

Monitoring data exchanges between information systems

Master Thesis Business Information Technology

Tycho Bom

November 10th 2010

UNIVERSITY OF TWENTE.



Master Thesis

Monitoring data exchanges between information systems

Place and date

Hilversum, November 10th 2010

Author

Tycho M. Bom
Master Business Information Technology
University of Twente

University supervisors

Dr. Marten J. van Sinderen
Faculty of Electrical Engineering, Mathematics and Computer Science

Dr. Maria E. Iacob
School of management and governance

CRM Resultants supervisor

Marc la Chapelle

UNIVERSITY OF TWENTE.



Abstract

Context

Information systems are more and more connected to each other and exchange more and more data with each other. As the number of data exchanges increases, it becomes increasingly difficult to maintain an overview of how these data exchanges are performing. This is especially a problem for application service providers which host systems for customers because they host multiple systems and each system can have multiple data exchanges. Although there are monitoring tools available that monitor the states of systems, there are no tools available to monitor the functional aspects of data exchanges. Due to this lack of oversight on the functional performance of data exchanges, problems can be unnoticed and customers can become unsatisfied. The purpose of this study is to create a reference architecture for systems that can provide central monitoring of data exchanges with the possibility of aggregating information on different data exchanges into useful diagnostic reports. The scope of this study is focused on monitoring only at the target system of a data exchange.

Approach

Our research combines information from literature and information from a case study at CRM Resultants. The combined information is about different aspects of data exchanges, possible failures in data exchanges, data quality and monitoring concepts. Based on this information we specify a reference architecture for monitoring functional aspects of data exchanges. When monitoring functional aspects we make a distinction between syntactic monitoring and semantic monitoring. Syntactic monitoring is about monitoring erroneous and successful transactions. Semantic monitoring also takes into account the contents of data in a transaction. To validate this reference architecture we create a prototype based on the reference architecture and test it in the environment of CRM Resultants.

Reference architecture and prototype

The reference architecture consists of three main components: a monitoring plugin, a data collection mechanism and a central monitoring server. The monitoring plugin is used to acquire monitoring information from a data exchange. This information is then collected by the data collection mechanism of the central monitoring server and the central monitoring server presents the data to the end user and is able to generate alerts if necessary. This architecture is capable of monitoring different types of data exchanges because for each type of data exchange a different monitoring plugin can be used and the possible differences in the data format the monitoring plugins generate can be compensated by the data collection mechanism.

In the prototype the monitoring plugin is implemented by Log4Net for monitoring web service data exchanges and by Scribe for monitoring Scribe data exchanges. Scribe is also used as a data collection mechanism to transfer data from the monitoring plugins to the central monitoring server. We use Microsoft Dynamics CRM as the central monitoring server to show the result to the end user.

Findings

Test results of the prototype show that the prototype works and that the reference architecture is valid. Most of the syntactic errors we wanted to detect were detected by the prototype. During the testing period the prototype even identified several real problems of which some might be unnoticed without the monitoring prototype. We did not detect a lot of semantic errors. This is due to the fact that our implementation for checking semantic correctness is limited and the fact that the field we monitored is often automatically checked before it is sent to the target system.

Benefits resulting from this research

Our research resulted in the following benefits.

- Based on our reference architecture monitoring solutions can be created for monitoring functional aspects of data exchanges.
- By monitoring data exchanges insight is gained in the transactions data exchanges execute.
- Errors in data exchanges can be detected before a customer complains about a problem. This can increase customer satisfaction.
- In our prototype all monitoring information is accessible from a single web based user interface. There is no need any more to check separate databases and folders with error logs. This can speed up the investigation on problems.
- Monitoring data exchanges can be offered as a service to customers. Customers can pay a monthly fee for this service or they can monitor their own data exchanges and pay a licence fee for the monitoring software. This can increase revenue for the application service provider.
- Although it has never been an objective of this research, using our prototype it is now possible to view which data exchanges put the most information in a CRM system. It was already possible to determine the records that were created in a specified period in the CRM system itself, but it could be hard to determine from which source the information originated.

Recommendations

We formulate recommendations for application service providers in general and for CRM Resultants in particular. The most important recommendations are the following.

General recommendations

- Implement our reference architecture to monitor data exchanges between information systems to maintain oversight of the successful transactions of data exchanges and errors in data exchanges.
- Create an implementation of our reference architecture that can monitor all technologies that are used in the environment that is to be monitored.

Recommendations for CRM Resultants

- Use the monitor for all new data exchanges that are developed and add it to existing data exchanges which are known to be prone to errors.
- Extend the alerting functions of the monitor to make them more reliable and to increase the ease of use.
- Include a service module to support the process of following up errors in data exchanges.
- Incorporate the monitor into the company's central CRM system to make the monitoring results easily accessible to all employees and to integrate it to the existing information about hosted CRM environments and customers.

Preface

Years ago companies did all administration on paper and used manually ordered systems to store information. The most well-known examples of these systems are card catalogs and large filing cabinets. The most important asset in this process of storing and retrieving information was the employee itself. The employee needed to know all necessary procedures exactly because all actions were executed manually. As a result the employee had a feeling of the quality of data and the frequency of transactions. In large organizations where multiple people worked in the same files, it was difficult to maintain oversight, but at least each individual transaction was checked by humans and each employee was able to signal abnormalities in his or her work. You could state that employees were in touch with the information they worked with.

Nowadays most companies have replaced their card catalogs and filing cabinets by computer systems which store all their data. These systems vary from simple Excel sheets to the most advanced online ERP systems. This brings numerous benefits such as the increased accessibility of information, the ability to quickly search through the information and of course possibilities to easily copy and back up data. Along with these replacements also more and more actions, which were executed manually in the past, are automated by IT systems. This includes data exchange between information systems. This saves a lot of work which was previously done by hand. It is less error prone and it is often a lot faster.

Automation of information systems certainly brings a lot of benefits to companies. Maybe the most important of these benefits is cost reduction. The downside of all this automation is that IT systems are black boxes to a lot of users. They enter data into the system, but they do not know what the system does with it and how it works in the background. Especially if data is exchanged with other IT systems most users have no clue on what is actually happening. Users expect that the systems takes care of the data and executes all necessary procedures but as the amount of data increases it becomes impossible for users to check if the data is still correct. As long as everything is functioning correctly this is not a problem, but things become different if IT systems do not behave as desired or expected. Users will often not be able to signal these problems directly.

In the old situation users knew exactly how things were related and what data was exchanged between their systems. In the new, automated situation only system administrators can view transaction logs and error reports. The problem is that they either have to look at a lot of different locations for error logs or they are overloaded by e-mails containing log files. As a result system administrators often do not signal problems in data exchanges until the user complains. To solve this problem we create a solution that can monitor multiple data exchanges and presents the results in one overview. This gives system administrators the opportunity to signal problems before the user complains and oversight of exchanged data between information systems is regained.

Acknowledgements

During the execution of this research I was supported by several people, who I want to thank for their support. I want to thank my supervisors from the University of Twente Marten van Sinderen and Maria Iacob for their valuable feedback and the easy way in which discussions were held. All meetings went very smooth in my opinion.

I completed this research at CRM Resultants. I would like to thank Roel Hilberink and especially Marc la Chapelle for providing me this opportunity. Marc has been a really flexible supervisor and has been a valuable sparring partner for discussing research topics and possible solutions.

Furthermore I would like to thank Bas van Sluis and Hanno Zwikstra from CRM Resultants. Bas gave me useful insights in how to use Scribe for both monitoring and acquiring monitoring data. A lot of his hints are used in this research. Hanno has been very helpful by explaining the Log4Net framework and making small modifications to it to make it suitable for monitoring purposes. Without Hanno I would not have been able to monitor custom built web service data exchanges. I would also like to thank the other colleagues at CRM Resultants. They have been really supportive and were always available for discussing my research. These discussions helped me thinking about the research and gave me new insights.

Table of contents

<i>Master Thesis</i>	II
Monitoring data exchanges between information systems	II
Place and date	II
Author	II
University supervisors	II
CRM Resultants supervisor	II
Abstract	III
Context	III
Approach	III
Reference architecture and prototype	III
Findings	III
Benefits resulting from this research	IV
Recommendations	IV
Preface	V
Acknowledgements	VI
1 Introduction	1
1.1 Motivation	1
1.1.1 Background of the topic	1
1.1.2 About the company	1
1.1.3 Current situation	2
1.1.4 Desired situation	2
1.1.5 Difference between the current and desired situation	2
1.1.6 Problem definition	2
1.2 Research objective	2
1.3 Research approach	4
1.4 Report structure	5
2 State of the art	6
2.1 Data exchanges between information systems	6
2.1.1 Architectures of data exchanges	6
2.1.2 Layers of data exchanges	7
2.1.3 Timeliness of messages	10
2.2 Possible failures in data exchanges	11
2.2.1 Failures in network communication	11
2.2.2 Application failures	11
2.3 Data quality	12
2.4 Monitoring concepts for data exchanges	15
2.4.1 Methods of monitoring	15

2.4.2	Levels of monitoring.....	15
2.4.3	Polling versus publish and subscribe.....	17
2.4.4	Reporting results of monitoring	18
3	Case study on failures of data exchanges	19
3.1	Web service data exchanges of the Hogeschool Utrecht	19
3.1.1	Reasons for failures that have occurred.....	20
3.2	Scribe data exchange Hogeschool Utrecht.....	21
3.2.1	Reasons for failures that have occurred.....	22
3.3	BizTalk data exchange between Peoplesoft and Microsoft CRM of ROC van Amsterdam ...	22
3.3.1	Errors that can occur	22
3.4	Combined results of this case study.....	23
3.4.1	Errors resulting from the contents of data.....	23
3.4.2	Errors in middleware	23
3.4.3	Connection and performance errors.....	24
3.5	Mapping of errors to data quality	24
4	Monitoring reference architecture	25
4.1	Components of the architecture	25
4.2	Monitoring requirements of the architecture.....	26
4.3	Monitoring process steps.....	26
4.4	Contents of monitoring information	27
4.5	Events that must be monitored	28
4.6	Requirements of the central monitoring server	28
4.7	How this reference architecture improves data quality	29
5	Assessment of technical alternatives	30
5.1	Hyperic.....	30
5.1.1	Monitoring plugin.....	30
5.1.2	Central monitoring server	31
5.1.3	Monitoring data collection mechanism	31
5.1.4	Conclusion	31
5.2	WSMonitor	31
5.2.1	Monitoring plugin.....	31
5.2.2	Conclusion	31
5.3	SoapKnox	32
5.3.1	Monitoring plugin.....	32
5.3.2	Conclusion	32
5.4	ManageEngine.....	32
5.4.1	Monitoring plugin.....	32
5.4.2	Central monitoring server	32

5.4.3	Monitoring data collection mechanism	32
5.4.4	Conclusion	33
5.5	Scribe	33
5.5.1	Monitoring plugin.....	33
5.5.2	Monitoring data collection mechanism	33
5.5.3	Conclusion	34
5.6	soapUI.....	34
5.6.1	Monitoring plugin.....	34
5.6.2	Conclusion	35
5.7	Log4Net	35
5.7.1	Monitoring plugin.....	35
5.7.2	Conclusion	36
5.8	Microsoft Dynamics CRM	36
5.8.1	Central monitoring server	36
5.8.2	Conclusion	36
5.9	Conclusion on technical alternatives for monitoring	36
6	Our prototype.....	38
6.1	Components and architecture.....	38
6.2	Requirements	39
6.2.1	Aggregate data from different sources which may use different technologies.....	39
6.2.2	Monitor syntactic and semantic aspects of data exchanges.....	39
6.2.3	Support real time and deferrable messages	40
6.2.4	Detect patterns in data exchanges.....	40
6.2.5	Report different types of failures	40
6.3	Monitoring web service data exchanges.....	40
6.4	Monitoring Scribe data exchanges	41
6.5	Monitoring semantics.....	42
6.6	Central monitoring server	43
6.7	User experience of the central monitoring server	44
6.7.1	General interface.....	44
6.7.2	Following up alerts	45
6.7.3	Viewing batch results	48
6.7.4	Viewing semantic results.....	52
6.8	Collection mechanism	53
6.8.1	Collection of data from web service data exchanges.....	53
6.8.2	Collection of data from Scribe data exchanges	54
6.8.3	Collection of semantic results	57
6.9	Summary of how the prototype meets the requirements.....	58

6.10	Monitoring the monitor	58
6.11	Security aspects.....	59
6.12	Interoperability.....	59
6.13	Test results of the prototype.....	59
6.13.1	Types of errors found by the prototype	60
6.13.2	Counts of monitoring results.....	60
6.13.3	Real problems that were found by the monitor	63
6.13.4	Test limitations	64
7	Validation and discussion	65
7.1	Validation	65
7.2	Discussion	66
8	Conclusion	68
8.1	Summary of results.....	68
8.2	Benefits.....	69
8.3	Recommendations.....	70
8.3.1	General recommendations.....	70
8.3.2	Recommendations for CRM Resultants.....	70
8.4	Limitations and future work	70
	References.....	72

1 Introduction

In this chapter we explain the background of this research and why this research is conducted. We first give a motivation and some background information about the topic. Then we continue with the research objective. How we reach the objective is described in the research approach. Finally we explain the report structure.

1.1 Motivation

In this section we explain the background of the topic. We explain the current situation, the desired situation and the difference between them. At the end we present our problem definition.

1.1.1 Background of the topic

Enterprise application integration and service oriented architectures (SOA) are becoming increasingly popular. These concepts mean that data from different enterprise systems is combined, resulting in added value and a total overview of all information that is available. To be able to combine data from different systems, data exchanges between information systems have to be created. When the number of data exchanges increases, the oversight on how they are performing can easily be lost.

Another upcoming trend in the past few years is that companies are shifting from hosting enterprise systems themselves to using systems that are offered by application service providers. Companies can use these systems without having to install and maintain their own servers. The application service provider takes away a lot of possible obstacles for implementing and using a system and in return the customer can pay per user or per usage. This concept is often referred to as software as a service or cloud computing. One of the key success factors for software as a service is reliability of the systems. If this reliability is lower than when the customer hosts the system himself, the customer will not use the hosted platform. Also because for most systems there are multiple application service providers, there exists competition between them. If the reliability is not at least as high as that of your competitor, you will lose the competition beforehand.

Combining these two trends one can imagine that systems that are offered by application service providers are often integrated with other systems the customer uses. Because reliability of these systems is so important, it is necessary to monitor these data exchanges. For monitoring servers itself and the application that is offered, there are normally sufficient monitoring tools. For monitoring data exchanges this is different. When monitoring data exchanges not only the connections have to be monitored, but also the functional aspects of data exchanges will have to be monitored. Among other aspects this includes monitoring quality of data, the number of executed transactions and the number of transactions that cannot be processed with the corresponding reason etcetera.

1.1.2 About the company

This research is carried out at CRM Resultants. Cases and experience from this company are used to gather information and to test the solution.

CRM Resultants executes two main activities: consulting and hosting. The consultancy activity is focused on implementing Microsoft Dynamics CRM. CRM Resultants analyzes the business needs of customers and implements the CRM system to suit the needs of the customer. Because each customer has different needs, the system is always tailor made. Often there is a need to exchange data between the CRM system and other systems customers use. During the implementation project also these data exchanges are specified and built.

After the implementation project, the relation between CRM Resultants and the customer does not end. From the KPN cyber center at Schiphol CRM Resultants offers hosted CRM and functions in this way as an application service provider to customers. Customers can use their system by paying per user without having to install and maintain their own CRM servers. Also related Microsoft products

such as SharePoint, Exchange and even Windows desktops can be hosted on the platform of CRM Resultants. Support on these systems is also included.

Although CRM Resultants does a lot of CRM implementations and is a major player in the Microsoft CRM market, it is a small company. It has around 20 employees. Most of these employees are consultant or project manager. A few of them are developers. The developers build custom solutions and data exchanges with other systems.

1.1.3 Current situation

In the current situation there are no standard monitoring tools for monitoring data exchanges between information systems. For some data exchanges there are execution logs available, but these reside at different servers and locations and are not frequently checked. This can result in problems not being noticed until the customer complains. In the meantime data can be lost. This results in customers becoming unsatisfied. In the end this can even result in a customer switching to in house hosting of the system or switching to a competitor. Both of these are undesirable.

1.1.4 Desired situation

ASP platforms should have a very high reliability and should outperform in house installations of enterprise systems. Monitoring tools have to detect possible errors in data exchanges and should notify support employees. These employees can then investigate the problem and fix it before the customer notices the problem and files a complaint. This results in satisfied customers who see that everything is under control.

1.1.5 Difference between the current and desired situation

The main difference between the current situation and the desired situation consists of tools and knowledge. There are no tools for central monitoring of data exchanges. Some data exchanges log errors somewhere, but because the information is fragmented, no checks are executed on the information. Also, there is no knowledge of what and how these tools should monitor. Nowadays there are only tools in use which monitor if a server is online and what the CPU load is etcetera, but if a server is online that does not mean that data exchanges are functioning properly.

This research will contribute in reaching the desired situation. It will focus on monitoring data exchanges between information systems. Monitoring tools can help in detecting possible errors and it is up to the system administrator to take actions on this monitoring information.

1.1.6 Problem definition

Analyzing the current situation, the desired situation and the difference between them, we formulate the following problem definition:

There is a lack of insight in how data exchanges are performing. This results in failures of data exchanges which are unnoticed and on which no corrective actions are taken.

1.2 Research objective

We formulate the following research objective:

Create a reference architecture for systems that can provide central monitoring of data exchanges with the possibility of aggregating information on different data exchanges into useful diagnostic reports.

In this section we explain the background of this objective. Also we state the main research question that is answered and which sub questions are used to answer the main question.

First we depict the relation between this research and the main target of an application service provider.

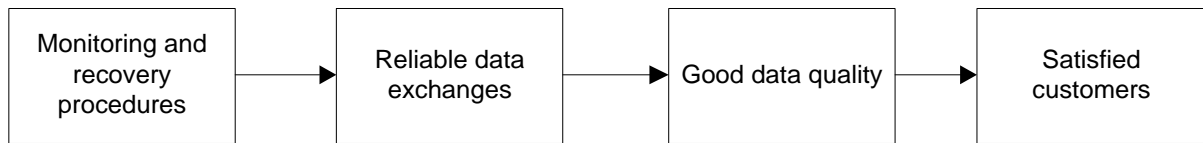


Figure 1 Causal relation

We state the main target of an application service provider as having satisfied customers. Satisfied customers will remain customers and will possibly attract more customers. These customers are of vital essence for an application service provider. We continue with the question: ‘How to make customers satisfied?’ Customer satisfaction depends on a lot of variables, but certainly one of them is good data quality. Customers use enterprise systems to store data of for example their orders. It is important that this data is as good as possible. If an application service provider can deliver better data quality than the customer can by hosting the system itself, it is even an additional reason for the customer to use the platform of the application service provider.

There are basically two ways in which data can be put into the enterprise system. The user can enter data or data can be entered by data exchanges with other systems. If these data exchanges are not reliable or enter wrong data, this degrades the data quality in the system. So to keep the data quality in the system high, the data exchanges need to be reliable.

Creating reliable data exchanges starts with a good design and a solid implementation. However, at runtime there are a lot of things that can go wrong. Therefore monitoring is needed. Monitoring does not directly increase the reliability of a data exchange, but if concrete actions are taken by support personnel if the monitoring tools signal a problem, the data exchange does become more reliable.

To summarize this section: monitoring tools will eventually lead to increased customer satisfaction. There are a lot of other opportunities to increase customer satisfaction, but in this research we focus on this opportunity.

A data exchange always has a source system and a target system. In our research we focus on monitoring the transaction results of data exchanges on the target system, because only at the target system we know the exact impact of transactions of a data exchange. We are interested in the errors of transactions and the successful transactions. By monitoring the errors we can identify potential problems and by monitoring successful transactions we can determine the effect a data exchange has on the data in the target system and detect the absence of transactions. Often an application service provider only has influence on one side of the data exchange. In our research we focus on the target system so we use the case where the target system is in the domain of the application service provider. This is schematically shown in Figure 2.

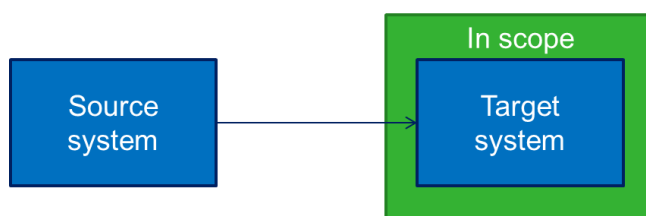


Figure 2 Overview of a data exchange and the scope of our research

In this research we make a distinction between syntactic and semantic errors. Syntactic errors are transaction errors. Semantic errors do not result in a transaction error, but results in not meaningful data in the target system. We do not focus on errors in network links between information systems, but if a link is completely down, we will detect the absence of transactions.

The main research question of the project is stated as follows:

How should a reference architecture that implements syntactic and semantic monitoring at the target system of data exchanges be specified?

To acquire knowledge to answer the research question, the following sub questions are used.

- What types of data exchanges are available and need to be distinguished?
- What problems can occur?
- How can data exchanges be monitored so that problems are detected?
- How should the architecture of a data exchange that includes monitoring functionality be specified?
- Which existing solutions are available which offer monitoring and recovery procedures and do they meet the requirements?

This research aims to develop a reference architecture for monitoring data exchanges. This should bring the current state of art closer to the desired situation as described in 1.1.4. Monitoring can be added to existing data exchanges and new data exchanges can incorporate monitoring from the design stage.

1.3 Research approach

The final product of this research is a reference architecture for monitoring data exchanges. In this reference architecture monitoring tools are specified and guidelines are given for what has to be monitored. To get to the final result, this research is split up into four phases. These phases are depicted in Figure 3.

In the first phase the state of the art concerning relevant topics in literature are investigated and a case study on real world errors is executed. In the literature study we focus on types of data exchanges, possible failures, data quality and monitoring concepts. In the case study we gather errors that have occurred in data exchanges of customers of CRM Resultants.

Based on the information we retrieved in phase one, we create our reference architecture in phase two. After we specified our reference architecture, we execute a case study on existing monitoring solutions. We compare them with our reference architecture and see if they meet the requirements.

Phase three consists of building our own prototype of a monitoring solution. We make use of a solution we found in our case study in phase two or a combination of solutions. By building and testing this prototype we validate our reference architecture.

If our prototype proves that our reference architecture is correct, we have validated our reference architecture. In phase four we draw conclusions on our findings, provide a discussion on our work and give recommendations for future work.

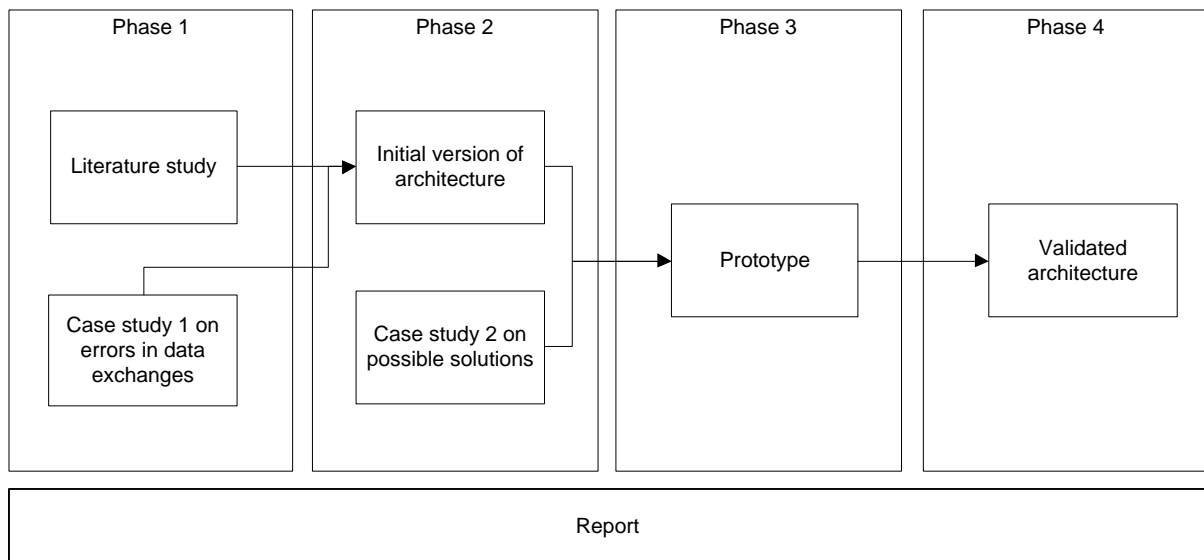


Figure 3 Schematic overview of our research approach

1.4 Report structure

In this section we describe the structure of the report.

Chapter 1 explains why this research is carried out. It gives background information on the topic, the company and the problems that exist. By analyzing the current and desired situation a problem definition is given. How the process of solving this problem looks like, is explained in the research approach.

Chapter 2 describes the current state of the art of relevant concepts. Results from a literature study are presented per topic.

Chapter 3 describes the results of the first case study about the problems that can occur in practice in data exchanges. This information is gained from experience of CRM Resultants.

Chapter 4 elaborates on the monitoring reference architecture. This reference architecture is based on the literature study from chapter 2 and the case study from chapter 3.

Chapter 5 concerns a case study on available monitoring solutions that might meet the requirements of our reference architecture. Solutions found in this case study are used to create our prototype.

Chapter 6 is about implementing and testing the prototype of the monitoring solution. Results from these tests are also presented in this chapter.

Chapter 7 validates if our prototype and reference architecture are correct by analyzing our results from the prototype and our experience with building it.

Chapter 8 gives a conclusion on our research project. Also the benefits and limitations of our solution are discussed and recommendations for future work are presented.

After chapter 8 a list of references used in our research is presented.

2 State of the art

In this chapter we present the state of the art concerning relevant topics for this research. We execute a literature review to find out what knowledge is already available [1]. We discuss this information from literature per topic and for each topic we explain why it is relevant for this research. At the end of each topic we relate the retrieved information to our research to make clear how we use the information.

In this research we do not strive to give a complete overview of all literature that is available about the relevant topics. We make use of the search engines Google Scholar [2], Scopus [3] and Web of Science [3] to gather the most important information about topics. Also we make use of books that give an introduction to topics and that can hint for other relevant topics.

2.1 Data exchanges between information systems

In this research data exchanges are the most important concept. Therefore we have to define what data exchanges are and what properties of data exchanges are important for our research.

In the past years enterprise application integration has gained more and more attention. By combining data from different information systems additional value can be created. Integrating data creates possibilities for data mining and for creating a complete overview of all data that is available to an organization [4]. When integrating enterprise applications, data exchanges have to be created.

We define a data exchange as follows: A data exchange is a process that transfers data from a source system to one or more target systems.

In the following sections we describe the following general properties of data exchanges.

- Architectures of data exchanges. This section explains the possible relations between involved systems in data exchanges.
- We refine the architecture of data exchanges by describing the layers of data exchanges and the layers of applications that exchange data. Also the relation between the application and the available network layers are explained.
- Timeliness of messages. This section explains the difference between real time messages and deferrable messages and explains batch data exchanges and real time data exchanges.

2.1.1 Architectures of data exchanges

The relations between information systems that exchange data can vary. We distinguish three types of architectures of data exchanges, which we based on Britton and Bye [5].

- Bus architecture. This architecture is well known under the name of enterprise service bus. In this architecture there is a central bus to which all systems connect. This bus is middleware that manages all communications between those systems. It is a tightly coupled architecture because all systems must use the same technology and follow the same protocol. This type of architecture has several advantages. It is fast because the hardware and software are made for this job. It is secure because all security can be built into this one piece of middle ware. It is flexible because new systems can easily be added.
- Hub architectures. Information systems can be connected using a hub. A hub is a server that routes messages between systems and can perform additional actions. These additional actions can be reformatting of the message, routing the message, adding information to the message and monitoring the message flow. In one enterprise architecture multiple hubs can be placed that serve different purposes and use different technologies and protocols.
- Web service architecture. This is a point to point architecture where web services interact directly with each other. Web services are in fact just middleware, but they are often part of an application. Therefore applications can talk directly to each other without the need for

additional middleware. This makes it possible to connect systems with different platforms with each other. Furthermore, they are designed to work over the internet, so they do not need additional connections between systems and do not have problems with firewalls.

What all three architectures have in common is that middleware is used. This middleware can consist of integration applications, but it can also consist of web services. In Figure 4 the architecture of a data exchange between a source system and a target system is shown. We based this generalized architecture on the book of Bussler [6].

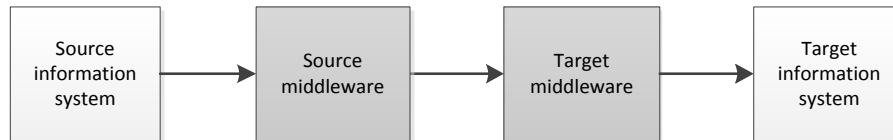


Figure 4 Layered data exchange using middleware

Now we take a look at what each piece of middleware does. First we take a look at the middleware of the source system. Its activities are shown in Figure 5. First it extracts data from the source system. If needed, it transforms this data to a specific format and finally it sends this data to the target system.

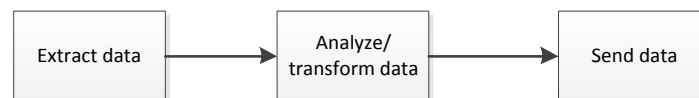


Figure 5 Process of middleware at the source system

The process of the middleware at the target system is similar to the process of the middleware in the source system. It starts with receiving data from the source system. Then it analyses and transforms that data to make it fit in the target system and finally it stores the data in the target system.

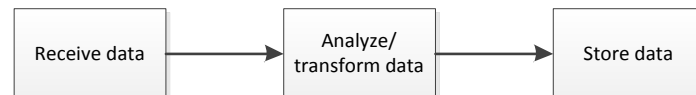


Figure 6 Process of middleware at the target system

Relation to this research

For this research it is important to note that data exchanges always have a source system and a target system. To make interaction between these systems possible, middleware is used. This middleware can transform data to make it compatible with middleware at the other side to the data exchange.

2.1.2 Layers of data exchanges

In our research we focus on monitoring syntactic and semantic aspects of data exchanges. Both applications and network interfaces are composed of layers. To make clear at which layer we want to do monitoring, we explain the relation between applications and network interfaces in this section.

The OSI model [7] is the layered architecture that is used to transport data over computer networks. This model is schematically shown in Figure 7. Each layer uses functionality from the layer below and provides functionality to the layer above. At the bottom there are three media layers. These layers are implemented by the network that is provided to the system that is connected to it. The upper four layers are implemented by systems that are connected to the network.

The OSI model is a very generic model, which can be applied to various technologies. The most common technology is TCP/IP. This technology only distinguishes four of the seven layers of the OSI model. In this model the application layer, presentation layer and session layer are simply called application layer. In this application layer protocols such as FTP, HTTP and SMTP are used.

