.org/

performed with the included tutorial. More advanced creations could also easily be created by using the visual designer. After a few days of work almost all identified processes were implemented in a proof-of -concept manner by using web services stubs. Furthermore, the created proof of concept orchestrations could be published as WSDL binding documents.

The only, quite major, disadvantage of OpenESB is that it is not supported anymore at the moment of the decision. The reason for this is that OpenESB is built using the Java language of Sun Microsystems. Since the acquisition of Sun by Oracle, the future of OpenESB was cancelled because Oracle sells its own ESB solution.

Proprietary

Due to the ease of which the orchestrations in OpenESB were created, the logical next step was to experiment with the product offered by Oracle: Oracle BPEL process manager[6]. The first step was the installation of the manager; this succeeded fairly quickly and was up and running in about an hour. The downside of this product is that if orchestrations have to be developed visually, another package has to be downloaded and installed: Oracle JDeveloper[7]. The installation of this second product was also performed in about an hour. The process of getting the two products to work together was not finished at all after several days of attempting to link both.

Open source revisited

When the proprietary approach ended in an unsatisfying way, the fourth approach was to search for an open source alternative that is still supported. The amount of options was diminishing fast but one open source option that was still supported and recently updated was Apache ODE[8]. ODE is a web application running in an Apache Tomcat installation. The installation of ODE was performed by simply placing a compressed archive in the correct Tomcat directory. After ODE was up and running, the next step was to install a visual design application, Eclipse. The installation and linking of Eclipse to the running ODE instance was performed quite easily due to the tutorials included with ODE.

The creation of orchestrations can mostly be performed using a visual editor but some elements have to be coded manually in the BPEL and WSDL documents. Another major disadvantage of ODE is that it only offers extensions to client functionalities, not a fully functional service bus. This means that clients are solely responsible for recovering orchestrations in case things fail and no quality of service or queuing options are available.

[6] http://www.oracle.com/technetwork/middleware/bpel/overview/index.html
[7] http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html
[8] http://ode.apache.org/

BizTalk attempt revisited

Because the alternative products included in this study do not match the expectations of the theoretical design, a final attempt was made to get a sample proxy orchestration working in BizTalk 2010. After another day of trying to include a service reference in orchestrations, a first little step of success was achieved. After a sample proxy could be created, the other basic building blocks followed rapidly because those other requirements can be linked to one of the strengths of BizTalk: 'Connecting different systems'.

7.2.3 Decision for a product

After the necessary building blocks have been gathered successfully for three different products, a decision should be made regarding which product to use. Table 10 lists the key characteristics of the four different products. The choice is made using the cut-off method described in (Bots, 2002) by formulating a set of hard demands on the packages. These hard demands used for the elimination process are based on the requirements posed by Tam Tam:

1) The product should still be supported; meaning that in the case a bug comes forward there is a realistic chance it gets fixed
2) It should cost no money; Tam Tam does not have the financial resources to pay for an expensive Enterprise Service Bus product

When these requirements are put forward one at a time, the first requirement eliminates the almost perfectly suited OpenESB. The second requirement, and the fact that it could not be gotten to work, eliminates the Oracle proprietary alternative to OpenESB. To make a decision between ODE and BizTalk, elements from the architectural design were brought forward again and they pointed in favor of BizTalk because the lack of fault tolerance capabilities in ODE.

Table 10: Comparison of COTS products

	OpenESB	apache ODE + Eclipse designer	Microsoft BizTalk 2010	oracle BPEL process manager + JDeveloper designer
configuration time	1 hour	1 hour	8 hours	8 hours
learning curve	flat	medium	steep	very steep
time to get simple proxy orchestration running	1 hour	1 hour	1 week	unknown
effort from simple proxy to full blown processes	little	medium	little	unknown
support available	none	medium	perfect	perfect
last update date	Q4 2008	Q2 2010	Q4 2010	Q2 2010
price	open source	open source	0, Microsoft partner	50k / CPU

monitoring capabilities	none	only number of instances	full blown	full blown
concurrency possibilities	perfect	none	Very good	unknown
binding options	many	very few	medium	unknown
Delivery guarantee	until something fails	until something fails	always	always

7.3 Further evaluation and assessment of the choice

With the basic building blocks available for BizTalk, the designed architecture can be implemented. The problem however, is that the implementation resulting from this way of working is very simplistic and every additional requirement brings a lot of uncertainty forward. Such situations are indicated by (Meredith & Mantel, 2009) as "the natural inclination of the customer to change the deliverables as they obtain better information about their needs over time" or simply as "scope creep". Scope creep is one of the risks that comes forward in a lot of software risk assessments, mostly it comes directly after a lack of commitment and involvement of stakeholders in the project (Keil, Cule, Lyytinen, & Schmidt, 1998), (Boehm & Port, 2001), (Schmidt, Lyytinen, Keil, & Cule, 2001), (Mazumder, 2006) & (Boehm B. , 1991).

Although there is a 'do not ask do not tell' mentality with software engineers to limit the legal accountability when something goes wrong (Boehm & DeMarco, 1997), the approach taken here is by using the heuristic stated by (Boehm & Port, 2001) to "resolve risks early to avoid extensive late rework". By holding several brainstorm sessions with stakeholders and especially non-stakeholders, a list of elements was identified that might jeopardize the correctness of the choice for BizTalk. The identified elements grouped per category together with a short description and the corresponding mitigation plans are listed in Appendix D: 'BizTalk risk mitigation plans'. The mitigation plans have been defined by implementing proof of concepts for all features longed for.

Because risk mitigation plans have been created for all identified risks, the decision for using BizTalk does not have to be revisited. Furthermore with respect to overhead, every service bus product does introduce some overhead. The overhead introduced with BizTalk, however, is not significant enough to revisit the decision as shown in the second section of Appendix D: 'BizTalk risk mitigation plans'.

7.4 Reporting experiences with implementation

Part of the design process is creating a proof of concept for all requirements, linking them together is performed in the implementation process discussed in this section. The first paragraph discusses the first few implementation steps taken towards the designed architecture. The next step is to use the aggregated working proof of concepts as an input for a go / no-go decision for taking the design into production. The third step is to prepare the production environment to be used as defined in the design. In the last step of this section an encountered opportunity to quickly proceed with the implementation is discussed. The progress of the implementation is depicted in Appendix G: 'Progress of

implementation', wherein 'partly implemented' means either half the designed functionalities are in place or only a proxy for functionalities is in place.

7.4.1 Separate Processes

During the work towards the go / no-go decision two different processes have been ported to BizTalk. The first was the process of creating new projects for customers, for which the first three steps of the transition path were carried out. The design had already cut the process into different atomic steps, which were easily translated into WCF service functionalities. The orchestration had to be recreated similar to the BPEL design because Microsoft uses its own definition of BPEL in BizTalk. Once the functionality was in place, the switch was made to test the BizTalk version in the production environment. After some successful tests, the original functionalities were restored again because of the development nature of the used servers.

The second ported process is used for creating new Tam Tam user accounts. The implementation of this orchestration was fairly straightforward from the design again. The underlying functionalities however, requested very specific knowledge of the Active Directory and Exchange servers. Due to a lack of such knowledge, progression was very hard to achieve. Luckily the functionalities already lie under web services so could easily be invoked. Although this workaround performs the job, the help of an expert should be included during the actual implementation to split the functionalities (#23).

> ***#23 Expert knowledge*** *is wise to be called in during the implementation to reduce effort necessary to master the specific knowledge necessary*

Several other problems were encountered which could more easily be solved. The first is that in the case a service is extended, the WSDL does not necessarily need to be reloaded in calling applications when the interfaces of the used functions do not change. Secondly, namespaces are hard to work with: once two orchestrations use the same namespace, very strange undesired functionality is the result. The last problem is the fact that due to the message based nature of BizTalk, no void parameters can be used to trick BizTalk into carrying out pre-defined steps

7.4.2 Go / no-go decision

With the transition of several processes, the next step is to make a go / no-go decision with several stakeholders and external experts. The next chapter presents all aggregated information used for the go / no-go decision in the form of scenario analysis, means valuation and expert reviews. The result of this aggregation was that Tam Tam wants to proceed with the implementation of the design created in this thesis.

7.4.3 Preparing production environment

After a go-ahead for implementing the design in the production environment, several steps were executed. The first was to clearly document a list of implementation milestones and necessary atomic tasks in the form of a product backlog. The main focus

of listing these tasks is clarity, because in such a way the transition path towards the new architecture can be carried out autonomously. Furthermore, this focus on clarity causes transparency in the implementation process.

Secondly the production BizTalk environment was set up in which the design could be implemented. This entailed porting all customizations made in the development environment to the production BizTalk server. These customizations range from importing the known low-level services in the UDDI server to establishing new Single sign-on user mappings.

Thirdly, several documents and example files have been created for sharing knowledge, all of which are listed in Appendix F: 'Key deliverables'. The documents that have been created are mainly the implementation plan and several different tutorials explaining how to perform certain actions. These tutorials present the 'missing links' of knowledge encountered during the design and implementation and strive to flatten the BizTalk learning curve. The example files are partly created by following tutorials but are provided to serve as a reference point or as a starting point for further implementation.

7.4.4 Subsidiary company

During the implementation, a new spin-off company was founded by Tam Tam. This new company should benefit from the same automated back office of Tam Tam by being integrated into the existing environment. The integration should be performed in such a way it can easily be split up again when the two companies are to be separated. Due to the insight into the currently in place architecture and the ongoing transition towards a new architecture, this task was prioritized in the implementation backlog. This request for additional functionality presented an ideal means to assess the created design.

Due to the focus on the capability to split, the components under focus should be assessed. The Active Directory and customer portal SharePoint can easily be re-used, and a database layer is already put over the administration database duplicating all entries and adding a corporation ID to the unique keys indicating one of both companies. The CRM in place however, has to be duplicated to create two separate CRM entities for the separate companies. This request for an additional CRM instance causes a need for some functionality to be disabled and some functionality to be added.

Because of the very strict planning for the addition of functionalities and components, the step of creating complete orchestrations in the transition path is skipped. The implemented parts act as a starting point for creating the fine-grained orchestrations. Because the interface of the orchestrations is already in place, only the underlying functionality has to be replaced.

Putting the adapted elements in production succeeded without many hurdles along the way, only some minor inconsistencies between the production environment and the coded functionalities were encountered. None of these issues persisted, so the first (parts of) processes are successfully using BizTalk as a communication means.

Chapter 8: Evaluation of results

The next step, after the design is completed and key elements of it are implemented, is to evaluate the newly designed architecture with the defined means and metrics. The contents of this chapter were used to decide on a go/no-go live implementation decision. In this chapter, the same evaluation steps are executed as in earlier evaluation iterations, with two additional steps. The first added step is rounding up all advantages and disadvantages of the new architecture by aggregating the results of the four evaluation iteration steps. The second added step is holding expert review sessions with several stakeholders who were not previously involved in the project to sustain their objectivity. With this aggregated evaluation in hand, a go/no-go decision could be made by creating a balance sheet and making a deliberate choice.

8.1 Aggregated evaluation iterations results

During the course of this thesis project, several evaluation iterations have been conducted at four key points throughout the project. The results of these for iterations is listed in Appendix E: 'Evaluation iterations'. The fourth of the following instances is the most elaborate and is conducted in preparation for the aggregation of advantages and disadvantages and the expert reviews of the next sections:

- Iteration 1: After insight has been gained into the current situation
- Iteration 2: After choosing the architectural style
- Iteration 3: After the design specifics have been laid out
- Iteration 4: After prototype implementation

The content of these iterations is primarily focused on conducting scenario analyses and metrics valuation. The results of all instances of these metric valuation sessions is depicted in Table 11 and represents quantitative measures for evaluating the progression achieved with the new architecture.

Table 11: Aggregated evaluation iteration results

Metric	Current situation	Enterprise Service Bus	ESB Design	BizTalk 2010
Number of interaction schemes	6	1 (+ interaction schemes in dedicated wrappers)	1 (+ interaction schemes in dedicated wrappers)	2 (+ interaction schemes in dedicated wrappers)
Number of hardcoded connection strings	21	Number of services (+ connection strings in dedicated services)	9 (+ connection strings in dedicated services)	34 + 1 for UDDI
Number of incoming method calls per	2.2	2.4	2.4	2.8 (+ direct data requests/number

component				of services)
Number of components with related functionality (duplication)	8	0	0	0
Speed of source code retrieval	2 hours	15 minutes	15 minutes	15 minutes
Speed of live location retrieval	45 minutes	15 minutes	15 minutes	5 minutes
Speed of understanding application interactions	Several hours	15 minutes finding + 30 minutes understanding	15 minutes finding + 30 minutes understanding	5 minutes finding + 30 minutes understanding
Speed of understanding process orchestrations	Multiple hours	2 hours	30 minutes	30 minutes
Speed of checking component status	Multiple hours	30 minutes	5 minutes	15 minutes
Speed of understanding component purposes	Several hours	1 hour	1 hour	1 hour

8.2 Benefits and drawbacks of created architecture

The values identified for the key metrics and the scenario analysis of the evaluation iterations show that the designed situation can have quite a few advantages. While leaping forward on one field, it is imminent that some disadvantages are present at other fields. These identified changes, summarized in Table 12, come from a variety of categories: from purely technical through business to changes for individuals. This section contains an elaboration on all identified advantages in the first paragraph and an elaboration on the disadvantages in the second paragraph.

Table 12: Identified advantages and disadvantages

Advantages	Disadvantages
Time consumption	Overhead of data requests
Removal of duplicate functionality by creating abstractions	Difficult to hide certain functionalities
Auditing and monitoring of processes	Changed way of working
Division of needed personnel	
Versioning	
Increased manageability	
Expose functionality for external use	
Security can be managed in one place	

8.2.1 Advantages

Time consumption

From the scenario analysis and key metrics it becomes clear that a lot of time can be saved with the new situation due to improved clarity (#24). Although some time is needed for some transition struggles towards the designed situation, in the long run, the amount of time saved will most likely be positive as indicated by most scenarios and the evaluation metrics.

> ***#24 Business requests are more easily achieved*** *because of the flexible nature of the design. In the case new requirements are posed, these can be quickly implemented*

Removal of duplicate functionality by creating abstractions

The introduction of several layers of abstraction removes some duplicate functionality. This can be attributed to the clear definition of contracts for interactions. By specifying exactly what function should perform what functionality, general re-usable functions can be created limiting the sensitivity for code (and functionality) duplication.

Auditing and monitoring of processes

The introduction of specific process orchestrations for all identified processes introduces the opportunity to audit and monitor all processes. This entails creating the opportunity to check how often a certain process is executed and by whom, allowing for more elaborate business intelligence activities. Next to that, the opportunity arises to identify points of failure in case the process does not execute properly.

Division of needed personnel

By creating strict interfaces for component interactions, no programmers are needed to create orchestrations. Non-programmers are able to craft and adapt business processes by using process orchestrations. The use of process orchestrations reduces the burden of understanding processes because they can be interpreted and created in a similar way to creating 'flow'-diagrams. The only task remaining for programmers is implementing the low level services with respect to the identified contracts and adapting applications to invoke the created process orchestrations.

Versioning

Another advantage, which is introduced by using centralized process orchestrations and specified interactions contracts, is allowing the use of versioning. Before new versions of process orchestrations and functions replace the currently in place versions, the old versions can still be kept active. This causes no processes to be taken offline when a new version should replace the existing one. Other layers that have the advantage of versioning are the identified contracts interface layers. In the case new functions are added to the interface, elements using the old functions do not have to reload the new contract specification document to keep on functioning.

Increased manageability

By creating several abstraction layers it becomes clear what functionality can be found in each layer. This decreases the effort necessary for understanding which component serves what purpose and provides guidelines for making several decisions while developing new components.

Expose functionality for external use

Next to advantages that play inside Tam Tam, the newly designed service bus can more easily expose functionalities to be used by third parties. In the current situation it is very hard to specify what functionality should be accessible by external entities. In the designed situation however, all aggregated functionality is accessed through the service bus allowing specific functionalities to be exposed to external entities.

Security can be managed in one place

In the current situation, access to components can be limited by configuring each component separately. This is used currently, but there are some credentials 'floating around' Tam Tam which have super-user rights to a lot of systems. Recently, a move can be identified towards a managed security organization. The design in this project helps reaching this goal by including a Single sign-on environment. This SSO environment causes access rights to be configured in one place (BizTalk server) and managed in another (Active Directory). The centralization also increases the ease with which security considerations can be implemented.

8.2.2 Disadvantages

Overhead of data requests

The idea of the designed solution was to relay all data requests through the service bus. Although this brings some advantages like monitoring and auditing, it also creates quite some overhead as can be seen in Appendix D: 'BizTalk risk mitigation plans'. Due to this, the choice is made for the implemented solution to only create an abstraction layer in the form of web service functions. The requests to these are made directly to the web services instead of being relayed through the service bus. A disadvantage of performing requests in this way is that it becomes harder to monitor and audit data requests.

Difficult to hide certain functionalities

The new situation makes it more difficult to hide functionalities because most functionality is performed through the service bus. In the current situation some functionality may be hidden by using obscurity, achieving the result that when no one knows about some functionality they will most certainly not use it. It might be necessary to limit the usage of certain functionalities, so this should be enforced by explicitly limiting access to certain applications or entities for example by using the Single sign-on functionality introduced.

Changed way of working

In the current situation, very little attention is given to the architectural landscape and a lot of requested functionalities are added in such a way they just perform the job. The new situation however, requests more careful thought about where to place the extended functionality and how its interactions take place. This forces a change in the way of working of developers and thus may cause some resistance.

8.3 Expert review

The last step of the thorough evaluation is to evaluate the design with external experts. The term 'external' in this context indicates Tam Tam employees who have not been involved in any part of the process of getting to the architectural design. This external status of experts causes their opinion to be unbiased and objective. The nature of the held interviews consisted mainly of verifying the results of the latest scenario analysis iteration. To reach a thorough evaluation of scenarios, the first step in the interview was to introduce the experts to the frame of reference used throughout the project. With that knowledge in place, the next step was to elaborate more on the current architecture and provide a clear image of the decisions and details of the designed architecture. The results of all interviews are elaborated on and after that some invaluable insights are aggregated in the last paragraph of this section.

8.3.1 Head of Operational Services

The first interviewed expert is the head of the operational services division of Tam Tam, the division responsible for maintaining customer applications that are in production. Due to this maintenance role, the expert has a clear view on maintainability of others' code and design; providing an excellent input for reflection.

On the technical perspective of the design, it is foreseen that the centralization of knowledge presents the biggest gain in saved time because everything can be performed at a single place. An additional advantage is that the system administration department can quickly assess what is going wrong, whereas in the current situation they have no handles like this. Although this is favorable, just like all other advantages and disadvantages, this has a turn side because the single point of contact also introduces a single point of failure. The last insight from the technical perspective is that a golden mean is longed for between a higgledy-piggledy architecture and an extremely solid and robust architecture. The main reasons for this are that the architecture is for internal use and should be easily maintainable.

From the organizational perspective some change is needed as well. The major change that needs to be implemented is to discourage 'working around' the new architecture. This could be reached by using technical efforts like correctly applying the UDDI registry or documenting specific know-how. Efforts from other categories consist of taking care of

*#25 **Budgets should be available** to enable knowledge retrieving and knowledge sharing*

sufficient budgets for implementation and knowledge retrieving and sharing (#25). Although these additional budgets might seem odd, BizTalk knowledge forms a valuable addition to Tam Tam because it has occurred several times customer BizTalk requests had to be declined by Tam Tam.

Additional key focal points are 1) the amount of BizTalk customization for that has a direct influence on the ability to upgrade or replace the service bus in the future, 2) how error notification is arranged because some processes need direct feedback when steps fail, and 3) the ability for rollback mechanisms.

8.3.2 Application developer

The second interviewed expert is an application developer who has developed quite a lot of applications that are in place in the current architecture. This developer already has some insights into the inner workings of the current architecture and thus has a clear standpoint for evaluating the new architecture.

A certain amount of skepticism surrounded the advantage of less needed time because of the seemingly cumbersome extra steps that need to be performed while carrying out otherwise simple actions. This can be overcome by creating clear step-by-step illustrated tutorials for all possible scenarios. The most time winning part will not be the implementation part, but the process of knowing what to implement. This advantage comes from the created clarity; it is known exactly what has to be changed when and where.

On a technical frontier, the division of data and logic of processes is very wise, given the fact that a lot of Tam Tam projects use a Model-View-Controller mechanism (#26). Careful attention should be given to data request functionalities. The implemented request functionalities should be as generic as possible, so instead of 'retrieve all projects on which developer x has worked half a day', methods like 'retrieve all projects', 'retrieve all customers' or 'retrieve all employees' should be used. Next to that, the use of workflow-based processes is a giant leap forward because it brings the advantage of centralization. Windows Workflow Foundation came up as an alternative, but was eliminated because it is a framework instead of architecture. This framework also allows decentralized solutions, limiting the advantages of centralization. The advantage of division of personnel is thought to be unnoticeable because, in the end, developers are the employees responsible for maintaining the architecture and processes.

> ***#26 Enterprise Service Busses*** *introduce a data layer and a process-layer similar to that of the Model-View-Controller programming paradigm*

On the organizational frontier, several key characteristics have been identified. These are mainly focused on knowledge, architectural ownership and standards. With respect to knowledge, the know-how with BizTalk should be kept available within Tam Tam even when employees stop working for Tam Tam. A lot of effort should be put in reducing the bus-factor of the new architecture. Furthermore, an architecture owner should be commissioned because BizTalk development should be 1) in line with developer capabilities, 2) understood completely, and 3) performed often to maintain agility. The last organizational aspect is that before the implementation starts, a clear naming convention should be devised. The rationale behind this is that the service bus forms the center of the architecture around which everything is build and should thus include clear conventions (#27).