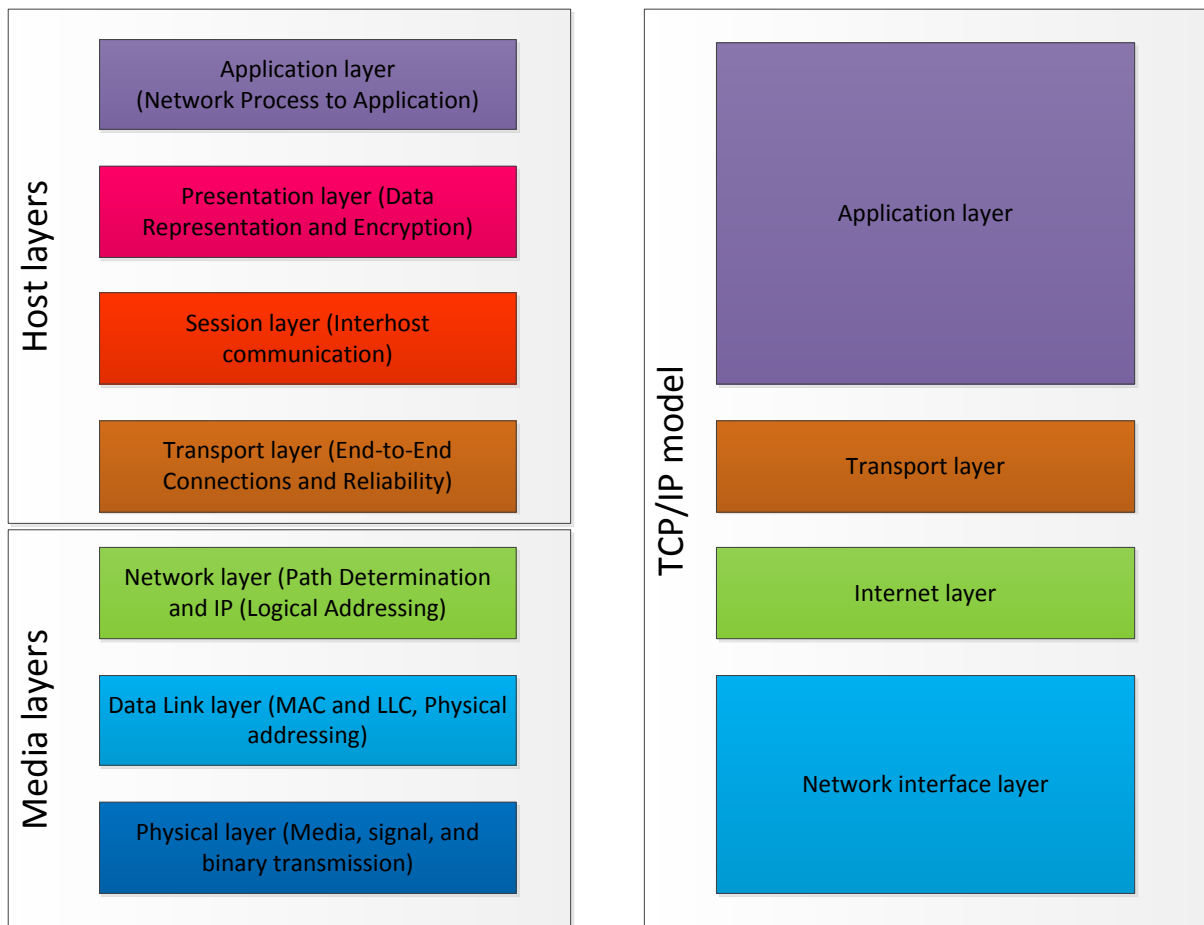


Figure 7 The OSI layered architecture and the TCP/IP model

Now that we described the layered architecture of computer networks, we take a look at the layered architecture of applications that interact with the application layer of the TCP/IP model.

There are various guidelines and architectures available for building layered applications. We make use of the architecture that originates from the Microsoft Application Architecture Guide [8]. There are other variants on this layered architecture available from competitors like IBM. We use the Microsoft version because we build our prototype in a Microsoft environment. We use this architecture as-is because the most important is to note that applications are composed of layers. The overview of the layered architecture is shown in Figure 8. Note that this is the architecture of an application and not of a computer network.

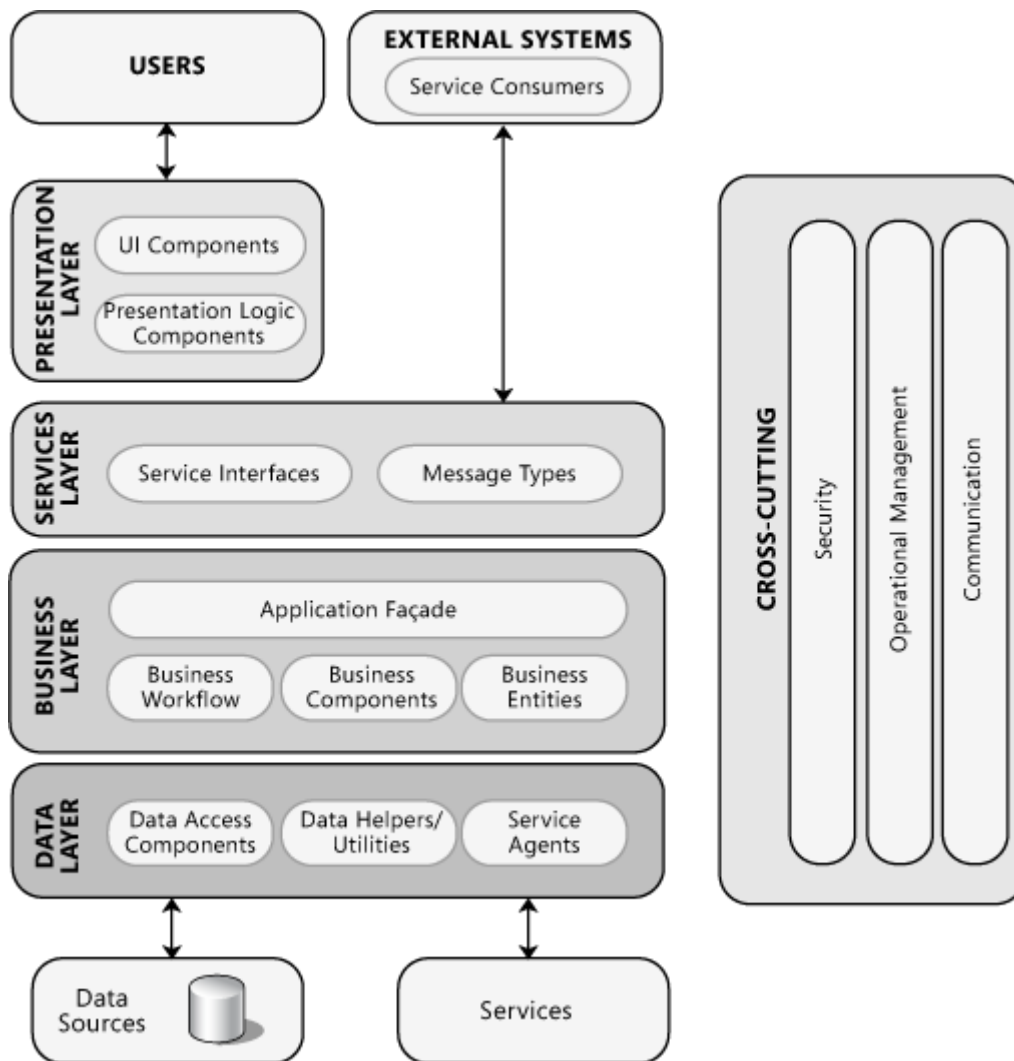


Figure 8 Layered architecture of Microsoft information systems [8]

In this architecture each layer uses functionality from the layer below and provides functionality to the layer above it. The basic three layers are the data layer, the business layer and the presentation layer. Because in this research the information systems often expose services to other systems and exchange data with other systems, we use the variant of the layered system that also incorporates a services layer. We shortly describe each layer, based on the Microsoft Application Architecture Guide [8].

- **Presentation layer.** This layer is responsible for providing the user oriented functionality and for managing the user interaction. The layer contains components that provide a bridge to the services layer, or if that is not available, to the business layer.
- **Services layer.** This layer is not explicitly available in all information systems. However, if the system must offer services to other systems as well as providing functions to support clients, the services layer is a common approach to expose business functionality of the system.
- **Business Layer.** This layer contains the core functionality of the system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other systems can use.
- **Data Layer.** This layer provides access to data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services. The data layer exposes generic interfaces that the components in the business layer can consume.

Now we relate the application architecture to the TCP/IP model. The relation between the architecture of the application and the TCP/IP model is that the services layer from the application uses the application layer from the TCP/IP model. So suppose the services layer from the source system needs to send data to the services layer of the target system, the data goes to the TCP/IP model and its lower layers and eventually reaches the services layer of the target system. The data flow in this situation is depicted in Figure 9. In the picture is not shown how the services layer of the source system acquires its data from lower layers and how the services layer of the target system stores its data because it is out of the scope of this research.

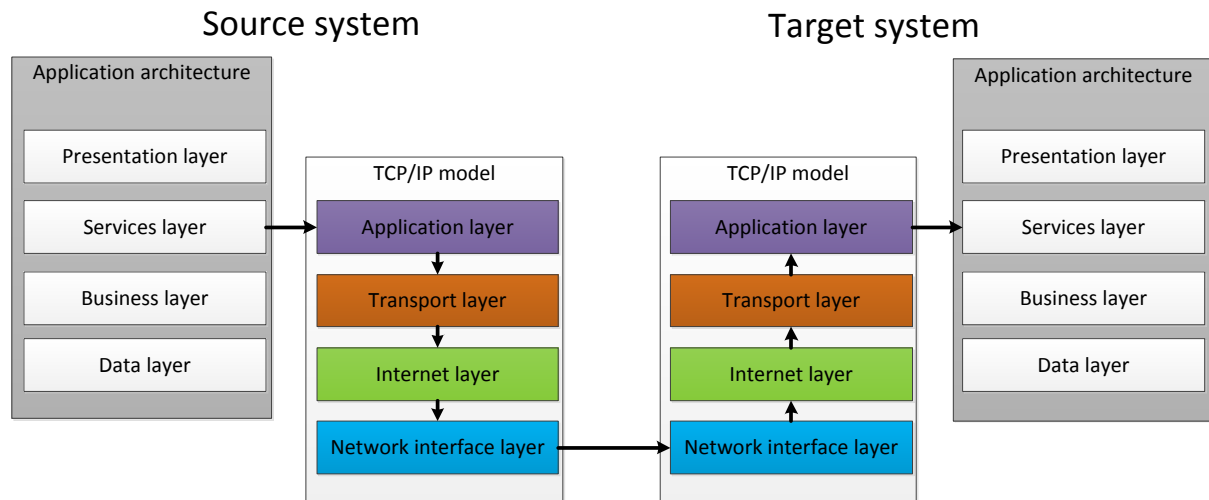


Figure 9 Overview of layers involved in a data exchange

Relation to this research

In our research we do not investigate monitoring on layers that are part of the TCP/IP model. For monitoring these layers, there are already a lot of tools available. We only focus on the services layer from the application architecture because we want to monitor the ultimate result of a transaction in the target system.

For this research the most important fact to note is that applications are composed of layers and that the services layer of an application interacts with the application layer of the TCP/IP model. The layers of the applications architecture and the layers of the TCP/IP model are not directly comparable.

2.1.3 Timeliness of messages

In data exchanges between systems, messages with information are exchanged. It is important to distinguish real-time and deferrable messages. Messages are real-time if the result of the message is needed immediately. So if the message cannot be processed the system must halt and report an error. Deferrable messages are messages of which the result is not immediately needed. This means that the system can process the message after the transaction which created the message has completed. If the message cannot be sent at a certain time it can be useful to report that, but it does not directly impose a problem because the system can retry it later [5].

In practice deferrable messages are often processed in batch transactions. These messages can each be stored in a separate file or can be combined in a large file. In this case multiple deferrable messages are consecutively processed. Data exchanges that use batch transactions are normally scheduled and started at certain times whereas real-time data exchanges execute the transaction directly when input is received.

Relation to this research

For monitoring purposes it is important to make the distinction between real-time and deferrable messages because the monitoring tool will have to monitor different aspects. For real-time messages actions can be monitored at the time the message is received, but for deferrable messages monitoring can be done when the batch process is started.

2.2 Possible failures in data exchanges

In this section we identify possible failures that can occur in data exchanges. We need to investigate these failures to be able to determine what should be monitored. Before we go into detail about possible failures, we first define the term failure.

A failure is defined as a deviation between the observed behavior and the desired behavior of a software system [9].

In IT systems failures can occur at various components of the system. Data exchanges consist of at least four components: a source system, middleware of the source system, middleware of the target system and the target system itself. The components use layered application architectures and layered network architectures. At both layered systems failures can occur.

2.2.1 Failures in network communication

Failures can happen at several layers of the data exchange. In section 2.1 we explained that application that exchange data make use of the TCP/IP model, also referred to as the OSI model. For this model there are numerous solutions available to monitor its status and process. Therefore we exclude errors at this level in this research. For this research it is sufficient to know that errors can occur in this part of the data exchange.

2.2.2 Application failures

In this research we focus on failures that affect layers in the application architecture. As mentioned before errors can also occur in the TCP/IP model, but most of those errors will not propagate to the application. However, if a link between two systems is completely malfunctioning, the application will suffer from these error and these errors can be monitored.

Robinson [10] identified a list of problems that can occur when using services. This illustrates why monitoring is important. According to Robinson the following problems can occur.

- Service Failure. A service can (silently) fail to produce any results.
- Tardy Service. A service can produce results after a specified deadline. Dissatisfaction may also be associated with results produced by the service.
- The Reluctant Service. A service can consistently produce late results.
- Service Spam. A service may make frequent requests of other services.
This failure cannot happen in the case we use in our research because all connections are secured and are not publicly available.
- Run on the Bank. A set of buyers, or an individual buyer, can overload a service with requests. For example, a collection of competing retailers could flood a vendor with requests for a popular product (e.g., “tickle me Elmo” at Christmas). Knowing this, a strategic retailer could abuse other retailers by using a Service Spam strategy to prevent them from placing requests.
This failure cannot happen in the case we use in our research because all connections are secured and are not publicly available.
- Doesn’t play well with others. An individual service may work fine in isolation. However, when combined with others, it may fail. Such scenarios will be commonplace as new traders enter the marketplace while web service standards evolve.

- The Gang that can't Shoot Straight. Just as an individual service may fail to work with another service, an aggregation of services may be prone to failure.
- Service Spoof. A non-legitimate trader can penetrate an electronic marketplace, in spite of security precautions. Thus, it will be important to monitor transactions for unusual behavior that can be associated with a spoofed service.
This failure cannot happen in the case we use in our research because all connections are secured and are not publicly available.
- Denial of Service. A non-legitimate trader can make frequent requests of services, as part of a denial of service strategy.
This failure cannot happen in the case we use in our research because all connections are secured and are not publicly available.

Some of the above failures are caused by software failures, which are errors in the program code of software. According to Britton and Bye [5] software failures are the worst kind of error. To minimize problems in software, programmers should write code that:

- Prevents bad data from getting in the database. All inputs should be checked and if any errors in the data are found, the transaction should not be committed.
- Checks for corrupt data when data is read from the database.
- Displays as much information as possible when errors are detected.

Relation to this research

In this research we do not concern communication failures, but only failures that can be detected at the services layer of an application. From the complete list which was given by Robinson, only a subset of the possible failures is applicable to this research because not all failures can happen or are important in this research. We use only the following failures.

- Service failure
- Tardy service
- The reluctant service
- Doesn't play well with others
- The gang that can't shoot straight

For failures that we do not use in our research, the reason why they are not used is explained in the complete list above.

2.3 Data quality

Monitoring tools should lead to better data exchanges, which lead to better data quality and which ultimately lead to improved customer satisfaction. This is explained in section 1.2. Because improved data quality is a target of this research, we state what we define as data quality. We use the definition of Wang and Strong [11]. Wang and Strong did an extensive literature study and a two-stage survey among data consumers to research what data quality is.

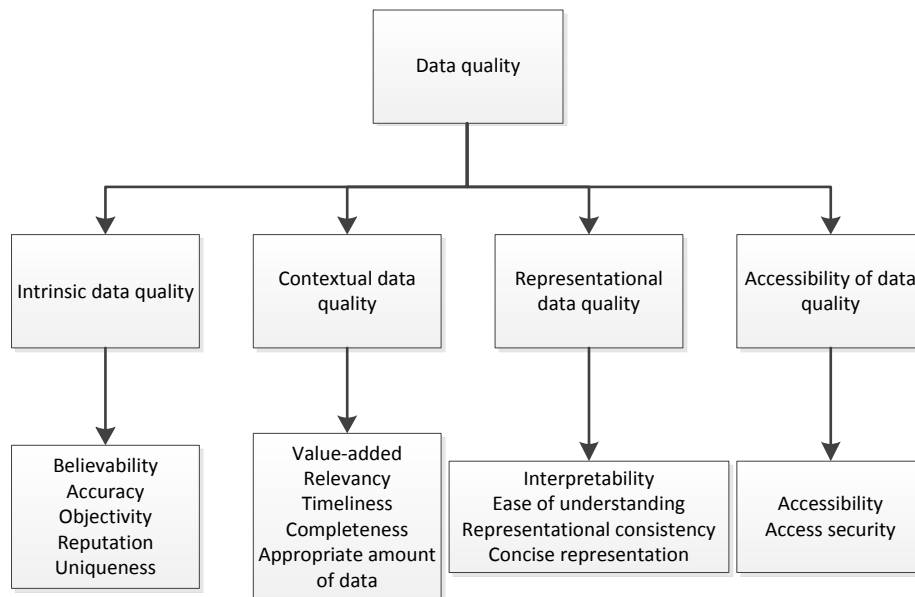


Figure 10 A combination of Wang and Strong [11] and Akoja et al. [12]

According to Wang and Strong data quality has four aspects as shown in Figure 10. Each of these aspects has several properties. These aspects are the following:

- Intrinsic data quality. This aspect includes accuracy and objectivity, but, in contrary to the traditional view, also believability and reputation. This is similar to some aspects of product quality. So when implementing a system you should also ensure believability and reputation of the data.
- Contextual data quality. This means that data quality must be considered within the context at hand. So the data consumer must see relevant data which is needed for the task that is executed. In the context of this research timeliness and completeness are the most important.
- Representational data quality. This aspect is about the format of the data and the meaning of data. This implies that for end users data must not only be concise and consistently presented, but also interpretable and easy to understand.
- Accessibility of data quality. Accessibility is presumed by consumers. Not all studies treat accessibility as part of data quality, but according to Wang and Strong there is little difference in treating accessibility as an aspect of overall data quality or separating it from other categories of data quality.

The same classification of data quality aspects is used by Batini and Scannapieco [13]. They call it the empirical approach. This approach is based on feedback from data consumers, so we use this approach in our research. In the end the target of this research is to make customers satisfied, so we can use their opinion on data quality.

Akoja et al [12] add to the framework of data quality the uniqueness property. Especially for CRM systems this is an important property. Because CRM systems often aggregate data from different sources, these data have to be combined instead of duplicated.

Batini and Scannapieco [13] also describe several methods to improve data quality. There are two approaches: data driven and process driven. Data driven approaches use existing data and try to improve the quality of that data. Process driven approaches look into the processes which enter data into the system and try to eliminate the root causes. In this research we focus on process driven

approaches by improving the process and monitoring of data entry in information systems. We do not strive to improve the data quality of existing data.

In section 2.2 we identified possible failures of data exchanges. We now map those failures to concepts of data quality. These concepts are taken from the lowest row of Figure 10 and only the affected aspects are shown. Each failure can affect other aspects of data quality. The mapping of failures to data quality aspects is shown in Figure 11.

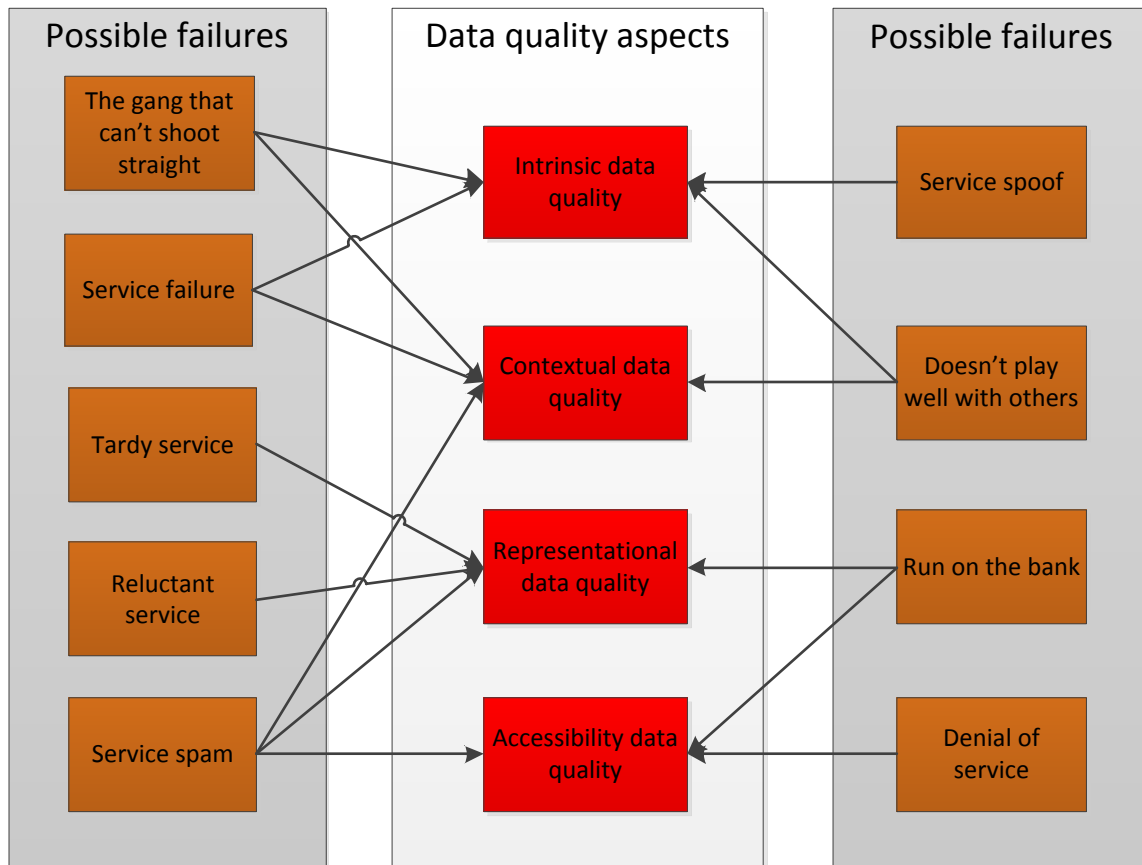


Figure 11 Mapping of failures to data quality aspects

Relation to this research

As can be seen in Figure 11 all four aspects of data quality are impacted by possible failures of data exchanges. In section 2.2 we explained that not all possible failures can happen in our case. Because service spam, run on the bank and denial of service failures cannot happen in the case we use in our research, accessibility of data quality is not important for this research. So if we look at the failures that can happen and the affected aspects of data quality only the following aspects are important for this research:

- Intrinsic data quality
- Contextual data quality
- Representational data quality

2.4 Monitoring concepts for data exchanges

Because this research focuses on monitoring, it is important to investigate existing concepts of monitoring. These concepts can be used to create the reference architecture for data exchanges with monitoring.

When software is developed several verification techniques such as testing, model checking and theorem proving can be applied to verify its theoretical correctness. To check the current execution of software, monitoring is needed. Monitoring helps in detecting possible failures and in providing additional information on failures. This information can be used to correct errors and speed up the recovery of data [9].

In organizations where systems are hosted for customers a robust monitoring system is needed. This system should provide the service desk with all information needed about the current state of systems [14].

2.4.1 Methods of monitoring

There are a lot of monitoring tools for information systems available. However, most tools monitor if the operating system is still running and if there is enough drive space etc. For monitoring functional aspects of an application other monitoring tools are needed that monitor different aspects.

If standard middleware is used, the default monitoring capabilities of the middleware can be used. If these do not suffice or if monitoring of functional aspects is needed, interceptors can be used. This is described by Morgan et al. [15]. These interceptors can be configured to analyze the data that is processed by the middleware. This results in the opportunity to monitor the data that is processed. It can for example count certain values and report them to another system which gathers the data and creates a visual representation of the monitoring data.

2.4.2 Levels of monitoring

Monitoring can be done by middleware or agents that are plugged into middleware. Monitoring can be done at different levels. For our research we are interested in syntactic monitoring and semantic monitoring. Both levels of monitoring are described in this section.

Syntactic monitoring

The term syntax is used to refer to a set of rules concerning the construction of natural languages, but also to computer programming languages. The term syntactic correctness can be used in various contexts. If we apply it in a natural language, a sentence which is grammatically correct is syntactic correct, but it may have no meaning. In our research we define that a transaction of a data exchange is syntactic correct if no error occurs. In most situations this means that the middleware of the target system can process the transaction and can execute the necessary steps to complete the transaction. With syntactic monitoring we want to monitor the number of transactions, the number of records that are created in these transactions and the errors that have occurred. The process of syntactic monitoring is shown in Figure 12.

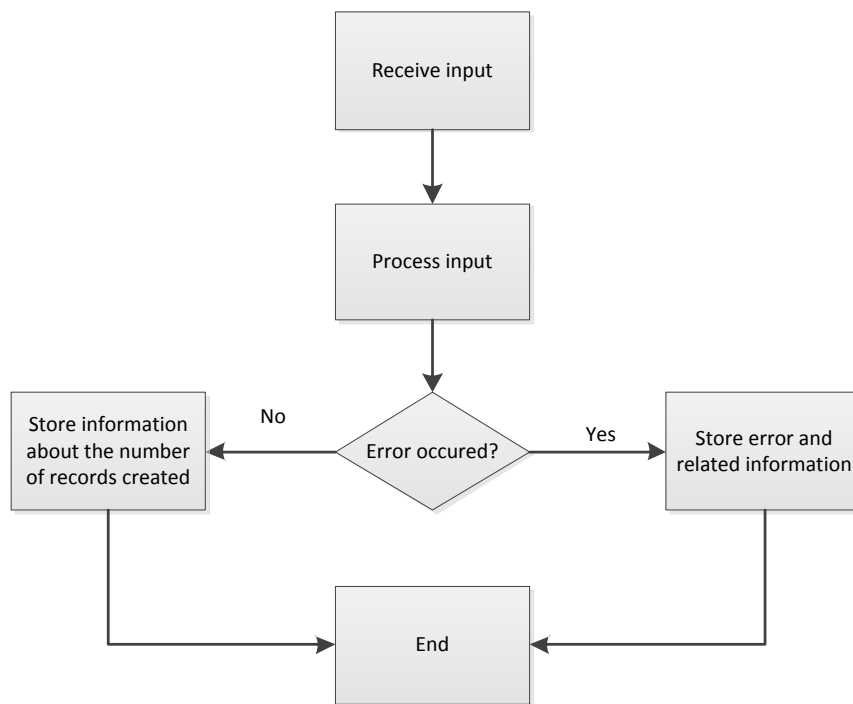


Figure 12 Process of syntactic monitoring

Semantic monitoring

Semantics is about the meaning of content. This content can consist of words, phrases or other types of data. In our research we define a semantic correct transaction as a transaction that is syntactic correct and has meaningful data as a result. To illustrate this, we give an example. Suppose there is a transaction that creates a person with a phone number of only 3 digits. Syntactic this transaction is correct because the transaction completes without errors, but the transaction is semantic incorrect because a phone number of 3 digits is not meaningful and obviously incorrect.

So in contrast to syntactic monitoring, semantic monitoring also analyzes the content of messages that are exchanged. The contents can be compared to rules that are specified beforehand. If the input matches the rules, no error occurs. If the input does not match the rules there is an error.

In some cases restrictions on the contents of messages are included in the syntax of the target system. If this is the case and the contents of the message do not meet the requirements the message will be rejected and a syntactical error will occur. In the case we use in our research most data fields are strings and will accept any input. Therefore separate monitoring has to be done to monitor semantics. So to summarize: if a transaction error occurs we call it a syntactic error. If the contents of data in the message do not meet the requirements we call it a semantic error.

The process of semantic monitoring is similar to the process of syntactic monitoring. In practice both types of monitoring can be executed by the same program. In Figure 13 the process of semantic monitoring is shown. Important in this process is the comparison of the input to rules that are specified beforehand. These rules are needed because the semantic correctness of a transaction cannot be determined by whether a transaction error has occurred.

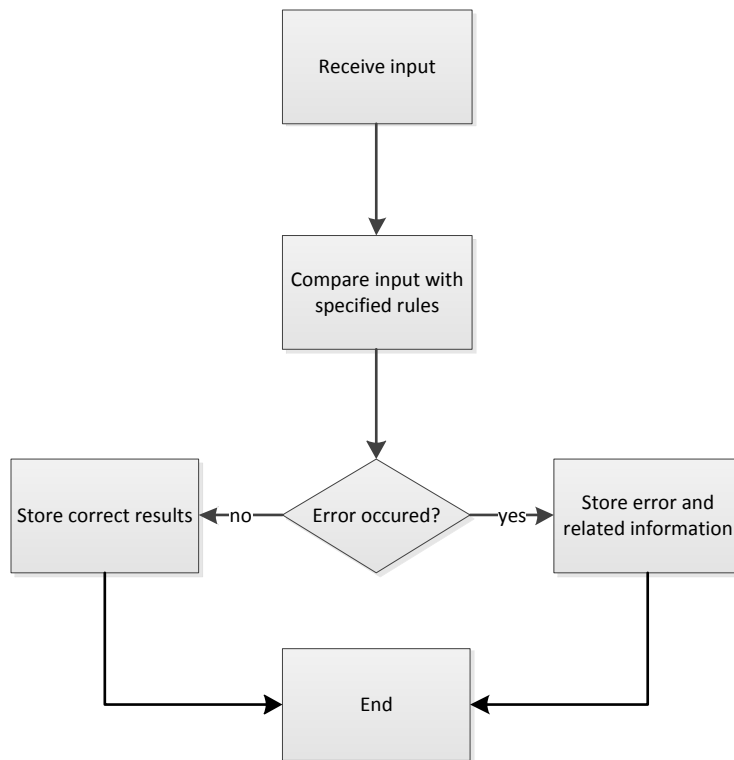


Figure 13 Process of semantic monitoring

2.4.3 Polling versus publish and subscribe

On the web there is a lot of debate on whether polling or publish and subscribe is the best solution to send monitoring results to a central server. There is no hard conclusion to draw on this topic and each solution has its advantages [16].

Publish and subscribe monitoring is the concept in which the system that is monitored, runs a monitoring agent which collects data and reports to a central server that is subscribed to the agent. This monitoring agent is plugged into the software that is to be monitored.

Polling, also called agent-less monitoring, works without an additional running process on the server that is monitored. In this concept the central server polls the servers that it wants to monitor and collects the data itself. Also in this case the server that is to be monitored, has to deliver the required data to the central server. So in some way there is still a monitoring agent, which collects the data and is able to send the data to the central monitoring server.

According to Roepke [17] the difference between publish and subscribe and polling lies in the fact whether the server that is monitored takes the initiative to send data or that the central server asks for data. If the server that is monitored uses a sort of interrupt based model and initiates the transfer of monitoring results, we call it publish and subscribe. If the central server initiates the data transfer to request data, we call it polling.

For our research both polling and publish and subscribe are possible solutions. Regardless of which mechanism is used, the server that is monitored needs some logic to gather information and to send information. We call this logic a plugin. For most general concepts of monitoring it does not matter whether this plugin uses an agent to send data or that it sends data in response to a request from the central server. So in descriptions where it is not relevant if the system is agent-based or agentless, we use the term plugin.

2.4.4 Reporting results of monitoring

At each data exchange a monitoring plugin gathers information. This information is sent to a central monitoring server to create an overview of all data exchanges that are monitored. In Figure 14 the relation between the data exchange, the monitoring plugin and the central monitoring server is shown. Middleware often has monitoring capabilities. These monitoring capabilities can be used to implement the monitoring plugin functionality. This functionality is used to gather information and to send useful information to the central monitoring server. We do not go into detail about the configuration of middleware and the format in which information is sent. This can be determined for each data exchange separately as long as the format is supported by the central monitoring server.

As can be seen in the figure, monitoring is done at the target system. This is because in practice we often only have influence on the target system, which in the case of CRM Resultants is Microsoft Dynamics CRM. Monitoring can also be done at the source system, but this would be the responsibility of the external administrator of the source system. In cases where our system functions as the source system, monitoring can be done by the external party that manages the target system. They can use the reference architecture we specify in this research to set up their own monitoring solution.

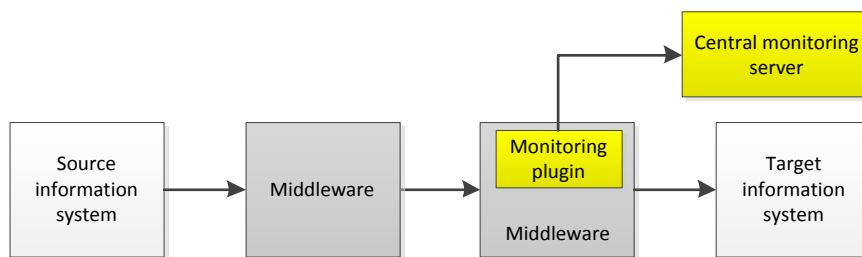


Figure 14 Monitoring architecture of a data exchange

The central monitoring server can be connected to multiple monitoring plugins. In this way it can monitor multiple data exchanges and create an overview of all data exchanges. This is further explained in our reference architecture.

Relation to this research

In this research we use the concept of plugins to monitor a data exchange. This plugin can be implemented by various types of middleware. The plugins have a connection to the central monitoring server to send the monitoring information to the central monitoring server. This central monitoring server is used to provide an overview of all data exchanges.

3 Case study on failures of data exchanges

In our literature review in chapter 2 we found several possible failures of data exchanges. These are possible failures for data exchanges in general. Because we execute this research on a specific case and we build our prototype for a specific case, we execute a case study to find failures that can occur in this specific situation.

This case study is focused on data exchanges that are built by CRM Resultants. We take some data exchanges of which is known that they do not always function as desired and we gather all failures that occur in these data exchanges. Gathering this information from practice helps in building the prototype and eventually validating if failures are correctly monitored.

3.1 Web service data exchanges of the Hogeschool Utrecht

The CRM system of the Hogeschool Utrecht has a custom built web service data exchange with the website of the Hogeschool Utrecht. The CRM side of the data exchange is built by CRM Resultants and the side of the website is built by Evident, the company which builds the website for the Hogeschool Utrecht. The following data is exchanged between CRM and the website.

- Requests for brochures. People who are interested in following a study at the Hogeschool Utrecht can fill out a form to request a brochure, which is sent by post. Information that is entered on this form is sent to CRM using web services. This information is sent in two steps. First information about the person is sent. This is for example the name, birthdate and address. Afterwards information is sent about the brochure that is requested and this information is related to the personal information that is sent in the previous step.
- Events. In CRM events can be created which are then sent to the website.
- Subscriptions for events. People can subscribe for events. These subscriptions are sent to CRM in two steps. First the contact information is sent and afterwards the information about the subscription is sent.
- Information about studies of the Hogeschool Utrecht is sent from CRM to the website.
- Courses are retrieved by CRM from the website.
- Information about postal codes is sent from CRM to the website. On the website in forms only the postal code and house number have to be filled in and the rest of the address is completed automatically.

An overview of the data that is exchanged is shown in Figure 15.

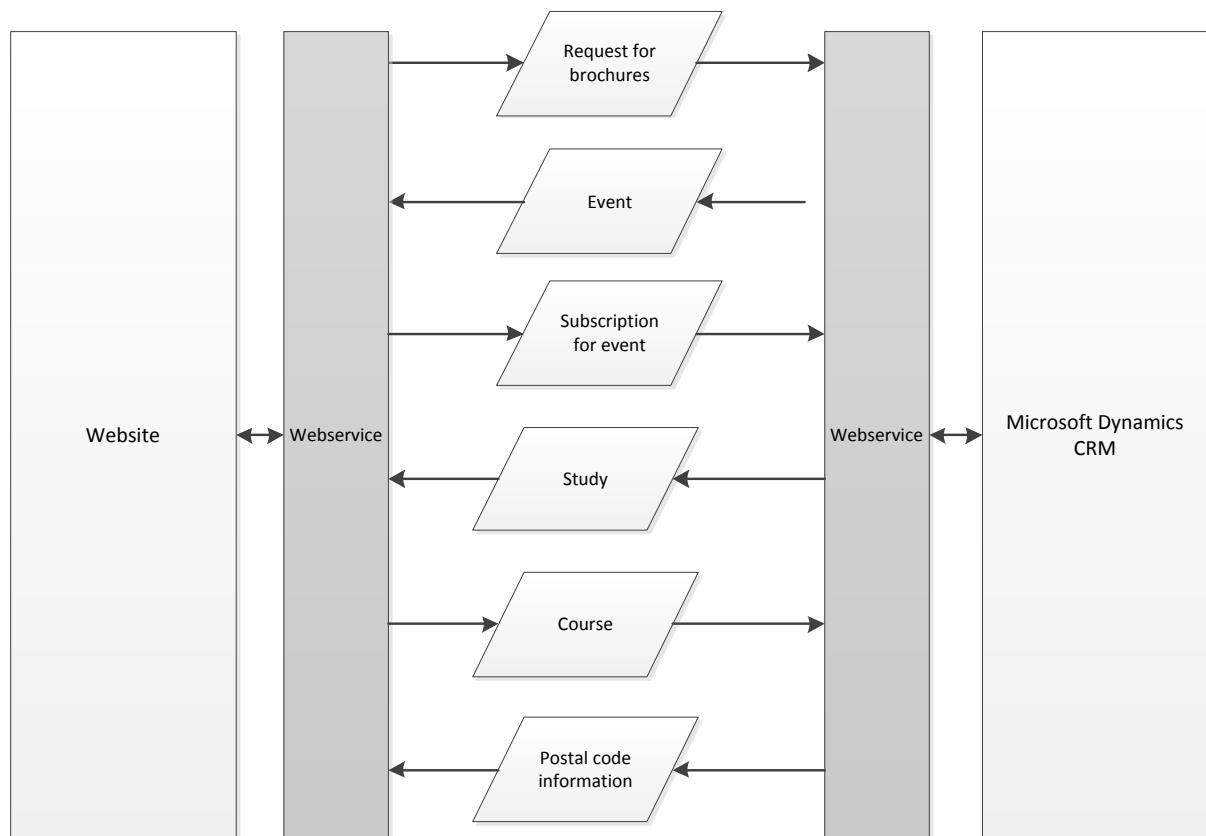


Figure 15 Overview of the web service data exchange between the website and CRM

Unfortunately these data exchange have had some problems in the beginning. By doing extensive analysis on these problems a lot of errors are identified and changes to the data exchange are made to prevent these errors from occurring. Because in this research we aim to gather as much information about errors as possible, we also list errors that cannot happen anymore.

3.1.1 Reasons for failures that have occurred

In this section we list the reasons for failures that have occurred in practice. Information about these failures is retrieved from employees of CRM Resultants and log files of data exchanges.

- Message cannot be processed because it is syntactically incorrect. This can happen if one party that builds the data exchange changes something to the specification without notifying the other party.
- The record that is referenced cannot be found. Because information about the brochure request has to be related to the contact information, the unique ID of the contact has to be referenced in the second message. It is possible that this record is not found.
- There is no reference specified to which the record should be linked.
- Multiple records found. Because information should not be duplicated in the database, before inserting records, a lookup is done based on certain criteria to check if the record already exists. If this is the case, the record is updated instead of a new record being created. If multiple records are found, it is impossible to determine which of them should be updated.
- Data wrongly formatted. The message is syntactically correct, but the data it contains does not meet the requirements of the data format of the target system.
- Data incomplete. The message is compatible with the specified WSDL, but some attributes are missing.
- Service fails silently. Occasionally the service fails without producing an error message or anybody noticing.

- Web service is very slow and responds after a reasonable amount of time. This does not directly result in a failure, but can cause long waiting times for the user.
- Time out. If the web service responds too slowly, a time out occurs and the transaction is aborted. This data is not sent again afterwards.
- Could not establish trust relationship for the SSL/TLS secure channel. The underlying connection for the data exchange did not work.
- SOAP exception because an invalid argument was passed to the web service.
- SOAP exception because the length of an attribute is exceeded.
- Send failure. A connection to send data cannot be opened.

3.2 Scribe data exchange Hogeschool Utrecht

The CRM system of the Hogeschool Utrecht has various data exchanges. Data exchanges are not only realized with web services, but most of them are built with Scribe [18]. Scribe is an integration product that can connect to various information systems and can handle various interfaces such as web services, FTP, e-mail and file systems. From this interfaces it accepts files in different formats such as XML, Excel and CSV. Most data exchanges that are built with Scribe, are batch exchanges in which a file with multiple records is imported in one job. Jobs can be scheduled to check for input at a certain time.

For each data exchange a definition has to be made in Scribe to define which steps have to be taken to transform and store the data. Scribe has built in functionality to export failed records to a file and to e-mail a summary of the job that is executed.

The Hogeschool Utrecht uses Scribe to exchange data with Osiris, a student administration system, and to import data from Excel files. In Figure 16 this data exchange is schematically shown.

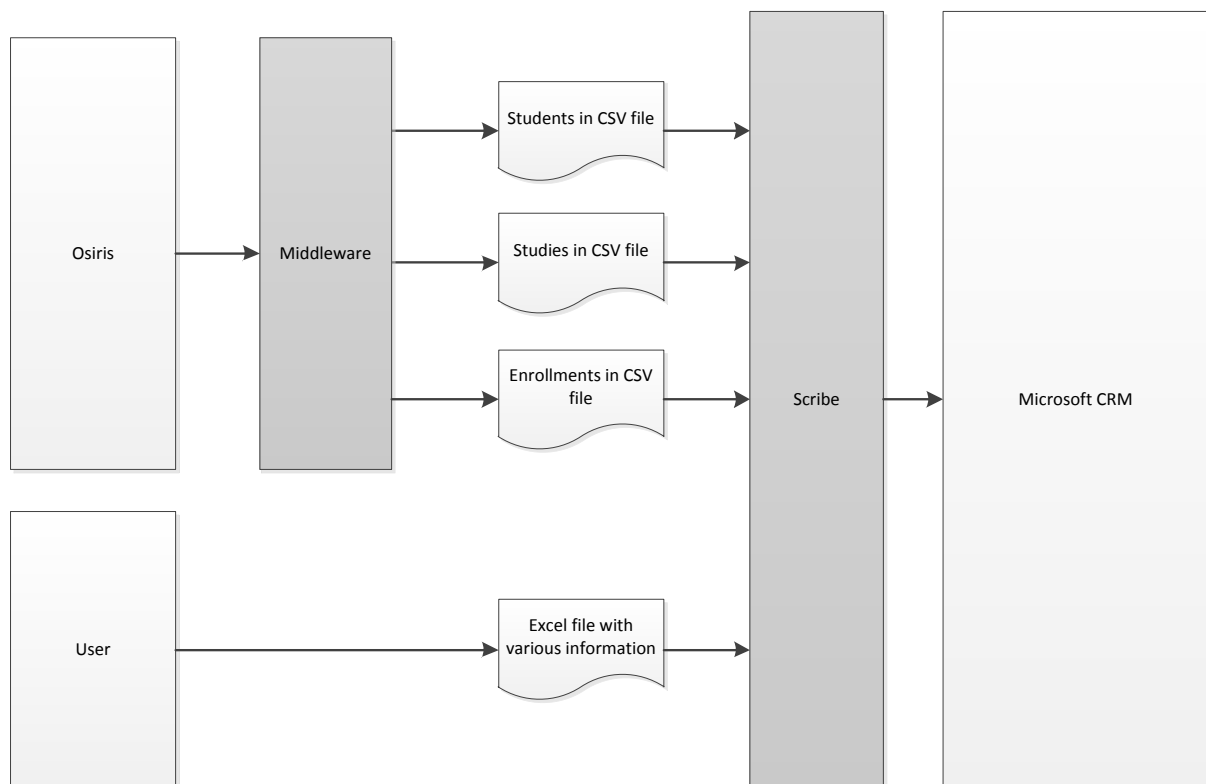


Figure 16 Data exchanges with Scribe

3.2.1 Reasons for failures that have occurred

In this section we list the reasons for failures that have occurred in practice. Information about these failures is retrieved from employees of CRM Resultants and log files of data exchanges.

- Multiple records found. Because information should not be duplicated in the database, before inserting records, a lookup is done based on certain criteria to check if the record already exists. If this is the case, the record is updated instead of a new record being created. If multiple records are found, it is impossible to determine which of them should be updated. This error can occur with different types of records.
- Generic SQL error. For some reason the SQL server is not able to process the request.
- Post job failure. After a job has completed, Scribe moves files from one folder to another. This can fail because of a lack of disk space or files being in use.

3.3 BizTalk data exchange between Peoplesoft and Microsoft CRM of ROC van Amsterdam

The ROC van Amsterdam has a data exchange between Peoplesoft and Microsoft CRM. This data exchange is built with BizTalk. The data exchange executes the following tasks:

- Inserting/updating student information from Peoplesoft to CRM
- Inserting/updating enrolment information from Peoplesoft to CRM
- Deleting enrolment information from CRM.
- Inserting/updating company information from CRM to Peoplesoft
- Inserting/updating training periods (internships) information from CRM to Peoplesoft

The architecture of this data exchange is shown in Figure 17.

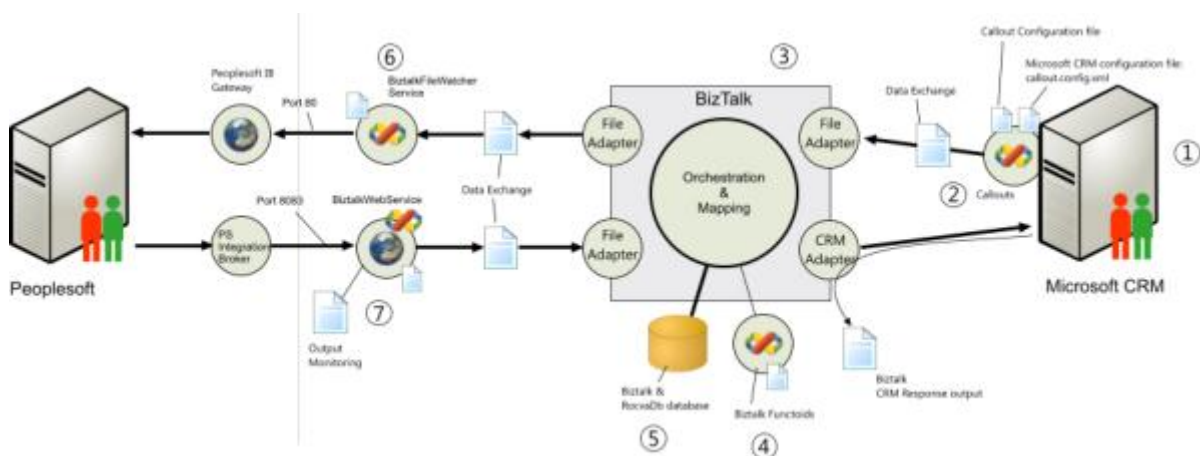


Figure 17 Data exchange between Peoplesoft and Microsoft CRM

In this data exchange the possible errors were specified beforehand. The result is that all possible errors are caught by the system. Therefore for this data exchange we list the errors that are caught by the system.

3.3.1 Errors that can occur

As can be seen in the figure above the complete data exchange consists of multiple components. At each component different errors can occur. Therefore we list the errors per component.

BizTalk file watcher

The BizTalk file watcher watches the contents of a folder on the server. If an XML file is placed in that folder, the file watcher picks it up and processes the contents of the file. This can result in the following errors.

- General exception in reading the file.
- SOAP exception which can be caused by an incorrect SOAP format.

BizTalk web service

The BizTalk web service processes the XML messages and stores the information in the target systems. The following errors can occur in this process.

- Format invalid. The format in which data is received does not meet the requirements.
- Invalid CRM ID. The record that should be updated or deleted is not found in the CRM database.
- ID from Peoplesoft is invalid. In CRM the Peoplesoft ID of a company is stored and in Peoplesoft the CRM ID of a company is stored. If a company in CRM already has a Peoplesoft ID, but this is different than the Peoplesoft ID that is provided, this error occurs and the request is discarded.
- System exception. The system can have a failure which makes it impossible to store data.
- Company not found. There is no company found in CRM with the supplied ID.
- Failure to connect to web service of CRM.
- SOAP exception. A general SOAP exception occurred while interacting with the web service.
- BizTalk web service exception. The web service of BizTalk has a generic failure.

BizTalk Functoids

Functoids in BizTalk are building blocks which are used in field mappings.

- Unable to open registry key. For some configuration tasks settings are stored in registry keys. This error can occur if the registry key is not found.

3.4 Combined results of this case study

In this case study we looked at three types of data exchanges. For each type of data exchange different errors can occur, but some errors overlap. Therefore in this section we create a list of all possible errors regardless of the type of data exchange and we list each type of error only once.

To create an overview of the errors we grouped them in four categories. Within each category we estimated which errors occur most frequently and we list the errors by descending frequency. We estimate that most errors result from the contents of data. We estimate that errors in the other categories do not differ much from each other if we compare them by frequency of occurrence.

Note that all ranking of errors is based on estimation because there is currently no monitoring executed on these errors. There is no statistical data available yet.

3.4.1 Errors resulting from the contents of data

1. Multiple records found.
2. The record that is referenced cannot be found.
3. There is no reference specified to which the record should be linked.
4. Message cannot be processed because it is syntactically incorrect.
5. Data wrongly formatted.
6. Data incomplete

3.4.2 Errors in middleware

1. Generic SQL error.
2. Post job failure.
3. General exception in reading the file.
4. System exception.
5. Unable to open registry key.

3.4.3 Connection and performance errors

1. Time out.
2. Could not establish trust relationship for the SSL/TLS secure channel.
3. A SOAP exception.
4. Web service is very slow and responds after a reasonable amount of time.
5. Send failure.

3.5 Mapping of errors to data quality

To determine the impact of the errors, we mapped the errors to the aspects of data quality that are impacted by the error. In Figure 18 the errors are shown in grey blocks and the aspects of data quality are shown in blue. The category of the aspects is shown between brackets.

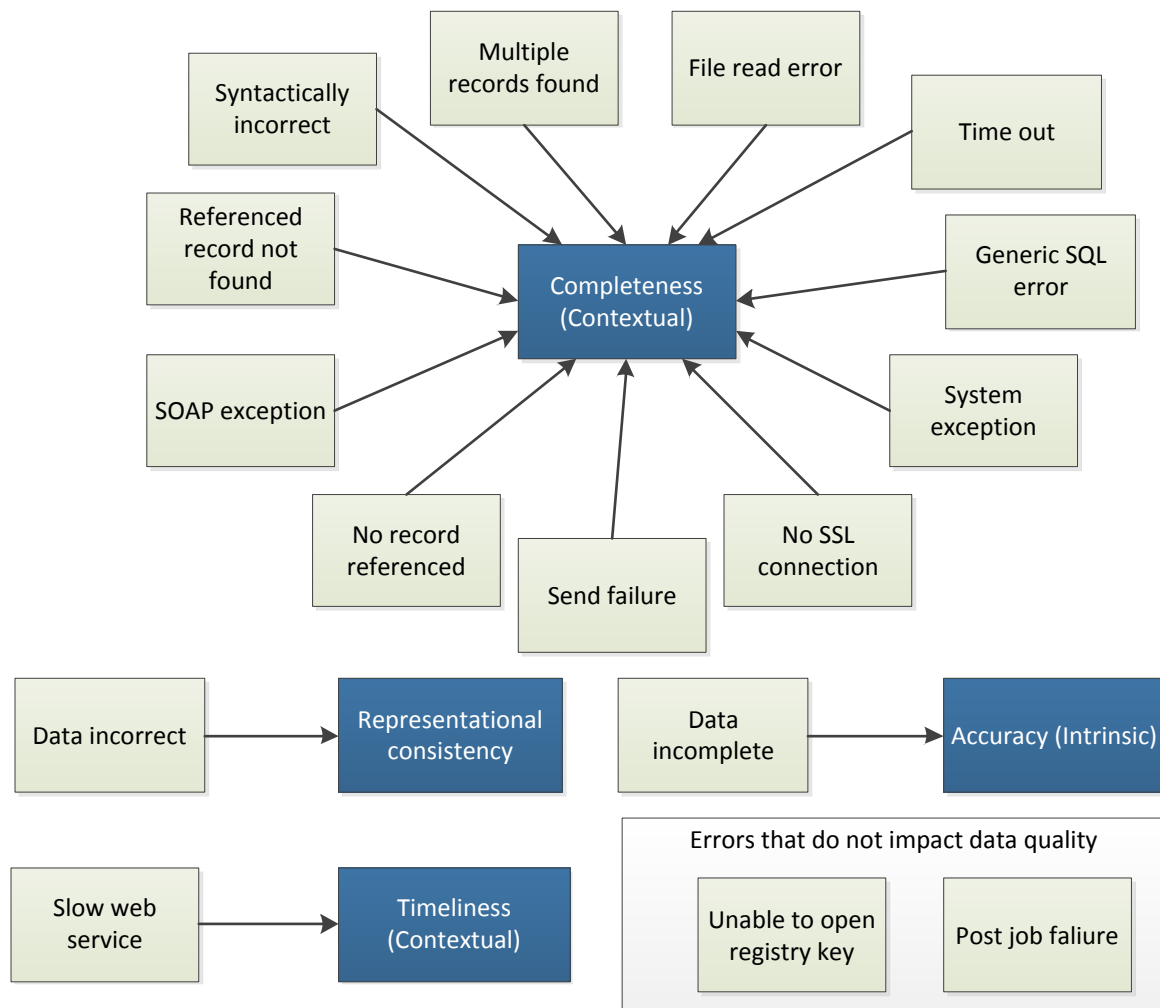


Figure 18 Mapping of errors to aspects of data quality

As can be seen in the diagram not all errors have impact on aspects of data quality. This is because the error does not affect transactions of the data exchange.

4 Monitoring reference architecture

Based on information from literature which is presented in chapter 2 and information from our case study in chapter 3 in this chapter we present our reference architecture. In this architecture we discuss the components that the monitoring solution has to incorporate and which requirements have to be met.

4.1 Components of the architecture

In the reference architecture we distinguish two types of components. First there are components that are needed by the data exchange itself and secondly there are components that are needed for monitoring those data exchanges.

Minimal components of data exchanges

- Source system. This is the system from which data is sent to the target system.
- Middleware of the source system. The middleware of the source system is responsible for extracting data from the source system, formatting it in the correct way and sending it to the middleware of the target system.
- Middleware of the target system. This middleware is responsible for receiving data from the middleware of the source system and processing it to store it in the target system.
- Target system. The target system is the system to which the data is stored.

Components for monitoring data exchanges

- Monitoring plugins. For each type of data exchange a plugin is needed that collects data about the transactions that are executed and the data that is exchanged. These plugins can be implemented by different applications. If middleware applications have built-in features for monitoring purposes, these can be used.
- Central monitoring server. This server is used to store monitoring information from different data exchanges and to present the information to the end user.
- A mechanism to collect data from the plugins. At this stage we do not impose restrictions on the choice if this mechanism is agent based or agent-less. The choice for one of the two options depends on the available monitoring solutions.

An overview of the reference architecture is depicted in Figure 19. In this architecture different data exchanges are shown. Each data exchange has its own plugin that reports monitoring information to the central monitoring server. This monitoring server aggregates this data and reports the information to the end user.

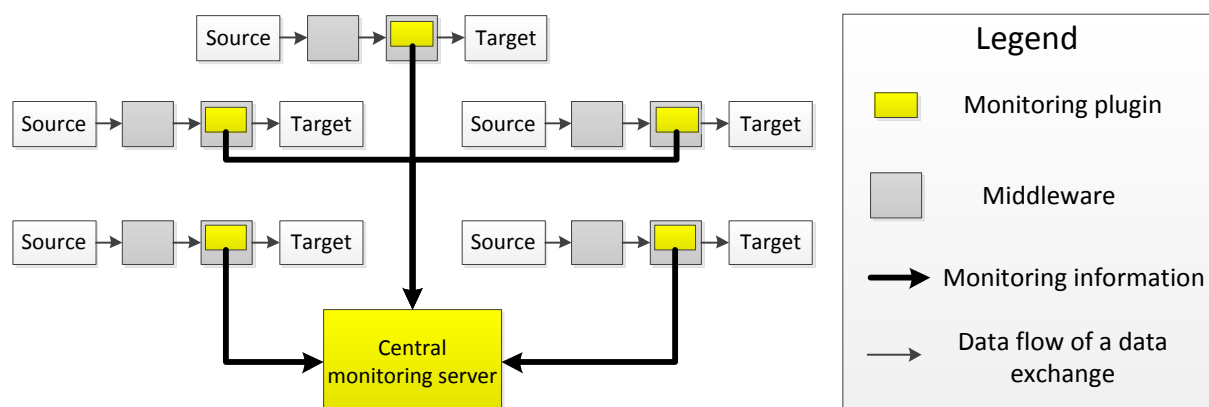


Figure 19 Reference monitoring architecture

Now we look at the process of an individual data exchange and the relation of the involved systems to the central monitoring server. If we zoom in on an individual data exchange, the architecture looks as depicted in Figure 20.

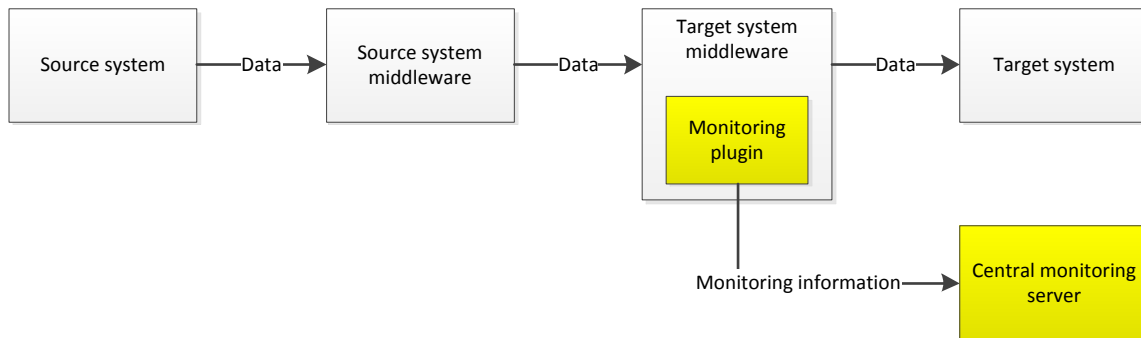


Figure 20 Reference architecture of monitoring an individual data exchange

4.2 Monitoring requirements of the architecture

Not all aspects of the reference architecture can be shown in a figure. Therefore we specify additional requirements which have to be met if a monitoring solution is built. We specify the following requirements.

- The solution must be able to aggregate data from different sources which may use different technologies.
- The solution must be able to monitor syntactic and semantic aspects of data exchanges.
- The solution must support real time and deferrable messages.
- The solution must be able to detect patterns in data exchanges.
- The solution must be able to report different types of failures.

4.3 Monitoring process steps

In Figure 21 a sequence diagram is presented on how the involved systems interact to exchange data and to process monitoring information. Each involved system has its own time line. The source system is the system from where the data originates. Both the source and the target system have middleware. The middleware of the target system is responsible for providing monitoring information to the central monitoring server. The central monitoring server receives this information and has to combine this information from different data exchanges. In this picture the initiative for sending monitoring information lies at the middleware target system but in practice also the central monitoring server can take the initiative to collect monitoring data.