> *#27 **Naming conventions** are very important with web services because this prevents creation of duplicate functionalities and offers users insight into the offered functionalities*

8.3.3 Developer

The third expert is a Tam Tam developer with no affiliation with the current architecture in place or the designed architecture whatsoever. This expert provides major insights into a future where responsibilities should possibly be transferred to other (new) employees.

To transfer the architecture into a workflow-based system is a very good idea. Although Windows Workflow Foundation (WF) is an alternative for reaching this goal, it is more prone to development efforts so it takes more time to implement with that technology. Furthermore, due to the fact that WF allows decentralization, the benefits of the centralized workflows are diminished when that approach is taken.

From an organizational perspective, a lot of people should become aware that the UDDI and BizTalk servers are in place so that no workarounds will be created. This mentality change is the most important organizational aspect because of the culture surrounding internal projects; these should be finished quickly and effectively. Next to a mentality change, time is a very important aspect. Due to the nature of the changed way of working, quite some time needs to be invested by developers to master the new situation. As long as Tam Tam offers such an opportunity, employees can start experimenting with BizTalk. The most advantageous situation is when new components need to be created; when components need to be adapted the threshold to use BizTalk is fairly high. Concluding, coping with legacy components is very important and because the design crafted in this project is only the first iteration of a successful effort, it is hard to justify the effort that needs to be put into it by developers.

8.3.4 Aggregated insights

The take-away from the different expert review sessions is elaborated on in this paragraph, first of all the major insights from the different expert sessions are

aggregated. Secondly, the results of investigating some leads that came forward from the interviews are presented.

Exit strategy

To keep the architecture flexible, the chosen Enterprise Service Bus implementation should be replaceable by another ESB or by a completely different architectural style. Due to this, the amount of customization of BizTalk is very important. The designed abstraction layers are causing components to only have knowledge about binding information to a service bus. This binding is in the form of a web service, thus providing the possibility to replace the BizTalk implementation by just replacing BizTalk and mapping the web services to the newly in place architectural style.

Organizational change

Next to technically changing the architecture in place, it is very important to also change some organizational aspects next to the identified responsibilities and decision rights. The major change should be a mentality change because developers should start using the Enterprise Service Bus and not work around that. Next to that, budgets need to be made available for knowledge management around BizTalk. Because BizTalk is a fairly complicated product, a lot of effort needs to be put in maintaining a 'product owner' and keeping knowledge available within Tam Tam.

Advantages and disadvantages

The first insight that comes forward from the experts in the category advantages and disadvantages is that every identified advantage can be translated into a disadvantage and vice versa (#28). From the interviews, the workflow-based approach came forward as an additional advantage. A new disadvantage is the single point of failure introduced by using an Enterprise Service Bus.

> ***#28 Advantages** always come with counter parting disadvantages and vise versa*

Leads from interviews

Some leads are identified that needed some further research. First of all, rollback mechanisms should be possible because otherwise when processes fail it is hard to retry because some steps can already be carried out successfully. BizTalk does provide this functionality by offering a 'Compensation' building block for process orchestrations, which allows undo functionality to be invoked. Secondly, the Windows Workflow Foundation framework[9] was mentioned but due to the decentralization possibilities it was no competitor for BizTalk in the context of this project. Thirdly, it might be advantageous to host several versions of the same web service on the same location. This can be performed by adding new bindings to the configuration of web services. Lastly, some error notification mechanisms need to be in place for alerting system

[9] http://msdn.microsoft.com/en-us/netframework/aa663328

administrators. This is available in BizTalk in the form of monitoring capabilities, which allow a complete overview of all stopped and currently running elements in BizTalk by the use of a database with all events in it.

Preliminary steps before implementation

Before the new architecture is to be implemented, a few elements have to be formalized. This entails creating a naming scheme, creating transferable know-how and specifying exactly what instances of what component have to run where.

Chapter 9: Evaluation of process

Thus far, most of this thesis was focused on the way of achieving results. The next step is to take a step backwards and see what can be learned from the process itself. This chapter elaborates on several of the key activities by providing a short recap on how the activities were performed and what the advantages and disadvantages are in both this and in other contexts.

Before diving into the specific activities, the first step is to overlook the complete process. The Agile Scrum way of working presented a clear insight in what should be performed in how much time and provided an easy way to compose task packages for pre-defined periods. The usefulness of this approach is limited to the ability to represent tasks in the form of needed time, which proved to be an issue for some show stopping research activities.

Although the Scrum approach is very helpful in identifying the amount of effort necessary for completing the identified tasks, the identification of tasks on all levels was performed by scaffolding the required work in a multi-level fashion. By scaffolding all tasks, a communication means presented itself because a framework within which certain tasks needed to be performed became clear (#29).

> ***#29 Scaffolding planned work*** *allows steps to be carried out quickly. By using a framework for the planned work, a communication means is created for planning and assessing the work performed*

The specific key activities that are present in this thesis are shown in Figure 18, the combination of those identified activities in a process flow resulted in the steps necessary for a transition towards a new systems architecture for the specific Tam Tam context. All identified activities are elaborated on in the following sections by referencing back to elements from Chapter 3:'Approach', the specific approach taken in this thesis.

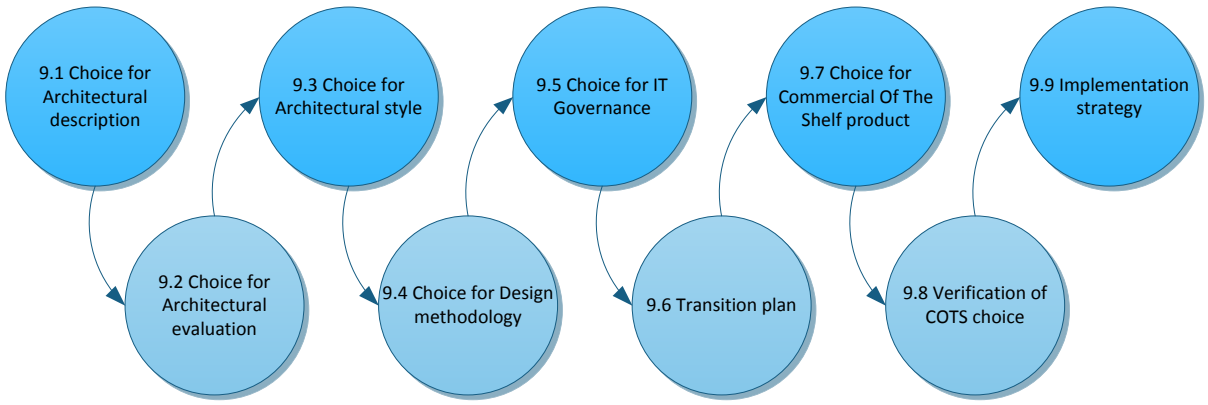

Figure 18: Thesis process flow

9.1 Choice for Architectural description

A lot is dependent on the specific context of, and the specific perspective on, the architecture that is studied. In the context of Tam Tam the goal was to reach the 'next step' of the internal systems architecture, where the specifics of the next step were far from clear. In situations where more clarity is readily available the approach taken will be a little different due to the identification of key characteristics and the substance to be presented while defining the description methodology.

The nature of the Tam Tam context was that some knowledge was available at some architects. This made the choice for an adaptable model in the form of a white board in combination with iteration ideal. These iterations lead to a well-supported aggregated collection of documentation from a wide variety of sources. In contexts in which less architecture communication possibilities are present, it might be wise to skip the analog white board model and directly create a digital visualization. By removing the analog steps the amount of overhead is reduced while, in those contexts, still achieving a complete picture.

The previously mentioned available knowledge presented a clear amount of clues that could be used, such as to where to look for in the source code and keeping a focus on component interactions. In situations with a lack of this knowledge, a breadth first search solution would have been a better approach. This search would then result in a web of components that presents the clues to investigate the relevant interactions between components.

#30 Definition of viewpoints already presents a gentle direction towards a solution

The perspective on the architecture used in this context, the back-end behind components, was explicitly defined in the approach. Other perspectives can focus more specifically on other aspects of the architecture. Although a wide variety of architectural viewpoints from the literature were included in the comparison, a change of perspective could cause more exclusive ways of visualization to be used in other contexts. An example of this is the chosen process visualization due to the focus on processes, with another focus the chance this viewpoint gets replaced is fairly big (#30).

9.2 Choice for Architectural evaluation

Although the calculation of the metrics can be seen as subjective, the reached result still stands out. On the field of selecting the specific means and metrics, the perspective is very important. Although these evaluation means can be of use in similar contexts, a better fit will arise when the four presented steps are carried out to derive a context- and perspective-specific version of the means and metrics.

In the evaluation iterations performed in this project, the majority of activities were based on measuring instead of questioning. This was performed in such a way because numerical values can more easily be compared than fuzzy statements like 'a moderate amount' or 'within reasonable time'.

The iteration aspect of evaluation is an important step included from the Agile Scrum methodology because a continuous reflection with the envisioned goal is performed. This allows the identification of needed adjustments to the new architecture to be made throughout the course of this project. By letting the evaluation come back at a regular interval, the results of it can prove a worthy source of intelligence for decision making.

The inclusion of expert review sessions helped test the amount of support that will be present when the designed architecture goes into production. Furthermore, these sessions presented unbiased opinions on the future situation, which could be used as reflection means. Key characteristic to look for in experts in other contexts is that they have not been involved in the design in any way so they are not biased with regard to the design.

9.3 Choice for Architectural style

A wide variety of different possible architectural styles have been listed. Although each option can be used to emulate every other option, the step taken was to choose for that alternative that presents the most saved time by its offered features.

The definition of trade-offs helped making a choice for the specific Tam Tam context in such a way they could also be seen as the first few decisions for the design. While performing the decision process, knowledge retrieved from the computer science domain was mostly used for the identification of key differences.

In order to use the decision making process in other contexts, the identified trade-offs can still be used for they present a means for classifying the different enterprise application integration options. If the context in which a decision needs to be made is very different than the Tam Tam context it is wise to supplement these trade-offs with several additional ones to arrive at a decision more tailored to that specific context.

When a best fit style is chosen, there will still be discrepancies between the desired and chosen style on some trade-offs. A way of taking this into account is to use another 'distance function' between two styles by, for example, giving higher weights to specific trade-offs.

9.4 Choice for Design methodology

The focus on re-engineering has both advantages and disadvantages, first of all it helps making the design assessable because all capabilities are stated explicitly by the visualized architecture. Secondly, the negative side of the re-engineering focus is that it limits future vision by limiting the capabilities by only allowing already existing functionalities and processes.

On a methodological view, the process of straightening out the reference frame by posing several key questions introduced much clarity for the design. The only steps remaining are to fill in the leftover details in three different fields: technology, organization and transition. This threefold proved a worthy choice but during the implementation several decisions were still remaining to be made or undocumented. In this project, these

shortcomings were not that much of an issue because the design and the major part of the implementation were in the hands of the same person. In situations where other people should take over the implementation, a complete specification can be advantageous but also limits the freedom of the developers (#31).

#31 More technological aspects are possible to be designed within the design instead of only the core technology to be used.

The posed questions were not supposed to cover the complete design but to strive for the idea that answers the posed questions would provide leads for the further design. With these answers providing a foundation for the design, the multi-level scaffolding approach is introduced again. First of all it is questioned what is desired to be reached and secondly the way of reaching that goal is defined.

9.5 Choice for IT Governance

The design for organizational change was mostly created by using the MIT matrix because said matrix specifies which areas of decision making rights and responsibilities are to be divided. The matrix is designed to be reusable in other IT situations and because it is at a reasonably abstract level, it can be used as communication means between both business and technology people.

Within Tam Tam, a momentum of organizational change was already started by outsourcing the system administration. On top of that, most of the responsibilities from the matrix were already at the entities specified in the design so the role of the design was to formalize them. The implementation of the desired rules and guidelines should be continued with the explicit formalization as a central goal. How this transition should be carried out is left outside of the scope of this project because of the very minor differences between the ongoing momentum and the designed rules and guidelines.

9.6 Transition plan

The goal of crafting a transition path was to limit the necessary downtime by keeping the exit strategies available. This focus on exit strategies was to enforce very small actions to be executed to make the switch when the new architecture receives a full go-ahead. The advantage of the crafted transition path in this situation is that it can both be used for the complete architecture at once and for little chunks of the designed architecture. Furthermore, it is created in such a way that it can be re-used in other contexts by keeping the amount of specifics very low. The crafted transition path can easily be used in situations with an Enterprise Service Bus and a Service Oriented Architecture but perhaps also in other Enterprise Application Integration situations.

On a more specific level, a worthy addition to the transition path can be to add 'eavesdropping steps' to the currently executing processes. These listeners can be used to log exactly what requests and responses are sent around to be used for testing the service wrappers.

The main disadvantage of this transition path is that it is unknown how much time the last step will take in situations with a lot of legacy code. In situations where less legacy code is present, it becomes clear very rapidly what operations are blocked from executing.

9.7 Choice for Commercial off-the-shelf product

The focus on the use of standards (BPEL, WSDL, etc…) limited the amount of possibilities by a large number. Next to that, when a decision should be made in a different context or by different people with other levels of experience, the resulting product will differ.

The part that can be re-used is the proof of concept driven product choice. This makes sure the chosen implementation form will work. By defining basic building blocks, the different options can be compared to make a choice. Furthermore, the creation of proofs of concept presented an insight into the characteristic differences between the different products forming a basis for a well-supported decision.

The decision for BizTalk in this context came as a surprise because of the given carte blanche and the fact that several attempts at BizTalk in the past have all failed. Due to the proof of concept approach however, the success of this implementation decision can be guaranteed with a great amount of certainty. The only thing that could have prevented the success of BizTalk was the list of identified risks in the verification of the choice, elaborated on in the next section.

9.8 Verfication of COTS choice

A choice for a specific product can be made fairly easily but a well-funded choice is the next step on the ladder of correctness. The step that follows the choice for a COTS product was to verify the decision to identify plausible future issues with the chosen product.

To identify the possible issues, several brainstorm sessions were carried out to identify a list of risks that had to be mitigated before carrying on with the implementation using BizTalk. Although very few issues were still present during the implementation, some more time should be invested to identify a complete list of issues with their corresponding mitigation plans (#32).

*#32 **Deployment** should be under focus in the design too, especially splitting it from development*

Within the context of this project it was an ideal option to use a risk analysis to verify the choice, but taking a step backwards is fairly common practice for every choice that should be made. Possible future failures should always be included in every decision.

9.9 Implementation strategy

Due to the nature of the design process, several parts of the implementation were already performed before the 'actual' implementation actually started. By merging the design and implementation in this way, it became clear that all designed requirements were feasible. What was left in the implementation phase was merging the different proof of concepts (#33).

> ***#33 Inverse proportionality*** *can be identified between the time spent while designing and the time spent while implementing*

When viewed from a project managerial perspective, this approach is ideal because it reduces the risk of insurmountable issues. For other situations this hands-on driven 'implement while designing' approach is very interesting to keep in mind, but the specific focus on re-engineering here allowed such an approach to succeed. For situations in which new architectures are to be made and the requirements are not clear, this approach might not work because there will be a possibility that the process will be stuck at the design stage.

What was missing in this approach is a step backwards to look at how the architecture should be maintained instead of only looking at how it should be like.

Part V:

Conclusion, wrapping up and contributions

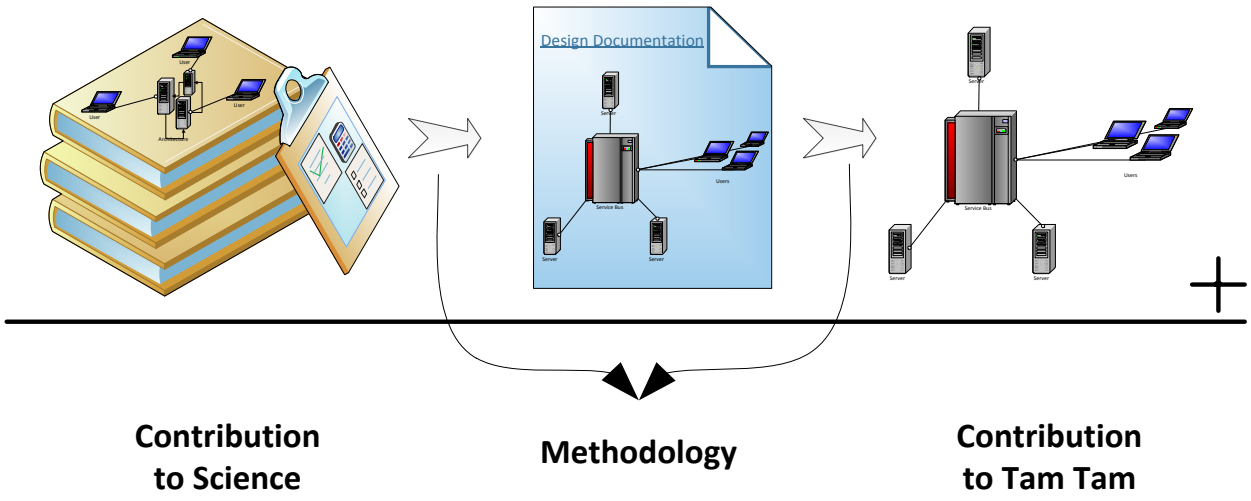

Figure 19: What can be learned from the process and results

Chapter 10: Contributions

Every new insight obtained in this thesis can be ascribed to one of two categories of areas, either science or Tam Tam. This chapter reflects on the specific contributions to both areas by elaborating on transferable and re-usable knowledge and insights. The goal is to present those leads this thesis provides that both science and Tam Tam are able to adopt. The first category of contributions, which is elaborated on in the first section, is the possible elements that can be derived from this thesis and adopted by science. After that, the second section presents the advantages Tam Tam received by hosting this thesis project.

10.1 Science

10.1.1 Aggregated insights

During the core chapters of this thesis several insights have been listed which present key learning points during the different stages of this thesis. Several sources of insights that were present are knowledge management, available resources, hands-on driven design, layered design, possibilities for adaptability and the use of scaffolding in all stages. These learning points can be used both as preventing means as well as explanatory means for common situations in other contexts.

10.1.2 ESB transition case study

In the relevant literature, documents describing carried out transitions to other forms of Enterprise Application Integrations are very sparse. A possible explanation for this is that most literature research only prescribes what should and should not be done while carrying out such a transition. This lack of case studies is decreased by the provided case study in this thesis.

10.1.3 Re-usable step-by-step approach

One of the key areas of focus of this thesis project was the process by which the results were to be reached. Due to this focus, a re-usable step-by-step approach was crafted on a theoretical foundation. One of the main aspects of this approach is that it is not static, but it presents a dynamical way to transition a currently in use systems architecture towards the best fitting Enterprise Application Integration style. The dynamical nature comes forward from the main idea that in a different context other specifics are to be adopted in every step taken.

10.1.4 Abstract ideas made tangible

The source of all taken steps lies in the available literature. This literature mainly discusses abstract ideas. The feasibility of this literature is tested in this thesis by carrying out the chosen steps. By carrying out said steps in a tangible real-life situation, it becomes evident whether or not the theoretically designed actions are applicable in practice. Furthermore, the gap between business and IT is bridged by linking the

abstract communication means to tangible solutions and presenting a way in which business requirements can be translated into real-life IT solutions.

10.1.5 Compilation of EAI research

Because one of the main focuses of this thesis is to provide a theoretically backed approach, several aspects of theory surrounding the Enterprise Application Integration topic are explored. The decision for each methodology to be followed uses a wide variety of possible alternatives in the comparison.

10.1.6 Scientific approach can tackle 'recurring' problems

The approach taken in this occurrence of the transition towards an Enterprise Service Bus by Tam Tam achieves the result that has been tried to achieve several times in the past. The taken approach consists of balancing alternatives on every single step taken in this thesis. This ensured the best possible fit was created for Tam Tam and it also reached a certain amount of support for the design to be successful. Key lesson learned here is that when a step backwards is taken, and extra time is invested, a successful transition can be imminent.

10.2 Tam Tam

10.2.1 Continued effort towards an EAI

In the past, several attempts were made to reach an Enterprise Application Integration solution. Most of the attempts failed and a new approach was searched for. The approach taken in this thesis resulted in both a feasible design and a working implementation. By carrying out the specified implementation plan further, future efforts towards an EAI can be ceased.

10.2.2 Adaptability and flexibility

The introduction of a layered approach and the Enterprise Service Bus itself increases both adaptability and flexibility of the architecture. Where in the current situation IT was always one step behind the business requests, the newly designed situation allows both to be leveled. Furthermore, the introduction of layers causes a very limited amount of end-users noticing changes because back-end components can now more easily be replaced with greatly reduced downtime.

10.2.3 Transferable BizTalk server 2010 knowledge

With the creation of tutorials for every task that has to be performed with BizTalk, know-how concerning BizTalk can easily be shared. This shared knowledge presents incentives for two actions. The first is that understanding can be created about what has been implemented in what way and the second is that BizTalk requests of customers can be fulfilled instead of being rejected.

10.2.4 Communication means for current architecture

Before a design could be made for the new architecture, the current architecture had to be thoroughly understood. Due to the created documentation of the current architecture, a communication means was created. This means for communication can be, and has been, used for identifying what functionality is offered how. Furthermore, the created communication means presented a starting point for designing what the future situation should look like.

10.2.5 Single point of contact

The introduction of an Enterprise Service Bus introduces a common knowledge of all available functionality. All offered functionalities are listed in a single location independent to the inner workings of these functionalities. This presents means for exposing functionalities both within Tam Tam as well as outside of Tam Tam. The other way around is valid too, by listing all functionalities in a single place only one place has to be allowed to connect to entities outside of Tam Tam.

10.2.6 Implementation plan to the 'next level'

One result of the step-by-step approach towards the next level is a clearly specified implementation plan. This plan, with specified atomic tasks, can be followed to achieve the designed improvements.