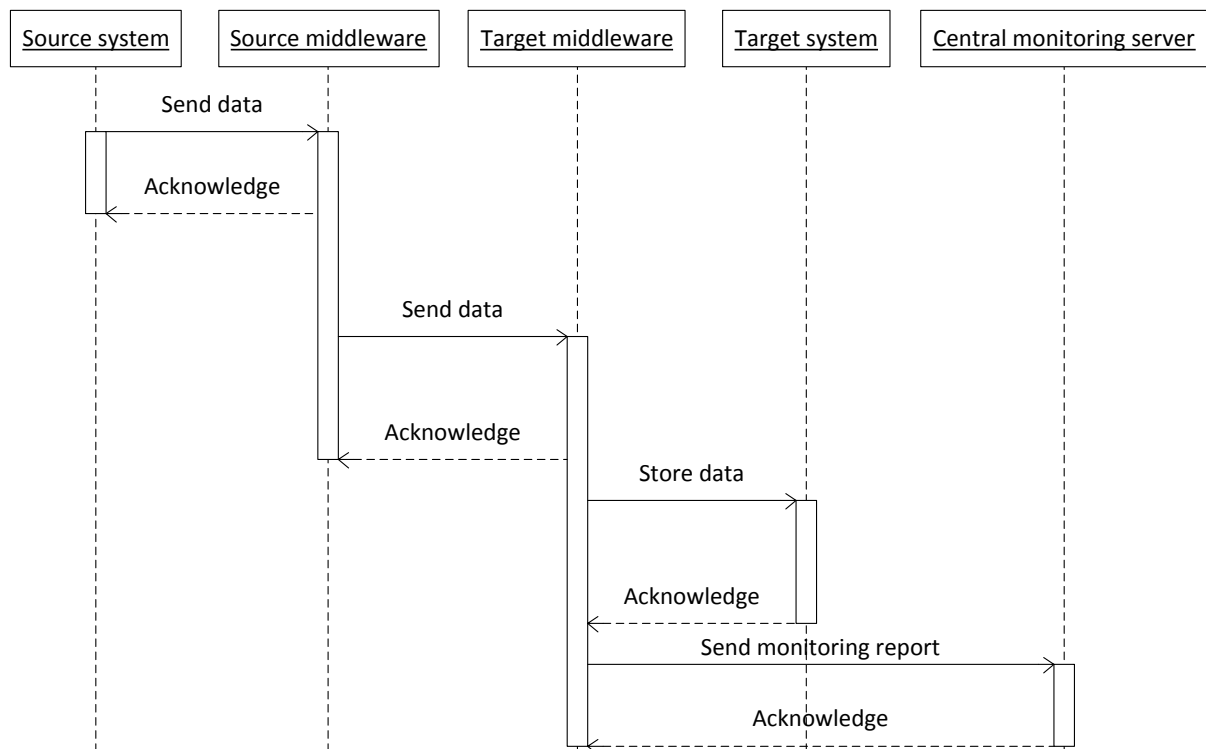


Figure 21 Sequence diagram of the monitoring process

4.4 Contents of monitoring information

The exact contents of monitoring information depend on the data exchange that is monitored. Below is a list of components that always need to be included.

- Server from which the information originates. Because the central monitoring server receives information from multiple servers, it needs to know from which server the data is coming to be able to relate the monitoring results to the correct server.
- Data exchange from which the information originates. Often there are multiple data exchanges running on a server. Therefore the central monitoring server needs to be able to determine from which data exchange the monitoring information originates.
- Date and time. The date and time of the transaction have to be stored in the monitoring results. This helps in finding the transaction that caused the error and in case the monitoring server has a downtime, it is still possible to determine when an error occurred.
- Error or success. The monitoring information should include information about whether a transaction was successfully executed or whether an error occurred.
- Error message. In the case of an error the type of error has to be specified.
- Context. To be able to locate the source transaction that caused an error, information has to be stored with which the transaction can be found. Normally this will be a transaction ID or a unique number from the source data.
- Information about the semantic correctness of a transaction. This information can be delivered for each transaction separately or a query can be executed at scheduled times that reports information about the semantic correctness of transactions that are executed.

4.5 Events that must be monitored

Depending on the type of data exchange a different set of events can be monitored. For each data exchange at least the following events have to be monitored and reported to the central monitoring server.

- A successful transaction. Successful transactions are needed to detect patterns in the data exchange and to detect a possible absence of transactions. If the source system does not send data to the target system, no error occurs, but based on the lack of successful transactions, it is still possible to detect an error.
- An error in a transaction. If a transaction cannot be executed this has to be reported to the central monitoring server including the type of error and its context.
- A fatal error in the middleware. It is possible that the middleware does not even try to execute transactions because the middleware itself encountered an error.

For real-time data exchanges it is preferred to report the events above for each transaction separately. For batch data exchanges it is also possible to aggregate the information and report them at once.

4.6 Requirements of the central monitoring server

The central monitoring server is the key component in our reference architecture. It is responsible for storing all monitoring data and providing a user interface to the end user. To be able to present the monitoring data in a clear way it must store the information in a structured way. A reference data model is therefore presented in Figure 22.

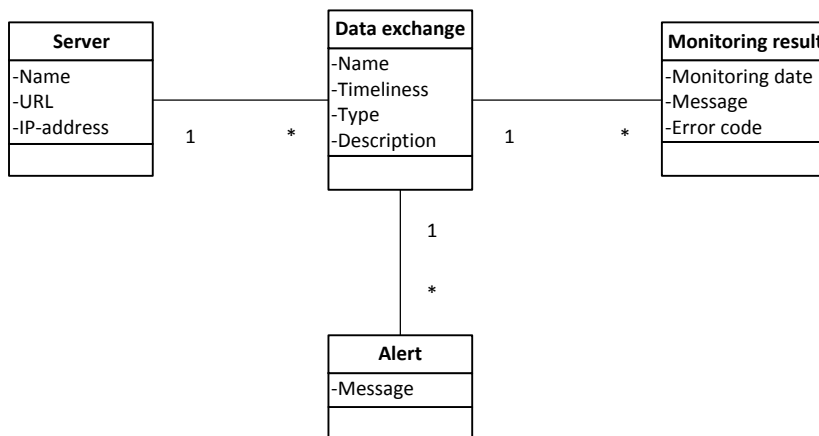


Figure 22 Reference data model of the central monitoring server

In the data model only the minimum required attributes are included. Depending on the environment additional attributes can be added. As can be seen in the data model the highest entity is the server. This server entity represents the server on which the target system that has data exchanges is running. This can be for example a CRM server. This server can have multiple data exchanges. All monitoring results are appended to these data exchanges and because data exchanges have a relation with the server, the monitoring results are indirectly linked to the server. This makes it possible to make selections of monitoring results at different levels. To provide an easy way for the end user to view potential problems, alerts can be appended to data exchanges. These alerts should be generated based on configured conditions.

Apart from storage of monitoring data, the central monitoring server must provide the following functionality.

- A user interface, preferably web based to make it easily accessible from different locations.
- Search functionality.
- Functionality to view results at the server level or at the data exchange level.
- Functionality to generate alerts on specified conditions.

4.7 How this reference architecture improves data quality

In section 2.3 we specified that in this research we focus on improving intrinsic, contextual and representational data quality. As can be seen in Figure 10 these three aspects of data quality are main categories of data quality. In section 3.4 we showed that errors that occur in practice affect the following subcategories of data quality. After the specific aspect of data quality the main category is specified between brackets.

- Completeness (intrinsic data quality)
- Accuracy (contextual data quality)
- Timeliness (contextual data quality)
- Representational consistency (representational data quality)

If we can monitor the errors that affect data quality as shown in Figure 18, we can improve data quality. Of course monitoring by itself does not improve data quality, but monitoring provides the opportunity to take actions if errors occur. Without monitoring most errors will be unnoticed and no corrective actions will be taken and that affects data quality in a negative way. In Figure 23 this reasoning is schematically shown. The system administrator will probably not take action on every single error, but each corrective action he does take, improves the data quality compared to the situation where there is no monitoring at all.

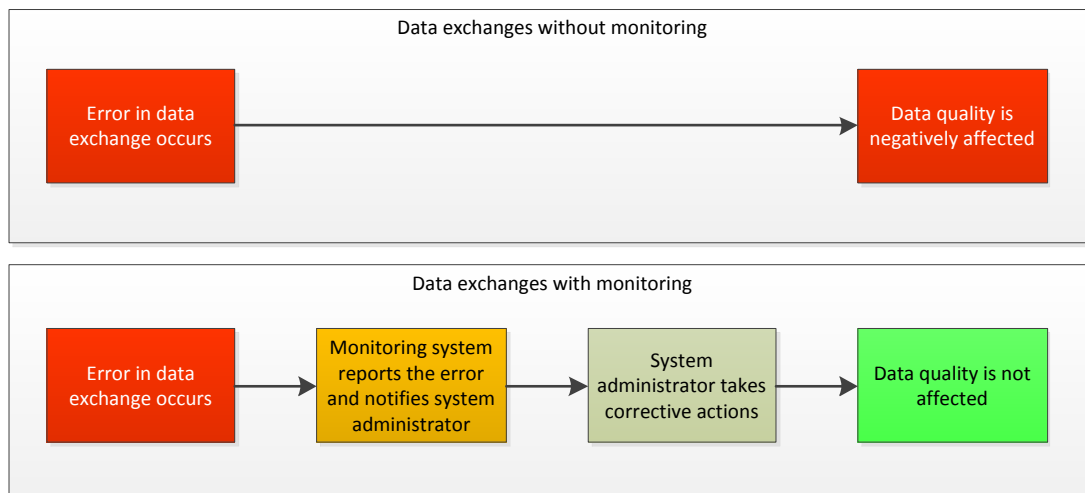


Figure 23 How monitoring improves data quality

5 Assessment of technical alternatives

In this chapter we present the results of our second case study. Based on our reference architecture we searched for existing alternatives that could meet the requirements. Not all alternatives are directly comparable to each other. Therefore for each alternative we specify which component of our reference architecture it covers and we describe each component separately, if applicable.

At the end of this chapter a summary of all solutions is presented. This summary contains an overview of the components that are included in the solution and the requirements that are met by the solutions.

5.1 Hyperic

Component	Covered
Monitoring plugin	Yes
Central monitoring server	Yes
Monitoring data collection mechanism	Yes

Hyperic [19] is a division of VMware, which offers enterprise monitoring solutions. The architecture of Hyperic is depicted in Figure 24. The way this architecture is built matches our reference architecture. All three components in our reference architecture are included in this solution.

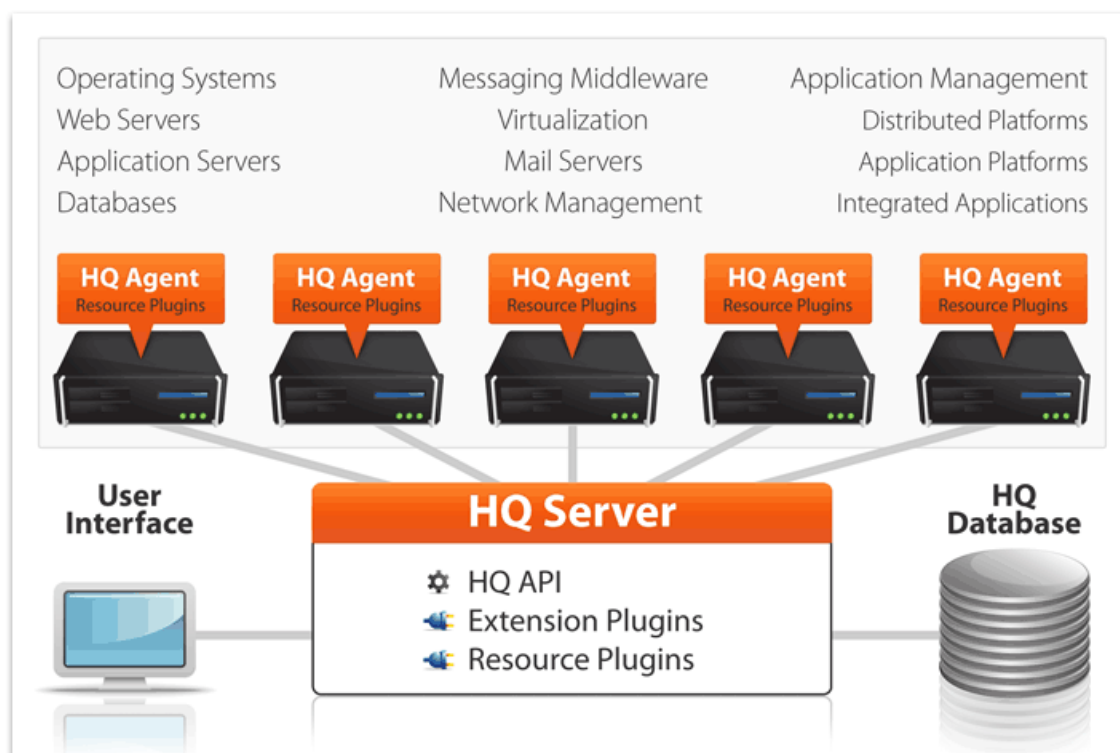


Figure 24 Architecture of Hyperic

5.1.1 Monitoring plugin

Hyperic makes use of so called HQ Agents which can be installed on the servers that have to be monitored. These HQ Agents detect the software that is installed and report to the central Hyperic Server. The agents are compatible with a wide range of applications including applications like IBM MQ, Microsoft SQL Server, Java Management Extensions and Internet Information Services. The agents however do not measure syntactic and semantic aspects of data exchanges. They are only capable of measuring system statuses.

5.1.2 Central monitoring server

Hyperic has a central server that stores monitoring results and presents it in a web based interface. It gives a complete overview of all system statuses and offers the opportunity to drill down on problems and analyze error logs. It is capable of detecting patterns and can notify the user if a pattern is different from the regular pattern.

5.1.3 Monitoring data collection mechanism

The data collection mechanism is implemented in the agents. Agents at the servers that are monitored take the initiative to send monitoring data to the central server.

5.1.4 Conclusion

Although Hyperic offers a lot of options, it is mainly focused on the system status. It can monitor applications at process level and show available disk space, cpu usage and related measures. It is not capable to monitor syntactic and semantic aspects of data exchanges. Unfortunately this product is not useful for our research.

5.2 WSMonitor

Component	Covered
Monitoring plugin	Yes
Central monitoring server	No
Monitoring data collection mechanism	No

WSMonitor [20] is an open source java project. It can capture and analyze HTTP requests and SOAP messages and present them in a graphical user interface. The interface of WSMonitor is shown in Figure 25.

Listen port: 4040			Target host: localhost			Target port: 8080		
Id	Time Sent	Request Encoding	Request Preamble	Request Length	Response Encoding	Response Preamble	Response Length	
1	19:01:51.333 2...	XML	POST /jaxws-fromjava/a...	215	XML	HTTP/1.1 200 OK	220	
2	19:01:52.004 2...	XML	POST /jaxws-fromjava/a...	216	XML	HTTP/1.1 500 Internal Ser...	521	
3	19:01:59.164 2...	Fast	POST /jaxws-fromjava/a...	139	Fast	HTTP/1.1 200 OK	139	
4	19:01:59.545 2...	Fast	POST /jaxws-fromjava/a...	141	Fast	HTTP/1.1 500 Internal Ser...	332	

HTTP Headers	SOAP
<pre><?xml version="1.0" encoding="UTF-8"?> <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"> <S:Body> <ns2:addNumbers xmlns:ns2="http://server.fromjava/"> <arg0>10</arg0> <arg1>20</arg1> </ns2:addNumbers> </S:Body> </S:Envelope></pre>	
<pre><?xml version="1.0" encoding="UTF-8"?> <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"> <S:Body> <ns2:addNumbersResponse xmlns:ns2="http://server.fromjava/"> <return>30</return> </ns2:addNumbersResponse> </S:Body> </S:Envelope></pre>	

Figure 25 Graphical interface of WSMonitor

5.2.1 Monitoring plugin

WSMonitor only has functionality that we require for the monitoring plugin. It captures the data by using port forwarding. So if a client makes a request, that request is copied to another port on which WSMonitor is listening. These results can show syntactic and semantic aspects of a request. It can however only report on the contents of requests that are done and cannot measure the effective results of a transaction in the target system.

5.2.2 Conclusion

Although WSMonitor provides a graphical user interface, it can only be used as a monitoring plugin for web services. The main problem is that it cannot monitor the effective results in the target system so for our research WSMonitor is not usable.

5.3 SoapKnox

Component	Covered
Monitoring plugin	Yes
Central monitoring server	No
Monitoring data collection mechanism	No

SoapKnox [21] is a monitoring solution that is capable of monitoring Java and .Net web services. The user interface is web based, so all information about the current status can be viewed in a web browser. This interface is however not capable of aggregating monitoring results from data exchanges that are not monitored by SoapKnox. Therefore we only recognize it as a monitoring plugin.

5.3.1 Monitoring plugin

SoapKnox has modules that can be plugged into various web servers. These modules can be used as the plugin component of our reference architecture. The modules do however only support web service technology.

5.3.2 Conclusion

SoapKnox can be used as a plugin to monitor web services. It is possible to define a lot of measures and this makes it capable of monitoring syntactic and semantic aspects of the data exchange. The monitoring results that it creates must however be collected by another application to be able to create an overview of data exchanges using different technologies.

5.4 ManageEngine

Component	Covered
Monitoring plugin	Yes
Central monitoring server	Yes
Monitoring data collection mechanism	Yes

ManageEngine [22] is a complete suite which includes all three components of our reference architecture. The architecture of the solution also matches our reference architecture in which plugins report to the central server.

5.4.1 Monitoring plugin

What we call plugins in our reference architecture, are called monitors in this solution. The available monitors can monitor various applications. The list of applications includes SQL Server, .Net Monitoring, SharePoint Monitoring, Database Query Monitor and Web Services (SOAP). The plugins are capable of monitoring traffic and statuses of services and systems, but they are not capable of monitoring syntactic and semantic aspects of data exchanges.

5.4.2 Central monitoring server

The results from all connected monitors are shown in the central monitoring server. This interface is web based. In this interface it is possible to define thresholds, set alarms, detect patterns and more. Because there are monitors available for different technologies, in this interface an overview of all data exchanges can be shown regardless of the technology that is used. An example of this interface is shown in Figure 26.

5.4.3 Monitoring data collection mechanism

Results from the monitors are sent to the central monitoring server using a mechanism of ManageEngine itself. This mechanism has auto discovery features that can detect installed monitors automatically.

5.4.4 Conclusion

The problem with this solution is that it monitors the status of web services and if they respond instead of the data that it exchanges. So although it is a nicely integrated solution that covers all our needed components, it monitors the wrong aspects of data exchanges.

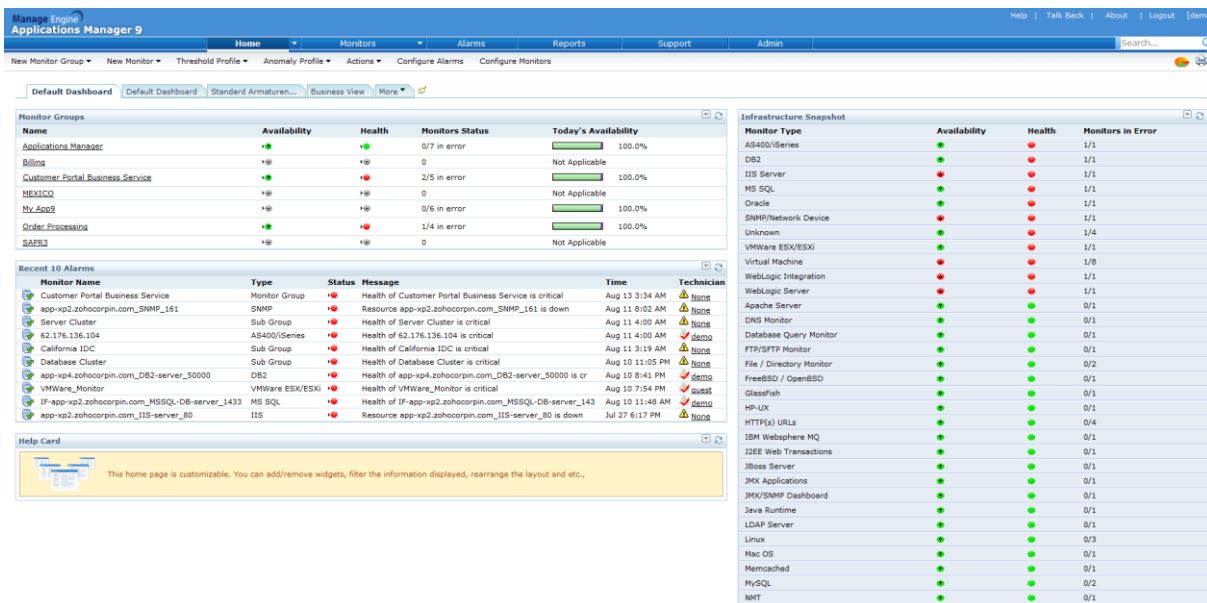


Figure 26 Web based interface of ManageEngine

5.5 Scribe

Component	Covered
Monitoring plugin	Yes
Central monitoring server	No
Monitoring data collection mechanism	Yes

Scribe [18] is a data integration and migration tool that accepts various source formats and can output data to SQL databases of various applications. It can for example insert data from XML, CSV and Text formats into the database of Microsoft CRM, Salesforce and many others. It is also capable of transforming data from one database directly into another database. In Figure 27 the position of Scribe between various applications is shown. Scribe is widely used by CRM Resultants to create data exchanges with CRM.

5.5.1 Monitoring plugin

Scribe can generate monitoring results of its own data exchanges. Each execution can be logged and within each execution all errors can be logged. By default these errors include only syntactic aspects but it is possible to add additional conditions on which errors must be generated. This adds the opportunity for monitoring semantic aspects. The monitoring features of Scribe are not able to monitor data exchanges that use other technologies than Scribe. The big advantage of using the monitoring features of Scribe is that it is able to monitor the effective results of transactions in the target system.

5.5.2 Monitoring data collection mechanism

Because Scribe is capable of using various source formats and because it can transform the data into various output formats, it is very useful to use as a monitoring data collection mechanism. It can gather and transform results from various sources and store the results in the central monitoring server. It can accept results from both real-time and batch data exchanges.

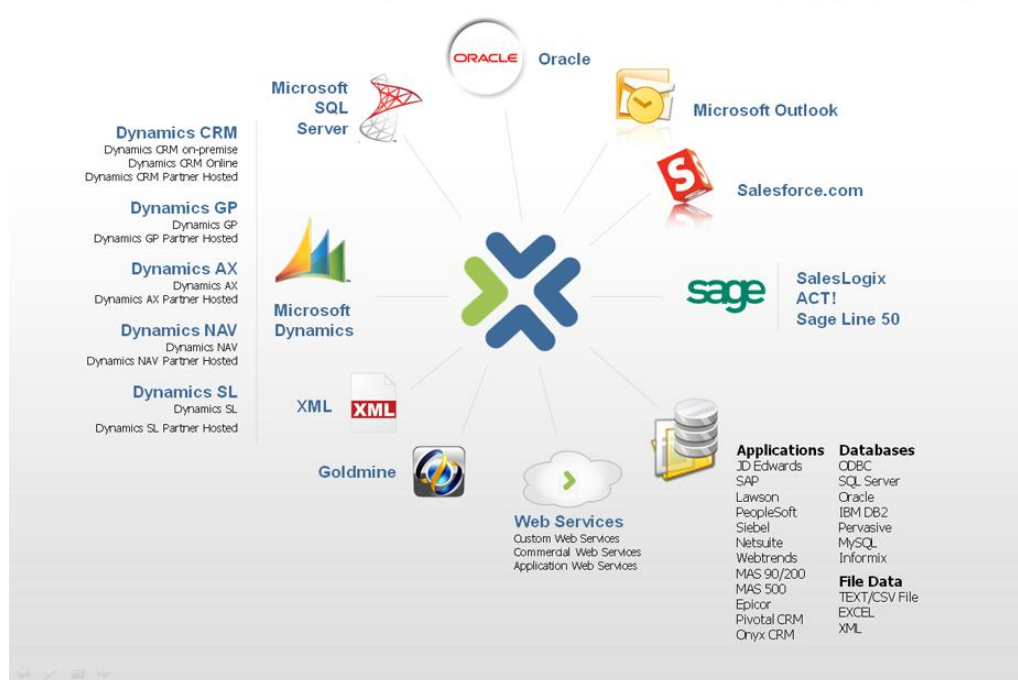


Figure 27 Scribe as a data integration tool

5.5.3 Conclusion

For Scribe data exchanges the default monitoring features of Scribe are very useful. Scribe can monitor the effective results of a data exchange in the target system and this makes it very suitable for our research.

The data integration features of Scribe can be used to collect monitoring data from various plugins. Because a lot of technologies are supported it makes the monitoring solution technology independent.

5.6 soapUI

Component	Covered
Monitoring plugin	Yes
Central monitoring server	No
Monitoring data collection mechanism	No

SoapUI [23] is an open source application which is created by Eviware. Its main purpose is functional testing of web services but it has much more features. It can simulate services, do load testing and use test cases to test quality of web services. This also includes testing data quality. It is possible to define certain assertions to which data has to match. SoapUI then runs the test and checks if the data matches the criteria and is able to report the results.

5.6.1 Monitoring plugin

Although soapUI is mainly targeted at testing web services at the design stage, it also features SOAP recording. This is a kind of monitoring in which soapUI intercepts web service messages between a client and server and can do several analyses on the message. This can be used as the monitoring plugin component of our reference architecture. One of the possibilities of this monitoring feature is to compare the actual web service requests with the WSDL specification of the web service. SoapUI is then able to show the coverage of all elements in the WSDL specification. In this SOAP monitoring it is also possible to define so called XPath constraints on values in the XML message. Based on these

constraints points can be given to the message and these points can then be used to evaluate the quality of the message. This makes both syntactic and semantic monitoring possible. The results of such analysis can be seen in the screenshot in Figure 28.

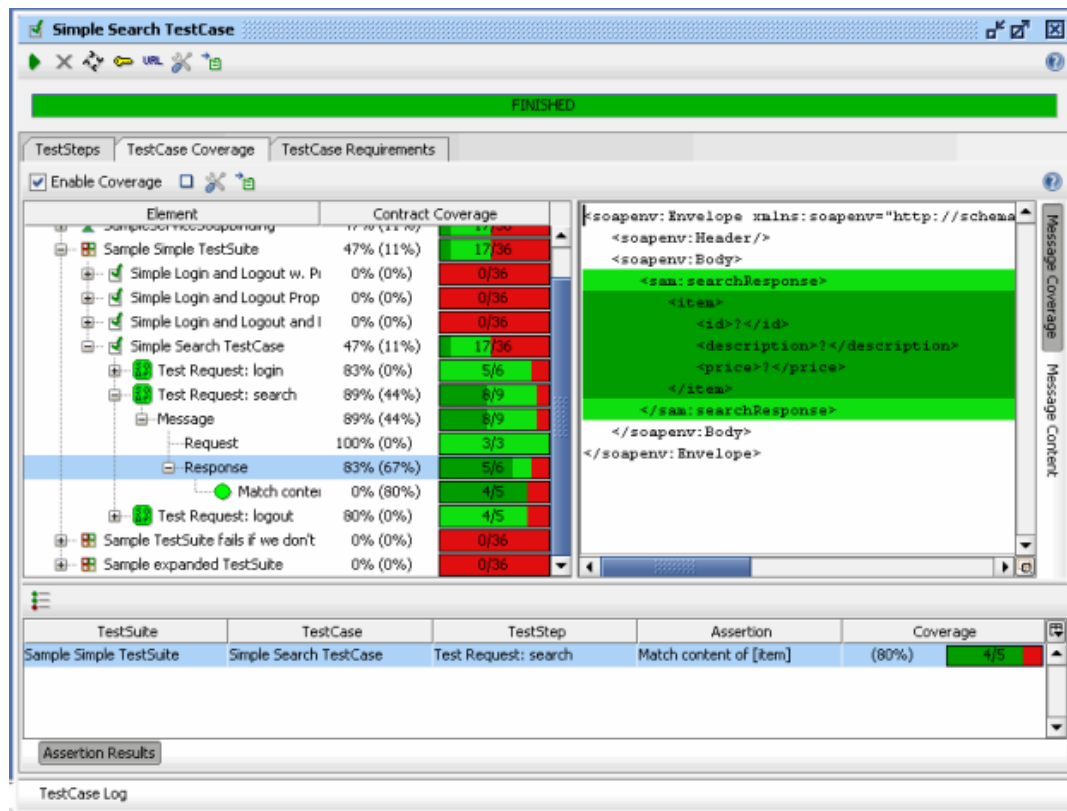


Figure 28 Analysis of values based on XPath constraints

5.6.2 Conclusion

The monitoring features of soapUI provide extensive features to report transactions of web services. The problem is however that it is not capable of monitoring the effects of a transaction in the target system. It can only monitor contents of requests without knowing if the request is correctly handled by the target system.

5.7 Log4Net

Component	Covered
Monitoring plugin	Yes
Central monitoring server	No
Monitoring data collection mechanism	No

Log4Net [24] is a logging tool that can be used in .Net frameworks. It can log actions to various output formats such as XML and SQL databases. It is already in use by CRM Resultants for debugging purposes at the design stage of custom built solutions.

5.7.1 Monitoring plugin

Log4Net can be added to web services to log the actions web services perform. This provides the opportunity to monitor both the incoming requests and the effective results a transaction has on the target system. It is capable of generating various error messages. These can be user defined or can be system errors. Log4Net supports only .Net code, so it is not technology independent.

5.7.2 Conclusion

Because Log4Net can be used to monitor the effective results of a transaction to a database, it is very suitable for our research. Although it does not support various technologies, it is very suitable in the case we use for validating our reference architecture.

5.8 Microsoft Dynamics CRM

Component	Covered
Monitoring plugin	No
Central monitoring server	Yes
Monitoring data collection mechanism	No

Microsoft Dynamics CRM [25] is in our research often the target system of a data exchange. It does not provide out of the box monitoring features, but it can be used to store data from various sources.

5.8.1 Central monitoring server

Although its name suggests that it can only be used as a CRM system, Dynamics CRM is in fact just a relational database with a built-in web interface. The system can be used for various purposes and it is possible to create custom database schemas and import data into those tables. This makes it possible to let Dynamics CRM serve as a central monitoring server into which all monitoring results are aggregated and shown. The advantages of using Dynamics CRM as a central monitoring server are that it is easy to configure, that it supports workflows and that it has reporting functionalities. The workflows can be used to define triggers that notify a user when a data exchange needs attention. The reporting functionalities can be used to provide an overview of the status of all data exchanges that are monitored.

5.8.2 Conclusion

Microsoft Dynamics CRM is suitable for functioning as a central monitoring server. It can contain all types of data and can report the results to the end user. Especially in the case we use in our research to validate the reference architecture Dynamics CRM can be useful because we already have experience with Dynamics CRM.

5.9 Conclusion on technical alternatives for monitoring

In our case study we did not find a solution that includes all of our three required components and matches all criteria of our monitoring reference architecture. There are tools available to intercept web service requests such as soapUI and WSMonitor and there are tools available that can integrate information from various monitoring agents such as ManageEngine and Hyperic. There is however no tool that can do syntactic and semantic monitoring and can aggregate these monitoring data into a central monitoring server. Most tools for creating overviews of monitoring information are aimed at server states instead of syntactic and semantic monitoring information. In addition, tools that can intercept web service requests are only focused on the contents of the request and not on the effective results of those requests in the target system. Therefore we have to build a prototype ourselves that combines some of the listed products above. An overview of the technical alternatives we assessed, their components and the requirements they meet, is shown in Table 1.

For the plugin component of our reference architecture we can make use of Log4Net and Scribe. As a collection mechanism only Scribe is suitable because the collection mechanisms of other solutions can only be used with their own monitoring plugins and central monitoring server. Scribe is capable of handling various sources and target systems so it is the only option from our case study. For storing monitoring results Dynamics CRM is the only possible option from our case study. Other central monitoring servers only work with their own collection mechanism and monitoring plugins. For the central monitoring server also other database solutions can be used, but we did not investigate other options.

	Components	Monitoring plugin	Central monitoring server	Collection mechanism	Requirements	Aggregate different technologies	Syntactic and semantic aspects	Real-time and deferrable messages	Detect patterns	Report different types of failures
Hyperic		•	•	•		•			•	•
WSMonitor		•					•			•
SoapKnox		•					•			•
ManageEngine		•	•	•		•			•	•
Scribe		•		•		•	•	•		•
soapUI		•					•			•
Log4Net		•					•			•
Dynamics CRM			•			•	•	•	•	•

Table 1 Overview of possible solutions

(• indicates that the solution includes the component or satisfies the requirement)

6 Our prototype

Because in our case study on possible solutions for monitoring we did not find a solution that has all three components and matches the criteria of our monitoring reference architecture, we build a prototype ourselves to test our reference architecture. The main purpose of this prototype is to validate our reference architecture for monitoring data exchanges. In this chapter we describe our test setup of the prototype and show the results of the prototype.

In our prototype we build a monitoring solution for monitoring data exchanges based on web services and for data exchanges based on Scribe. We use these two types of data exchanges because 90% of all data exchanges that are built by CRM Resultants are covered by these two types. In our prototype we make use of one central monitoring server and two separate solutions to acquire monitoring information from web services and from Scribe. First we describe the overview of our monitoring server and the requirements it meets, then we describe the implementation of our prototype and finally we provide the test results of our prototype.

6.1 Components and architecture

In our prototype we make use of the following solutions:

- Monitoring plugins: Log4Net and Scribe
- Monitoring data collection mechanism: Scribe
- Central monitoring server: Microsoft Dynamics CRM

The complete overview of the used components and their relations to each other are shown in Figure 29. In this picture only one Scribe data exchange is shown and only one web service data exchange is shown. In practice there are multiple data exchanges that are monitored. The components of this architecture that are used for monitoring are coloured yellow.

The results from the Log4Net logging database and the Scribe internal database are stored in the CRM database of the central monitoring server. The Scribe instance of the central monitoring server is responsible for collecting the data from the data exchanges that are monitored.

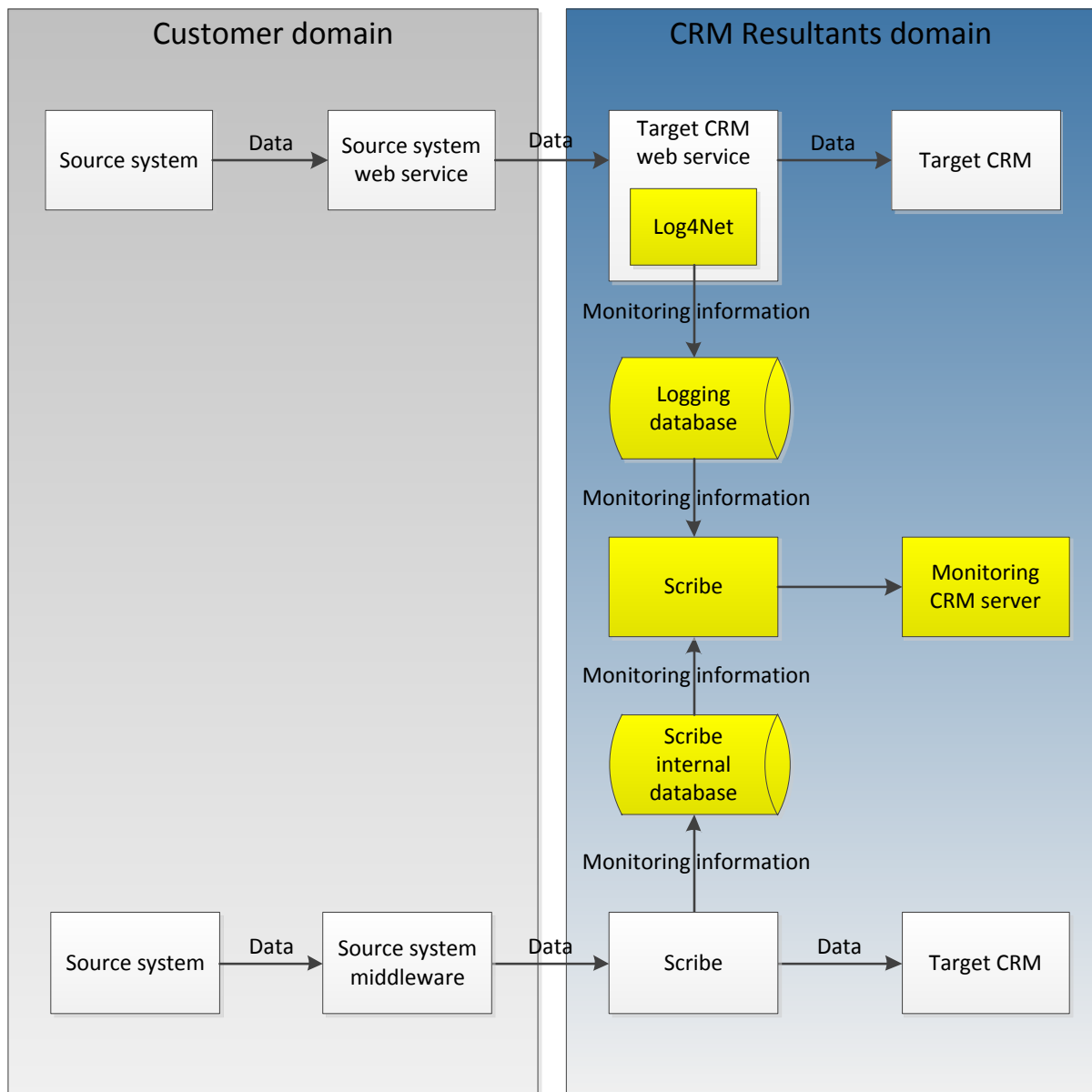


Figure 29 Overview of the architecture of the prototype

6.2 Requirements

In our reference architecture we specified a number of requirements that have to be met. We explain per requirement how our prototype meets the requirement.

6.2.1 Aggregate data from different sources which may use different technologies

In our prototype we make use of Scribe to retrieve the monitoring results from different data exchanges. Scribe can handle a lot of formats and this makes the prototype technology independent. Also difference in the structure in which data exchanges report monitoring results can be handled by Scribe. Scribe can transform the data so that all data can be stored in a uniform way in the monitoring CRM system.

6.2.2 Monitor syntactic and semantic aspects of data exchanges

Because Log4Net is included in the .Net code that is responsible for handling web service requests, all types of errors and success messages can be generated. Syntactic errors in which the .Net code encounters an error are automatically generated. Semantic errors in which data is incorrect must be configured and those error messages must be manually programmed into the .Net code.

Scribe automatically logs all transactions error to a database. In this way all syntactic errors are covered. For generating semantic errors additional checks will have to be built into the data exchange. Scribe can then log semantic errors.

In our prototype we do not build checks for semantic errors into the data exchange. We do not do this because we are monitoring data exchanges that are in production and because these checks will decrease the performance of data exchanges badly. Instead we use Scribe to generate a summary of all semantic correct transactions and semantic incorrect transactions every day. This summary is then stored in the central monitoring server.

6.2.3 Support real time and deferrable messages

All results from both real time and batch data exchanges are stored in the logging database and the Scribe internal database. The Scribe instance of the monitoring server can handle both types of monitoring results.

6.2.4 Detect patterns in data exchanges

The functionality of detecting patterns is limited in our prototype. It is however capable of generating alerts when no successful transactions are registered for a specified amount of time. Also it is possible to generate charts from the monitoring results. The pattern can then be observed by the end user.

6.2.5 Report different types of failures

The prototype is capable of reporting all types of failures as long as the monitoring plugin can log them. The prototype stores error messages as text in the monitoring server. By doing this it supports all errors that are generated by the monitoring plugins in the data exchanges.

6.3 Monitoring web service data exchanges

For monitoring data exchanges that make use of web service technology, we use the tool Log4Net. Log4Net is a tool that can be configured to log information about the execution of .Net code. It supports logging events at five different levels. The following levels are supported.

- **Debug.** This level is designated for usage while building the data exchange and contents of the message should provide internal parameters.
- **Info.** This level is designated to provide information about data that is retrieved from external sources and to confirm that responses are sent to external systems. Also messages that confirm that the transaction is completed have this level.
- **Error.** This level is used to log that an error has occurred and that the transaction is aborted.
- **Warn.** This level is used to log that something went wrong, but that the transaction can continue.
- **Fatal.** This level is used if no communication with the other system of the data exchange is possible.

Apart from the level also a message is logged. This can be a message that is automatically generated by Log4Net or it can be a message that is specified by the developer of the data exchange. In this way also successful transactions can be logged. In Figure 30 an example is shown of how data that is generated by Log4Net looks like.

Id	Date	Level	Application	Logger	Context	Message
961	2010-10-26 16:09:...	INFO	HU-website-PostcodeCheck	PostcodeService	SearchPostcode	Incoming request: Postcode + Nr: 3
962	2010-10-26 16:11:...	INFO	HU-BrochureaanvragenSiteCoreSvc-Productie	SiteCore.BusinessComponents.SiteCoreService	CreatePreStudy	<?xml version="1.0" encoding="utf-8"
963	2010-10-26 16:09:...	INFO	HU-website-PostcodeCheck	PostcodeService	SearchPostcode	Sending query to webservice.nl: 3825
964	2010-10-26 16:09:...	INFO	HU-website-PostcodeCheck	PostcodeService	SearchPostcode	Postcode query result: Postcode: 382
965	2010-10-26 16:23:...	INFO	HU-website-PostcodeCheck	ASP.global_asax	(null)	Application ends, going to persist Req
966	2010-10-26 16:23:...	ERROR	HU-website-PostcodeCheck	ASP.global_asax	(null)	RequestCounter cannot be persisted,
967	2010-10-26 16:11:...	INFO	HU-BrochureaanvragenSiteCoreSvc-Productie	Framework.Core.CrmProcessor	CreatePreStudy Check...	Query returned [0] records.
968	2010-10-26 16:11:...	INFO	HU-BrochureaanvragenSiteCoreSvc-Productie	Framework.Core.CrmProcessor	CreatePreStudy Create...	A new cmrow_prestudy record was cr
969	2010-10-26 16:27:...	INFO	HU-BrochureaanvragenSiteCoreSvc-Productie	SiteCore.BusinessComponents.SiteCoreService	CreateAspirant	<?xml version="1.0" encoding="utf-8"

Figure 30 The Log4Net table in SQL Server Management studio

In our prototype we monitor the web services of the Hogeschool Utrecht that we took earlier as a case to gather possible errors in section 3.1. Because Log4Net generates way more info messages than the number of successful transactions, we applied a filter on the messages that we use for monitoring. We describe this filter later where we go into detail on the collection of monitoring results.

6.4 Monitoring Scribe data exchanges

Scribe is middleware that has built-in logging features. Scribe can log execution job results to a database and send reports in XML format by e-mail. For our prototype we make use of database logging. Scribe logs its execution results to two tables: Executionlog and transactionerrors.

In the executionlog table results are stored each time a job is executed. In this table a summary of the job is stored including the number of successful transactions and the number of errors. For real time data exchanges each transaction is a separate job so the executionlog contains a row for each individual transaction. For batch data exchanges a job can consist of multiple transactions so the executionlog contains a summary of the job. So for batch data exchanges there can be multiple successes and errors per row in the table, whereas for real time data exchanges the maximum number of successful transactions and errors is one. An example of the executionlog table is shown in Figure 31. In this screenshot not all columns are shown.

	SERVERNAME	COLLABORATION	STARTTIME	SOURCEROWS	SOURCEROWSREJECTED	FATALERRORCODE	FATALERRORMESSAGE
922	HU	Icares	2010-10-25 18:05:14.000	1	0	0	NULL
923	HU	Icares	2010-10-07 12:34:27.000	1	0	0	NULL
924	HU	Icares	2010-09-21 19:22:25.000	0	0	306	Error opening data source XMLAdapte
925	HU	Icares	2010-09-27 22:10:57.000	1	0	0	NULL
926	HU	Osis	2010-04-03 00:01:15.000	22	0	0	NULL
927	HU	Icares	2010-10-04 12:13:55.000	1	1	0	NULL
928	HU	Icares	2010-10-23 14:45:58.000	1	0	0	NULL
929	HU	Icares	2010-09-21 19:21:40.000	0	0	306	Error opening data source XMLAdapte
930	HU	Osis	2010-10-02 07:00:32.000	103	0	0	NULL

Figure 31 The Scribe executionlog table in SQL Server Management Studio

In the transactionerrors table the errors of both real time and batch data exchanges are stored. This table has a row for each input record that resulted in an error. Successful transactions are not stored in this table. So the job that is logged in the executionlog of a real-time data exchange can have zero or one rows in the transactionerrors table and the job of a batch data exchange can have zero or more rows in the transactionerrors table. An example of the transactionerrors table is shown in Figure 32. Not all columns of this table are shown.

	SOURCEROWNUMBER	STEPNUMBER	TARGETTABLE	OPERATION	ERRORCODE	ERRORMESSAGE	REJECTEDROWNUMBER
1	1	5	campaignresponse	I	384	CreateEntity failed: Campaign With Id = c75a587f...	1
2	9	1	lead	S	697	User error: Meerdere aspiranten gevonden!	1
3	15	9	cmrow_interest	S	697	User error: Meer dan 1 belangstelling gevonden v...	2
4	49	1	lead	S	697	User error: Meerdere aspiranten gevonden!	3
5	55	1	lead	S	697	User error: Meerdere aspiranten gevonden!	4
6	69	9	cmrow_interest	S	697	User error: Meer dan 1 belangstelling gevonden v...	5
7	74	9	cmrow_interest	S	697	User error: Meer dan 1 belangstelling gevonden v...	6
8	79	1	lead	S	697	User error: Meerdere aspiranten gevonden!	7
9	84	9	cmrow_interest	S	697	User error: Meer dan 1 belangstelling gevonden v...	8

Figure 32 The Scribe transactionerrors table in SQL Server Management Studio

6.5 Monitoring semantics

The data exchanges that we monitor in our prototype are in production and cannot be changed. We could create some additional data exchanges for testing purposes, but when checking transactions for semantic correctness in the data exchange itself, it would impact the performance of a data exchange very badly. Especially for Scribe data exchanges this has a big impact.

Because of these reasons we generate the semantic results of transactions once a day. Each day a Scribe job, at the server that is monitored, queries the lead records that are modified in the CRM environment in the last 24 hours and checks their correctness. In our prototype we implemented this in a limited way. We check if a postal code of an address which is located in The Netherlands consists of four numbers followed by a space and two capital letters. If a record does not match these criteria we can guarantee that it is incorrect. If a record matches the criteria we can however not guarantee that it is correct because the postal code might not exist or might not match with the city that is entered. In our case study we say that a postal code is correct if it is correctly formatted.

The query we use to generate the daily summary is shown in Textbox 1. In this query the GUID of the countrypid represents The Netherlands. The result of this query is a number of correct postal codes and a number of incorrect postal codes.

```
SELECT
    (SELECT COUNT (*)
     FROM Lead
     WHERE modifiedon > GETDATE()-1
     AND address1_postalcode LIKE '[0-9][0-9][0-9][0-9] [A-Z][A-Z]'
     AND (crmrdr_countryid = '3082FD92-0D28-DD11-B0A3-005056AE5887' OR
          crmrdr_countryid IS NULL)
    ) AS [CorrectPostalCodes],
    (SELECT COUNT (*)
     FROM Lead
     WHERE modifiedon > GETDATE()-1
     AND address1_postalcode NOT LIKE '[0-9][0-9][0-9][0-9] [A-Z][A-Z]'
     AND (crmrdr_countryid = '3082FD92-0D28-DD11-B0A3-005056AE5887' OR
          crmrdr_countryid IS NULL)) AS [IncorrectPostalCodes]
```

Textbox 1 SQL query used to generate daily semantic summary

The results of this query are stored in the Scribe internal database and a monitoring date is appended. This information is retrieved by the Scribe instance of the central monitoring server.

6.6 Central monitoring server

For our central monitoring server we use Microsoft Dynamics CRM. Dynamics CRM fulfils all requirements of the reference architecture. It has a web based user interface, search functionality, functionality to view results at different levels and functionality to generate alerts on specified conditions.

The advantages of Dynamics CRM are that it already has a user interface which is data driven and it is easy to set up. Also it is capable of defining workflows. These workflows can be used to notify administrators by sending e-mails or creating tasks if there is a potential problem. Eventually this can even be extended with a ticketing system for keeping track of fixing problems. In Figure 33 the data model of the CRM database that is used to store monitoring results is schematically shown. This data model is an extended version of the reference data model from our reference architecture.

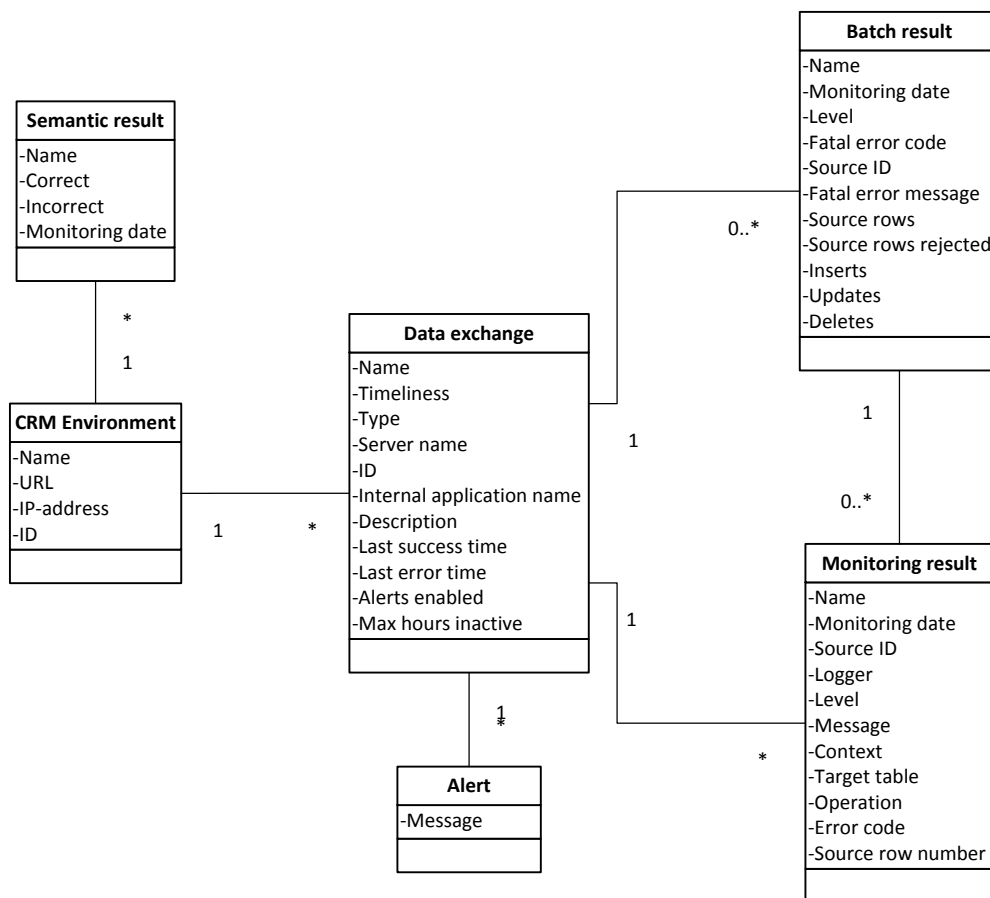


Figure 33 Data model of the central monitoring server

As can be seen in the data model of the central monitoring server a CRM Environment can have multiple data exchanges. If a data exchange is real-time it cannot have batch results. For each transaction the results are stored in the Monitoring result. In the monitoring result the level indicates whether the transaction was successful or resulted in an error.

Results of batch data exchanges are stored in a different way from results of real-time data exchanges. For batch data exchanges it is possible to store a summary of an executed job instead of creating a separate monitoring result record for each transaction. Therefore the results of batch data exchanges are stored in a batch result record. Only the errors of the batch job are stored in monitoring result records. This minimizes the number of created records and improves the ability to create an overview of the results of batch data exchanges.

When monitoring multiple data exchanges a lot of monitoring results are stored. This can make it difficult for a user to quickly signal potential problems. Therefore each data exchange can have multiple alerts. These alerts can be generated based on certain conditions. In our prototype we included functionality that generates alerts if no successful transaction was executed for a specified time. These alerts are generated by Scribe. On the data exchange the date and time of the last successful transaction and the last erroneous transaction are stored. Also the maximum number of hours that a data exchange is allowed to have no successful transaction can be specified. Based on this information and the field which specifies whether alerts are enabled or not, a scheduled job checks these conditions and generates alerts. These alerts can then signal the user and the user has the ability to dig further into the potential problem by viewing individual results of a data exchange.

6.7 User experience of the central monitoring server

In this section we show the user experience of the central monitoring server. We do not cover every detail, but show the most important aspects. Other actions can be performed in similar ways. First we present the general interface and afterwards we provide some examples of how actions can be performed.

6.7.1 General interface

The interface of the monitoring server is completely web based. This means that only a web browser is needed to work with the system. The main screen of the monitoring module of the system is shown in Figure 34. This screen has several areas. On the left side there is a list of the different types of records in the system. By clicking on them the list of records is shown. In the example the list of data exchanges in the system is shown. In this view the data exchanges are ordered by the time of the last successful transaction. So the lower the data exchange is listed, the longer ago a successful transaction was executed. Because inactive data exchanges go down in the list automatically, it is easy to recognize them.

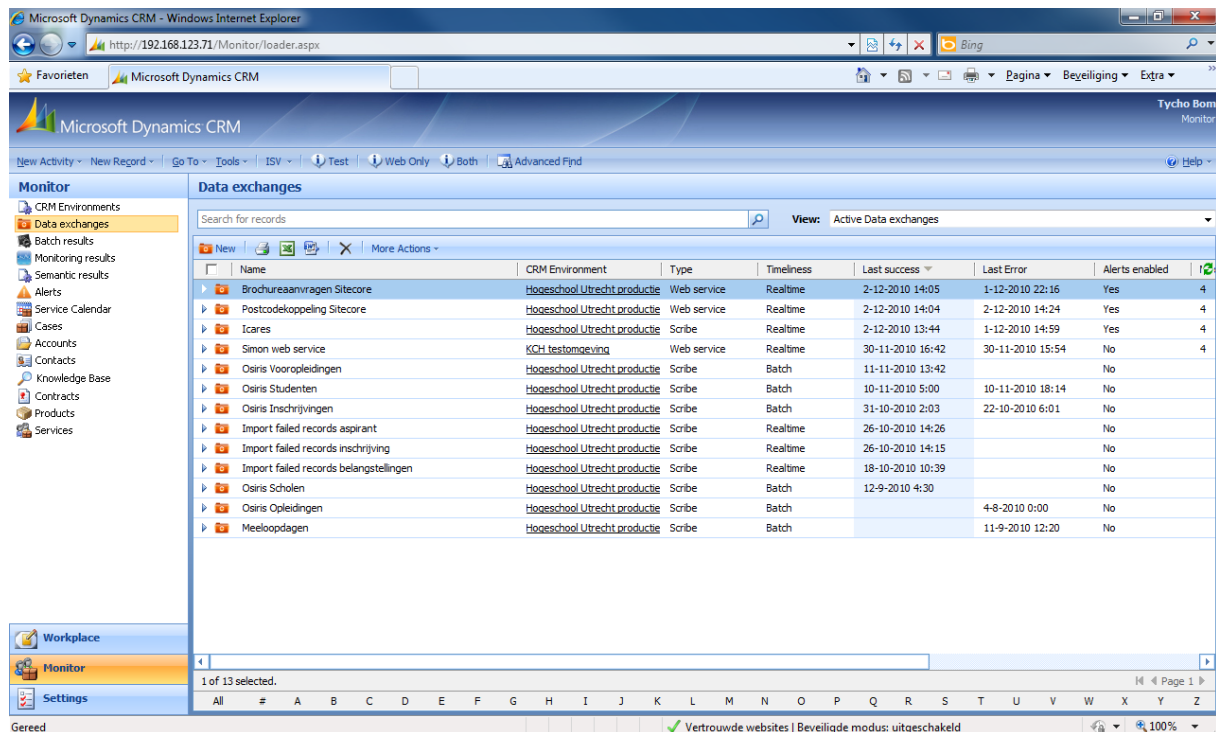


Figure 34 General view of the monitoring user interface

In the interface it is also possible to view overview reports of data in the system. By going to the Workplace module and clicking on the DashBoard button, various graphs are available concerning

monitoring information. An example of these graphs is shown in Figure 35. By viewing these graphs problems can easily be noticed and trends can be viewed.

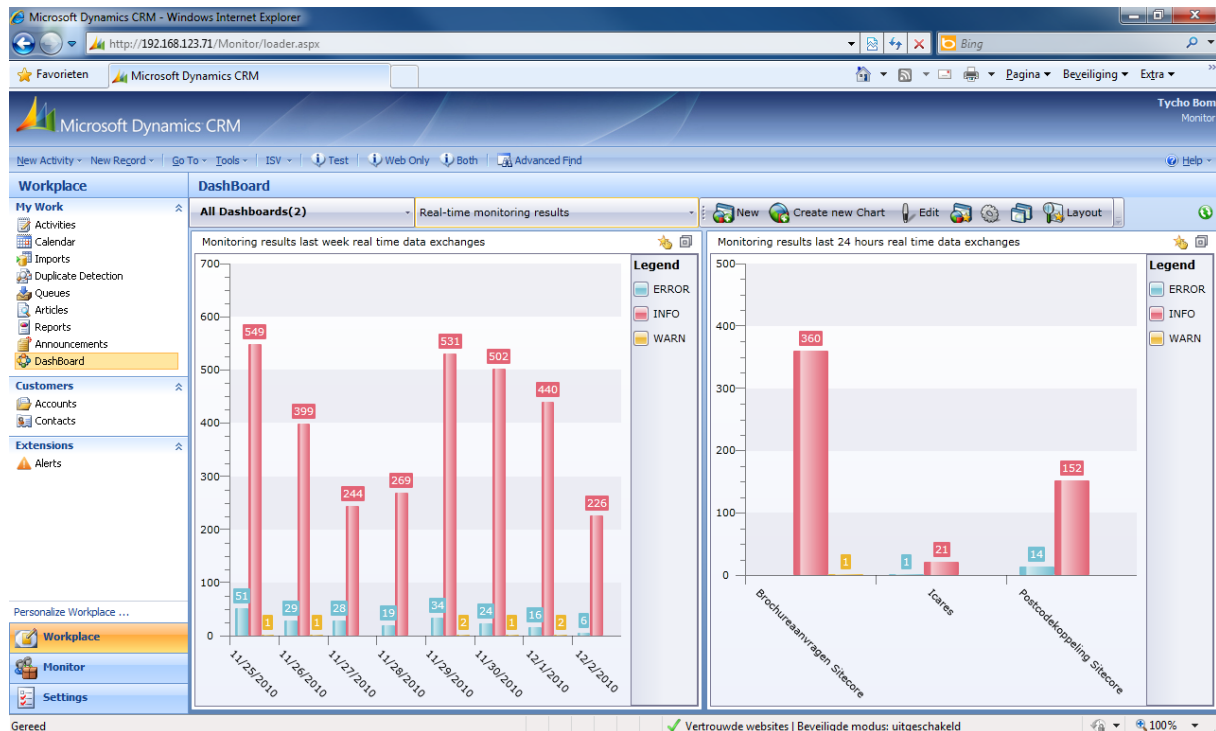


Figure 35 Two graphs of monitoring results of real-time data exchanges

6.7.2 Following up alerts

To view alerts that are generated by the system, the view of alerts can be opened. This is shown in Figure 36.

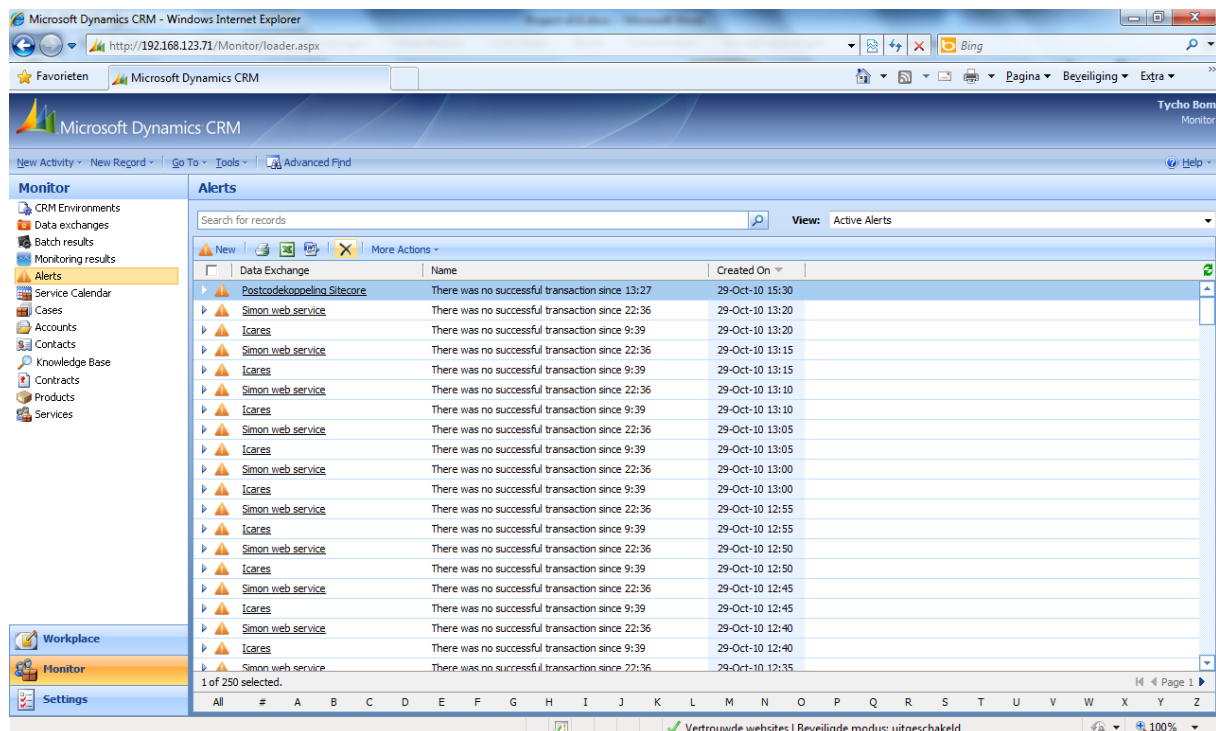


Figure 36 Main screen of the monitoring server

In the list of alerts the data exchange to which the alert is applicable and the message of the alert are shown. Alerts can be opened by clicking on them. In our prototype an alert looks like depicted in Figure 37. In our prototype we created a very basic implementation alerts. In future versions this can be extended with data to store whether the alert has been resolved etcetera.