Chapter 11: Conclusion

New attempts to move from an ad-hoc based systems architecture towards a fitting Enterprise Application Integration style came and went within Tam Tam. After most of these attempts failed, a rigorous alternative approach was longed for. This thesis project fulfilled this need by starting with a carte blanche to investigate the question:

"How can Tam Tam benefit from the theoretically 'logical' next step for the internal systems architecture?"

To be able to answer the stated question, seven different research parts have been identified with their results elaborated on in chapters 2 through 9. A summary of each of these research parts is presented in the next section. In the section following the summaries, several leads for further work are identified.

11.1 Summarizing research parts

11.1.1 What does the current situation look like?

To gain an understanding of the current situation, Chapter 2: 'Current Situation' describes the extract, abstract & present threefold used as iteration elements. By including additional sources of information in each iteration of the threefold, a complete mental image could be created. The next step was to make this mental image communicable by defining means of visualization understandable to both business people as well as IT people. This definition was created by thoroughly comparing the different possibilities and selecting a hybrid form consisting of elements of different frameworks and architectural description languages. After three selected viewpoints were completely visualized, the last step was to gain an understanding about the rationale for the desire for change. This rationale was formed from the acquired insight into the architecture as well as the a priori posed requirements.

11.1.2 What lessons can be learned from previous attempts at creating an integrated architecture?

During the process of understanding the current situation, several previous attempts for reaching a new systems architecture were identified. These identified previous attempts are elaborated on in Chapter 3: 'Approach' by providing the goals and possible reasons of failure for all of them. These reasons of failure presented invaluable lessons learned that were used as main drivers for the approach taken in this thesis.

11.1.3 How can different architectures be compared?

The last step of the surrounding environment in which the new design is to be created, is the definition of means and metrics for comparing architectures elaborated on in Chapter 4: 'Evaluation of architectures'. The way in which architectures are compared in this thesis consists of comparing a diverse variety of architecture and software

evaluation related literature. The approach used here is a defined four-step plan. The first step was to define the perspective from which the architectures should be viewed. The second step was the definition of objectives that should be reached with the new architecture. These objectives were translated into plausible scenarios in the third step and the last step was to derive quantitative metrics from the scenarios.

11.1.4 What is the theoretically 'logical' next step for Tam Tam?

The process of identifying the theoretically 'logical' next step is divided into two parts. First of all Chapter 5: 'Choice for architectural style' decides on the architectural style, after which Chapter 6: 'Future architecture design specifics' lists the design specifics. To decide on the architectural style, a framework is used for comparing architectural styles based on the definition of trade-offs. These trade-offs represent continua of characteristics on which available alternatives can be mapped. All identified Enterprise Application Integration alternatives are mapped on the combined trade-off vectors. The vector for an Enterprise Service Bus has the least distance from the desired vector defined by the Tam Tam architects so this style is chosen. Some slight differences between both vectors were mitigated to create the best fitting solution.

The design specifics were decided on by answering a selected set of questions to give the scaffolding for the implementation more form. The next step was to define which functionalities should be offered in what place and how. To give the purely technical design more momentum, organizational rules and a transition path have been defined to enforce the successful adoption of the designed architecture within Tam Tam.

11.1.5 Is the designed next step feasible in practice?

To test the design for feasibility, the theoretical designed systems architecture is linked to the practice in Chapter 7: 'Implementation'. The implementation plan used has a focus on exit strategies by providing several opportunities for reflecting on the chosen path. The first step in this path was to choose a Commercial off-the-shelf product that should reduce the time necessary for implementation. After having experimented with several products to implement the basic building blocks necessary for the design to be implemented, BizTalk seemed to be the best option. This choice is reflected upon by carrying out a risk analysis, which should prevent the decision for BizTalk to be regretted due to future requirements. After mitigation plans have been created for the plausible future requirements, the choice for BizTalk was made final and several parts of the design have been implemented according to a defined list of atomic implementation steps.

11.1.6 What advantages and disadvantages does the new architecture offer?

After having implemented several processes which cover all different aspects of the design, the partial new situation is reflected upon to derive the benefits and drawbacks in Chapter 8: 'Evaluation of results'. In said chapter a summary is given of several of the evaluation iterations carried out throughout the project. These iterations were introduced to allow adjustment of the direction at several points. The summarized results are extended with the addition of the most thorough instance of the evaluation

iterations by carrying out the scenario analysis and valuating the defined metrics. After this last iteration, advantages and disadvantages have been identified and some unbiased experts were invited to participate in evaluation sessions. From these sessions several insights were derived which are used in the implementation process that took place after the go/no-go decision was made. Several advantages were identified, ranging from an introduced single point of contact through transferable BizTalk server 2010 knowledge to future evolution possibilities.

11.1.7 How can pieces of the chosen process be used in other contexts?

The process followed, resulting in the results of the previous questions, is evaluated in Chapter 9: 'Evaluation of process'. The nature of every step, which the taken process is composed of, is very abstract. These abstract steps can only be used in other contexts after the key characteristics have been tailored to the specific context within which the steps are to be used. From this need for adaption to a specific context, it is derived that the major part of the followed process can be easily re-used in other scenarios. The only exception is the taken transition path, this can most probably only be used in contexts with service oriented designs.

11.2 Further research leads

When a step backwards is taken from the work performed in this thesis to look at a broader scope, several elements can be identified that need further research. Each of these elements is elaborated on shortly.

- **The sketched methodology should be tested with other cases and in other contexts.** Because the methodology is only used in a single context it still needs verification on the usage in other contexts
- **How can big IT governance changes be introduced?** In the current context the amount of IT governance changes that had to be introduced was very limited. Research has to be performed to figure out how major governance changes can be put into practice
- **The influence of selected visualization means on the design and vice versa.** As was noted as a learning point, the selection of the processes viewpoint had an influence on the use of orchestrations. It might be that more general causalities for this relation exist or that choices for viewpoints are influenced by the wishes of clients.
- **Re-usage of the four-step evaluation methodology.** The four-step evaluation methodology presents a step-by-step approach for defining the evaluation means and metrics. Research could be performed into the re-usage of this four-step methodology for other fields where designs are made for elements subject to change.
- **Are the defined architectural style trade-offs representative for Enterprise Application Integration?** Research effort could be investigated in verifying whether the defined trade-offs offer enough support for every context or if this set of trade-offs is only usable in a similar context as this project.

- **Can a taxonomy be created that classifies available designs?** The approach of asking strategic questions used in this context might be generalized by formulating a taxonomy that classifies all available options.
- **Formalizing the complete step-by-step approach.** By formalizing the step-by-step approach, the same objective can be reached in other scenarios and contexts. Before being able to formalize the approach, several of the previous future work elements need to be carried out to achieve a well-supported formalization.

References

Abowd, G., Bass, L., Clements, P., Kazman, R., & Northrop, L. (1997). *Recommended Best Industrial Practice for Software Architecture Evaluation.* CMU/SEI-96-TR-025.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., et al. (2007). *Web Services Business Process Execution Language Version 2.0.* OASIS.

Armour, F., & Kaisler, S. (2001). Enterprise Architecture: Agile Transition and Implementation. *IEEE IT Professional, volume 3 issue:6*, 30-37.

Assmann, M., & Engels, G. (2008). Transition to Service-Oriented Enterprise Architecture. In R. Morrison, D. Balusubramaniam, & K. Falkner, *Software Architecture - Lecture Notes in Computer Science* (pp. 346 - 349). Springer Berlin / Heidelberg.

Babar, M., Zhu, L., & Jeffery, R. (2004). A framework for classifying and comparing software architecture evaluation methods. *Software Engineering Conference, 2004. Proceedings. 2004 Australian* (pp. 309 - 318). IEEE Computer Society Washington, DC, USA ©2004.

Basili, V., Caldiera, G., & Rombach, H. (1994). The Goal Question Metric Approach. In J. Marciniak, *Encyclopedia of Software Engineering* (pp. 528 - 532). John Wiley & Sons.

Boehm, B. (1991). Software Risk Management: Principles and Practices. *IEEE Software, volume 8 issue:1*, 32 - 41.

Boehm, B., & DeMarco, T. (1997). Guest Editors' Introduction: Software Risk Management. *IEEE Software, volume 14 issue:3*, 17 - 19.

Boehm, B., & Port, D. (2001). Educating software engineering students to manage risk. *Proceedings of the 23rd International Converence on Software Engineering* (pp. 591 - 600). Washington, DC: IEEE Computer Society.

Bots, P. (2002). *Introduction to Systems Engineering and Policy Analysis: A practical guide to systematic problem solving.* Delft: Delft University of Technology, Faculty of Technology Policy and Management.

Brisebois, R., Boyd, G., & Shadid, Z. (2007). What is IT Governance and Why is it Important? *5th Performance Seminar of the INTOSAI IT Standing Committee*, (pp. 30-35).

Channabasavaiah, K., Holley, K., & Tuggle, Jr., E. (2003, 12 16). *Migrating to a service-oriented architecture, Part 2*. Retrieved 11 11, 2010, from IBM developerWorks: http://www.ibm.com/developerworks/library/ws-migratesoa2/

Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S. (2007, June 26). *Web Services Description Language (WSDL) Version 2.0 Part1: Core Language*. Retrieved January 2011, 2011, from W3c: http://www.w3.org/TR/wsdl20/

Clements, P., & Northrop, L. (1996). *Software Architecture: An Executive Overview.* Hanscom AFB.

de Haes, S., & van Grembergen, W. (2005). IT Governance Structures, Processes and Relational Mechanisms: Achieving IT/Business Alignment in a Major Belgian Financial Group. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences.* IEEE Computer Society.

de Leusse, P., Periorellis, P., & Watson, P. (2007). *Enterprise Service Bus: An overview.* Newcastle upon Tyne: University of Newcastle upon Tyne.

Deepview case study. (2010, 1 4). *SOA in practice: IBM uses an integrated approach*. Retrieved 11 5, 2010, from IBM: http://www-01.ibm.com/software/success/cssdb.nsf/CS/CHUY-7YTKXL?OpenDocument&Site=default&cty=en_us

Deursen, A. v., Hofmeister, C., Koschke, R., Moonen, L., & Riva, C. (2004). Symphony: View-Driven Software Architecture Reconstruction. *Proceedings Fourth Working IEEE/IFTP Conference on Software Architecture (WICSA 2004)* (pp. 122-132). Oslo: Centrum voor Wiskunde en Informatica.

Dobrica, L., & Niemel, E. (2002). A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering, volume 28 issue:7*, 638 - 653.

Durchslag, S., Donato, C., & Hagel, J. (2001). *Web Services: Enabling the Collaborative Enterprise.* San Fransisco: Grand Central Networks Inc.

Educause Centre for Applied Research. (2004). IT Governance. *IT Alignment in Higher Education, volume 3*, 57-66.

Emmerich, W. (2000). Software engineering and middleware: a roadmap. *Proceedings of the Conference on The Future of Software Engineering* (pp. 117-129). Limerick: ACM.

Erasala, N., Yen, D., & Rajkumar, T. (2003). Enterprise Application Integration in the electronic commerce world. *Computer Standards & Interfaces, volume 25 issue:2*, 69-82.

Gallagher, K., Hatch, A., & Munro, M. (2008). Software Architecture visualization: An Evaluation Framework and Its Application. *IEEE Transactions on Software Engineering, volume 34 issue:2*, 260-270.

Goel, A. (2006). *Enterprise Integration --- EAI vs. SOA vs. ESB.* Infosys Technologies White Paper.

Grundy, J. (2001). Software Architecture Modelling, Analysis and Implementation with SoftArch. *34th Annual Hawaii International Conference on System Sciences (HICSS-34)* (p. 9051). Maui: IEEE Computer Society.

Grundy, J., & Hosking, J. (2000). High-level static and dynamic visualisation of software architectures. *Proceedings of 2000 IEEE Symposium on Visual Languages* (pp. 5-12). Seattle: IEEE Computer Society.

Grundy, J., & Hosking, J. (2003). SoftArch: tool support for integrated software architecture development. *International Journal of Software Engineering and Knowledge Engineering, volume 13 issue:2*, 125-151.

Hagel, J., & Brown, J. S. (2001). Your Next IT Strategy. *Harvard Business Review, volume 79 issue:9*, 105-113.

Hardy, G. (2003). Coordinating IT Governance - A New Role for IT Strategy Committees. *Information Systems Control Journal, volume 4*.

Hazra, T. (2002). Building enterprise portals: principles to practice. *Proceedings of the 24th International Conference on Software Engineering* (pp. 623 - 633). Orlando: ACM New York.

ISO/IEC 28500. (2008). *Corporate governance information technology.* ISO/IEC.

Janssen, M., & van Veenstra, A. (2005). Stages of Growth in e-Government: An Architectural Approach. *The Electronic Journal of e-Government, volume 3 issue:4*, 192 - 200.

Janssen, M., Gortmaker, J., & Wagenaar, R. (2006). Web Service Orchestration in Public Administration: Challenges, Roles, and Growth Stages. *Information Systems Management, volume 23*, 44 - 55.

Johannesson, P., & Perjons, E. (2001). Design Principles for Process Modeling in Enterprise Application Integration. *Information Systems, volume 26 issue:3*, 165 - 184.

Kazman, R., & Carrière, J. (1999). Playing Detective: Reconstructing Software Architecture from Available Evidence. *Automated Software Engineering, volume 6 issue:2*, 107-138.

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. *Software, IEEE, volume 13 issue:6*, 47 - 55.

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for Architecture Evaluation.* Pittsburgh: Carnegie Mellon University.

Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., et al. (2004). *Patterns: Implementing an SOA Using an Enterprise Service Bus.* IBM Redbooks.

Keil, M., Cule, P., Lyytinen, K., & Schmidt, R. (1998). A framework for identifying software project risks. *Communications of the ACM, volume 41 issue:11*, 76 - 83.

Kruchten, P. (1995). Architecture blueprints- the '4+1' view model of software architecture. *IEEE Software, volume 12 issue:6*, 42-50.

Kruchten, P., Selic, B., & Kozaczynski, W. (2001). Describing Software Architecture with UML. *Proceedings of the 23rd International Conference on Software Engineering* (pp. 715-716). Toronto: IEEE Computer Society.

Lam, W. (2005). Investigating success factors in enterprise application integration: a case-driven analysis. *European Journal of Information Systems, volume 14 issue:2*, 175-187.

Lankhorst, M. (2004). *ArchiMate Language Primer.* Enschede: Telematica Instituut.

Lassing, N., Rijsenbrij, D., & van Vliet, H. (1999). The goal of software architecture analysis: confidence building or risk assessment. *Proceedings of the 1st Benelux Conference on State-of-the-art of ICT architecture* (p. 6p). Amsterdam: Vrije Universiteit, Amsterdam.

Leenslag, W. (2006). *Werkwijze ter beoordeling van IT governance.* Enschede: Universiteit Twente.

Leist, S., & Zellner, G. (2006). Evaluation of current architecture frameworks. *Proceedings of the 2006 ACM symposium on Applied computing* (pp. 1546-1553). Dijon: ACM.

Lindvall, M., Tvedt, R., & Costa, P. (2003). An Empirically-Based Process for Software Architecture Evaluation. *Empirical Software Engineering, volume 8 issue:1*, 83 - 108.

Lung, C.-H., & Kalaichelvan, K. (2000). An Approach to Quantitative Software Architecture Sensitivity Analysis. *International Journal of Software Engineering and Knowledge Engineering, volume 10 issue:1*, 97 - 114.

Mazumder, S. (2006). *SOA: A Perspective on Implementation Risks.* SETLabs Briefings.

McKeen, J., & Smith, H. (2002). New Developments in Practice II: Enterprise Application Integration. *Communications of the Association for Information Systems, volume 8 issue:1*, 451-466.

Menge, F. (2007). Enterprise Service Bus. *Free and open source software conference*, (pp. 1 - 6).

Meredith, J., & Mantel, S. (2009). *Project Management: A Managerial Approach; 7th International student edition.* John Wiley & Sons Ltd.

Microsoft Patterns & Practices Team. (2009). *Microsoft Application Architecture Guide (Patterns & Practices), 2nd Edition.* Microsoft Press.

Natis, Y. (2003). *Service-Oriented Architecture Scenario.* Gartner, Inc.

Papazoglou, M., & Ribbers, P. (2006a). *e-Business: Organizational and Technical Foundations.* Chichester: Wiley.

Papazoglou, M., & van den Heuvel, W.-J. (2006b). Service-Oriented Design and Development Methodology. *International Journal of Web Engineering and Technology, volume 2 issue:4*, 412 - 442.

Papazoglou, M., & van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *International Journal on Very Large Data Bases, volume 16 issue:3*, 389 - 415.

Perry, D., & Wolf, A. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes, volume 17 issue:4*, 40-52.

Petre, M., Blackwell, A., & Green, T. (1998). Cognitive questions in software visualization. In J. Stasko, J. Domingue, M. Brown, & B. Price, *Software visualization: Programming as a multimedia experience* (pp. 453-480). Cambridge: The MIT Press.

PricewaterhouseCoopers. (2006). *IT Governance in Practice: Insight from leading CIOs.* PricewaterhouseCoopers.

Rau, K. (2004). Effective Governance of IT: Design Objectives, Roles and Relationships. *Information Systems Management, volume 21 issue:4*, 35-42.

Richardson, G., Jackson, B., & Dickson, G. (1990). A Principles-Based Enterprise Architecture: Lessons from Texaco and Start Enterprise. *MIS Quarterly, volume 14 issue:4*, 385-403.

Ross, J. (2003). *Creating a Strategic IT Architecture competency: Learning in Stages.* Massachusetts Institute of Technology (MIT), Sloan School of Management.

Sambamurthy, V., & Zmud, R. (1999). Arrangements for Information Technology Gorvernance: A Theory of Multiple Contingencies. *MIS Quarterly, volume 23 issue:2*, 261-288.

Schmidt, M.-T., Hutchinson, B., Lambros, P., & Phippen, R. (2005). The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal, volume 44 issue:4*, 781-797.

Schmidt, R., Lyytinen, K., Keil, M., & Cule, P. (2001). Identifying Software Project Risks: An International Delphi Study. *Journal of Management Information Systems, volume 17 issue:4*, 5 - 36.

Simonsson, M., & Johnson, P. (2006). *Defining IT Governance -- A Consolidation of Literature.* Stokholm: Royal Institute of Technology (KTH).

Smolander, K., Hoikka, K., Isokallio, J., Kataikko, M., Mäkelä, T., & Kälviäinen, H. (2001). *Required and Optional Viewpoints - What Is Included in Software Architecture?*

Trowbridge, D., Roxburgh, U., Hohpe, G., Manolescu, D., & Nadhan, E. (2004). *Integration Patterns.* Microsoft.

Webber, J. (2005). Guerilla SOA: How to fight back when a vendor tries to take control of your enterprise.

Weill, P. (2004). Don't Just Lead, Govern: How Top-Performing Firms Govern IT. *MIS Quarterly Executive, volume 8 issue:1.*

Weill, P., & Ross, J. (2004a). *IT Governance on One Page.* MIT Sloan Management Review.

Weill, P., & Ross, J. (2004b). *IT governance: how top performers manage IT decision rights for superior results.* Harvard Business Press.

Weill, P., & Ross, J. (2005). A Matrixed Approach to Designing IT Governance. *MIT Sloan Management Review, volume 46 issue:2*, 25-34.

Weill, P., & Woodham, R. (2002). *Don't Just Lead, Govern: Implementing Effective IT Governance.* MIT, Sloan School of Management.

White, S. (2004). *Introduction to BPMN.* IBM Cooperation.

Wikipedia Community. (2010). *Orchestration (Computing)*. Retrieved January 31, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Orchestration_(computing)

Wikipedia Community. (2010). *Universal Description Discovery and Integration*. Retrieved January 31, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration

Wikipedia Community. (2011). *ACID*. Retrieved March 11, 2011, from Wikipedia: http://en.wikipedia.org/wiki/ACID

Zachman, J. (1999). A framework for information systems architecture. *IBM Systems Journal, volume 26 issue:3*, 454-470.

Abbreviations

ACID	~	Atomicity, Consistency, Isolation and Durability
BPEL	~	Business Process Execution Language
CIO	~	Chief Information Officer
COTS	~	Commercial off-the-shelf
CRM	~	Customer Relationship Management
DB	~	Database
DNS	~	Domain Name System
EAI	~	Enterprise Application Integration
ESB	~	Enterprise Service Bus
GAC	~	Global Assembly Cache
HRM	~	Human Resource Management
IIS	~	Internet Information Services
ODE	~	Orchestration Director Engine
QoS	~	Quality of Service
SOA	~	Service Oriented Architecture
SQL	~	Structured Query Language
SSO	~	Single sign-on
UDDI	~	Universal Description Discovery Integration
UML	~	Unified Modeling Language
URL	~	Uniform Resource Locator
WCF	~	Windows Communication Foundation
WF	~	Windows Workflow Foundation
WSDL	~	Web Service Definition Language
XML	~	Extensible Markup Language

Appendix A: Description of the current situation

The focus of this appendix is to create insight in the context used in this thesis; this appendix presents the description of the current architecture with the means defined in Chapter 2:'Current Situation'. The way this appendix is divided is by first looking at several identified life cycles within Tam Tam and describing all processes surrounding those. After that, the identified components are elaborated on and their relation with the individual processes is explained. The last two sections discuss the low-level interactions between components and a global overview of the architecture currently in place.

A. Starting point: life cycles

Within Tam Tam several life cycles are in place, these are identified high-level workflows represented by a concatenation of different processes. First of all these life cycles are presented after which the specific processes that form the cycles are described shortly.

Employee life cycle

The employee life cycle presents the whole contact period of an employee with Tam Tam. Starting from their first contact in the form of job interviews via becoming and being an employee to the moment the employee quits their job to search for a challenge elsewhere.