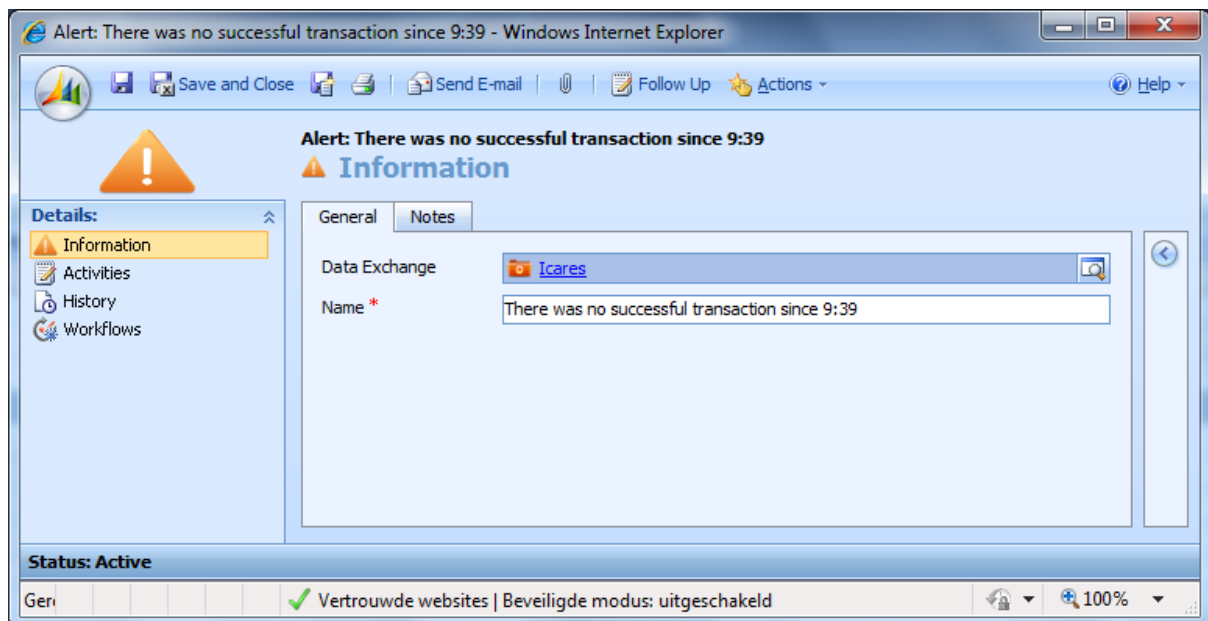


Figure 37 Alert record

To check whether there is really a problem with the data exchange, the data exchange can be opened by clicking on it. The information pane of the data exchange is shown in figure Figure 38. On this screen the general information of a data exchange is shown and in the status section information regarding its latest success and latest error are shown. Also alerting for this data exchange can be configured here.

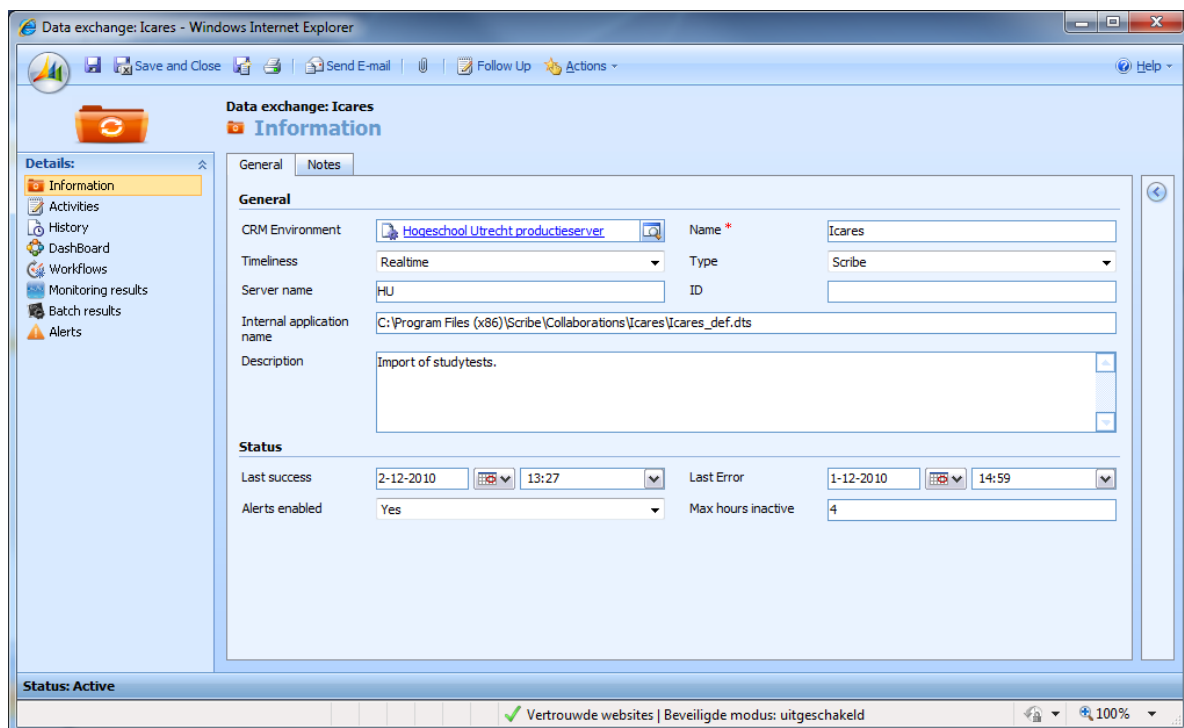


Figure 38 The information pane of a data exchange record

To show the monitoring results of this data exchange, at the left side of the screen there is a menu option for monitoring results. After clicking on it, the related monitoring results of this data exchange are shown, as can be seen in Figure 39.

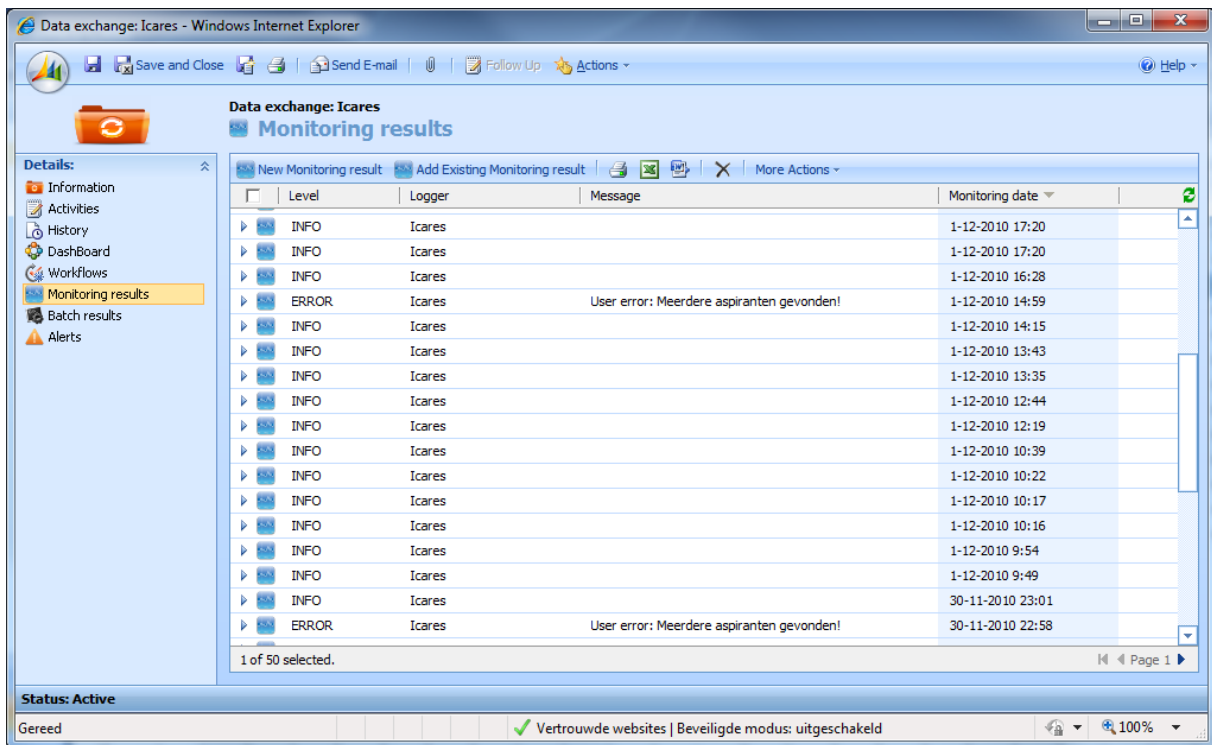


Figure 39 Related monitoring results of a data exchange

The information of related monitoring results can also be viewed in a graph by clicking on the Dashboard button. This is shown in Figure 40.

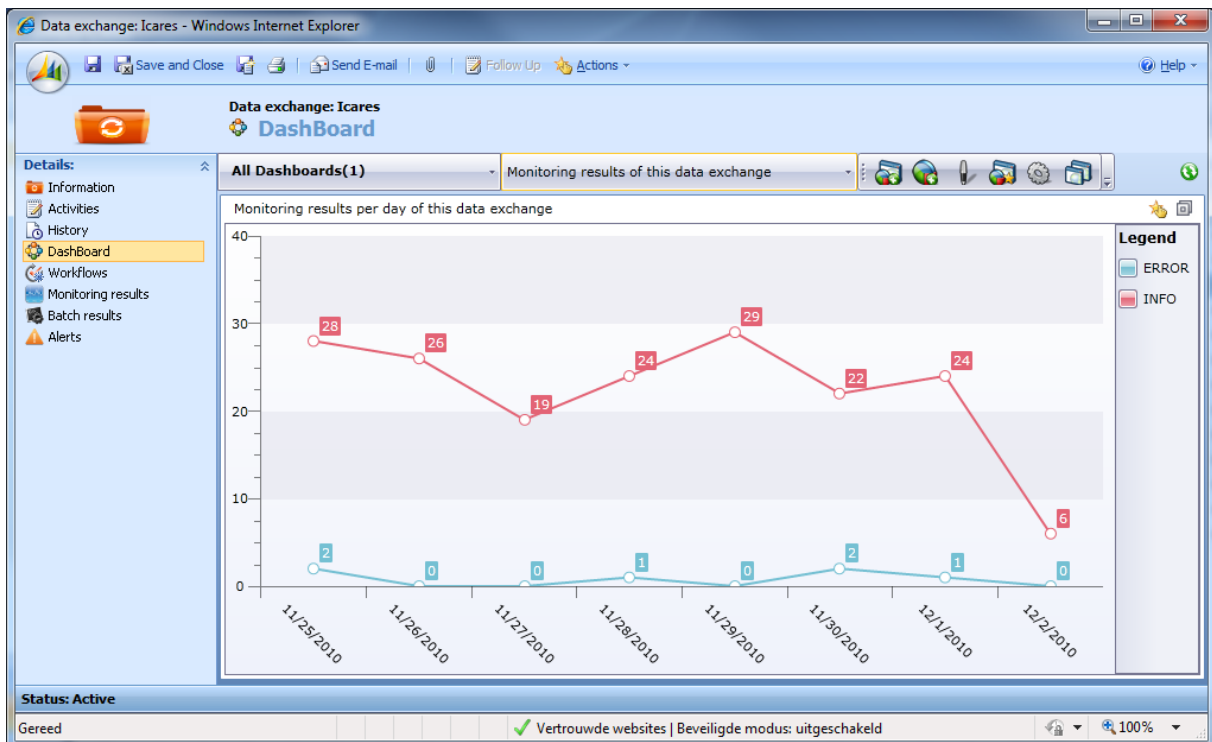


Figure 40 Dashboard of the monitoring results of this data exchange

From the screen in Figure 39 it is possible to open an individual monitoring result. An example of a monitoring result is shown in Figure 41. On the monitoring result information regarding the error is shown. Depending on the type of error and the source of the monitoring results, the displayed fields can contain different data. Based on the number of errors that are linked to the data exchange and the information in the monitoring results, the user can determine if action is required.

The screenshot shows a web application window titled "Monitoring result: ERROR: lead". The window has a menu bar with options like "Save and Close", "Send E-mail", "Follow Up", and "Actions". On the left, there is a sidebar with "Details" and "Information" sections. The main content area is divided into "General" and "Notes" tabs. The "General" tab contains the following fields:

- Name: ERROR: lead
- Data Exchange: Icares
- Monitoring date: 29-Oct-10 09:53
- Logger: Icares
- Message: User error: Meerdere aspiranten gevonden!
- Context: ADP
- Scribe context: Target table: lead, Operation: Seek, Error code: 697, Source row number: 1
- Batch result: (empty)
- Source id: 0e9fab1d-7e9b-4837-91b6-5ca0aa0be2f4
- Level: ERROR

At the bottom, the status is "Active". The browser's address bar shows "Gereed" and the status bar shows "Vertrouwde websites | Beveiligde modus: uitgeschakeld".

Figure 41 Monitoring result

6.7.3 Viewing batch results

To improve the overview of monitoring results of batch data exchanges, we store summaries of batch executions in batch result records. These records are linked to batch data exchanges and possible errors of a batch result are stored in the monitoring results, similar to real-time data exchanges. Successful transactions are not stored in separate monitoring result records.

To view batch results, users can either open a batch result directly, or select a data exchange first and afterwards view its related batch results. In Figure 42 the list of data exchanges that are monitored is shown. A data exchange can be opened by clicking on it.

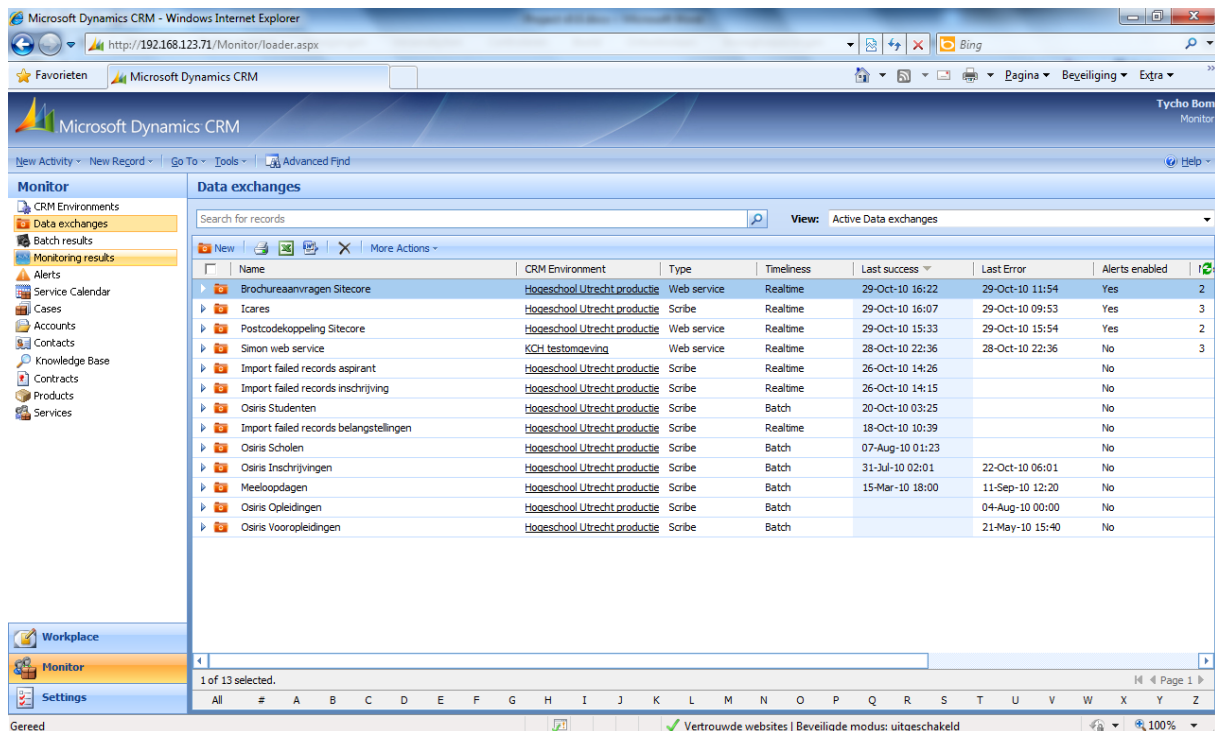


Figure 42 List of monitored data exchanges

The record of a batch data exchange is similar to the record of a real-time data exchange. The main difference is that the entity batch result is used. At the left side of the record the user can click on the batch results button to view the related batch results. This is shown in Figure 43.

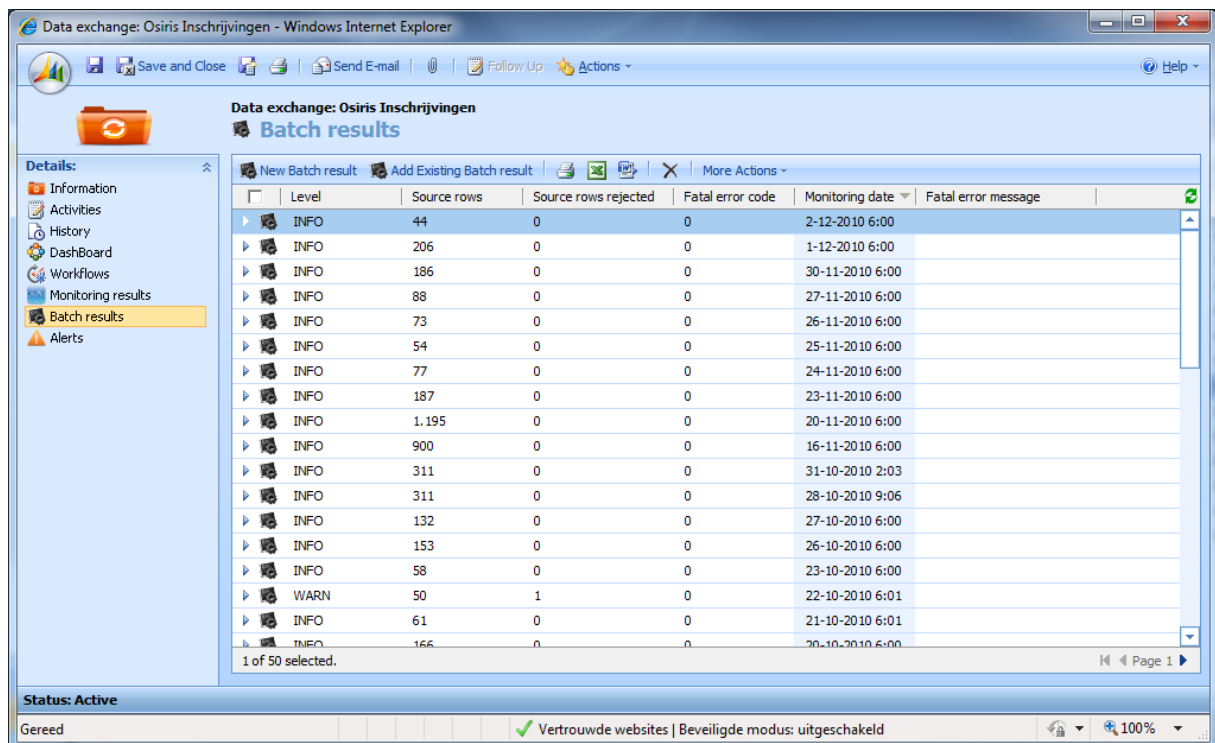


Figure 43 Related batch results of a data exchange

To easily view trends in batch results of a batch data exchange, there is a dashboard available. This can be opened by clicking on the Dashboard button. This is shown in Figure 44.

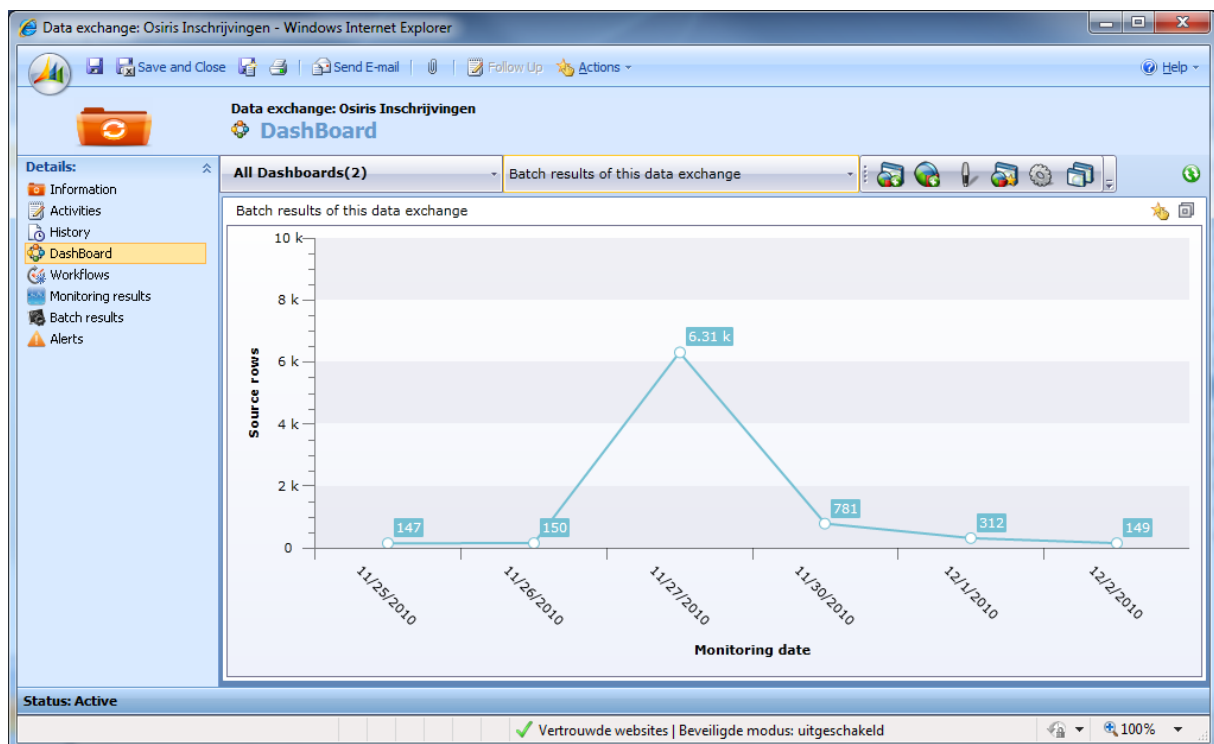


Figure 44 Graph of source rows in batch results of this data exchange

From the screen in Figure 43 batch results can be opened to view its details. The general view of a batch result is shown in Figure 45.

Batch result: Batch result 40470.2505555556																									
Information																									
<table border="0"> <tr> <td>Name *</td> <td colspan="3">Batch result 40470.2505555556</td> </tr> <tr> <td>Data Exchange *</td> <td>Osiris Inschrijvingen</td> <td>Monitoring date</td> <td>19-Oct-10 06:00</td> </tr> <tr> <td>Level</td> <td>WARN</td> <td>Fatal error code</td> <td>0</td> </tr> <tr> <td>Source ID</td> <td colspan="3">ad79cff1-5dfd-4478-8507-b273946bf775</td> </tr> <tr> <td>Source rows</td> <td>1,158</td> <td>Source rows rejected</td> <td>23</td> </tr> <tr> <td>Fatal error message</td> <td colspan="3"></td> </tr> </table>		Name *	Batch result 40470.2505555556			Data Exchange *	Osiris Inschrijvingen	Monitoring date	19-Oct-10 06:00	Level	WARN	Fatal error code	0	Source ID	ad79cff1-5dfd-4478-8507-b273946bf775			Source rows	1,158	Source rows rejected	23	Fatal error message			
Name *	Batch result 40470.2505555556																								
Data Exchange *	Osiris Inschrijvingen	Monitoring date	19-Oct-10 06:00																						
Level	WARN	Fatal error code	0																						
Source ID	ad79cff1-5dfd-4478-8507-b273946bf775																								
Source rows	1,158	Source rows rejected	23																						
Fatal error message																									
Operations																									
Inserts	98	Updates	1,037																						
Deletes	0																								

Figure 45 General information of a batch result

On the batch result record the total number of rows or records that are processed is shown in 'Source rows'. The number of errors is shown in 'Source rows rejected'. To view the errors the monitoring results of this batch result can be shown by clicking on monitoring results. This is shown in figure Figure 46.

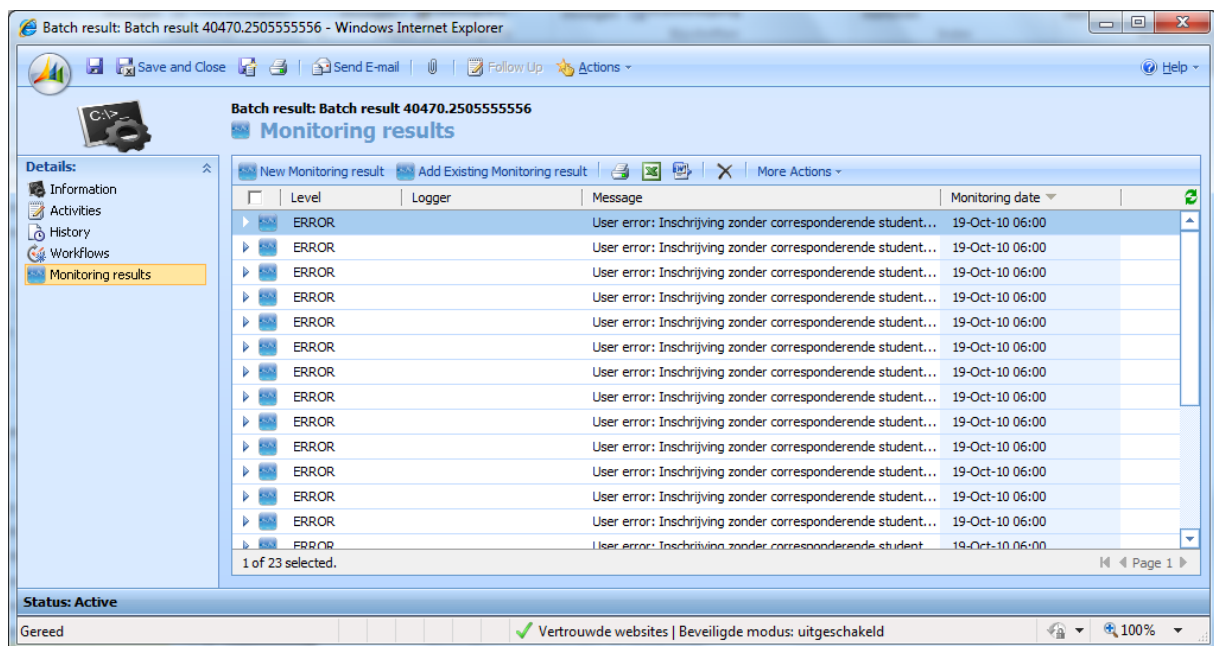


Figure 46 Errors of a batch result

The individual monitoring result is the same as the monitoring result of real-time data exchanges. Note that monitoring results are in this case related to both the batch result and the data exchange. This means that these errors can be viewed directly from the data exchange or from the batch result.

The errors of batch results can also be viewed graphically. This can be done by clicking on the DashBoard button. This is shown in Figure 47.

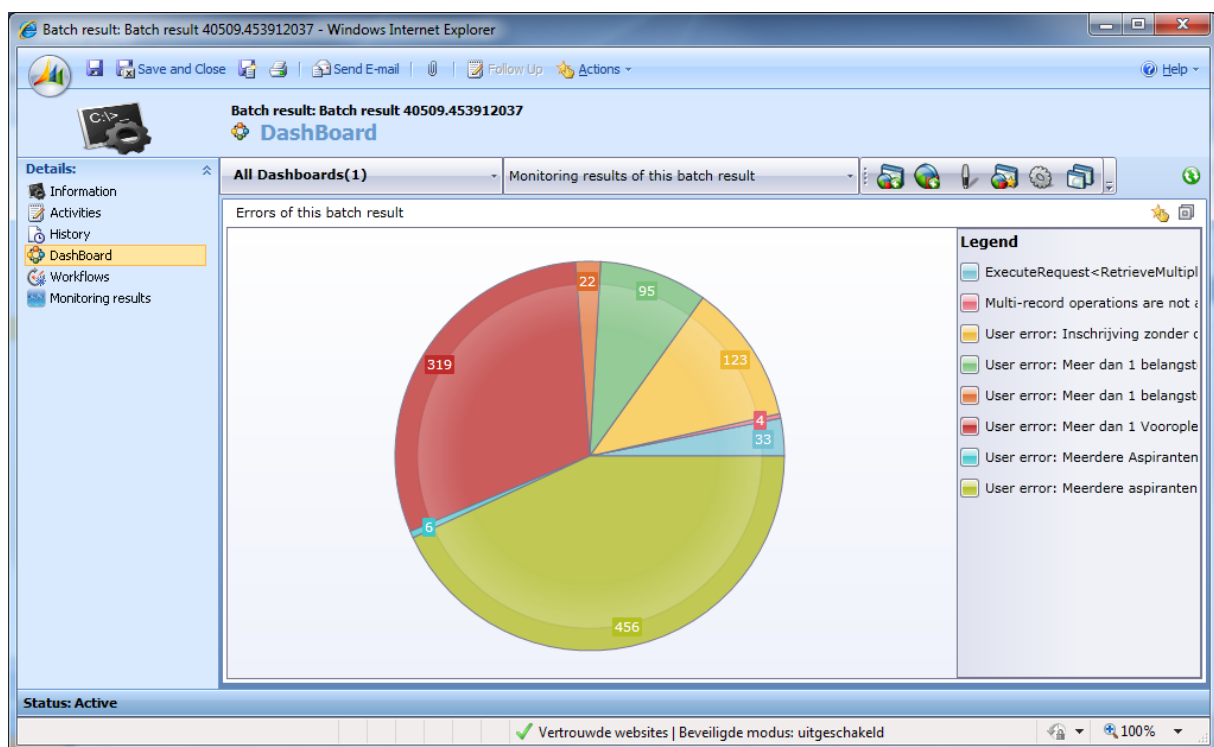
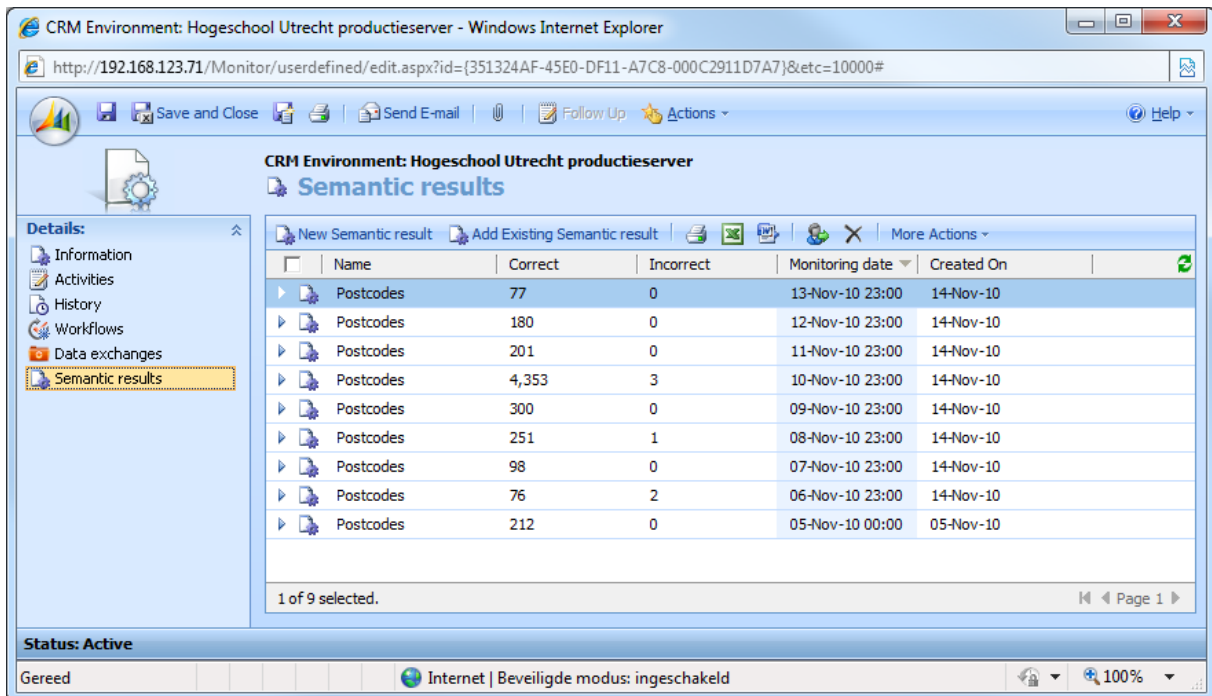


Figure 47 Graph of errors of a batch result

6.7.4 Viewing semantic results

Semantic results are linked to CRM environments instead of data exchanges. There is no information available on the semantic results per data exchange because we only generate daily summaries. If we open a CRM environment, we can see that the semantic results are linked to the environment in the menu on the left side. By clicking on it the results can be viewed. This is shown in Figure 48.