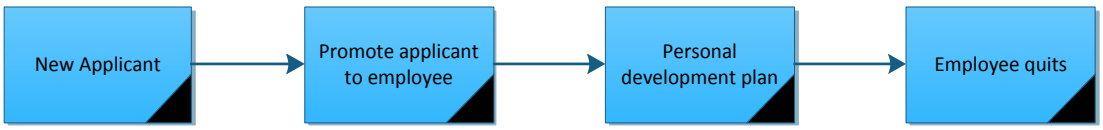

Figure 20: Employee life cycle

Customer life cycle

The second life cycle presents the process starting with acquiring new customers through offering opportunities to them and carrying out projects for those customers. The last step, found in the next cycle, is handing over the created products and services to the service and support department.

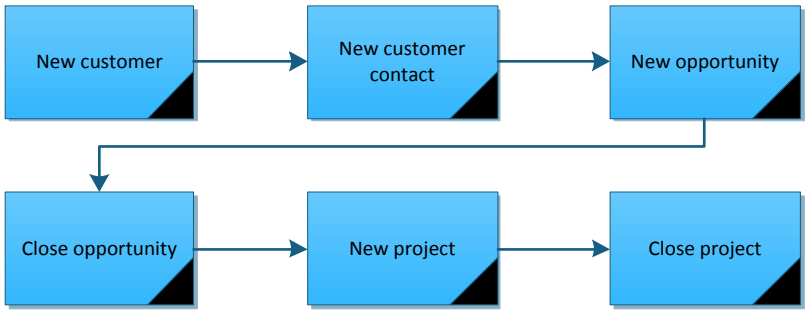

Figure 21: Customer life cycle

Service life cycle

After projects are carried out for customers and new products are delivered, usually the 'operational services' department of Tam Tam takes over the project and provides postproduction maintenance and services. Inherent part of this is keeping track of issues that popped up while using the product.

Figure 22: Service life cycle

Administrative life cycle

Another category is supporting the employees to keep performing work for Tam Tam's customers by carrying out the administrative tasks surrounding the other processes. This life cycle includes processes that keep the cash flow within Tam Tam going.

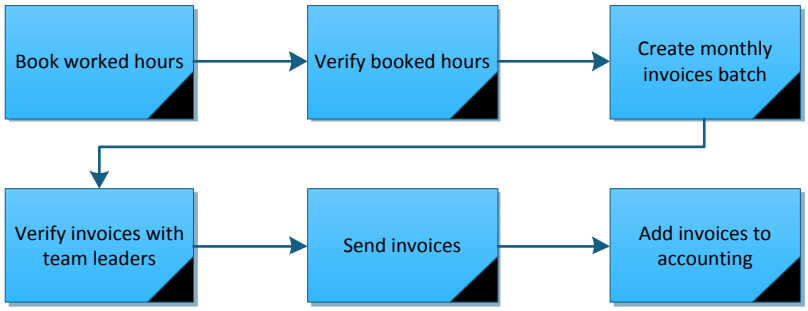

Figure 23: Administrative life cycle

B. Search for individual processes

Table 13 presents all processes that are present in the life cycles as well as additional processes that exist around the life cycles but are not part of the 'main flow' in the specified life cycles. In this table the goal of each of the processes is given. Some processes are indicated with a sign: a percent (%) indicates that the process is carried out on external systems; a star (*) indicates that the process is not carried out; and an

ampersand (&) indicates that the process is carried out by hand. The next section lists for each component what processes are using that specific component; this information is used as a starting point to describe the component orchestration.

Table 13: Identified processes

Life Cycle	Process	Id	Goal
Employee	New applicant%	#1	Store applicant details and progress of job interviews
	Promote applicant to employee	#2	Hire the employee, store details and create user accounts
	Personal development plan&	#3	Six-monthly iteration of knowledge and plans of employees, supported by a SharePoint workflow
	Employee quits	#4	Employee stops working at Tam Tam, all information except administrative details should be removed
Customer	New customer	#5	Customers are added to the CRM to assign contacts, opportunities and projects
	New customer contact	#6	Contact details of customer employees are saved to enable contact
	New opportunity	#7	Create place to store internal documents about offerings to customers
	Close opportunity*	#8	When an offering is accepted/rejected an opportunity can be closed down
	New project	#9	In the case an offering is accepted, a place for communicating documents and progress with customers is created
	Close project	#10	After development is finished the project site can be closed down
Service	Hand over project to supporting service&	#11	To be able to provide services for projects, the service department should be taught what the product does and how it works
	Keep track of new issues	#12	Whenever a customer encounters a bug or deficit in the product an issue should be created
	Fix issues	#13	The progress of fixing issues is kept track of
Administrative	Book worked hours	#14	Tracking worked hours is necessary for sending correct invoices to customers
	Verify booked hours&	#15	Team leaders have to confirm the correctness of the declared bookings
	Create monthly invoices batch	#16	At the end of each month invoices are created to be send to customers

	Verify invoices with team leaders&	#17	After the invoices are crafted they have to be checked
	Send invoices&	#18	The invoices are printed and send to customers
	Add invoices to accounting	#19	The last step is to keep track of which invoices are paid and send reminders
Additional	Customer contact logs in for the first time	#20	A customer contact needs to verify its email address and set a password to be able to log in to the Customer Portal SharePoint and Issue Tracker
	Customer contact logs in	#21	Customer contact credentials need to be verified
	Change customer details	#22	If a customer is changed, the details should be changed in several places
	Change customer contact details	#23	If a customer contact is changed, the details should be changed in several places
	Change opportunity details	#24	When an opportunity is changed, the URL to be used to communicate with the customer should be changed as well
	Change project details	#25	When a project is changed, the URL to be used to communicate with the customer should be changed as well
	Create budgets for service department	#26	Before being able to book hours to issues a budget needs to be initiated
	Book lunch and travel	#27	To keep track of declarations, employees must fill in lunch participation and travel declarations
	Pay salary	#28	Employees need to be paid every month in proportion to work performed
	Change Employee details	#29	When information about an employee change it has to be changed in several locations

C. Identification of components

The components section is divided into three different parts representing the three different specimens of components in place in the current architecture. The first part discusses the databases in place, the second are applications and the third are the services and supporting peripherals. Each component lists by which processes it is used, this usage is used in the fourth section of this appendix by visualizing component usage in executing specific processes.

Databases

All databases that are in place are elaborated on and Table 14 presents the specific database details.

Administration DB

The Administration DB used to be the core database of Tam Tam, all data was stored in this one place. Recently several things have changed with this set-up by moving employee, customer and customer support data to separate databases. At the moment the Administration DB still acts as a portal to some data from those separate databases by 'linking' the databases.

HRM DB

The HRM DB was crafted in conjunction with the HRM application and serves the goal to store all employee data which does not change much. This category of employee data includes personal details, terms of employment, salary scale and vacation days. Data which changes regularly is represented by the booked hours which the employee has worked for a specific purpose. This dynamic data is stored in the administration database.

CRM DB

The CRM DB is the place in which all customer contact is stored. Per customer, customer contacts can be added representing the employees of a specific customer with whom Tam Tam has ties.

IssueTracker DB

The IssueTracker DB is used to store all bugs and feature requests from customer contacts. The service department within Tam Tam carries out the created requests and hereby uses this database for keeping track of their progress and communicates with customer contacts about specific requests.

CustomerPortal DB

The CustomerPortal DB acts as a message oriented service bus, the actions that are stored on this message bus are SharePoint operations. This is done because SharePoint operations usually take too much time in executing to let users wait for the completion. This database is also used for storing one time keys for customer contact account activation.

Data Ware House

The data ware house presents opportunities for checking Tam Tam's operations. This is done through specific business intelligence functions which report about data in the several databases (linked through the administration database).

Customer Portal SharePoint DB & Portal SharePoint DB

The SharePoint databases are used to store all SharePoint data, including pages and workflows. For example the personal development process workflow is carried out in the SharePoint portal database.

Table 14: Identified database details

Database	Runs on	Contains	Used in
Administration	DFT-SQL-001	Projects, booked hours, declarations, budgets and invoice data	#5, #9, #10, #14, #16, #22, #25, #26, #27, #28
HRM	DFT-SQL-003/hrm	Employee info, terms of employment, bonus information and personal development plan	#2, #3, #4, #28, #29
CRM	DFT-SQL-003	Customers, customer contacts and opportunities for customers	#6
IssueTracker	DFT-ISSUE-(010/011)	Issues reported by customer contacts for projects	Only outgoing calls
CustomerPortal	DFT-SQL-003/prod	Queue for customer SharePoint operations and one time keys for customer contact account activation	#7, #9, #20, #22, #24, #25
Data Ware House	DFT-SQL-004	Business intelligence data	#27
Customer Portal SP	DFT-SQL-003	SharePoint data	Dedicated DB
Portal SP	DFT-SQL-010	SharePoint data	Dedicated DB

Applications

All applications that are in place are elaborated on and Table 15 presents the specific details.

CRM

The CRM application is used to manage customer and customer contact details. Next to that it is used to create opportunity sites. Several CRM plugins are in place which take care of callouts to the CustomerPortal web Service when certain operations are executed.

CustomerPortal Auth App

The CustomerPortal Auth App is an extension to a Microsoft ISA server and can be seen as the castle gate that serves the function to only let through legit employees and customer contacts to internal applications. The ISA server takes care of the authentication and the CustomerPortal Auth App of the authorization.

IssueTracker

This application is the issue tracker that employees and customer contacts use to post (comments to) issues to be fixed by the service department. Next to that this application is used to book hours to certain budgets so those hours can be declared through invoices send to customers.

Project Toolkit

The project toolkit is used to create new projects, when these projects are created a site creation action is passed to the CustomerPortal web Service. Additional tasks of the project toolkit are the creation of budgets and changing key information about projects, i.e. lead developer/responsible people.

Queue Runner

The queue runner pops the 'slow' operations from the queue in the CustomerPortal DB and carries them out. These operations range from the creation of project sites to updating customer details in the Customer Active Directory.

Hours Application

The Hours application is the main place where employees indicate the number of hours they have worked and under what category those hours belong. The second part of this application allows employees to indicate on which days they have joined the company lunch and declare their work-home and home-work kilometers.

Facturering

The Facturering application creates the opportunity for the finance department to automatically create the batch of invoices for specific months. Next to that it can be used to see and edit specific invoices. When an invoice batch is created, the output is a set of PDF files on a pre-defined network share and an XML file to be imported into the accounting software.

HRM Application

This specifically crafted application for the HRM department allows HR employees to fill in new employees and keep track of their progress as employees. Important elements of this application are a lot of data showing features, i.e. what are upcoming birth days or employee contract ends.

ADP

ADP is an externally licensed application in which the payroll information is stored by the finance department.

Exact

Exact is the externally licensed application for the finance department to manage Tam Tam financial goods. Invoices are imported in here through the use of XML exports generated by the Facturering application.

Portal SP

This application presents the spine of the intranet used within Tam Tam. One of the workflows present in this application is the Personal Development Plan workflow that indicates what steps should be taken in what order to complete the six-monthly employee evaluation process.

Customer Portal SP

This application presents the spine of the project pages used by Tam Tam to store documents about carried out projects. Customer contacts use this application too for commenting on the stored documents and collaborating with Tam Tam employees.

Table 15: Identified application details

Application	Runs on	Performs	Used in
CRM	DFT-CRM-003	Manage customer and customer contact details	#5, #7, #22, #23, #24
CustomerPortal Auth App	DFT-INT-004	The gate through which all external users need to pass to gain access to Tam Tam internals	#20, #21
IssueTracker	DFT-ISSUE-001	Keeping track of the progress of fixing issues	#14
Project Toolkit	DFT-INT-002	Creation and closing of project sites and creating budgets for projects / services	#9, #10, #25, #26
Queue Runner	DFT-INT-003	Carries out 'slow' operations with the CustomerPortal SharePoint server	#7, #9, #22, #24, #25
Hours Application	DFT-ISSUE-001	Allows employees to book hours and indicate lunch participation and travel details	#14, #27

Facturering	DFT-FIN-001	Allows users to look up invoices and details about those invoices	#16
HRM	DFT-INT-002	Enables all HRM processes	#2, #3, #4, #29
ADP	EXTERNAL	Managing the salary paying process and progress	#2, #29
Exact	EXTERNAL	Accounting application	#19
Portal SP	DFT-INT-002	Program that forms the backbone of the intranet in place within Tam Tam	#3
Customer Portal SP	DFT-INT-(003/004/005)	Main part of the project pages used to communicate with customers about projects	#7, #9, #22, #24, #25

Services and supporting peripherals

All services and supporting peripherals that are in place are elaborated on and Table 16 presents the specific details.

IssueTracker web Service

This web service provides the opportunity to contact the issue tracker functionality from other components. It provides means to add and edit customer details. The customer details are stored in the CRM database, but since the issue tracker is a stand-alone application it should be kept in sync to allow CRM customer contacts to log in to the IssueTracker as well. This knowledge is used to allow contacts to interact about certain service agreements.

CustomerPortal web Service

This web service can be seen as the center point in the web of customer related operations; it takes care of queuing SharePoint operations, updates active directory details, keeps the issue tracker synchronized and maintains project information in the administration database.

Accounts web Service

The accounts web service takes care of the actions that need to be performed after a new employee is created in the HRM application. It creates an active directory account, a mailbox and enables the office communicator account for the new employee.

Exchange Server

Contains the mailboxes for all employees and contains some account details, i.e. if a user is allowed to use office communicator.

(Customer) Active Directory

These contain user information for customer contacts as well as employees. This is the one place in which the passwords of user accounts are stored, thus these components help in authenticating users for the use of several (web) applications. Next to that it contains groups and security policies to enforce the prohibition of certain actions or access to applications.

Table 16: Identified services and supporting peripherals details

Service or supporting Peripheral	Runs on	Performs	Used in
IssueTracker web Service	DFT-ISSUE-001	Interface for other applications to interact with the Issue Tracker	#6, #20, #23
CustomerPortal web Service	DFT-INT-004	Handles customer contact and SharePoint operations	#5, #6, #7, #9, #20, #22, #23, #24, #25
Accounts web Service	DFT-XCH-005	Creation of new employee accounts	#2
Exchange Server	DFT-XCH-005	Keeping track of employee mailboxes and settings	#2
(Customer) Active Directory	DFT-DC-(001/002/003/ 010/011/012)	Authentication of employees and customer contacts	#2, #5, #6, #20, #21, #22, #23, #29

D. Specification of viewpoint 2: Visualization of component usage

In this section one example of component usage is given, all other component usage visualizations are available within Tam Tam internal documents. The visualization consists of several key elements as defined in the BPMN language (White, 2004). A circle denotes the start of a process and a circle with a bold ring denotes an endpoint of a process. Dashed lines denote the passing of messages and straight lines denote process flow. Tasks are defined by rectangles and decision points are depicted by diamonds. Furthermore, the visualization is divided into pools with possible some swim lanes in it. Pools define categories of actors and swim lanes define specific parts of actors.

Promote applicant to employee

Figure 24 presents the component interactions carried out while adding an employee to Tam Tam. The process is initiated by the HRM department by filling in employee details

in the HRM application. This application stores the details in the HRM database and calls the accounts web service to start creating the necessary user accounts. The creation of user accounts is done by first of all creating an active directory account, followed by the creation of a mailbox and lastly enabling the user to use Office Communicator. The last step of the accounts web service is to give an error if something went wrong or an 'ok' if all is carried out as planned. This reflection is sent back to the HRM department through the HRM application to be dealt with. Finance is directly notified that a new employee has to be added to their accounting service and the HRM department can decide to contact system administration depending on the reflection given by the accounts web service.

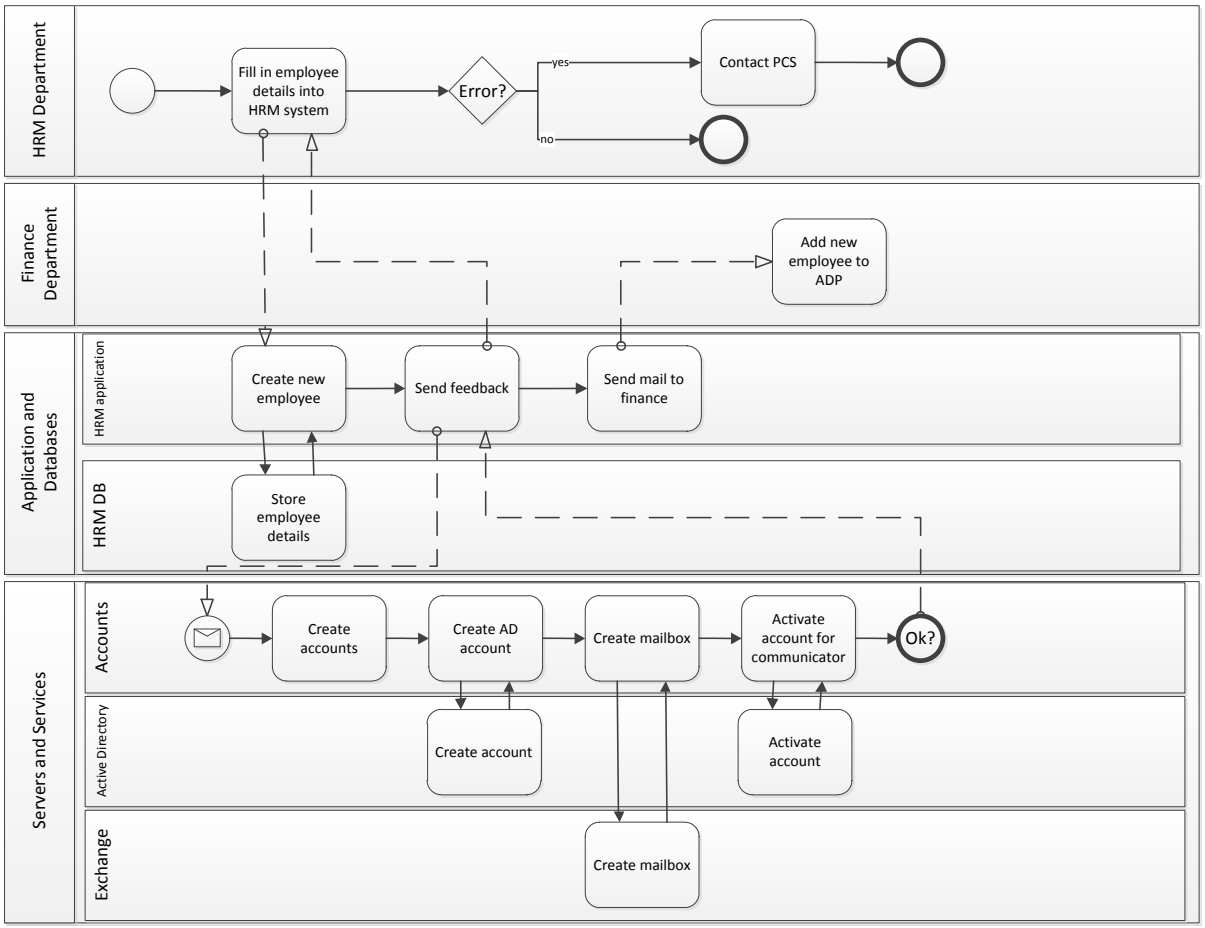

Figure 24: Promote applicant to employee

E. Specification of viewpoint 3: Component interactions

The main topic in this paragraph is the lowest level of abstraction of the current architecture. The component choreography is presented in this section by giving one example; all other choreographies are available within Tam Tam internal documents.

Active Directory: Accounts web service – new Active Directory user

When a new employee is added to the HRM system, said HRM system calls the Accounts web service to create accounts for the new employee. The creation of these accounts is done by inferring several functions on the Active Directory server. The entity

'TamTam.HRM.Management.Powershell' is the executing class of the accounts web service and carries out the following commands as can be seen in Figure 25:

1) Connect to the Active Directory server
2) Create a new Active Directory user
3) Verify that the user is created correctly by invoking the 'Get-QADUser' function, if this method call returns a valid user it means the creation was successful
4) After that the newly created user account should be enabled
5) The last step is to add the employee to a certain group to grant pre-defined permissions and apply a pre-defined security policy. A group can be defined as a department within Tam Tam
6) Lastly the connection to the Active Directory server is closed down again

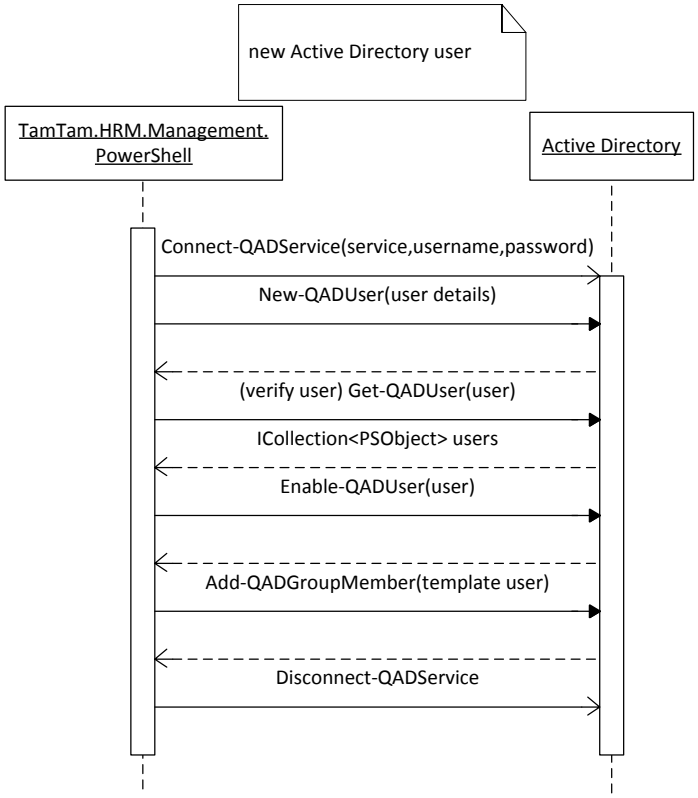

Figure 25: Active Directory: accounts web service - new Active Directory user

F. Specification of viewpoint 1: Global overview

The last step is putting all gathered information together into one global overview of the current architecture. Figure 26 presents the data from the last sections aggregated in one big overview. All three different categories of components have their own color; databases are green; applications are red; and services and supporting peripherals are blue. A non-dashed arrow from A to B means that component 'A' initiates an interaction with component 'B' indicating a coupling between the two components. Dashed arrows indicate a stream of information, either through linking two databases or exporting files to certain locations.