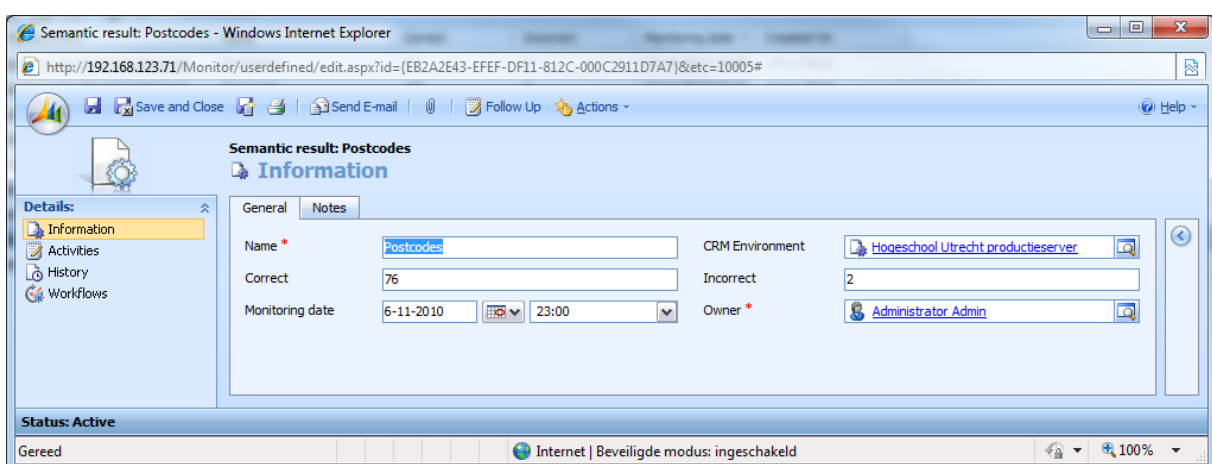


The screenshot shows a web application interface for 'CRM Environment: Hogeschool Utrecht productieserver'. The left sidebar contains a 'Details' menu with options: Information, Activities, History, Workflows, Data exchanges, and Semantic results (highlighted). The main content area is titled 'Semantic results' and contains a table with columns: Name, Correct, Incorrect, Monitoring date, and Created On. The table lists 9 records for 'Postcodes'. Below the table, it says '1 of 9 selected.' and 'Page 1'. The status bar at the bottom indicates 'Status: Active' and 'Gereed'.

Name	Correct	Incorrect	Monitoring date	Created On
Postcodes	77	0	13-Nov-10 23:00	14-Nov-10
Postcodes	180	0	12-Nov-10 23:00	14-Nov-10
Postcodes	201	0	11-Nov-10 23:00	14-Nov-10
Postcodes	4,353	3	10-Nov-10 23:00	14-Nov-10
Postcodes	300	0	09-Nov-10 23:00	14-Nov-10
Postcodes	251	1	08-Nov-10 23:00	14-Nov-10
Postcodes	98	0	07-Nov-10 23:00	14-Nov-10
Postcodes	76	2	06-Nov-10 23:00	14-Nov-10
Postcodes	212	0	05-Nov-10 00:00	05-Nov-10

Figure 48 Semantic results of a CRM environment

Individual semantic result records can be viewed by clicking on them. A semantic result record is shown in Figure 49. On this record the number of correct results and the number of incorrect results are shown. The monitoring date shows the date and time on which the result was generated. The name of the record indicates the field that is monitored. In our prototype we only monitor the postal code field of lead records.



The screenshot shows a detailed view of a semantic result record for 'Postcodes'. The left sidebar is the same as in Figure 48, but the 'Information' tab is selected. The main content area is titled 'Semantic result: Postcodes' and 'Information'. It contains a form with fields for Name, Correct, Incorrect, Monitoring date, and Owner. The values are: Name: Postcodes, Correct: 76, Incorrect: 2, Monitoring date: 6-11-2010 23:00, Owner: Administrator Admin. The status bar at the bottom indicates 'Status: Active' and 'Gereed'.

Name	Correct	Incorrect	Monitoring date	Owner
Postcodes	76	2	6-11-2010 23:00	Administrator Admin

Figure 49 Semantic result record

6.8 Collection mechanism

Scribe is used to collect monitoring information from different sources and to store that information in the central monitoring CRM database. Scribe connects to databases that store results about web service data exchanges and to databases that store information about Scribe data exchanges. Apart from the data collection function Scribe also has a function to generate alerts.

6.8.1 Collection of data from web service data exchanges

In Figure 50 the relation between involved systems in the monitoring process of web service data exchanges is shown. Log4Net logs to a database. Multiple loggers can use the same database, as long as this database is not blocked by firewalls and virtual LANs. In practice there are some blockades, so the Scribe instance of the monitoring server will have to query multiple databases but this is not shown in the figure.

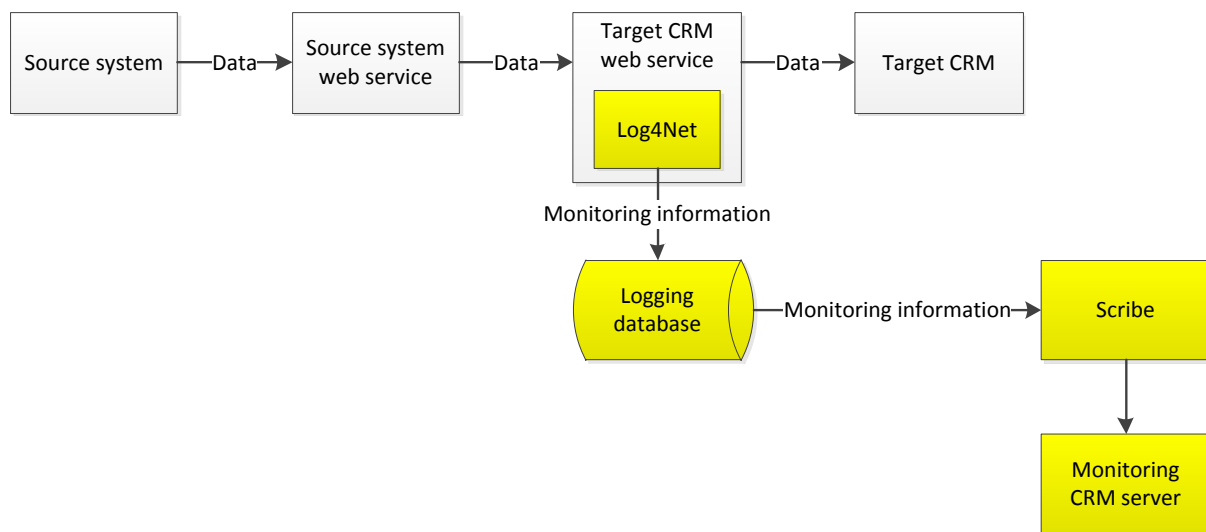


Figure 50 Overview of the architecture for monitoring web services

As stated before Log4Net classifies its logging messages with different levels. We are only interested in the levels info, warn, error and fatal. The debug level is only used at the development stage and contains too much information for our purpose. Also the messages from the info level contain too much data. Each transaction can generate multiple info messages containing information about the lookups that are performed and other local variables. We are only interested in the records that are created, so we apply a filter on the info messages. We do not apply a filter on the levels warn, error and fatal because we are especially interested in things that go wrong. The query that we use to retrieve data from the Log4Net database is shown in Textbox 2. In this query we included the LastRunDateTime. This is an internal Scribe variable that contains the date and time of the last time the job has run. In this way we only get the new results. To compensate for possible differences in system times of the monitoring server and the server of the source database, we include records from one hour earlier than the last time the job has run. Although this creates some overlap, this does not result in duplicate records in the monitoring database because we perform update/insert operations. This means that for each operation first a lookup is done to check if the record already exists. If that is the case, it is updated. If no record is found, a new record is created.

In our prototype we filtered on info messages that contain the text “Postcode query result” or “A new % record was created” where % functions as a wildcard. This is a filter specific for the data exchanges that are monitored in the prototype. For new data exchanges it is recommended to agree on a keyword that is used to generate info messages that represent a successful transaction.


```

SELECT *
FROM [Log4Net].[dbo].[Log]
WHERE (([Level] = 'INFO' AND ([Message] LIKE '%Postcode query result%')
      OR [Message] LIKE '%A new % record was created%')
      OR ([Level] = 'ERROR') OR ([Level] = 'WARN') OR ([Level] =
'FATAL'))
      AND [Application] != '(null)'
      AND [Date] > DATEADD ("hh", -01, :LastRunDateTime)
ORDER BY [Id] DESC

```

Textbox 2 SQL query to retrieve Log4Net information from HU production server

6.8.2 Collection of data from Scribe data exchanges

In Figure 51 the relation between the involved systems in the process of monitoring Scribe data exchanges is shown. In this figure a data exchange between a source system and a target CRM system is shown. The middleware of the source system is not specified, but the middleware of the target CRM system is Scribe. Scribe logs the monitoring results of the data exchange to the Scribe internal database. From this database the monitoring information is collected by the Scribe instance of the central monitoring CRM server. Each instance of Scribe needs its own database because also its settings are stored in the database. In practice this means that the Scribe instance of the central monitoring server will have to retrieve data from multiple Scribe internal databases.

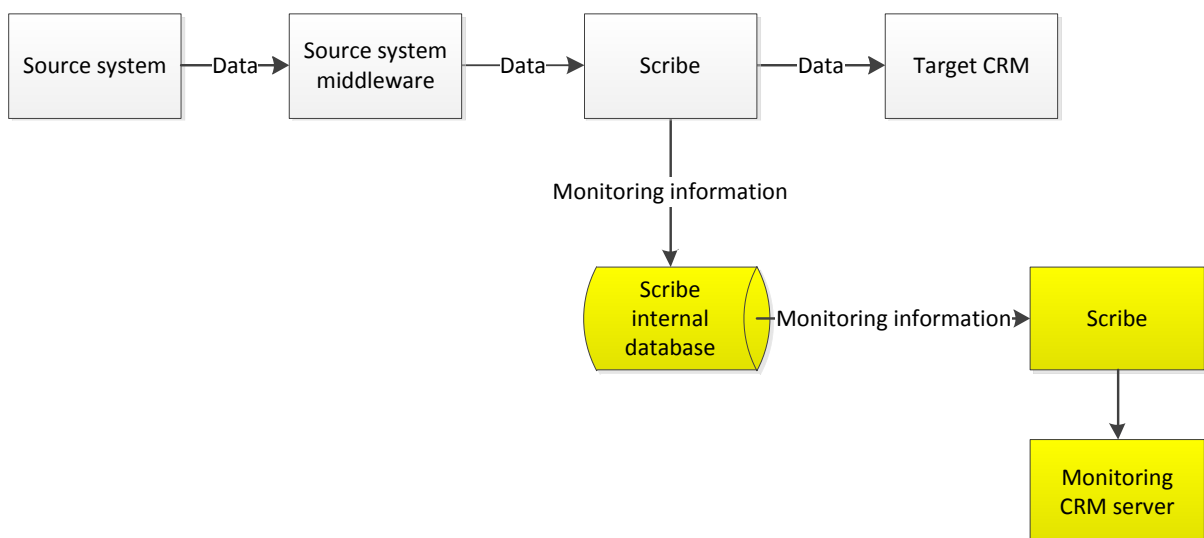


Figure 51 Overview of the architecture for monitoring Scribe data exchanges

As stated in section 6.4 Scribe stores the monitoring results in two tables. Also there is a distinction between real-time and batch data exchanges. Therefore we need multiple queries to retrieve the data.

Realtime data exchanges

To retrieve monitoring results from realtime data exchanges, we retrieve the successful transactions from the executionlog table and we retrieve the errors from the transactionerrors table. Therefore we use two queries. These queries are shown in Textbox 3 and Textbox 4. In both queries the LastRunDateTime is used to retrieve only the results that are created after the last date and time the job has run minus one hour, similar to the query for results from Log4Net.


```

SELECT [EXECID]
      , [SERVERNAME]
      , [COLLABORATION]
      , [STARTTIME]
      , [ENDMILLISECONDS]
      , [JOBSPECNAME]
      , [MESSAGE]
      , [FATALERRORCODE]
      , [FATALERRORMESSAGE]
FROM [ScribeInternal].[SCRIBE].[EXECUTIONLOG]
WHERE SOURCEROWS <=1
AND SOURCEROWSREJECTED = '0'
AND [STARTTIME] > DATEADD ("hh", -1, :LastRunDateTime)

```

Textbox 3 Query for retrieving successful transactions of real-time data exchanges

```

SELECT [ScribeInternal].[SCRIBE].[EXECUTIONLOG].[EXECID]
      , [SOURCEROWNUMBER]
      , [TARGETTABLE]
      , [OPERATION]
      , [ERRORCODE]
      , [ERRORMESSAGE]
      , [ERRORSOURCE]
      , [STARTTIME]
      , [SERVERNAME]
      , [COLLABORATION]
      , [JOBSPECNAME]
FROM [ScribeInternal].[SCRIBE].[EXECUTIONLOG] JOIN
[ScribeInternal].[SCRIBE].[TRANSACTIONERRORS]
ON [ScribeInternal].[SCRIBE].[EXECUTIONLOG].EXECID =
[ScribeInternal].[SCRIBE].[TRANSACTIONERRORS].EXECID
WHERE [COLLABORATION] = 'Icares' AND [STARTTIME] > DATEADD ("hh", -1,
:LastRunDateTime)

```

Textbox 4 Query for retrieving errors of real time data exchanges

In the query to retrieve errors of a data exchange, a join is needed to be able to identify the data exchange from which the error originates. In this example we also filter on the collaboration called Icares because that is the only real-time Scribe data exchange that we monitor in this prototype.

Batch data exchanges

For batch data exchanges we use only one query to retrieve both the summary and the errors. This results in a table where the summary is displayed multiple times in case there are multiple errors. The query used is shown in Textbox 5.

```

SELECT
[ScribeInternal].SCRIBE.executionlog.execid,
[ScribeInternal].SCRIBE.executionlog.SERVERNAME,
[ScribeInternal].SCRIBE.executionlog.COLLABORATION,
[ScribeInternal].SCRIBE.executionlog.STARTTIME,
[ScribeInternal].SCRIBE.executionlog.JOBSPECNAME,
[ScribeInternal].SCRIBE.executionlog.SOURCENAME,
[ScribeInternal].SCRIBE.executionlog.SOURCEROWS,
[ScribeInternal].SCRIBE.executionlog.FATALERRORCODE,
[ScribeInternal].SCRIBE.executionlog.FATALERRORMESSAGE,
[ScribeInternal].SCRIBE.executionlog.SOURCEROWSREJECTED,
[ScribeInternal].SCRIBE.executionlog.INSERTS,
[ScribeInternal].SCRIBE.executionlog.UPDATES,
[ScribeInternal].SCRIBE.executionlog.DELETES,
[ScribeInternal].SCRIBE.TRANSACTIONERRORS.SOURCEROWNUMBER,
[ScribeInternal].SCRIBE.TRANSACTIONERRORS.TARGETTABLE,
[ScribeInternal].SCRIBE.TRANSACTIONERRORS.OPERATION,
[ScribeInternal].SCRIBE.TRANSACTIONERRORS.ERRORCODE,
[ScribeInternal].SCRIBE.TRANSACTIONERRORS.ERRORMESSAGE
FROM [ScribeInternal].[SCRIBE].[EXECUTIONLOG]
LEFT JOIN [ScribeInternal].[SCRIBE].[TRANSACTIONERRORS]
ON ScribeInternal.SCRIBE.EXECUTIONLOG.EXECID =
ScribeInternal.SCRIBE.TRANSACTIONERRORS.EXECID
WHERE SOURCEROWS > '1' AND [STARTTIME] > DATEADD ("hh", -6,
:LastRunDateTime)
ORDER BY SOURCEEXEETIME DESC

```

Textbox 5 Query to retrieve results from Scribe batch data exchanges

We created logic that first creates the summary in CRM and afterwards appends the errors to the summary. The logic first checks if a summary exists and depending on that result it creates a summary or adds the error to the existing summary. The steps that Scribe executes to insert the batch summaries and the individual errors are schematically shown in Figure 52. In this figure the green lines depict the number of results that are found in the seek step.

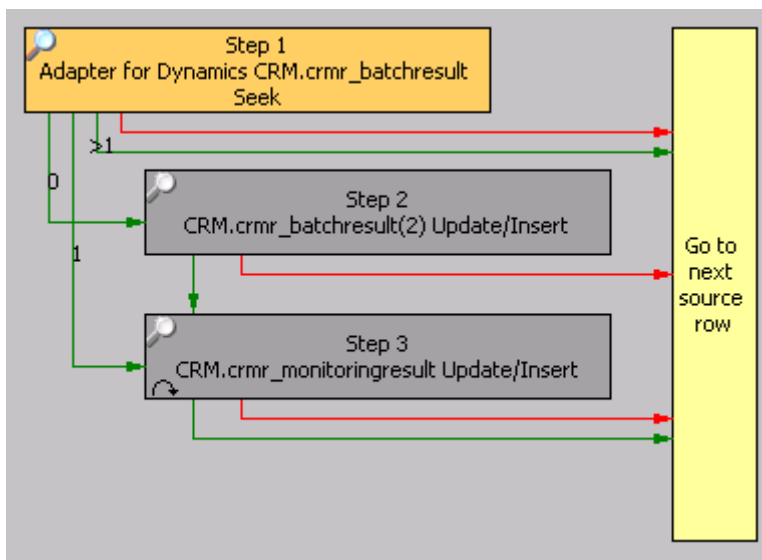


Figure 52 Steps executed to insert batch results into the CRM database

6.8.3 Collection of semantic results

The summaries of semantic results are stored once a day in the Scribe internal database of the server that is monitored. This information is collected by the Scribe instance of the monitoring server. This situation is shown in Figure 53.

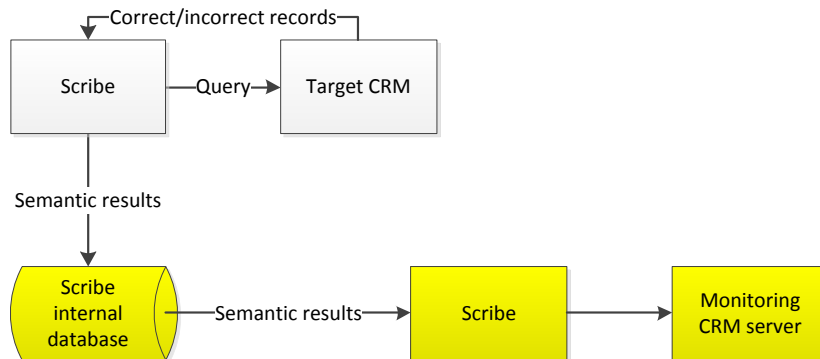


Figure 53 Collection of semantic results

The query used to retrieve the semantic results is shown in Textbox 6. Again in this query the date and time the query has run are used to collect only the new results.

```
SELECT (*)
FROM [ScribeInternal].[SCRIBE].[MONITOR_SEMANTICRESULTS]
WHERE [MONITORINGDATE] > DATEADD ("hh", -1, :LastRunDateTime)
```

Textbox 6 SQL query to retrieve semantic results

6.9 Summary of how the prototype meets the requirements

In the sections before we explained each part of our prototype in detail. In this section we provide an overview of which requirements of the reference architecture are met and how the components are implemented. This summary is shown in Table 2.

Component	
Monitoring plugin	Log4Net and Scribe
Central monitoring server	Microsoft Dynamics CRM
Collection mechanism	Scribe
Requirements	
Aggregate different technologies	Yes
Syntactic and semantic aspects	Yes
Real-time and deferrable messages	Yes
Detect patterns	Yes
Report different types of failures	Yes
Contents of monitoring information	
Originating server	Yes
Originating data exchange	Yes
Date and time	Yes
Error or success	Yes
Error message	Yes
Context	Yes
Semantic correctness	Daily summary
Events that are monitored	
Successful transactions	Yes
Error in a transaction	Yes
Fatal error in the middleware	Yes

Table 2 Components and requirements of the prototype

As can be seen in the table all requirements are met. We already explained these in section 6.2.

If we look at the contents of monitoring information, all information we specified in the reference architecture is available in the prototype. There is however a difference between the implementation of semantic correctness in the reference architecture and our implementation. In the reference architecture semantic correctness is monitored as part of the execution of a data exchange whereas we generate daily summaries in our prototype.

In our reference architecture we specify a minimum of three types of events that must be monitored. All three of these events are supported by our prototype.

If we look at the complete comparison between the reference architecture and our prototype, we can see that the prototype meets all requirements that are demanded by the reference architecture. Only the semantic monitoring of data exchanges is limited in our prototype.

6.10 Monitoring the monitor

The monitor we created is built to monitor data exchanges, but by introducing this monitor we created additional data exchanges. Of course also in these data exchanges errors can occur. Formatting errors are unlikely to happen because the source databases have static schemes which are compatible with the target system, but other types of errors can still occur. To monitor if the monitor is still working, there are three mechanisms implemented.

- If fatal errors occur in retrieving data from the source databases, an e-mail is sent to the system administrator.
- If the monitor cannot retrieve information from a data exchange, eventually an alert will be generated because the maximum time of inactivity is reached. When the system administrator checks the problem he or she will quickly see that the data exchange is functioning but that the monitor is not working.
- In the views in CRM, data exchanges are ordered based on the date and time of their last activity. In this way it is easy to see when the last results from data exchanges are received and complete malfunctioning of the monitor will easily be noticed.

6.11 Security aspects

CRM systems often contain sensitive information that may not be made public. In our project we retrieve monitoring information from multiple servers on which CRM is running so this could possibly introduce security issues. We have taken the following measures to ensure the security of data in the CRM systems is not affected.

- Both the monitor server and the CRM servers are running in a protected environment. This environment is secured by firewalls. All data remains in this protected environment.
- To retrieve monitoring information from SQL servers on the CRM systems, a special account is used. This account has no access to the CRM database but only read access to databases containing monitoring information. So even if this account is used by a hacker he or she will only be able to access the non-sensitive monitoring information.
- No CRM data is stored in the monitoring results unless the error is related to the input data.

6.12 Interoperability

The main components of our central monitoring server consist of Scribe and Microsoft Dynamics CRM. Both run on a Microsoft platform and will not run on other platforms. The prototype is however still capable of monitoring other platforms. In our prototype we only use Log4Net and Scribe as monitoring plugins, but these can be replaced by other monitoring plugins. Because we make use of Scribe as our data collection mechanism, we can accept almost every input format. As long as there is a plugin available that can monitor a data exchange and store results, Scribe can collect the results and store them in Dynamics CRM. Also when the structure of monitoring results is different from the structures we used in our prototype, Scribe can transform that data into the structure we need in Dynamics CRM.

Our prototype is not even limited to monitoring data exchanges. In fact it can monitor all types of software. As long as there is a plugin available that generates monitoring results, we can store them in Dynamics CRM and monitor the software. In that case the data exchange in the monitoring CRM is not a data exchange but another piece of software. In the case of CRM Resultants it can for example be used to monitor the execution of all custom built .Net code.

6.13 Test results of the prototype

In this section we provide the results of the testing period of our prototype. We present an overview of the type of errors that are found, the number of errors that are found and the real problems that the prototype already solved.

We started monitoring with our prototype on the 25th of October 2010. For some data exchanges it was however possible to retrieve data from transactions in the past. We did however not use results from before the 1st of October 2010. For each table or graph we present we specify the period of the results that are used.

6.13.1 Types of errors found by the prototype

To get as much distinct errors as possible, we used all available monitoring data from the 1st of October 2010 till the 11th of November 2010. We found the distinct errors that are shown in Table 3. We grouped errors in which actual values are shown into one error.

	Error	Count
1	Error opening data source Scribe.MSCRMAdapter30.Database	2799
2	A(n) SoapException occurred while executing the requested operation.	472
3	The profile for account with ID [value] could not be retrieved!	289
4	RequestCounter cannot be persisted, method is not yet implemented! Current counter value is [value]	241
5	The operation has timed out	94
6	User error: Meerdere aspiranten gevonden!	48
7	The following query returned 2 results, where 1 was expected	39
8	No account found with number [value]	37
9	User error: Inschrijving zonder corresponderende student in CRM	24
10	A(n) IndexOutOfRangeException occurred while executing the requested operation.	15
11	input value: [value]	15
12	Writing out properties of record..Entity name: account	12
13	Error opening data source HUTEST CRM database.	9
14	Postal code incorrect (semantic error)	6
15	No study could be found for provided id: 25DW00700502-D-MTIBIN!	4
16	Index and count must refer to a location within the string.	4
17	The following query returned 3 results, where 1 was expected	3
18	No werkrelatie could be retrieved for the provided ID ([00000000-0000-0000-0000-000000000000])	2
19	Multi-record operations are not allowed. At least 2 records matched.	2
20	Contact record was not created	2
21	CreateEntity failed: Generic SQL error.	1
22	The following query returned 5 results, where 1 was expected	1
23	An error occurred while retrieving KvK data	1
24	Account was not created	1
25	Call failed!	1
26	An uncaught exception occurred	1
27	<soapVal xsi:type="xsd:string"	1
	Total	4124

Table 3 Distinct errors from the period 01-10-2010 till 11-11-2010

As can be seen in the table we found 27 distinct errors in the period we analysed. For semantic results only one error can occur. The value is either correct or incorrect. We do not make a distinction in which part of the value is correct.

6.13.2 Counts of monitoring results

In the previous section we only listed the errors and not the successful monitoring results. To be able to relate the number of errors to the number of successful transactions we show graphs of

monitoring results for real time data exchanges and batch data exchanges. All monitoring results with the level INFO are successful transactions. All results with other levels are warnings or errors.

Results from real-time data exchanges

Because we were not able to retrieve monitoring information from real-time data exchanges before we started monitoring them, we use data from the period 25-10-2010 till 11-11-2010. In Figure 54 the distribution of the results over the monitoring result levels is shown. In the graph there is a relatively large amount of fatal errors. This is mainly caused by a real-time Scribe data exchange that could not connect to its source. Scribe then keeps retrying every minute and generates a fatal error at every attempt.

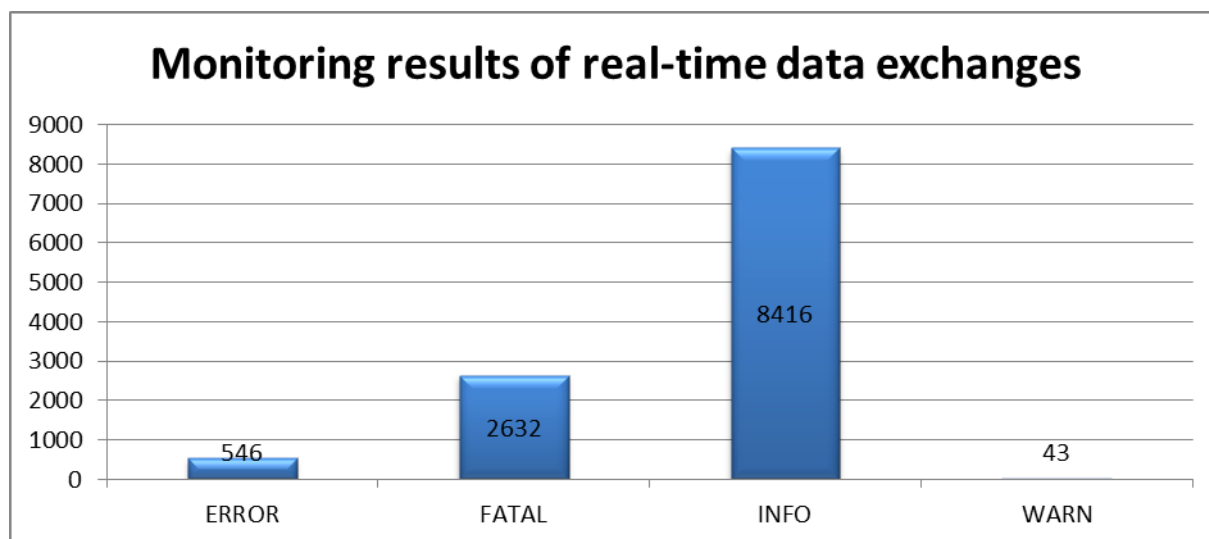


Figure 54 Monitoring results of real-time data exchanges by level from 25-10-2010 till 11-11-2010

If we look at how the monitoring results are distributed over the most active real-time data exchanges, we see that most fatal errors are indeed caused by the Scribe data exchange called Icares. This is displayed in Figure 55.