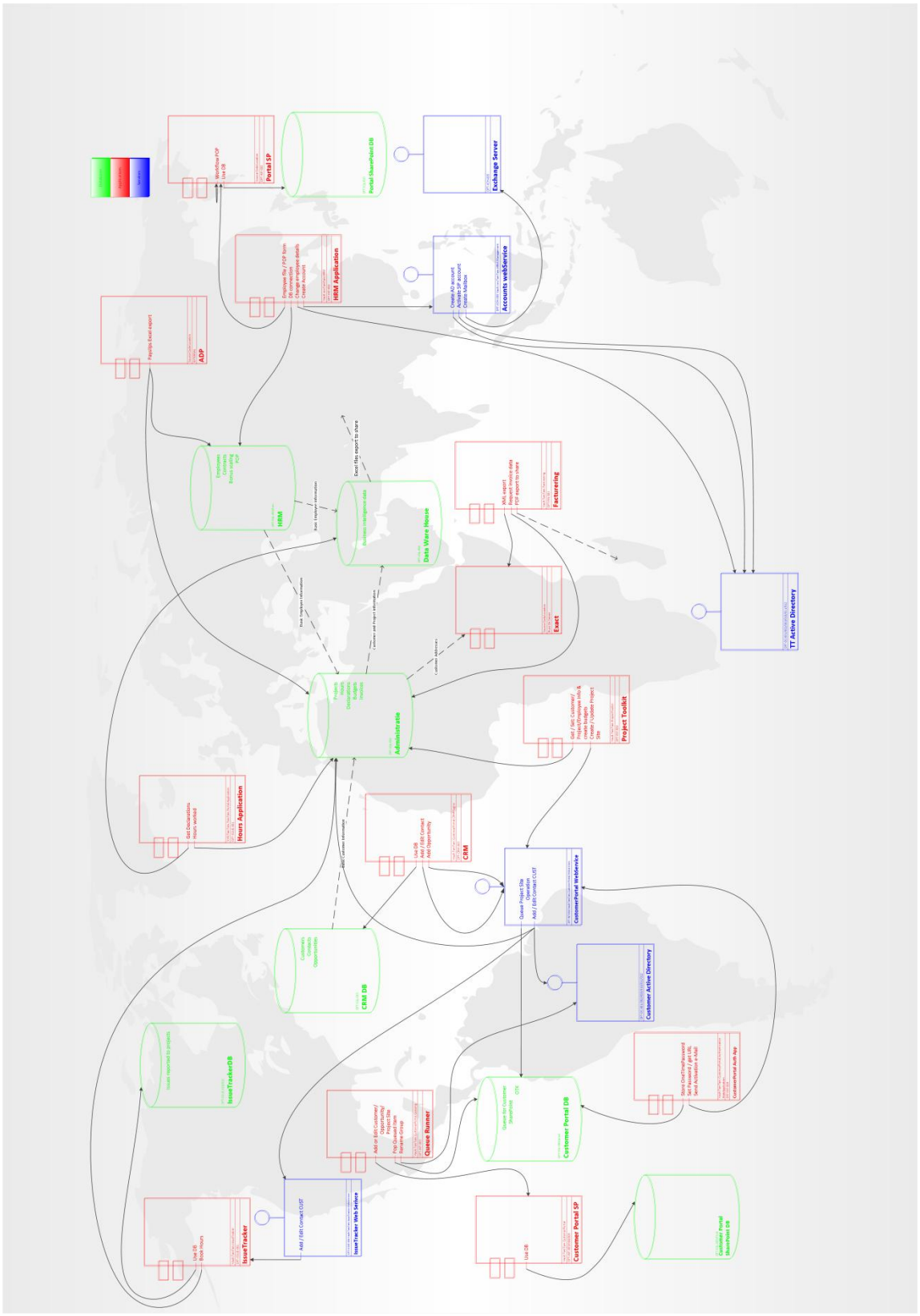

Figure 26: Aggregated overview of current Architecture

Appendix B: EAI styles

Enterprise Application Integration can be seen as the evolution from ad-hoc architectural decisions to a situation in which certain guidelines are followed by crafting a general idea behind the enterprise architecture. The main purpose of crafting, and adhering to, such a general idea is to "create a seamless whole" (Johannesson & Perjons, 2001) of the architecture in place supporting enterprise operations.

According to (Lam, 2005), reasons for companies to create a transition towards an EAI are because with ad-hoc architectures the adaptability to new business requirements is fairly low, the maintenance costs of interfaces kept increasing and the ripple effects of application changes were increasing. (Erasala, Yen, & Rajkumar, 2003) lists other advantages for EAI in the form of better adaptability to electronic commerce opportunities, easier mergers and consolidations and a free choice for departments for software solutions they want to integrate.

As can be derived from the requirements in section 2.4, the main goals for Tam Tam to initiate a transition towards an EAI are first of all to increase maintainability by defining a standardized way to implement component interfaces and interactions. A second reason for Tam Tam is to gain insight into what components are used for what processes to decrease the effort needed to investigate when some component is offline. A third reason is that a carefully crafted EAI can reduce coupling between the available components by disabling the possibility for direct calls to component functionality.

Several different EAI implementation types are available in the relevant literature; this appendix provides an overview of the commonly used types by presenting their key characteristics and the general idea behind the different types.

A. Point-to-Point

The first EAI implementation type elaborated on here is the Point-to-Point architectural style. Key characteristic of this topology is that there is no general idea behind the architecture composed of the components in place. The reasoning behind this kind of topology, as (Trowbridge, Roxburgh, Hohpe, Manolescu, & Nadhan, 2004) and (McKeen & Smith, 2002) put it, is the need for components to be coupled. This coupling is performed by coding the connections on a one-by-one basis by including the location and specifications of the called components in the calling components. The advantage of this is that it is very easy to implement interaction between several components. With a small number of components this stays an advantage, but with an increasing number of components the effort needed for maintenance is getting bigger and bigger. Reasons for this maintenance penalty are a lot of duplicate code (Trowbridge, Roxburgh, Hohpe, Manolescu, & Nadhan, 2004) and the fact that the number of interactions becomes overwhelming (Johannesson & Perjons, 2001) causing a lot of components need to be changed in the case the details of one component are changed.

B. Middleware

To tackle the drawback of a large amount of interactions, a step in research was taken towards middleware. The basic idea of middleware is to create an abstraction layer on top of the raw connections layer; this can be done by either creating a central entity that handles all connections between components, or standardizing the mechanism for communicating with called components. Advantages of such middleware according to (McKeen & Smith, 2002) are that the middleware specifies a standard way of interacting between components and allows the integration of otherwise disparate components. The main downside of this architectural form is that the software that represents the middleware can become a legacy system itself in a couple of years, thus diminishing its effectiveness. Several forms of middleware are available and are discussed one by one.

Transactional (Emmerich, 2000) (Papazoglou & Ribbers, 2006a)

Transaction-oriented middleware present a means for components to carry out transactional operations over a number of distributed components. Key idea behind transactions is that those operations are either fully executed or not executed at all, as defined in the ACID properties (Atomicity, Consistency, Isolation and Durability) (Wikipedia Community, ACID, 2011).

Message oriented (Emmerich, 2000) (Papazoglou & Ribbers, 2006a)

Message oriented middleware is based on the 'hub and spoke' idea (Papazoglou & Ribbers, 2006a) which consists of a central entity that keeps track of all messages being sent and ensures the messages are delivered to the correct components. The key use of such a system is when asynchronous communication is needed. Main reasons for this are that the central entity needs to convert messages to a format suitable for the called component and the central entity is responsible for the messages to arrive at their destination, even if that destination has some downtime. As (Johannesson & Perjons, 2001) put it, the use of a central entity reduces the number of connections necessary by creating a message broker to which all components have a connection.

Procedural (Emmerich, 2000) (Papazoglou & Ribbers, 2006a)

Procedural middleware is often represented by Remote Procedure Calling. The key idea behind this is to allow access to certain methods and functions from other components. This is done by creating an interface for components with pre-defined functionality that can be used in other components. These interfaces are represented in the source code of calling components by stubs that act as proxies for the connection between two components. The server side has stubs too for returning the results of the called functionality.

Object and component (Emmerich, 2000) (Papazoglou & Ribbers, 2006a)

Object oriented middleware is an evolution of Remote Procedure Calls by providing the extra functionality of inheriting object-oriented principles. With such middleware in

place, components have the option to send objects back and forth to other components. The objects can be seen as an advanced version of the stubs in place with Remote Procedure Calls, but additionally these can be sent across the network. An additional advantage is that methods executed at remote components can have objects that refer to a third component as a return value.

Data-access (Papazoglou & Ribbers, 2006a)

The last type of middleware is the data-access type. The main idea behind this type is to create an abstraction layer on top of the data in place. This means that access to data in databases and other components is performed with a single defined syntax through one single manner of interaction. The advantage of such an abstraction is that calling components do not need specific knowledge for specific data storage decisions.

C. Event-driven

The event-driven approach is based on events that trigger some functionality. This can be done in two different ways, firstly by centralizing the process knowledge and secondly by decentralizing the process knowledge. The first approach is an extension to the message oriented middleware approach by increasing the functionality of the central entity to also contain the process knowledge (Johannesson & Perjons, 2001). When a process needs to be executed, an event is sent to the process broker which in turn calls all components that should cooperate in successfully executing the desired process.

The second approach is to have the process knowledge in the individual components. The way a process is executed using this approach has two different options. The first is by using a message bus to which the event is posted and all other components listen to. The individual components know how to react to certain posted events. The second approach is by using the publish/subscribe methodology (Papazoglou & Ribbers, 2006a) and (Trowbridge, Roxburgh, Hohpe, Manolescu, & Nadhan, 2004) by letting components subscribe to certain events. In the case an event is given to the central entity, this central entity knows which components to notify.

D. Service Oriented Architecture

Service Oriented Architectures are based on the idea that computational logic should be divided into autonomic entities which have a predefined objective (Goel, 2006). These entities are called services and are distributable and loosely coupled with defined standards-based interfaces (Microsoft Patterns & Practices Team, 2009). These services should also have the characteristic to be most general (Papazoglou & van den Heuvel, 2007) because their interface can be inferred from all places, both within a company as outside. This means no knowledge should be needed within services about calling components.

Several roles play an important part in Service Oriented Architectures; the first is the service provider which provides a service. The second role is the service requestor, an entity that wishes to use a certain service. Lastly, the third role is the service registry that acts in a similar way as the 'yellow pages' by allowing service providers to publish

services and service requesters to query for certain services. These three roles are performing three different operations that indicate the lifecycle of service usage: publishing services, requesting services and binding services. (Papazoglou & Ribbers, 2006a)

Plenty of advantages of Service Oriented Architectures can be found in the literature. The main category of advantages is found in the field of development of components for a SOA environment. (Goel, 2006) states that "SOA brings cost effective, reusable and low lead time solutions to an organization" meaning it enables developers to create components and implement interactions more easily. (Microsoft Patterns & Practices Team, 2009) endorses this idea by stating that developers with no prior knowledge should be able to use interfaces of components faster and that, in principle, service implementations can be purchased from external parties. Next to that, SOA allows an incremental development and deployment process (Natis, 2003), because all referenced services can be added one at a time. (Natis, 2003) also states that maintenance and extension of business applications can be done more easily because the services are loosely coupled. The other field of SOA advantages is that SOA sets the first step in allowing companies to move towards other technologies, for example (Channabasavaiah, Holley, & Tuggle, Jr., 2003) identify that SOA can be seen as the first step towards grid computing and on-demand business. (Microsoft Patterns & Practices Team, 2009) extends said possibilities by stating that the step from a SOA to a cloud-based solution is very small. Furthermore, a SOA has an effect on every aspect of a company for it facilitates a great amount of agility to business processes (Durchslag, Donato, & Hagel, 2001).

E. Enterprise Service Bus

The origin of Enterprise Service Busses (ESBs) can be found in the combination of middleware, web services and orchestration technologies (Papazoglou & Ribbers, 2006a) & (Menge, 2007). An ESB is comparable to a middleware message bus by providing a connectivity layer between services and has two main tasks: 1) managing meta-data about endpoints and 2) matching endpoints (Schmidt, Hutchinson, Lambros, & Phippen, 2005). More technically speaking, the ESB is a messaging infrastructure that performs protocol conversion, message transformation, message routing and accepting and delivering messages (Goel, 2006). Due to this one single place in which all interactions are coordinated a single point of contact is created for external entities to use internal services (Keen, et al., 2004).

The advantages of an ESB are that the ESB coordinates all interaction between components by forming the one place in which process orchestration is stored (Menge, 2007). This coordination is based on the incoming message content, origin, destination and some predefined rules (de Leusse, Periorellis, & Watson, 2007). Another advantage stems from the nature of the message transformation capabilities; due to this, many different protocols can be used to interact with the ESB allowing components with otherwise incompatible message formats to interact (Microsoft Patterns & Practices Team, 2009). The only downside is that due to the message handling functionality the interactions should be performed asynchronously (Papazoglou & van den Heuvel, 2007).

Appendix C: Key design elements

This appendix gives examples of the specific details of the designed architecture. The first is a list of identified services, the second are a few example BPEL documents and the third is an example of how the process of binding the web services is performed.

A. Identified services

The fundaments on which all orchestrations build to provide aggregated functionalities are the methods offered by several services. The services that are designed are listed in Table 17 with their functionalities, outgoing connections and main sources where their functionality lies in the current architecture. The specific implementation details can be decided on at implementation time or when a service is replaced in the future, as long as the implementation supports publishing its interface as a WSDL document.

Table 17: Designed services

Service	Functionalities	Outgoing connections	Main existing sources
Active Directory	User account control	Customer and Tam Tam AD domain	CustomerPortalWebService, HRM application, AccountsWebService
Admin DB	Project operations	Administratie DB	CustomerPortalWebService, Project toolkit
CRM	Account name access	CRM DB	Already existing service, included with application
Customer Data	Customer information for use mainly in finance processes	Administratie DB	CustomerPortalWebService
CustomerPortal SharePoint	Opportunity and project management	Customer Portal SharePoint	CustomerPortal Queue executor
Data Warehouse	Retrieve collected declarations	Data Ware House	Hours application
Exchange	Handles employee mailboxes	Exchange server	AccountsWebService
Finance	Handle budgets, invoice data, booked hours and export invoices	Administratie DB	Project toolkit
IssueTracker	Manages issuetracker user accounts	IssueTracker DB	Partly existing, CustomerPortalWebService

B. BPEL Examples

In essence there is a continuum of different processes to be identified between purely synchronous and purely asynchronous. In this part the design of three different process orchestration types will be covered. The first is saving a budget, a synchronous orchestration that passes messages back and forth between the requested service method and the calling application. The second is the creation of a new opportunity, a purely asynchronously executed orchestration. The third is creating a new project which is a hybrid orchestration, in other words between the two ends of the continuum.

Synchronous: Saving a budget

To save a budget, the project toolkit needs to have an interaction with the Finance Web Service method 'saveProjectBudget'. Figure 27 shows the created pass-through orchestration for storing a budget. When the orchestration is called, four steps are executed. First the input message is mapped to a service request message (see Figure 28). The next step is to invoke the saveProjectBudget web method resulting in a response message. The third step is to map this response message to the orchestration output message. The fourth step is to return this output message to the calling service.

The quality of service binding details of this specific orchestration are that only one instance of the orchestration can be created and calls that come in while the instance is still running should be queued. Otherwise concurrency issues might arise.

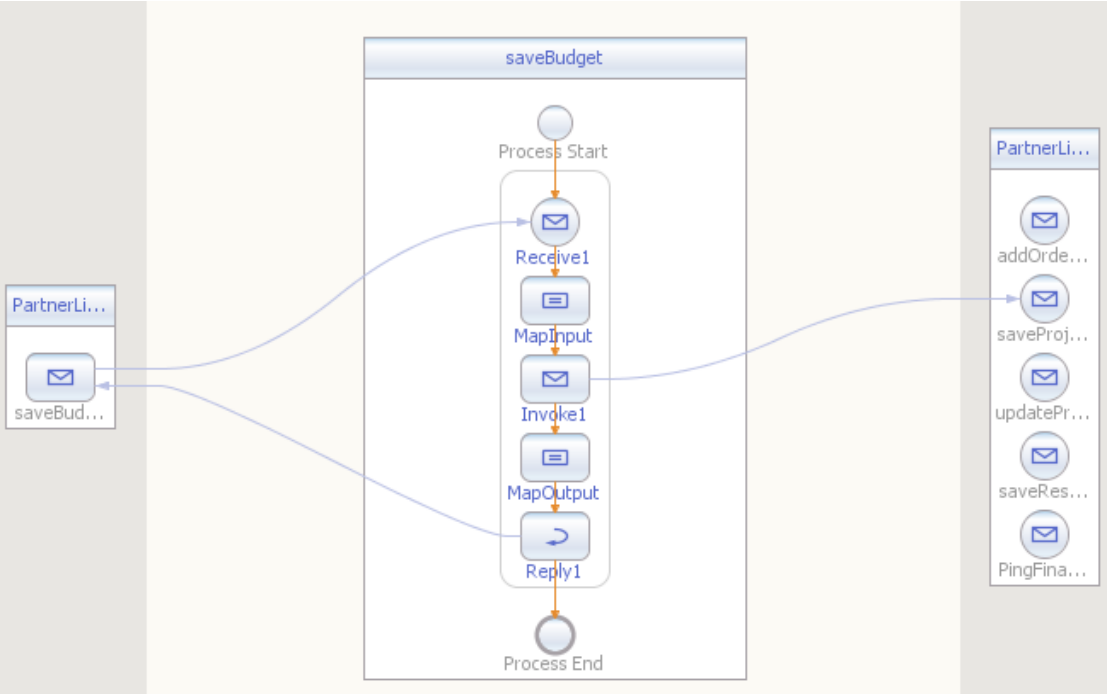

Figure 27: Pass-through orchestration

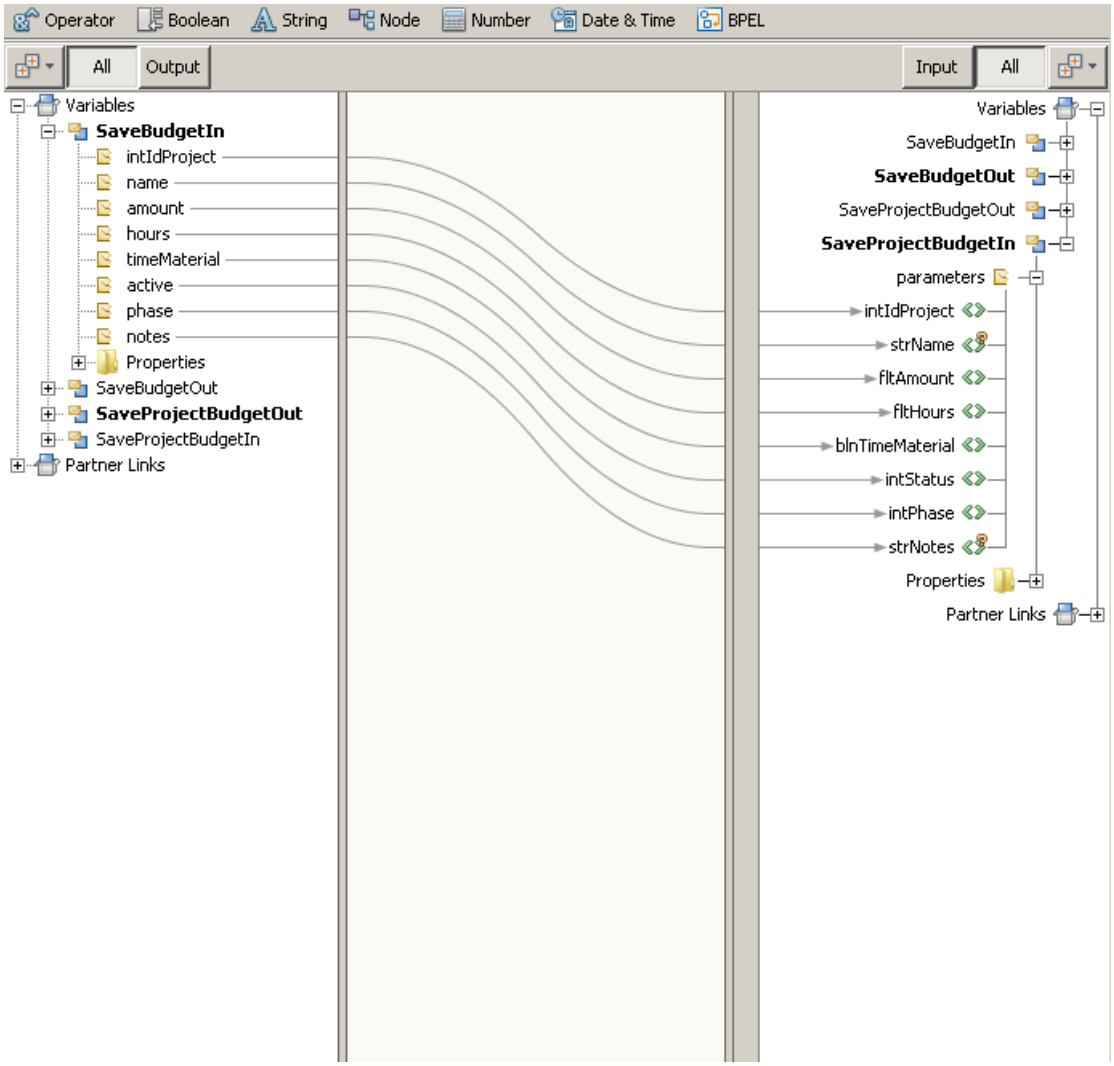

Figure 28: Message mapping

Asynchronous: Creation of a new opportunity

The next orchestration is a purely asynchronous process, as shown in Figure 29. When the orchestration is called, a new instance is created and the control flow is given back directly to the calling application. The application can continue its operations while the orchestration instance in the BPEL-engine lives on to complete the necessary operations.

When binding this type of orchestrations it is possible to use instances of the orchestration as queue entries by removing the instance limit from the quality of service values for the incoming request. To overcome the hurdle of concurrency issues, the outgoing requests can be constrained with quality of service demands. In this example the first call to retrieve some information need not be limited, but calling the SharePoint service method should theoretically be limited to one request at a time because the functionality changes things, but in practice this is not necessary because the underlying technology only allows one operation to be executed at the same time.

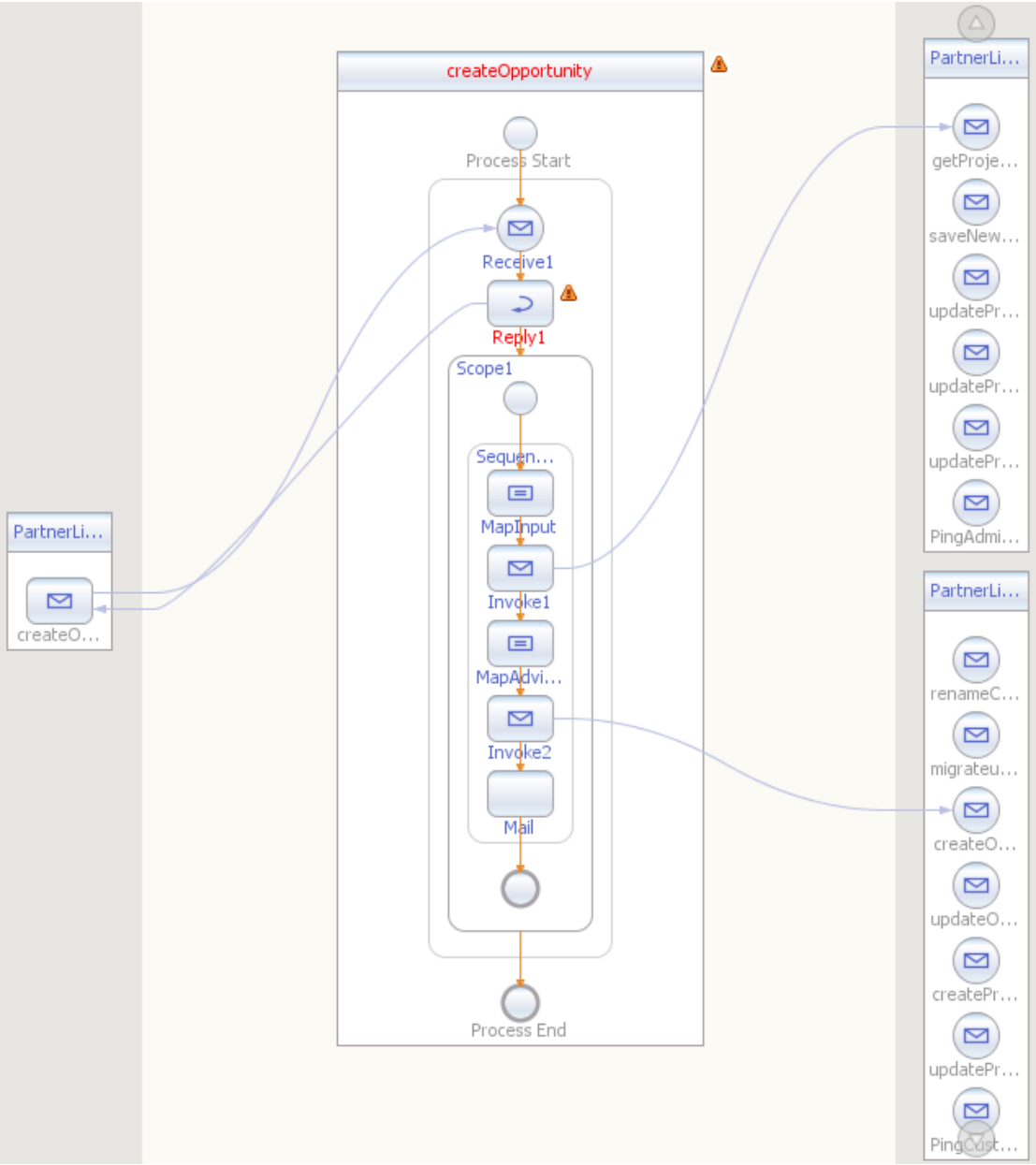

Figure 29: Asynchronous orchestration

Hybrid: Creation of a new project

The last of the discussed orchestrations is a hybrid form of the last two. The presented example deals with the creation of a new project. When a new project is created the project ID of the newly created project should be returned to the called application directly. The creation of this project ID is performed in the first invoke action in the depicted orchestration in Figure 30. After the project ID is created by the CustomerDataWebService, this id is returned to the calling application and a method of the CustomerPortalSharepointWebService is invoked to create a site for the new project.

With regard to binding this orchestration, the quality of service values for the SharePoint connection should be limited. Furthermore, the incoming request should be limited by allowing only one request-response path to be executed at the same time

(when a response is returned, new instances can be created because concurrency is evaded by the quality of service values of the second invoke action).

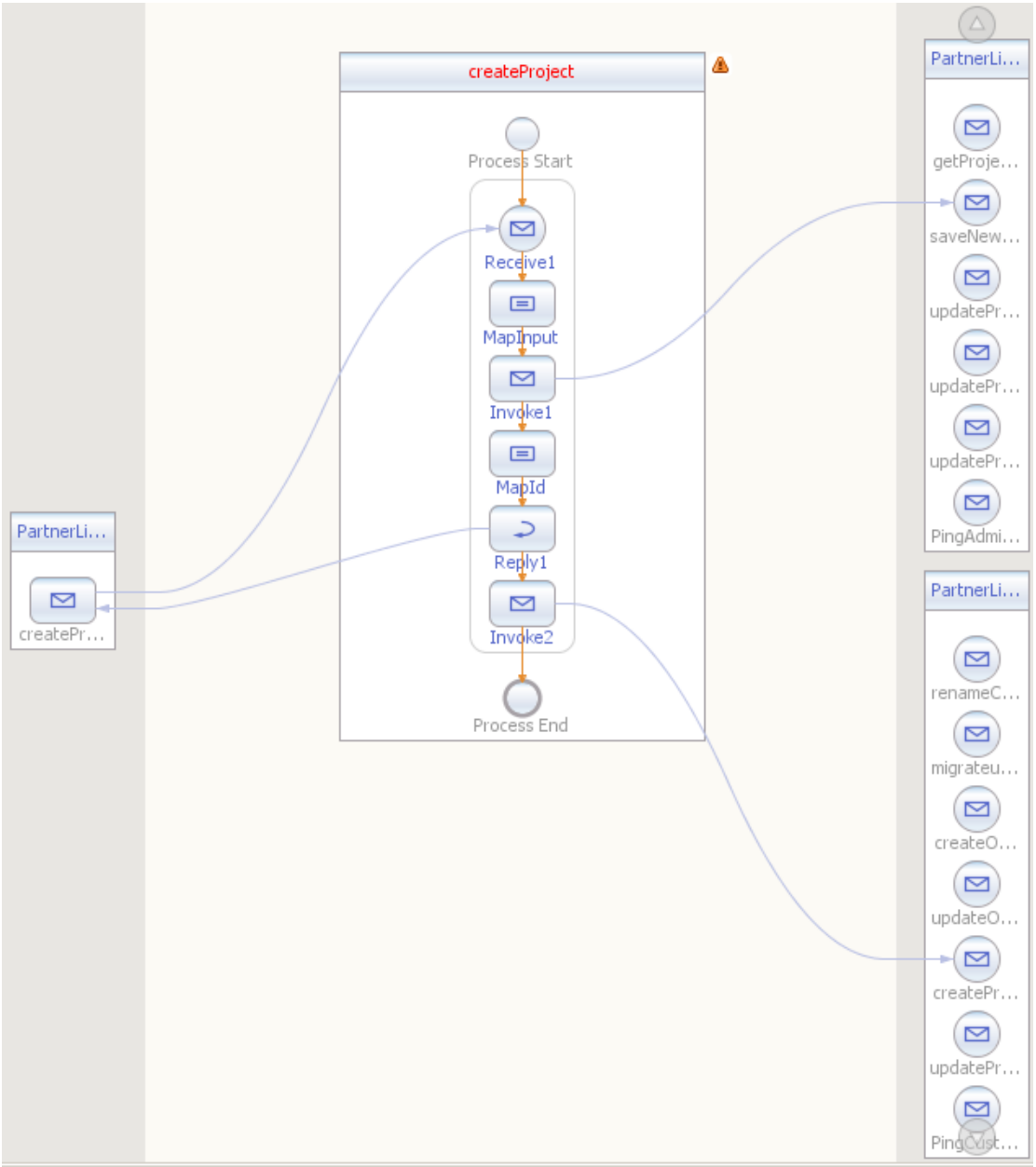

Figure 30: Hybrid orchestration

C. Binding web service example

As an example of binding an orchestration, Figure 31 shows how a binding can be performed in 'OpenESB' by creating a CompositeApplication which can be loaded in the 'GlassFish ESB' server. The actions that need to be performed are:

1) Import BPEL and WSDL documents
2) Connect outgoing method requests to imported WSDL specifications
3) Assign QoS values for connections, examples of which are: max number of retries, waiting time and max concurrency limit

4) Publish the CompositeApplication to the GlassFish ESB server

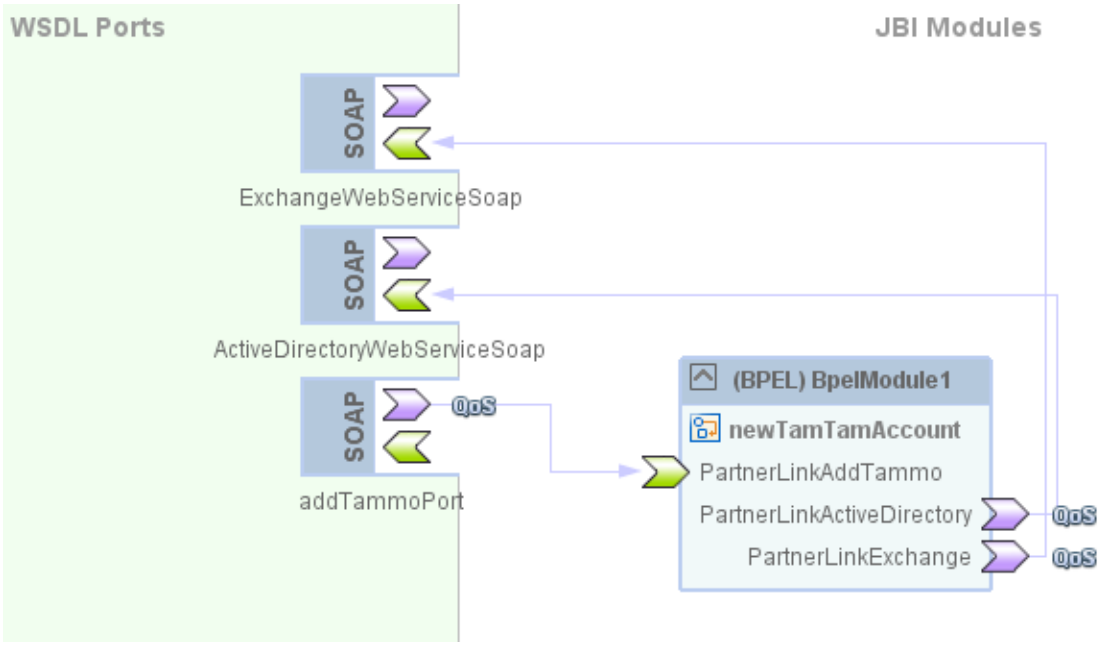

Figure 31: Binding an orchestration

Appendix D: BizTalk risk mitigation plans

This appendix describes the results of reflecting on the decision for BizTalk, the first section provides a table with all risks and their possible mitigation plans. The second section provides some details about the amount of overhead introduced with BizTalk.

A. BizTalk risk mitigation table

Table 18: BizTalk Risk mitigation plans

Name	Description	Possible	Mitigation plan
Developing	Functionalities that might be longed for while developing orchestrations		
Developing on own Machine	Is it possible to develop for BizTalk on another machine and deploy to the BizTalk server	Yes	1) Develop and build on own machine 2) Deploy orchestration to BizTalk server 3) Deploy assemblies to the GAC 4) Run WCF publishing wizard
Documentation	Documentation of functionalities offered is very important	Yes	Next to a specific design document, an UDDI server contains necessary information to consume orchestrations and services
Implementation details	What are the details of the implementation	Yes	An UDDI server is deployed in which all details are stored regarding owner, creator, description, etc.
Separate test environment	Next to the live environment another environment can be necessary for testing	No	The nature of an ESB allows multiple versions of orchestrations to be run at the same time; still it can be useful to create a separate testing environment when the implementation is in use.
Failure handling	What happens when things fail during execution		
Exceptions	How should exceptions be handled	Yes	1) BizTalk->services: retries after a specified time 2) Client->BizTalk: manually, watch out for timeouts due to BizTalk retry delay

Logging	To what level do logs specify what went wrong	Yes	When tracking is enabled, everything executed action is logged (both pre and post actions)
Resume on error	How does BizTalk recover from SQL failures	Yes	BizTalk is message oriented and everything is stored in the database. ESBs serve a contract to guarantee a delivery. The only issue is the duration it takes is not specified.
Rollback	How can started orchestrations be rolled back on the initiative of both administrators and failure to comply	Partial	Can be performed by calling created counterparts of service functions
Statistics	Is there a way to show statistics about carried out orchestrations	Yes	The BizTalk Administrator console provides elaborate statistics
Orchestrations	Additional functionality to be used in future orchestrations		
Date encoding	How dates are encoded; do things succeed when different formats are used	Yes	Dates are mostly formatted using UTC[10]
Delay & Pre conditions	Can an orchestration wait for an event before continuing	Yes	Listen shapes wait for specific events; Delay shapes delays the execution for a specific amount of time
Complex data types	How can complex data types be send over 'the wire'	Yes	Define Data Contracts in WCF services; Only use serializable elements
Loops with condition	Can loops be used in orchestrations to handle multiple elements one at a time	Yes	Cannot be done visually, but by programming some functionality[11]
Recursion	Can Recursion be used	Yes	Loop functionality is available, plus orchestrations can be invoked from other orchestrations
Send mail	How can mails be sent from orchestrations	Yes	1) Dynamic ports allow binding to specific recipients

[10] http://geekswithblogs.net/michaelstephenson/archive/2010/08/29/141542.aspx

[11] http://blog.eliasen.dk/PermaLink,guid,6c7ac8ec-3f3e-49e4-a15a-76c736d30654.aspx

			2) Promote fields in schemas to be used in mail (Message parts of called web services is harder to achieve) 3) Programmatically compose mail
Stop & go	When does an orchestration instance know when to continue	Yes	Listen shape waits for events, in the case of resume by failure it just waits until the retry period is over
Daylight saving time	What happens with the 'spring forward, falls back' events	Yes	Since BizTalk 2006 there were some issues fixed with the delay shape and daylight saving time
Workflow	Can the human parts of processes be included in orchestrations too	No	Not in a neat way, BizTalk offers system-to-system integration, for human parts Windows Workflow Foundation and SharePoint should be used
Security	Security considerations at all levels of abstraction		
Access restriction	Some actions may only be performed at night	Yes	Service windows can be set to ports in which no operations are executed with those ports; Furthermore, receive ports can be disabled during a specified range of dates
Encryption	How can the send data be encrypted	Partial	Use HTTPS, because HTTP is always vulnerable to man-in-the-middle attacks
General Authentication	How to limit access to using elements in the architecture	Yes	Host WCF services behind HTTPS
Method level Authentication	How to limit access to specific functionalities	Partial	Create a Single sign-on application for each WCF service with a corresponding 'users' group in the Active Directory. Problem is that ACID[12] cannot be guaranteed anymore when different services are called
Usage	Functionalities surrounding the usage of orchestrations		
Concurrency	What happens with multiple orchestration	Yes	Send ports have the option 'ordered delivery' which takes care of the correct sequence of

[12] http://en.wikipedia.org/wiki/ACID

	requests on the same service		function invokes
Orchestration priority	Is it possible to have some orchestrations that are handled earlier than others	Partial	Only send ports can be given a priority over others, allowing a separation of functionality that needs to be executed quickly on a web service level
Restriction on license	What are the restrictions to use BizTalk for Tam Tam internal systems	Yes	"Just use it, we are allowed to use it for testing and development"
Scheduling	Specify times at which orchestrations are invoked automatically	Yes	Let Windows Scheduling put an XML in a folder monitored by BizTalk
Stress and Load	What overhead is introduced by starting to use BizTalk	Partial	Some stress and load tests are performed, see section B BizTalk stress and load tests for the results
Web Services	Can the scope of web services be broadened		
ASP.NET support	Does BizTalk 2010 support legacy web services	Yes	Can be done by a somewhat cumbersome solution; mappings between messages have to be programmed manually instead of using the visual interface
Binding options	Are other binding options available next to ASP.NET and wsHTTP	Yes	The created metadata when consuming a web service is not linked to specific bindings; Ports can be bound to any adapter
External web services	How to comply with changes in web services not under Tam Tam maintenance	Yes	1) Interface change: same actions as internal services 2) Location change: firstly change in the UDDI server and secondly in the specific send port
Service description pages	How to declare what an endpoint offers and who is responsible for it	Yes	An UDDI server is used in which descriptions, owners and maintainers can be attached to web services

B. BizTalk stress and load tests

For testing the overhead that BizTalk introduces, three different tests have been run. The general idea behind the tests is that a dummy web service method is invoked, in two cases via a proxy orchestration and in the third case directly from the server running the orchestrations. Figure 32 illustrates the setting for the tests.

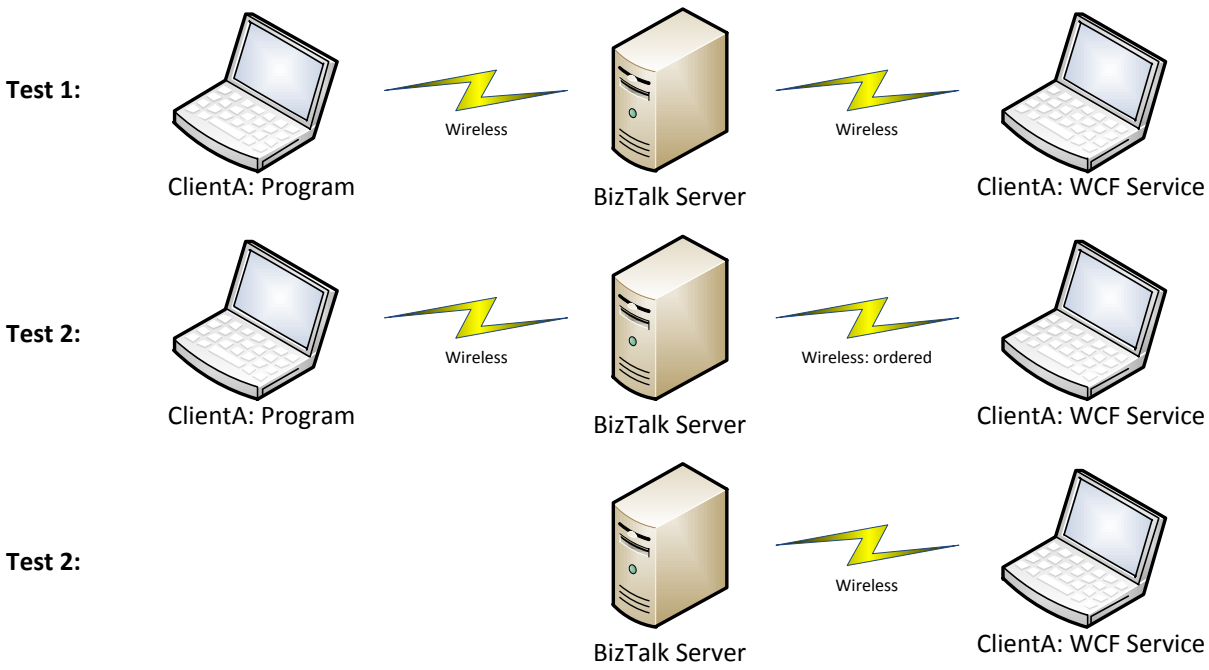

Figure 32: Stress test organization

For each test setting a program is executed that has a number of threads simultaneously calling the orchestration (or web service directly in the third test) 52 times in a sequence. All data about how long each call takes is accumulated in Excel documents and used to create graphs by performing the following steps:

1) Delete the first five and last five values of each thread, because a random wait is introduced before the first request is made and some threads can finish early and thus provide a wrong image.
2) Some key values are calculated over the listed data: median, first and third quartile & minimum van maximum value
3) The calculated values are used to create the graphs shown in Figure 33, Figure 34 and Figure 35. The values on the left indicate the range for the medians, the values on the right indicate the range for other values and each column identifies the number of threads at the same time. Furthermore, all used values are in milliseconds.