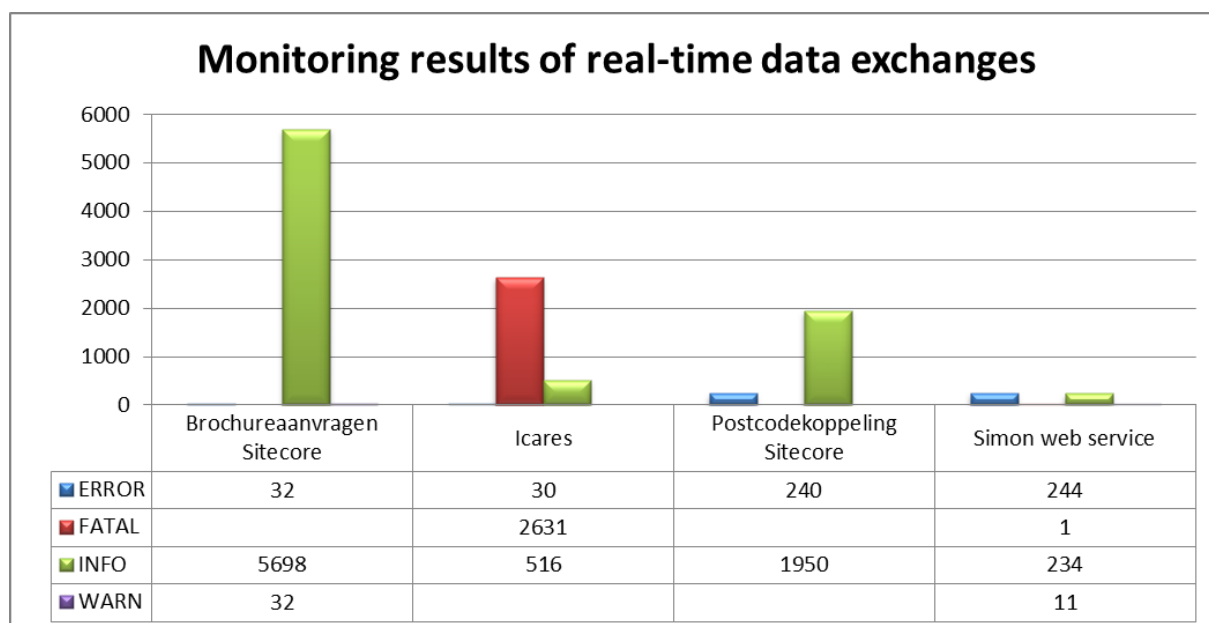


Figure 55 Monitoring results of real-time data exchanges by data exchange from 25-10-2010 till 11-11-2010

We can also view the monitoring results by date. In this way we can see when successful transactions were executed and when errors have occurred. This is shown in Figure 56. We can see that most fatal errors occurred on the 28th of October. This was caused by the fact that the SQL-server of the CRM environment crashed. The monitor detected this error properly. In this graph we can also see a pattern in the activity of data exchanges. We can see that there is less activity in the weekends, which are on 30-31 October and 6-7 November.

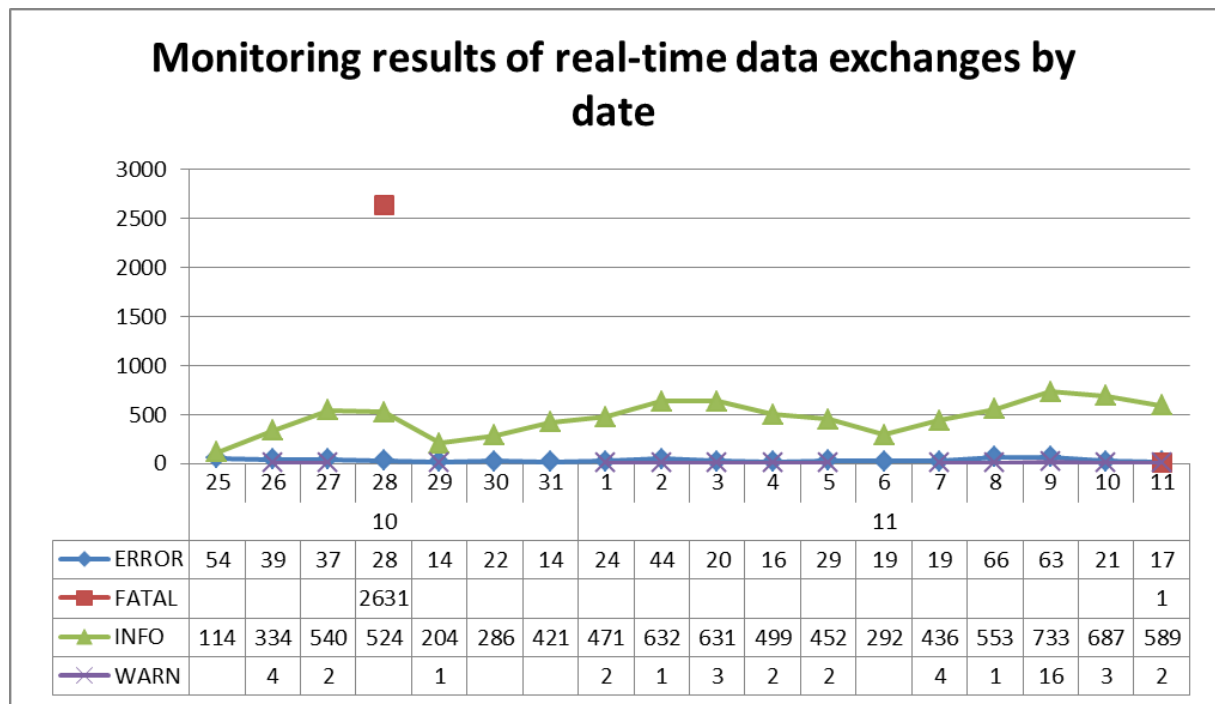


Figure 56 Monitoring results of real-time data exchanges by date from 25-10-2010 till 11-11-2010

Results from batch data exchanges

For the batch data exchanges we monitored it was possible to use results from the 1st of October till the 11th of November. If we look at the batch results of the three most active batch data exchanges we can see that the number of successful transactions is much larger than the number of errors. Errors are equal to rejected source rows. This is shown in Figure 57.

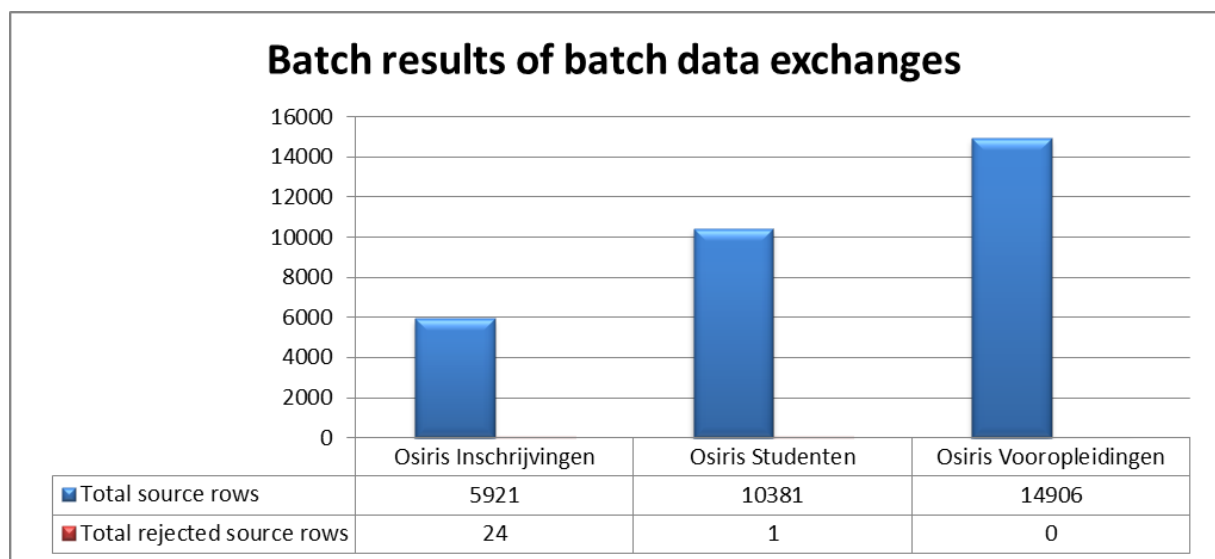


Figure 57 Batch results of batch data exchanges from 1-10-2010 till 11-11-2010

The batch results used in the graph in Figure 57 are the summaries of batch executions. If the execution is not able to start at all, a normal monitoring result is created containing a fatal error. Now if we look at the monitoring result records from batch data exchanges, we get the graph displayed in Figure 58. In this graph the errors correspond to the rejected source rows in Figure 57. Fatal errors are errors that prevented the batch execution from starting. This is quite a large amount of errors because Scribe data exchanges retry the execution of a batch job every 60 seconds and for each failed retry Scribe logs a fatal error.

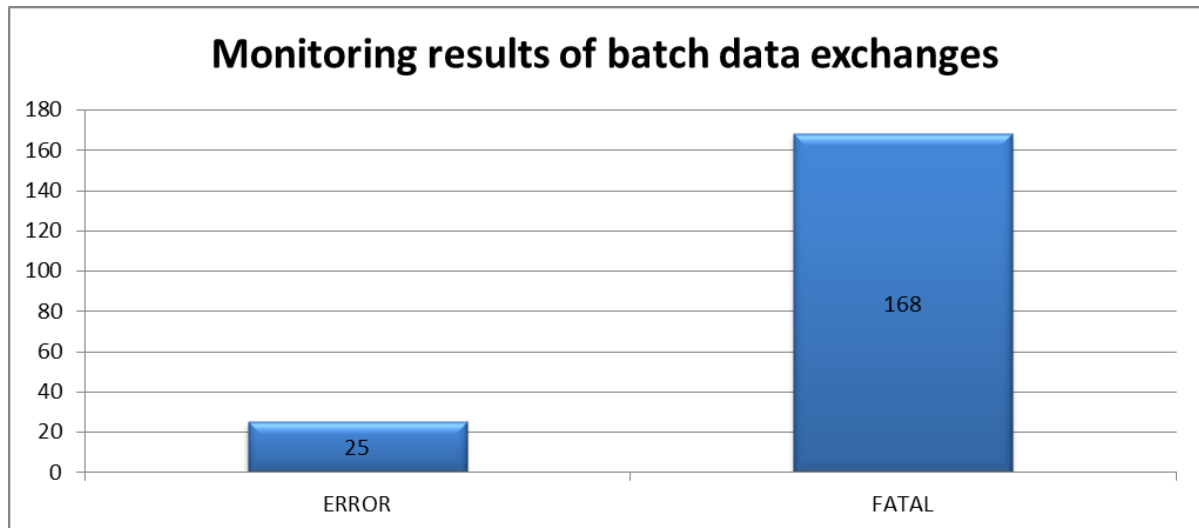


Figure 58 Monitoring results of batch data exchanges from 1-10-2010 till 11-11-2010

Semantic results

In our prototype the functionality for monitoring semantic aspects of data exchanges is very limited. As a result also the test results of this functionality are limited. In a testing period of 8 days we got the results shown in Table 4 for the semantics of postal codes. We expect that the low ratio of incorrect postal codes is due to the fact that at most input forms an automatic script is used to check postal codes and to retrieve the corresponding street and city. Therefore most users will correct the postal code before submitting the form.

Type	Number of results
Correct	5536
Incorrect	6

Table 4 Semantic results of postal codes

6.13.3 Real problems that were found by the monitor

The monitor stores a lot of data and of course the question is if this data helps in solving real problems. During our testing period the monitor already found some problems with data exchanges that needed attention.

- On Monday the 25th of October we saw that the last successful transaction of the Icares data exchange was more than 24 hours ago. This was unusual for this data exchange, so we looked deeper into the problem. It turned out that one of the services responsible for the data exchange was not running. By turning it on, we fixed the problem.
- On Thursday the 28th of October the monitor reported a lot of fatal errors from the Icares data exchange. It turned out that the SQL Server of the CRM environment stopped working. We already received a call from the customer because the whole environment was down, but at least the monitor detected the problem too.

- On Monday the 1st of November the complete hosting platform was cut off from the internet. Our monitor was still accessible because it runs in a separate environment. Because the monitor could not retrieve data from the data exchanges, it reported fatal errors. In this case we already knew the platform was down, but again the monitor also noticed it.
- The monitor showed that the batch data exchanges called Osiris Inschrijvingen and Osiris Vooropleidingen do not run every night as they should do. We never noticed that before. The reason turned out to be that no files were received. The problem was caused outside our domain, but at least we were able to notify the administrator who is responsible for sending the files.
- Based on the semantic results we were able to detect that a small number of lead records in the CRM environment of the Hogeschool Utrecht has an incorrect postal code. The number of errors is however so low that it does not need attention.

6.13.4 Test limitations

With our prototype we monitored real data exchanges that are running on a production environment. This made it impossible to create errors in the data exchange ourselves because this would harm the production environment of the customer. Because we had no influence on the errors that occurred ourselves we were only able to test the monitor on errors that occurred in practice. Therefore we assume that we did not test all possible errors. We have however no indication that other errors than we tested will not be caught by the monitor.

In our test results we show real numbers of monitoring results that were stored by our prototype. The main purpose of these numbers is to show that our prototype is working. You can however not directly compare the numbers of errors between data exchanges because not all data exchanges generate errors in the same way. If you look at Figure 55 you might conclude that Icares is the worst performing data exchange because it has the most fatal errors. In practice however it is one of the most reliable data exchanges. The difference is caused by the way it treats a complete downtime. In our monitoring period the complete CRM system has been down a couple of hours. In this period none of the data exchanges was functioning. If the system is down, the Icares data exchange keeps retrying to store data every couple of seconds and every time it fails it reports a fatal error. The other data exchanges are also down but they do not retry transactions so fewer errors are reported.

As described in section 6.13.3 we found some real problems with data exchanges that needed attention. Because during our testing period we paid a lot of attention to the monitor, we quickly saw potential problems. It is possible that a normal administrator would not have seen these problems in the monitor.

Our test results of semantic monitoring are very limited. We choose to monitor the number of correct and incorrect postal codes of lead records that were created. We have chosen this value because it was the easiest value to monitor. The problem with this value is that on most input forms there is an automatic check on the postal code. This results in a very low number of errors and as a result the potential of semantic monitoring does not show off. To show the real potential of semantic monitoring additional testing should be done on other values.

7 Validation and discussion

In this chapter we validate our prototype and eventually our reference architecture. The prototype is specified in chapter 6 and the reference architecture is specified in chapter 4. We relate the errors from literature and from our case study to the results that we presented in section 6.13 to see if our prototype and our reference architecture are valid.

7.1 Validation

Before we specified our reference architecture and built our prototype, we looked for possible failures in literature and we executed a case study to find errors that occurred in practice. We related these errors to aspects of data quality in Figure 11 and Figure 18. By monitoring the errors that affect data quality we have the opportunity to increase data quality. So if we are able to monitor the errors, our prototype and implicitly our reference architecture are valid.

In the test results of our prototype we listed the distinct errors that were detected by our monitor. We numbered them in Table 3 and in Table 5 we relate the errors we wanted to detect to the errors the monitor detected in practice.

Errors from literature	Error numbers from prototype
Service failure	1, 2, 21, 25 and more
Tardy service	5
Reluctant service	5
Doesn't play well with others	27
The gang that can't shoot straight	
Errors from case study	Errors numbers from prototype
Multiple records found	6, 7, 17, 19, 22
The record that is referenced cannot be found	3, 8, 15, 18
There is no reference specified to which the record should be liked	18
Message cannot be processed because it is syntactically incorrect	2, 27
Data wrongly formatted	11, 14
Data incomplete	14
Generic SQL error	21
Post job failure	
General exception in reading the file	1, 13
System exception	
Unable to open registry key	
Time out	5
Could not establish trust relationship for the SSL/TLS secure channel	
A SOAP exception	2
Web service is very slow and responds after a reasonable amount of time	
Send failure	

Table 5 Mapping of errors found by the prototype to errors from literature and our case study

As can be seen in Table 5 most of the errors we wanted to monitor are covered by the results of our prototype. Because we investigated possible errors of data exchanges with three different technologies and only monitored data exchanges with two different technologies, it is a logical result that not all errors are found by the monitor. It is also possible that not all errors occurred during the

testing period. We expect that apart from performance errors, the errors that were not detected in our testing period will be detected if they occur.

Performance problems can only be detected if they result in an error. If the performance of a data exchange is low but there is no data lost, our prototype does not detect the problem. In the future functionality for detecting performance issues can be added to the prototype. In general we can say that most of the errors we specified in our state of the art and our case study can be detected. These errors affect data quality so by monitoring these errors with our prototype the data quality of data exchanges can be improved.

By building the prototype we also showed that it can handle different technologies. We used Log4Net to monitor custom built web service data exchanges and we used Scribe to monitor Scribe data exchanges. These Scribe data exchanges can also use different technologies. This proves that our prototype is not technology dependent.

In our research we made a distinction between real-time and deferrable messages. The latter is also referenced to as a batch data exchanges. Both types of data exchanges can be monitored by our prototype so also at this point our prototype is valid.

Although we only tested semantic monitoring of postal codes, we show that the architecture for monitoring semantics works. In our prototype the semantic results were measured per CRM environment and not per data exchange to maintain the performance of data exchanges. This implementation is a bit different from the way the reference architecture is specified, but the same components are used and the data flow is also the same.

Because our prototype is based on our reference architecture, by testing our prototype we implicitly validated our reference architecture. The prototype we built is only one implementation of the reference architecture, but it proved that if the guidelines of the architecture are followed and the requirements are met, the monitoring solution works. Therefore we state that our reference architecture is valid.

7.2 Discussion

Our reference architecture is specified as generic as possible. This means that it can be implemented with various tools and should work in various environments. We have however tested this reference architecture only in the environment of CRM Resultants. In this environment it showed that it was valid, but there might be situations where this reference architecture does not work. This can be the case if for example the middleware that is involved in the data exchange cannot produce monitoring information or log results. In our case Scribe itself could log transaction results and for web service data exchanges Log4Net could produce monitoring information, but a similar solution might not be possible in all cases.

In our project we assumed that there is somebody who checks the monitoring results and takes action on potential problems. The reliability of data exchanges is of course only improved if there really is somebody taking action on the results. This means that there is not only a technical aspect on monitoring, but also an organizational aspect. We did not investigate the organizational aspect, but we can recommend making one person responsible for checking the results and that person can delegate potential problems to other people who might have more knowledge about the specific data exchange.

In our prototype we had set the conditions to generate alerts very tight. This was mainly to test if the mechanism of alerting was functioning correct. In real situations these settings should be tweaked over time to make each alert more meaningful. The risk of generating too many alerts is that the system administrator does not take them seriously anymore.

As explained before in section 6.5 our monitoring of semantics can only determine which values are incorrect. It assumes that all values that are correctly formatted are correct. In reality the values can still have no meaning. So to make semantic results more reliable additional checks should be done. Also for showing off the real potential of semantic monitoring, additional field should be monitored.

In our research we monitored at the target system to which the data of a data exchange is stored. In this way we are sure about the results of the transaction. It is also possible to monitor at the source system, but then it is only possible to monitor if data is being sent, without knowing what the results of the transaction are in the target system. In cases where CRM is the source system, our prototype can easily be used to monitor if data is being sent to the target system. We did however not investigate this possibility because in cases where CRM is the source system, a similar monitoring solution as we provided can be implemented on the target system.

In literature we found more possible errors than in our case study. We were only able to test our reference architecture on errors that occurred in the case of CRM Resultants. Although we could not test if all types of errors could be reported, we expect no problems here. As long as there is a plugin that can store the error somewhere, our central monitoring server is able to pick up the error and report it to the end user of the central monitoring server.

8 Conclusion

In this section we present the conclusions of our research, point out the limitations of our research results and give recommendations for future work.

8.1 Summary of results

The objective of this research is to create a reference architecture for systems that can provide central monitoring of data exchanges with the possibility of aggregating information on different data exchanges into useful diagnostic reports. In this section we summarize how we accomplished this objective.

We started by specifying the state of the art on modelling of data exchanges and identified classes of data exchanges, failures and the concepts of data quality and data exchange monitoring. Data exchange involves multiple systems which are composed of functional layers. A distinction can be made between data exchanges with real-time requirements and data exchanges that exchange deferrable messages. The latter are called batch data exchanges. With the state of the art study we answered our first research sub question about what types of data exchanges are available and need to be distinguished.

After we acquired knowledge about data exchanges in general, we focused on errors that can occur in data exchanges. We found a list of errors in literature and we executed a case study on errors of data exchanges in practice. We related these errors to data quality aspects to determine their impact on the target system of a data exchange. The target system of a data exchange is defined as the system to which data from a source system is stored. We found that there are three categories of errors: errors resulting from the contents of data, errors in middleware and connection and performance errors. These errors have impact on intrinsic, contextual and representational data quality. By identifying the errors and their effects on data quality, we answered our second research sub question about what problems can occur in data exchanges.

To determine possible options for monitoring data exchanges we looked at monitoring concepts proposed in literature. We found out that most monitoring approaches focus on system states such as CPU usage and disk space. We want to monitor functional aspects of data exchanges in which we make a distinction between syntactic and semantic monitoring. Syntactic monitoring is about the correct execution of transactions of a data exchange. Semantic monitoring is about the meaning of data that is exchanged. To acquire monitoring results from different sources into one central system there are two main mechanisms available: polling versus publish and subscribe. In our prototype we use polling to transfer monitoring results from plugins to the central monitoring server. This answers the third research sub question about how data exchanges can be monitored so that problems are detected.

To create a reference architecture that satisfies our requirements three components are needed in addition to the data exchange itself. First a monitoring plugin is needed. This component collects information about the transactions that are executed by the data exchange and stores this information. Second a central monitoring server is needed to store the results from different data exchange and to present the results to the end user. Third a data collection mechanism is needed to retrieve the monitoring information from the monitoring plugins and store the information in the central monitoring server. Apart from the components there are requirements on the information that is contained in the monitoring results. This reference architecture answers the fourth research sub question: "How should the architecture of a data exchange that includes monitoring functionality be specified?"

To test our reference architecture we looked for existing solutions with which we could implement our reference architecture. We did not find a solution that includes all the three needed components and meets all the requirements of the reference architecture. Therefore we used a combination of

products and built a prototype ourselves. We used Microsoft Dynamics CRM as our central monitoring server and Scribe as our data collection mechanism. Depending on the type of data exchange we used Scribe or Log4Net as the monitoring plugin. By monitoring both successful transactions and errors, we were able to detect not only errors, but also absence of transactions. Absence of transactions can be caused by the source system that does not send data or by problems that prevent data from the source system reaching the target system. Because no data is received, no error occurs, but by analysing the number of transactions, the problem can still be detected. Our prototype generates alerts if no successful transaction is executed for a specified amount of time. We tested our prototype by monitoring real data exchanges that occur in a production environment of a customer. The results of this prototype show that the prototype works and that the reference architecture is valid. This answers our fifth research sub question: "Which existing solutions are available which offer monitoring and recovery procedures and do they meet the requirements?"

By answering all research sub questions we answered our main research question. We specified a reference architecture that implements syntactic and semantic monitoring at the target system of data exchanges. By implementing the reference architecture the data quality of data exchanges can be improved.

8.2 Benefits

We started our research by declaring that integration between information systems becomes more and more important. As a result data exchanges are becoming more important and thus they need to be monitored. Our reference architecture and our prototype can help to monitor these data exchanges. The most important benefits that result from our research are the following.

- Based on our reference architecture monitoring solutions can be created for monitoring functional aspects of data exchanges.
- By monitoring data exchanges insight is gained in the transactions data exchanges execute.
- Errors in data exchanges can be detected before a customer complains about a problem. This can increase customer satisfaction.
- In our prototype all monitoring information is accessible from a single web based user interface. There is no need any more to check separate databases and folders with error logs. This can speed up the investigation on problems.
- Monitoring data exchanges can be offered as a service to customers. Customers can pay a monthly fee for this service or they can monitor their own data exchanges and pay a licence fee for the monitoring software. This can increase revenue for the application service provider.
- Although it has never been an objective of this research, using our prototype it is now possible to view which data exchanges put the most information in a CRM system. It was already possible to determine the records that were created in a specified period in the CRM system itself, but it could be hard to determine from which source the information originated.

8.3 Recommendations

Based on the findings of our research we give recommendations for application service providers in general and for CRM Resultants in particular. The general recommendations do not take our prototype into account but only the reference architecture. The recommendations for CRM Resultants do take the prototype into account.

8.3.1 General recommendations

- Create an implementation of our reference architecture for monitoring data exchanges between information systems to maintain oversight of the successful transactions and errors of data exchanges.
- Create an implementation of our reference architecture that can monitor all technologies that are used in the environment that is to be monitored.
- Make somebody responsible for checking the results and taking actions on the results. This does not mean that this person has to solve all problems, but he or she can also delegate the problems to people who have more knowledge about the specific problem.
- Make somebody owner of the monitoring server. This person has to manage the system, make necessary changes and setup monitoring for new data exchanges.

8.3.2 Recommendations for CRM Resultants

- Use the monitor for all new data exchanges that are developed and add it to existing data exchanges which are known to be prone to errors.
- Extend the alerting functions of the monitor to make them more reliable and to increase the ease of use.
- Include a service module to support the process of following up errors in data exchanges.
- Incorporate the monitor into the company's central CRM system to make the monitoring results easily accessible to all employees and to integrate it into the existing information about hosted CRM environments and customers.
- For monitoring web service data exchanges some keywords have to be agreed on between the owner of the monitoring server and the developers of data exchanges. These keywords can then be used to only include relevant messages in the monitoring server.

8.4 Limitations and future work

Our research has some limitations and there are opportunities for future work. Both are presented in this section.

Our reference architecture and our prototype are based on the assumption that we have influence on only one side of the data exchange. In the case of CRM Resultants this is true, but in other cases there can be situations in which you have influence on both the source and the target system of a data exchange. This provides additional opportunities for monitoring and can be researched in future work.

Because we monitor at the target system of a data exchange, errors at the source system can only be detected by the absence of transactions at the target system. This means that it takes at least the specified maximum time of inactivity before the error is detected. Monitoring at the source system can detect these errors faster and should therefore be further researched.

Semantic monitoring of data exchanges is very limited in our research. We only check the representation of data. There are additional opportunities for checking semantic correctness of data. Researching these opportunities can be done in future work.

We tested our prototype only in the environment of CRM Resultants. To test if it is indeed capable of monitoring data exchanges in other environments, additional testing should be done.

Our reference architecture and prototype can only detect problems but not correct them. The next step would be to investigate if it is possible to automatically correct problems or at least suggest solutions.

By building our prototype we created a solution for monitoring data from different sources. In this research these sources are data exchanges, but we expect that it can be extended to monitor other custom built software. This can be further researched.

During our research we found out that monitoring of syntactic and semantic aspects is not very common yet. Keeping the trends of service oriented architectures and integrating systems with each other in mind, we expect that this area of research will gain more attention in the next years.

References

- [1] Hart. C, *Doing a literature review: releasing the social science research imagination*. London, United Kingdom: Sage Publication, 1998.
- [2] Google Scholar. [Online]. <http://scholar.google.com>
- [3] Scopus. [Online]. <http://www.scopus.com>
- [4] William A Ruh, William J Brown, and Francis X Maginnis, *Enterprise Application Integration: A Wiley Tech Brief*.: John Wiley & Sons, Inc., 2001.
- [5] Chris Britton and Peter Bye, *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems (2nd Edition)*.: Pearson Addison Wesley, 2004.
- [6] C Bussler, *B2B integration: concepts and architecture*.: Springer-Verlag New York Inc, 2003.
- [7] H Zimmermann, "OSI reference model--The ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 28, pp. 425--432, 1980.
- [8] Microsoft Patterns & Practices Team, *Microsoft Application Architecture Guide (Patterns & Practices)*, 2nd ed.: Microsoft Press, 2009.
- [9] N Delgado, A Q Gates, and S Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Transactions on Software Engineering*, vol. 30, pp. 859 - 872, dec. 2004.
- [10] William N. Robinson, "Monitoring Web Service Requirements," in *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, Washington, DC, USA, 2003, pp. 65-75.
- [11] R Y Wang and D M Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of management information systems*, vol. 12, p. 33, 1996.
- [12] J Akoka et al., "A framework for quality evaluation in data integration systems," , 2007.
- [13] C Batini and M Scannapieco, *Data quality: Concepts, methodologies and techniques*.: Springer-Verlag New York Inc, 2006.
- [14] Suparno Biswas and Xavier Idrovo. (2006, January) ITSM Watch. [Online]. <http://www.itsmwatch.com/itil/article.php/3581491/The-Role-of-Application-Monitoring-in-ITIL.htm>
- [15] G Morgan, S Parkin, C Molina-Jimenez, and J Skene, "Monitoring middleware for service level agreements in heterogeneous environments," *Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government*, pp. 79--93.
- [16] Baron Schwartz. (2008, August) Xaprb. [Online]. <http://www.xaprb.com/blog/2008/08/21/is-agent-based-or-agentless-monitoring-best/>
- [17] Kay Roepke. (2008, August) Kay Roepke's Blog. [Online]. http://blogs.sun.com/kay/entry/agent_vs_agent_less_monitoring

- [18] Scribesoft. Scribe. [Online]. <http://www.scribesoft.com/>
- [19] (2010) Hyperic. [Online]. <http://www.hyperic.com/>
- [20] GlassFish. (2010, October) WS Monitor. [Online]. <https://wsmonitor.dev.java.net/>
- [21] SoapKnox Inc. (2010, October) SoapKnox. [Online]. <http://www.soapknox.com/>
- [22] ZOHO Corp. (2010, October) ManageEngine. [Online]. <http://www.manageengine.com/>
- [23] Eviware. (2010, September) soapUI. [Online]. <http://www.soapui.org>
- [24] Apache Software Foundation. (2010, October) Log4Net. [Online]. <http://logging.apache.org/log4net/index.html>
- [25] Microsoft. (2010, October) Microsoft Dynamics CRM. [Online]. <http://crm.dynamics.com/>