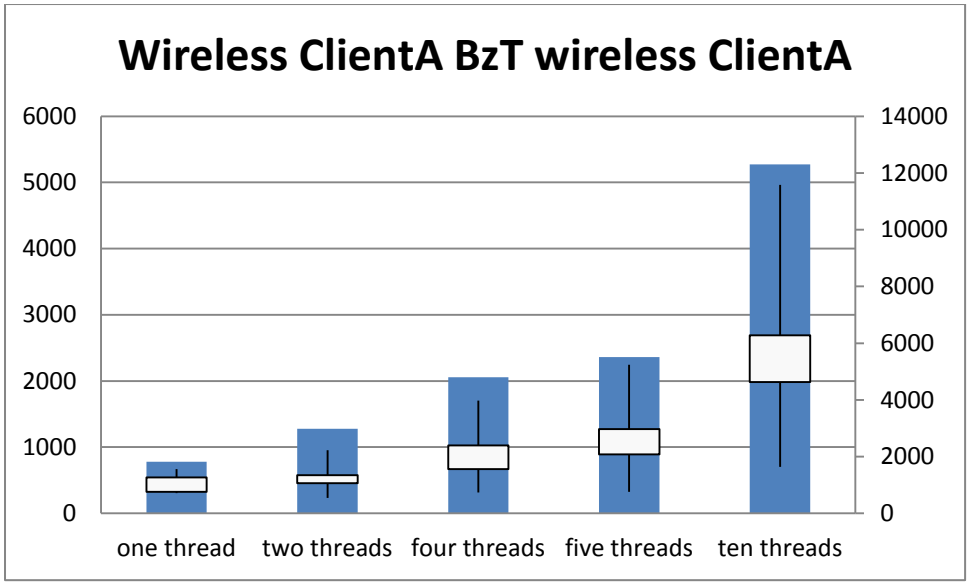

Figure 33: Wireless Client to BizTalk to Wireless Client Web Service

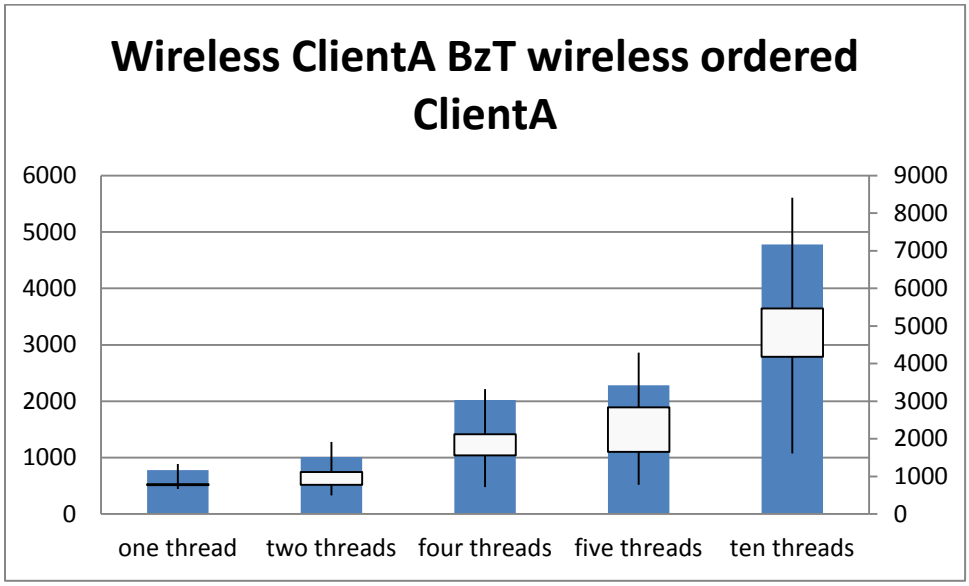

Figure 34: Ordered Wireless Client to BizTalk to Wireless Client Web Service

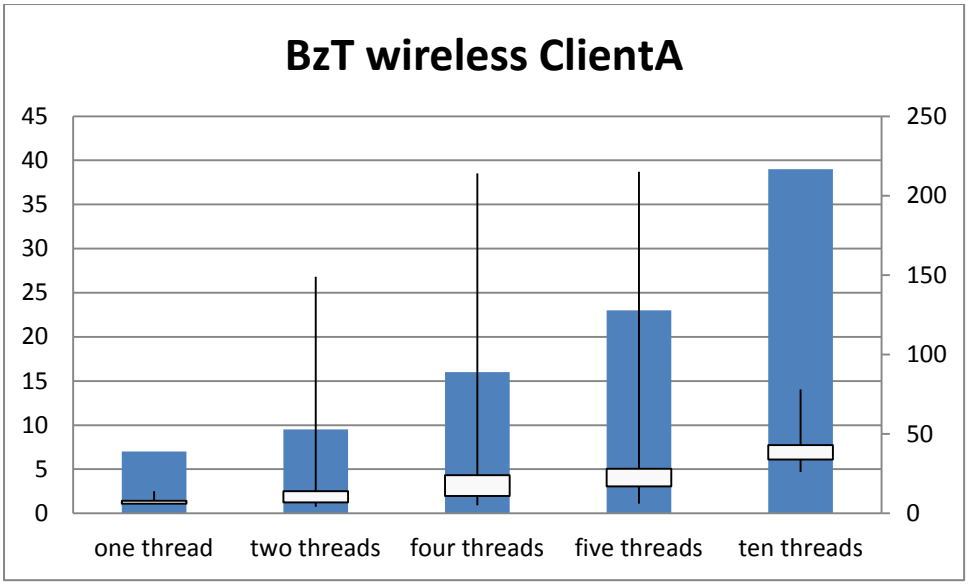

Figure 35: BizTalk to Wireless Client Web Service

Conclusions

When comparing the results for all three tests, the following conclusions can be drawn:

1) The introduction of BizTalk to act as a proxy causes a big increase in access times. This is mainly due to the fact that everything in BizTalk is stored in a database, so a lot of database access is performed for each request causing extra time necessary to execute.
2) When ordered delivery is used in BizTalk, the time needed to execute decreased. The reason for this is that only one worker thread contacted the web service from BizTalk, resulting in diminishing waiting times for no other threads are using the web service (#-1).
3) When more threads are invoking requests at the same time, the average waiting time with the proxy in place increases because of the queuing mechanism. In the case there is no proxy, the increase in waiting time is caused by hardware limitations for there is no regulation on how many requests are executed at the same time at the service location.
4) Even in the highly unlikely event when ten requests are executed at the same time, the average increased time for processing to finish is about five seconds.

As a wrap-up the overhead increased by using BizTalk is not a show stopper since most processes that are to be executed by BizTalk are asynchronous. This prevents the end user to notice the overhead by increasing the waiting time.

*#-1 **Queuing mechanisms** slow down individual requests, but the global optimum will be of improved quality*

Appendix E: Evaluation iterations

This appendix lists all four different carried out evaluation iterations at key points during the thesis project.

A. Evaluation iteration 1 – Current Architecture

After having defined the means and metrics to evaluate architectures, this is the first of several evaluation iterations. The focus of this iteration is to evaluate the performance and the possibility to manage the architecture currently in place. Measuring values for the identified metrics and interpreting the scenarios can be performed because the current architecture has been documented completely. This means an explanation how a certain scenario is carried out can be estimated by the current knowledge about the architecture. The second step of this section is estimating the values for the individual metrics by looking at how much time it has cost to gain an understanding from scratch.

Scenario analysis

Scenario 1: Addition of an extra component

The example used in this scenario is the addition of a component with new functionality, for example a dynamic 'handover' documentation system that contains all implementation details about projects. Such a system should contain data for all projects which is mainly kept up to date by hand, but when new projects are created the new system should be updated as well.

In the current situation the developer of the new system should search for the specific components that add or update projects and insert some function calls to the new system. Components that might be affected in this case are the project toolkit and the CustomerPortal Web Service. The time it takes to perform this action is the time necessary to understand both components and the process orchestration. After gaining an understanding the function calls should be inserted at the correct location.

Scenario 2: Moving a component to another location

Sometimes the live location of components might need to change when, for example, a new physical server should be used to run a specific component. In the case this happens, the hardcoded references to the old location should be updated to reference the new server.

In the current situation a lot of hardcoded connection configurations are present, causing the developer to search for those and alter them where necessary. This is a very time consuming operation because there is no documented knowledge about which component is used where.

Scenario 3: Extension / withdrawing functionality of a component

In the history of the current architecture the operation of splitting a component into two separate components has occurred several times. When such an event takes place, incoming function calls to the old component should be updated to call a specific new component.

In the current situation when the functionality of a specific component needs to be split into two different components all referring components need to be adapted. The most time consuming operations are figuring out which calling components to change and after that updating those components.

Scenario 4: Carrying out maintenance on a component

When a bug is found in one of the components and needs to be fixed, a lot of work is put into reproducing said bug.

In the current situation the developer should first of all search for the live location to reproduce the bug. After that, the source code should be retrieved and an inspection should take place if the bug stems from this particular component or from the interaction with another component. Due to the nature of the interactions that are currently in place, it might be cumbersome to find all interactions that can cause the bug.

Scenario 5: Change the functionality of a component

The previous scenarios all focused on local change of functionality. In the case some functionality changes which has an effect on the calling components, for example an additional necessary parameter to all requests, more elements need to be adapted.

In the current situation when such a change is necessary, the component itself should be adapted and all calling components need to be adapted as well. The main time consuming activity is searching for the source code of all components and searching for places in which an interaction with the changed component is present.

Scenario 6: A process cannot be executed anymore

When a process stops working, some investigation needs to be performed to find out the root cause of this failure.

In the current situation there is no documented knowledge about what components contribute to executing specific processes. Therefore to fix the failing process, all components which are suspected to have an influence on the broken process are to be inspected.

Scenario 7: Replace a component

It might occur that a component needs to be replaced by another version or a whole different component that offers similar functionality. For example Tam Tam could stop

using Microsoft CRM as a relationship management application and start using another package.

When this happens in the current situation, the developer should retrieve all locations where direct calls to the CRM application are present and update those to refer to the new package. Furthermore, all calls from the CRM application should be recreated in the new product.

Scenario 8: A certain process needs to be extended

It is not unforeseeable that some processes might need to be extended. An example of this, which came forward in a discussion with an account manager, is the creation of a special folder for each new customer project in Tam Tam Dropbox account.

In the current situation the developer needs to find out where the needed process is initiated and where it is carried out. When those locations are found, the developer can decide where to put the new operation calls. With each new addition, the decentralization of process logic can be increased due to this freedom for the developer.

Scenario 9: A server is going offline

When a specific server needs to be taken offline, the actor performing this action should be sure no critical components are running on it anymore.

In the current situation, documented knowledge of live locations is not complete. This means that it is very hard to determine which components are running where in the landscape of internal processes.

Metrics valuation

This paragraph consists of an elaboration on the values identified for the metrics with respect to the current situation. A key assumption taken here is that the values are listed for a developer with no prior knowledge about the architecture or components at all. At the end of this paragraph Table 19 gives a summary of the identified values

Number of interaction schemes

In the current architecture several different forms of interactions are present: Web service calls, Active Directory PowerShell calls, SharePoint interactions, Exchange method calls, database stored procedures and direct SQL invokes.

Number of hardcoded connection strings

In the described current architecture documentation a total of 21 different application interactions can be counted. Each of these application interactions is enabled by storing a connection string in the location which uses them; this means that there are 21 places in which a hardcoded connection string is placed.

Number of incoming method calls per component

An average of 2.2 different calls from source components is present. This means that when a component is changed, invokes from about 2.2 different other components need to be changed. The effect of this is that when a random component changes, about three different invoking components need to adapt their functionality.

Number of components with related functionality (duplication)

This metric is about the total number of components that share some functionality. This is measured by looking at the source code and identifying chunks of code that resemble the same functionality. In the current architecture at least 8 duplicate chunks of code can be identified. This means that when a bug is solved around a component invocation, it does not necessarily mean said bug is completely gone.

Speed of source code retrieval

Before a developer can start to perform any maintenance on a component, the source code location should be retrieved. Tam Tam uses three different systems for their source control (vault, Subversion and team foundation server), each of which contains both internal projects as well as projects for customers. The estimated time to find the source code location will be influenced by an internal Tam Tam project that shows dynamic 'handover' documentation of projects. Currently however, no information is available apart from consulting original developers. The average time it took in this project to find the source code of several components was about 2 hours.

Speed of live location retrieval

To be able to change the version of a component that is currently running, the application developer should have some knowledge about the location of said component. The estimated time to retrieve the live location is three quarters of an hour. This fairly quick time is accomplished by looking at the hardcoded connection configurations in other components which interact with the component in question. These hardcoded configurations are represented by DNS entries which can be 'pinged', thereby retrieving the physical address of the server that is running the component.

Speed of understanding application interactions

In the current architecture, developers should look through the source code and search for places in which call-outs are present. Retrieving all interactions with a specific component is a very cumbersome task and may lead to several hours of work.

Speed of understanding process orchestrations

For some processes the orchestration is stored in single components. For most processes however, multiple components need to be examined to gain an understanding. To derive how a certain process works, the developer needs to understand several application interactions meaning the understanding of a single process can take multiple hours.

Speed of checking component status

An essential request for the new architecture is the ability to check on the status of a component in place. At the moment the only way of figuring out a certain component is not working properly is by directly testing that component. When a process depending on the component does not work anymore one can only guess what component causes the problem. The process of understanding the inner process working can lead to multiple hours of work.

Speed of understanding component purposes

To understand what functionality a certain component has, it is necessary to find out the specific call-ins from other components. This information is currently retrieved by searching for call-outs of components to the said component. The amount of time needed for this is somewhat reduced by the hardcoded connection strings because when there is no connection string, that component does not need further investigation. This figure is related to the speed of understanding application interactions, so the amount of time should resemble the several hours of that figure.

Table 19: First evaluation iteration metrics

Metric	Current situation
# of interaction schemes	6
# of hardcoded connection strings	21
# of incoming method calls per component	2.2
# of components with related functionality (duplication)	8
Speed of source code retrieval	2 hours
Speed of live location retrieval	45 minutes
Speed of understanding application interactions	Several hours
Speed of understanding process orchestration	Multiple hours
Speed of checking component status	Multiple hours
Speed of understanding component purposes	Several hours

B. Evaluation iteration 2 – Architectural style

The second evaluation iteration serves the goal to verify whether the chosen architectural style does indeed bring advantages for Tam Tam. This evaluation iteration is a stripped down version of the evaluation because only the key metrics are possible to be evaluated for a theoretical Enterprise Service Bus. The scenarios are excluded because the added value of performing a scenario analysis with almost the same output as the scenario analysis that will be performed after the design is complete is very limited. After an elaboration on the values for the metrics, a summary of this evaluation

iteration is presented in Table 20, which uses Table 19 presented in Evaluation iteration 1 – Current Architecture as a basis with an additional column. The identified values are retrieved by performing a desk research into the relevant literature.

Number of interaction schemes

The basic idea behind an Enterprise Service Bus is that functionality should be wrapped in services, which renders that only web service calls need to be made from other components to perform some functionality. An additional advantage here is that those calls are only made at one single place.

Number of hardcoded connection strings

With an Enterprise Service Bus, the direct connection configurations are placed in the wrapper services. The location of the created wrappers is only stored in one place, namely the service bus. By creating a single point for the connection strings to be stored, the amount of hardcoded connection strings equals the number of services plus the number of connection strings in the dedicated services.

Number of incoming method calls per component

In the situation with the Enterprise Service Bus, all components (and orchestrations) use pre-defined interfaces in the form of wrapper services. This leads to two categories of impact, when the implementation of an original component changes only the wrapper needs to adapt its implementation. In the case a wrapper changes it needs to be changed in the orchestrations. Next to that, the process logic is stored in the centralized orchestrations so each component just calls the number of processes that have their origin in that component. There are a total of 19 process orchestrations that might be changed with a total of 45 interactions which is around 2.4 incoming calls. The advantage here is that the changes only need to be performed at a single place hereby not jeopardizing the functioning of other orchestrations at the same time.

Number of components with related functionality (duplication)

An Enterprise Service Bus is designed to provide a separation of functionality, which is enforced by the introduction of services. With these additional services the duplicate code entries from different components should be aggregated into one single place, rendering the total number (close to) zero.

Speed of source code retrieval

In the Enterprise Service Bus situation, assuming the documentation is kept up to date, or the internal Tam Tam dynamic handover project is completed, the time necessary to retrieve source code will be around 15 minutes because everything can be found on one centralized location.

Speed of live location retrieval

In the Enterprise Service Bus situation, the service registry can be used to locate the called components. To retrieve the location of calling components, either the –

assumingly up-to-date– documentation can be used or a log of orchestration calls. Both can be performed in a maximum of fifteen minutes.

Speed of understanding application interactions

In the Enterprise Service Bus situation, the specification of contracts in the form of services shows exactly how interactions are to be performed. The only necessary steps are to find the location of the service and examine the interface, which are about 15 minutes for the retrieval part and 30 minutes for examining the interface.

Speed of understanding process orchestrations

One of the key ideas behind an Enterprise Services Bus is the use of a business process executing engine (most commonly in the form of a BPEL-engine). This engine allows the process orchestrations to be stored centrally, thereby reducing the amount of time necessary to be able to understand these orchestrations (#-2). For there are some fairly complicated processes, the amount of time needed is about two hours.

> ***#-2 Easily extendable business processes*** *allow initiatives to be carried out quicker by allowing a just do it mentality*

Speed of checking component status

In an Enterprise Service Bus situation, the orchestrations and component (service) locations are stored locally. This centralization means that an automated test can be used to test the called components, resulting in much less time needed for investigation.

Speed of understanding component purposes

In the Enterprise Service Bus situation, the process orchestration is centralized giving the developer several interactions to look at. From the orchestration knowledge, the developer can derive the invoked functionalities resulting in about an hour of work. Next to that the wrapper services are a good starting point to get to know the offered low level functionalities.

Table 20: Second evaluation iteration metrics

Metric	Current situation	Enterprise Service Bus
Number of interaction schemes	6	1 (+ interaction schemes in dedicated wrappers)
Number of hardcoded connection strings	21	Number of services (+ connection strings in dedicated services)
Number of incoming method calls per component	2.2	2.4
Number of components with related functionality (duplication)	8	0
Speed of source code retrieval	2 hours	15 minutes
Speed of live location retrieval	45 minutes	15 minutes

Speed of understanding application interactions	Several hours	15 minutes finding + 30 minutes understanding
Speed of understanding process orchestrations	Multiple hours	2 hours
Speed of checking component status	Multiple hours	30 minutes
Speed of understanding component purposes	Several hours	1 hour

C. Evaluation iteration 3 – Post design

In 'Evaluation iteration 1 – Current Architecture' the current architecture is evaluated and in 'Evaluation iteration 2 – Architectural style' a preliminary evaluation of the designed architecture is presented. This section complements the evaluation of the designed architecture by carrying out a scenario analysis and revisiting the theoretical ESB values for the identified metrics. The main focus of this evaluation iteration is to thoroughly test the designed architecture, mainly by theoretically supporting the identified characteristics.

Metrics re-valuation

In the second iteration Table 20 was filled with measured values. The result of re-assessing the identified values is shown in Table 21, underneath the elaboration on the updates.

Number of interaction schemes

In the designed future architecture, all interaction with and by the service bus is done through the use of WSDL documents so the base value of 1 remains the same. The addition of interaction schemes between the created services and components also remains the same as no defined guidelines are posed for those interactions to limit the constraints on developers.

Number of hardcoded connection strings

Where the number of services was unknown in the second evaluation iteration, in the future architectural design 9 different services are identified that need to be implemented. The connection strings in these dedicated services are still present because the wrapper services need some reference to whichever component they provide a wrapper for. Components that are added in the future should have such a wrapper included in the implementation, diminishing their effect on this metric.

Number of incoming method calls per component

Due to the wrapping function of the identified services, the metric used here indicates from how many locations a service is called. The nature of the designed Enterprise Service Bbus situation is that all interactions are handled and relayed towards the services by the service bus. The effect of this is that services are called from a single

location, the service bus. The moment the implementation of the component behind the wrapper changes, only the service functionality needs to be altered. When, however, the interface of the service changes, some orchestrations might need to be changed as well but they can all be found in one location and can be changed by non-programmers due to the orchestrations' high-abstraction nature.

Number of Components with related functionality (duplication)

The value of (close to) 0 remains the same here because the nature of the centralized orchestrations enables the reuse of functionality. In effect this means that all duplicate code will be merged into the centrally published orchestrations.

Speed of source code retrieval

The amount of time to retrieve the source code is not altered with the definition of design specifics, because the focus is on the live situation.

Speed of live location retrieval

Due to the nature of the designed Enterprise Service Bus situation, the live location of all components can be found in three steps: 1) locate the service bus, 2) retrieve service location, and 3) retrieve component location from service. This step-by-step approach should not take longer than the stated 15 minutes.

Speed of understanding application interactions

This measure also remains the same because the design was created in such a way it is consistent with the theoretical foundation of Enterprise Service Busses. By creating interaction contracts in the form of services, programmers (and non-programmers) know exactly how to interact with components.

Speed of understanding process orchestrations

The process orchestrations are created by using BPEL documents which are in essence specifically purposed XML documents. To gain an understanding of those XML documents some time is needed, but when a BPEL designing application is used, the orchestration is split up in, and visualized as, atomic operations. Even with no prior knowledge at all, a thorough understanding can be achieved by reading through the orchestration as a 'flow-chart'. This simplicity of orchestrations diminishes the speed of understanding processes drastically.

Speed of checking component status

The designed service interfaces all contain a 'ping' operation which can be used to check the status of a component. In the case the service is not reachable anymore the ping orchestration becomes aware of that. When implemented in a neat manner, the ping operation should test the component status lying behind the service and return that situation report. Just by invoking the ping orchestration the status of the components can be retrieved.

Speed of understanding component purposes

Due to the highly cohesive nature of services, the speed of understanding component purpose is very fast. The speed of understanding the purpose of underlying components can be retrieved by looking at which services use the specific component for which functionality. Because of this need for aggregating purposes, the value of this metric remains about the same.

Table 21: Third evaluation iteration metrics

Metric	Current situation	Enterprise Service Bus	ESB Design
Number of interaction schemes	6	1 (+ interaction schemes in dedicated wrappers)	1 (+ interaction schemes in dedicated wrappers)
Number of hardcoded connection strings	21	Number of services (+ connection strings in dedicated services)	9 (+ connection strings in dedicated services)
Number of incoming method calls per component	2.2	2.4	2.4
Number of components with related functionality (duplication)	8	0	0
Speed of source code retrieval	2 hours	15 minutes	15 minutes
Speed of live location retrieval	45 minutes	15 minutes	15 minutes
Speed of understanding application interactions	Several hours	15 minutes finding + 30 minutes understanding	15 minutes finding + 30 minutes understanding
Speed of understanding process orchestrations	Multiple hours	2 hours	30 minutes
Speed of checking component status	Multiple hours	30 minutes	5 minutes
Speed of understanding component purposes	Several hours	1 hour	1 hour

Scenario analysis

In addition to the elaboration of the scenarios for the current situation, this paragraph elaborates on how the identified scenarios are handled in the designed situation with a focus on the changed aspects from the current situation. The examples used in the different scenarios resemble those used in section Evaluation iteration 1 – Current Architecture.

Scenario 1: Addition of an extra component

When adding a new dynamic 'handover' documentation system, several orchestrations need to be extended by including an invoke operation to the web service of the new component. This can be done far quicker for no programming is required, the only time consuming aspect is retrieving, adapting and redeploying the orchestrations.

Scenario 2: Moving a component to another location

To relocate a component the WSDL binding document of the old location should be replaced by a new WSDL document containing the new location. This is an operation that can be performed at a single location: the service bus; the only steps that need to be taken are replacing the WSDL definition and redeploying the orchestrations.

Scenario 3: Extension / withdrawing functionality of a component

In the designed situation several steps need to be taken to split functionality. The first is to divide the service into two cohesive entities with all functions that 'belong' together. These new functions should replace the old service reference in the service bus and the last steps are to update the orchestrations to call the same functions on other services and redeploy the orchestrations.

Scenario 4: Carrying out maintenance on a component

In the designed situation, the retrieval time of the live location and source code location does not change much, but finding the live location of services is diminished a little by listing all live service location in one place. The advantage of the designed situation lies in the extraction of the component interactions into orchestrations from the presentation logic. This makes it easier to find out if interactions with other components are to blame.

Scenario 5: Change the functionality of a component

In the designed situation a change of an interface is expressed by a change in the service for that specific component. A ripple effect can be traced through the service bus. When all orchestrations that use the specific changed functionality have been adapted, the published orchestration WSDL can be adapted too. The last step is to search through source code where the changed orchestrations are invoked and adapt those parts. The advantages of this situation is that the developer knows exactly what search phrases to use to retrieve the correct locations in the applications and the fact that old orchestration versions can be kept up and running at the same time.

Scenario 6: A process cannot be executed anymore

In the designed situation, the service bus management possibilities can be used to indicate which step of an orchestration causes the failure of an orchestration. Furthermore, the 'ping' operation can be used to test the statuses of all available services reducing the time needed to search by a fast amount.

Scenario 7: Replace a component

When replacing the CRM system, all that has to be performed is to adapt the implementation of the 'CRM web service'. Everything else stays exactly the same. This can be performed by replacing the implementation of the new CRM system according to the defined WSDL contract. The service bus just carries on calling the same functionality on the same interface, but the entity that performs the work is replaced behind the scenes.

Scenario 8: A certain process needs to be extended

When adding a Dropbox component for automatically creating customer project folders, a new service needs to be created that acts as a proxy to the Dropbox servers. This service is added to the service bus and a new invoke operation is added to the correct orchestration. The most time consuming part is creating the wrapper service that communicates with Dropbox because adding a method call to an existing orchestration is a matter of minutes.

Scenario 9: A server is going offline

In the designed situation, the location of components which are used is stored centrally in the service bus. This information prevents the need to investigate in the field of internal processes and thus limits the chance of unwanted downtime and time necessary to investigate.

D. Evaluation iteration 4 – Post implementation

The fourth evaluation iteration, presented in this section, concludes the iteration quartet. The nature of this fourth iteration is unlike the second and third iteration not based on theory but based on real world measurements. This evaluation presents the means to evaluate the actual progression made for Tam Tam by implementing the designed new architecture.

Final metric values

This is the last iteration over the identified metrics; Table 22 presents a complete comparison chart used for comparing the different architectural stages and situations. For each metric an elaboration is given with the rationale behind the values.

Number of interaction schemes

The number of different interaction schemes has increased by one due to the addition of dynamically retrieving the end point location from the UDDI server. This interaction scheme in addition with the WCF services used for web services and process orchestrations makes a total of two plus the interaction schemes in the low level functionalities behind the web services.

Number of hardcoded connection strings

Due to the incompatibility of BizTalk to query endpoint addresses from the UDDI server at the moment, the number of hardcoded connection strings is doubled. In addition to that, each process orchestration gets an own hardcoded connection string in the UDDI server. This makes a total of 34 individual hardcoded connection strings: 16 (processes) + 9 (services in BizTalk) + 9 (services in UDDI). Such an increase from 21 might seem as unnecessary, but it is the absolute maximum number of strings because those 34 are only used in the specified places. In the old situation several duplicate connection strings existed throughout several different components where in the new architecture only the UDDI connection string is duplicated.

Number of incoming method calls per component

The average number of incoming method calls per component turned out to be bigger as well. This can also be ascribed to the fact that data access requests are performed directly on the web services instead of through BizTalk. The total number is the original direct data access occurrences divided by the number of components plus 45 interactions from the process orchestrations divided by 16 processes.

Number of components with related functionality (duplication)

The value of (close to) 0 remains the same here because the abstraction layers take care of removing duplicate functionalities.

Speed of source code retrieval

The amount of time necessary to retrieve the source code is not altered with the implementation of the design, because the focus is on the live situation.

Speed of live location retrieval

Locating the live location of components can be achieved in a two simple steps: 1) locate the UDDI server (hosted on a standardized DNS entry), and 2) retrieve service endpoints. These steps can be performed in about 5 minutes.

Speed of understanding application interactions

The speed in which application interactions can be understood is decreased by the same amount of the measure for the time required for the live location retrieval. The amount of time required to understand all interactions stays about the same, the published web service interface helps greatly.

Speed of understanding process orchestrations

In essence process orchestrations are basic flow diagrams that represent actions that are carried out. With the assumption that the naming of building blocks is chosen carefully, about half an hour is needed for understanding a complete process.

Speed of checking component status

The status of components for which new web services are created can be checked by using the implemented 'ping' operation. By retrieving all web services from the UDDI server and invoking said ping method, most of the web services can be checked. Externally crafted web services cannot be forced to implement a ping method of some sort so in those cases it would take some more time to investigate.

Speed of understanding component purposes

With the introduction of wrapper web services, knowledge about what functionality a component performs can be retrieved from one single location, the interface description file. While this interface description file specifies what functions a web service offers and provides detailed descriptions in the form of comments, the number of functions might be large so the amount of time will be about an hour on average.

Table 22: Fourth evaluation iteration metrics

Metric	Current situation	Enterprise Service Bus	ESB Design	BizTalk 2010
Number of interaction schemes	6	1 (+ interaction schemes in dedicated wrappers)	1 (+ interaction schemes in dedicated wrappers)	2 (+ interaction schemes in dedicated wrappers)
Number of hardcoded connection strings	21	Number of services (+ connection strings in dedicated services)	9 (+ connection strings in dedicated services)	34 + 1 for UDDI
Number of incoming method calls per component	2.2	2.4	2.4	2.8 (+ direct data requests/number of services)
Number of components with related functionality (duplication)	8	0	0	0
Speed of source code retrieval	2 hours	15 minutes	15 minutes	15 minutes
Speed of live location retrieval	45 minutes	15 minutes	15 minutes	5 minutes
Speed of understanding application interactions	Several hours	15 minutes finding + 30 minutes understanding	15 minutes finding + 30 minutes understanding	5 minutes finding + 30 minutes understanding
Speed of understanding process orchestrations	Multiple hours	2 hours	30 minutes	30 minutes

Speed of checking component status	Multiple hours	30 minutes	5 minutes	15 minutes
Speed of understanding component purposes	Several hours	1 hour	1 hour	1 hour

Implemented scenario evaluation

After the last revision of the identified metrics, this paragraph presents the last version of the scenarios analysis used for comparison. Each scenario is elaborated on with the BizTalk 2010 implementation as reference point with precise durations of key actions that need to be taken to complete every scenario.

Scenario 1: Addition of an extra component

To add a new component to the architecture, all process orchestrations that should interact with the new component need to be adapted. Steps to be taken are as follows:

1) Add new web service to the UDDI server (15 minutes)
2) Importing the new web service into the process designer (5 minutes)
3) Extending and adding maps to map values for the new invoke (30 minutes)
4) Create a new orchestration for all new processes (1 hour apiece)
5) Re-deploy the orchestrations (5 minutes)
6) Publish receive location with the web service publishing wizard (5 minutes)
7) Include send port in BizTalk by importing an automatically created XML document (5 minutes)
8) Enlist new receive location to be used in the process orchestrations (5 minutes)
9) Add new process orchestration to the UDDI server (15 minutes)

In total it would take two and a half hours to add a new component with one additional process.

Scenario 2: Moving a component to another location

With BizTalk and an UDDI server in place, the activity to relocate a component entails changing a configuration string in two different places. First of all in the UDDI server to inform components that directly invoke the component. Secondly in the BizTalk server Administration Console to let orchestrations know.

The total amount of time necessary to perform those two operations is as follows:

1) Retrieve the UDDI server location (hosted on a standardized DNS entry) and adapt the configuration string (5 minutes)
2) Retrieve the BizTalk server location (hosted on a standardized DNS entry) and adapt the send port configuration in the BizTalk Administration console (10 minutes)

In total it would take about a quarter of an hour where the main time consuming factor is the user log in process on the BizTalk server.

Scenario 3: Extension / withdrawing functionality of a component

When a web service is split into two (or more) different parts, the references to the old web service should all be updated to refer to the correct new web service. An assumption is made here that the operation is only to split the service, no change or addition of functionality takes place.

The actions that need to be performed are as follows:

1) Delete the reference to the old web service in the workflows project (5 minutes)
2) Import the new web services into the process designer (10 minutes)
3) Change maps to map values from the new service definitions (15 minutes)
4) Change Send port types in the workflows to match the newly added web services (10 minutes)
5) Re-deploy the orchestrations (5 minutes)
6) Delete old send port and include new send ports in BizTalk by importing the automatically created XML documents (10 minutes)

In total this operation, next to splitting the web service in two separate services, takes about an hour. In this hour, all processes that relied on the old web service are updated to use the newly created web services

Scenario 4: Carrying out maintenance on a component

When maintenance needs to be carried out on the service representing a component, a service window can be configured that refrains BizTalk from sending messages to the specific component and postpones the delivery of those messages until after the service window. The nature of the design created in this thesis specifies that the newly created services are only invoked from BizTalk. This is the main reason that all sent messages are logged and a considerable easy assessment can be made for the origin of the investigated bug.

The activities that need to be performed are as follows:

1) Set up service window in the BizTalk Administration console (5 minutes)
2) Retrieve message logs from the BizTalk Administration console to define whether the bug is in the process orchestration or in the web service (5 minutes)
3) Evaluate the messages with the specified interface (15 minutes)
4) Time to fix the bug in the functionality underlying the web service (no change in time) or change the process orchestration (about two hours, see scenario 5)

When the bug lies in the underlying service implementation, the new situation takes the same amount of time to fix as the old situation. The only difference is that the identification of the bug origin can be carried out in about half an hour. When the introduction of process orchestrations causes the bug, the bug fixing time is about two and a half hour.

Scenario 5: Change the functionality of a component

When the functionality of a component including its interface should be changed, a ripple effect will be visible from the bottom functionality layer to the top level process invoking layer. The change of functionality itself is left out of the equation here because that time is equal to the time necessary to change functionality in the current situation.

The actions that need to be performed are as follows:

1) Find source and live location of the web service (30 minutes)
2) Adapt the service interface and re-publish the service (5 minutes)
3) Delete the reference to the old web service in the workflows project (5 minutes)
4) Import the new web services into the process designer (10 minutes)
5) Change schemas to include the adapted parameters (15 minutes)
6) Change maps to map values from the new service definitions (15 minutes)
7) Re-deploy the orchestration (5 minutes)
8) Republish receive location with the web service publishing wizard (5 minutes)
9) Delete old send port and include new send ports in BizTalk by importing the automatically created XML documents (10 minutes)
10) Enlist new receive location to be used in the process orchestrations (5 minutes)
11) Change calling locations by searching for applications that use a specific UDDI entry (less than current situation)

The two key assumptions are made here are as follows: 1) time necessary to change underlying functionality is not changed and 2) searching for specific UDDI entries takes far less time than 'searching for referring components'. The additional combined time is about two hours, excluding the amount of effort it takes to carry out changing the underlying functionality.

Scenario 6: A process cannot be executed anymore

When assessing why a process fails to be executed, the BizTalk Administration console proves to be of assistance here because it tracks all instances that are completed, are running or have failed. Furthermore, BizTalk ensures the processes can be resumed after the necessary interventions have been performed.

The following steps are used to identify what component causes the issues:

1) Retrieve message and instance logs from the BizTalk Administration console to define where the malfunction stems from (15 minutes)
2) Investigate the logs to find out what the origin of the failures is (15 minutes)

In total a maximum of half an hour is needed to identify the sour spot in the architecture.

Scenario 7: Replace a component

The only steps necessary to replace a component is implementing the communication from the existing web service interface to the new component and changing the location

of the old web service to the newly created service. This moving is in essence the same as that of scenario 2 so next to the implementation of the new connection it takes about a quarter of an hour to replace a component.

Scenario 8: A certain process needs to be extended

After having implemented all processes in BizTalk, the effort it takes to add a specific functionality invoke to a workflow is very minimal. As said in the design evaluation, the only time consuming aspects are retrieving, adapting and redeploying the orchestrations. The key assumption made here is that the to-be-added functionality is already represented by a web service, which otherwise would by far take the most effort to be created.

The specification of time necessary to perform such an addition is as follows:

1) Retrieving location of process orchestrations (15 minutes)
2) Importing the new web service into the process designer (5 minutes)
3) Creating a map to map values for the new invoke (15 minutes)
4) Re-deploy the orchestration (5 minutes)
5) Include send port in BizTalk by importing an automatically created XML document (5 minutes)

In total it would take less than an hour to perform the addition of extra functionality to a process.

Scenario 9: A server is going offline

In the case a server needs to be shut down for maintenance or deletion, it becomes clear by the UDDI server whether some components are dependent on it. In the case of maintenance, a maintenance window can be specified so nothing is disrupted. In the case of deletion, it is known exactly which components are still running on the machine that needs to be deleted. In summary the new situation increases the certainty with which servers can be turned off.

Appendix F: Key deliverables

Through the course of the thesis project several deliverables have been created. This appendix lists all deliverables and gives a short explanation on what is the use of the specific documents or implementations.

Systems Architecture, *Visualization of the current internal architecture.*

Description of the internal systems architecture in place within Tam Tam in the time span September until October 2010. Firstly is decided how to visualize the architecture and after that the process of describing it is elaborated on.

Design document, *What does the to be architecture look like*

This document presents an elaboration on the rationale behind the design, including the design specifics. This information is supplemented with the designed organizational change, a transition plan and the advantages and disadvantages of the new architecture.

BizTalk 101 ~ Creating your First BizTalk 2010 building block, *Create an orchestration acting as a WCF Proxy*

This document walks users through the creation of a proxy BizTalk orchestration to decrease the learning curve to start using BizTalk. Goals of this document are to create comprehension of the steps necessary to create a sample WCF service, use this service in a BizTalk orchestration and how to deploy the created orchestration.

BizTalk 102 ~ Relocating a service, *Informing BizTalk and UDDI of the new location*

This document walks users through the steps that need to be performed when a new web service is to be used within Tam Tam. This document covers publishing the web service, consuming the web service and updating the binding location of the web service.

BizTalk 103 ~ Enterprise Single sign-on, *How to limit Service access to specific users or groups*

This third tutorial describes the way Single sign-on can be used to limit the access to specific WCF functionalities. BizTalk orchestrations are run with a dedicated account, limiting the possibilities to verify if the calling user has the right privileges. By including SSO in the picture, the called functionality can decide whether or not the original caller has the right privileges. This document describes the step-by-step way to set up the environment for SSO and the use of it in orchestrations and WCF services.

BizTalk 104 ~ Remote deployment, *How to develop and deploy BizTalk applications from workstations*

This document walks users through the process of deploying a BizTalk application to a different machine running BizTalk Server 2010. Motivation for this tutorial is the desire

to split the BizTalk environment in three different parts: 1) machine for development, 2) machine for production, and 3) monitoring capabilities.

BizTalk 105 ~ Remote monitoring, *How to monitor a BizTalk 2010 server from a different machine*

This document indicates the different possibilities available for monitoring the BizTalk environment. It stretches from direct SQL access through checking-up on completed or terminated instances to real-time monitoring capabilities. The steps indicated in this tutorial help both system administrators and developers to figure out what causes which problems.

Configured BizTalk environment, *An environment in which applications can run*

This deliverable consists of a fully functional environment which is to be used to host the designed implementation. Every part necessary for the design is present here, including but not limited to UDDI/BizTalk/SSO/IIS – servers and the correct privileges are set.

Implementation plan, *Steps necessary to finish the transition towards BizTalk 2010*

This document lists all atomic tasks which have to be carried out to reach the designed architecture from the current situation. Key elements here are a backlog of activities with their estimated amount of time and the already performed tasks.

WCF Sample service, *To be used for developing services*

This sample service includes all standard elements to be used in web services. Elements offered are a configured configuration file, logging capabilities and a formalized way of creating functionalities.

WCF Sample proxy orchestration, *Result of BizTalk 101*

This presents a working example of the first BizTalk 2010 tutorial to be used as a building block or reference point for verifying the progression of following the tutorial.

Appendix G: Progress of implementation

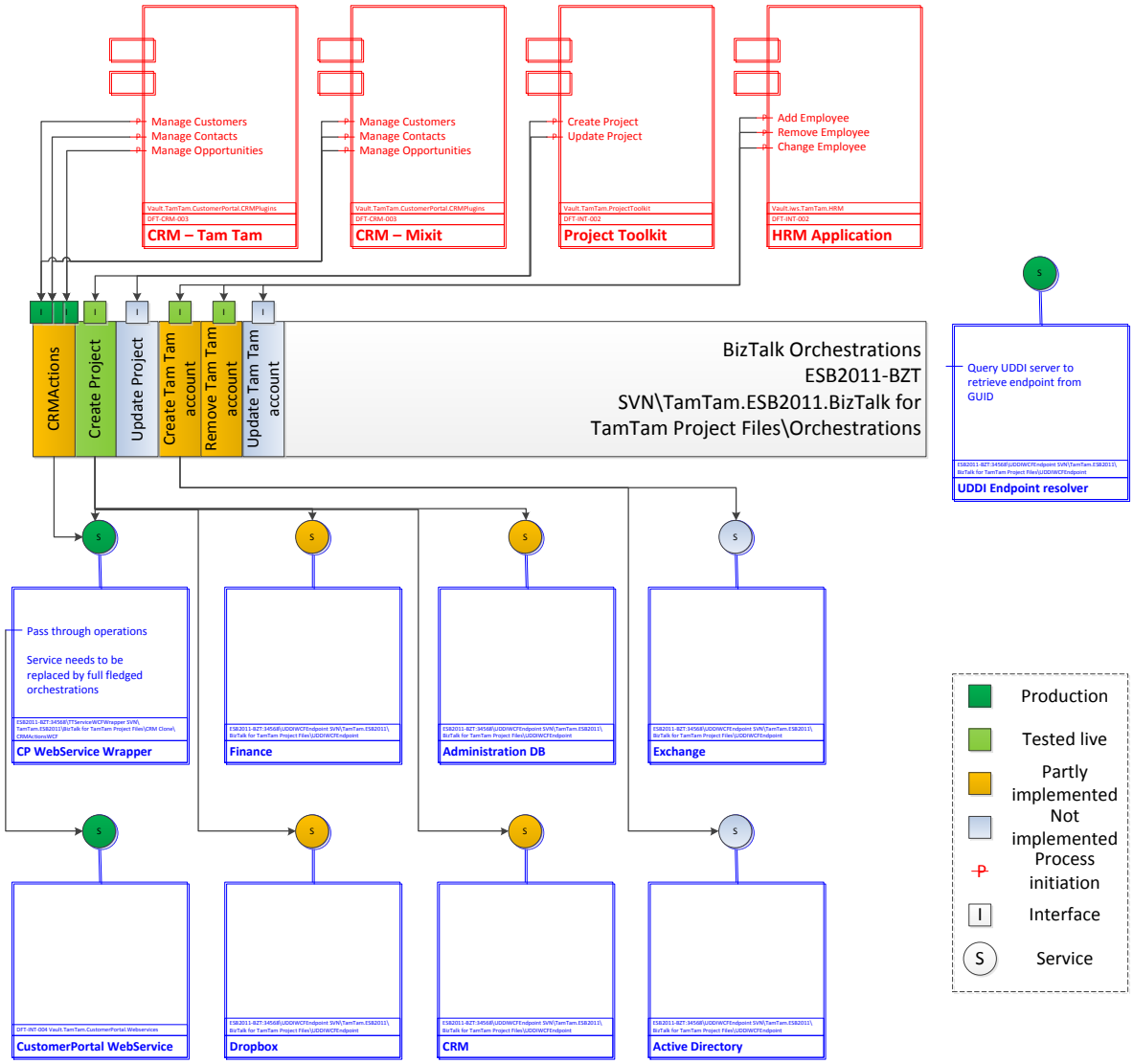

Figure 36: Progress of implementation

Researching a transition to an organized chaos in enterprise system architectures

Master's Thesis, June 2011

Ronald van Etten