Data Integration and Warehousing in a Employee Status Information System at LSH

Daniel Rönström
Jonas Sidborn

26 June 2007

Master's Thesis in Computing Science, 20*2 credits
Supervisor at CS-UmU: Michael Minock
Examiner: Per Lindström

Abstract

In larger companies it is hard to keep track where the employees are located at the moment. They can be on meetings, education etc. In most companies there already exist several systems that hold information about employees, for example Outlook Calendar, Telephone Systems, Security Systems and simple Whiteboards. But mostly they do not integrate the status information that is stored there. Instead, the most common way is that companies have a whiteboard placed outside each department where employees can write down their current status. The accuracy of the update interval is highly personal and it does not get a response if the meeting takes longer than planned. This Master Thesis presents a system that integrates several existing independent status systems in order to get as much information as possible of an employee and presenting it on a presentation screen outside the department and hopefully solve many problems that ordinary whiteboards has. This thesis also presents an in-depth study on data integration focused on materialized views and data warehousing.

Contents

List of figures

1. Introduction

This report describes the Master's thesis work performed at the Department of Computer Science at Umeå University between January and June 2006. The commissioner of the work was Land System Hägglunds (LSH) in Örnsköldsvik. LSH is a company that has about 1300 employees and is the leader in developing tanks and off road vehicles. A couple of years ago they became a part of the big concern BAE Systems; a British company specialized in defense systems. [BAE]

Many big companies worldwide have problems to keep track of where the employees are located at the moment without intrusions on a persons identity; this was also the problem for LSH. At the moment the different departments at LSH has a whiteboard located outside each department. It shows who are working there and employees are supposed to write down if they are on meetings, vacation, etc. The accuracy of the update interval is highly personal and it does not get a response if the meeting takes longer than planned. The meaning of this work is to develop an integration system that gathers status information of an employee from already existing systems at LSH and present it through a GUI.

1.1 Structure of the report

Including with this introduction chapter, this thesis will consist of 6 chapters. The content of each chapter is described below.

- *Chapter 2*: Problem description
- *Chapter 3*: Data Integration and Data warehousing
- *Chapter 4*: Overview of used techniques
- *Chapter 5*: Implementation
- *Chapter 6*: Result
- *Chapter 7*: Conclusions

2. Problem description

This chapter describes the requirements of the prototype developed at LSH that shows the current status of an employee.

2.1 Goals

The goal with this Master thesis project is to create a tool that increases the efficiency of work for employees at LSH and provide a more dynamic way of presenting status of an employee than the scenario described in previous section where the employee had to write down the status on the whiteboard.

2.2 Description

The system is divided into four parts. The first part collects information about the status of an employee from different systems. The second part uses an integration engine which transports data messages between different business systems. The third part was to design a new database that holds all information about the employees. The last part is to present the information on a Graphical User Interface (GUI).

2.2.1 Status information systems

An important concern is that the systems used to gather status information on an employee should not be integrity insulting. Therefore systems that were already public for employees were used. Three systems that matched this description are listed below.

- The Telephone system, SQL Server
- The Check in system, Oracle Server
- The Outlook calendar, Exchange Server

There are also other systems that are used to gather information for the application. These systems contain additional information about the employees. For example which department they are working at, different mail group's that they are members of, etc.

- Active Directory, AD
- Personal Data Register, HR

A Technique called consolidation is used to gather this information and push it to a new database (Data Warehouse).

2.2.2 Web Services

Web services are used for the communication with company data sources. Each web service should have a three layer structure. The first layer is the public methods that are supported by the Web Service. The Second layer is called Business Object Layer (BOL). This layer handles all logic like conversions, calculations on data, etc. The last layer is the Data Access Layer (DAL) that handles all communication with the data source. A picture describing the structure is presented below.

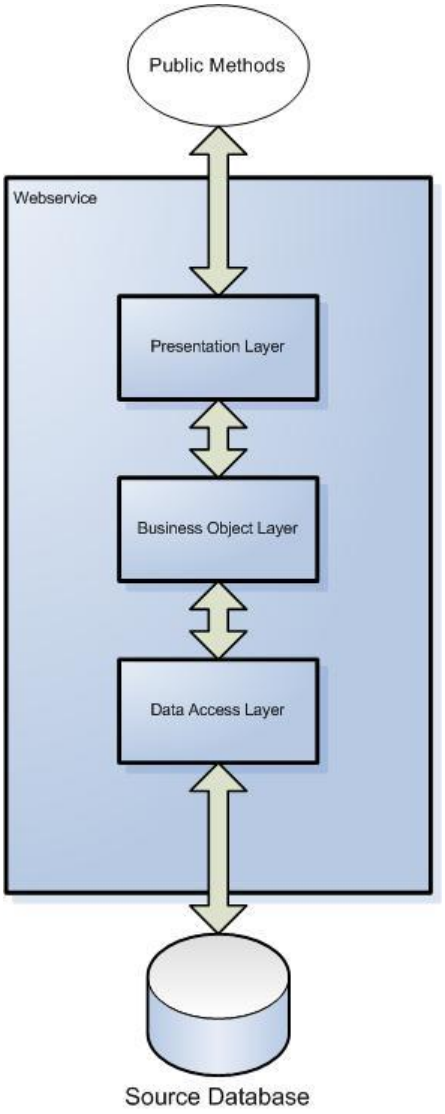

Figure 1: Picture describing web service layer structure

2.2.3 Integration engine

BizTalk is an integration platform developed by Microsoft. LSH is using it to export and import data between different business systems. All communication between Web services connected to different business systems passes through BizTalk.

2.2.4 Database

According to the programming guide at LSH the database is supposed to have all of its intelligence in itself by using stored procedures, triggers and dependencies between tables. It should also be designed in a way that makes it easy to make changes in the future.

2.2.5 GUI

In order to present the status information of an employee a GUI was to be developed. The requirements stated that it should be viewed as two different themes. One theme for presentation displays outside each department and one for the internal web accessible for every employee. Each employee should be able to create/edit/delete their own status board of employees and subscribe on other ones by adding them as favorites. A user should also only

have to logon once in the system to be authorized to view their own status boards. The GUI should have a three layer web architecture as described earlier.

2.3 Limitations

The limitation of the system that was stated before the application was developed is listed below

- 3'rd party system – changes in the main systems will cause changes in this application
- Human factor – e.g. employees that forgot to check in, put off their phone, missed to cancel a meeting etc.
- Consultants – They don't use the check in system so it's impossible to know if they are available or not.
- It only works on the Windows platform.
- It is only in Swedish.

2.4 Methods

The following software will be used when developing the system

- *Visual Studio 2005* – All code will be written with this tool except for the BizTalk application.
- *Visual Studio 2003* – All BizTalk 2004 code will be written in this tool.
- *SQL Server 2000* – The database constructed is made in this tool. Including stored procedures etc. also holds information about employee data and employee telephone status.
- *BizTalk 2004* – The integration engine that defines the flow that a message takes.
- *Exchange* – Holds information of outlook calendars for employees.
- *Oracle* – Holds information about check in/out times for employees.
- *Active Directory* – Holds information about user email, mail groups etc.

2.5 Overview of the system

As described earlier the system consists of four main parts:

- Retrieve data from different status systems.
- Integrate data into a global schema specified in the data warehouse.
- Designing a data warehouse that contains information about employees.
- A GUI that will provide functionality on the constructed data warehouse.

On top of each data source there is a web service. The functionality of the web services is to retrieve data from each data source and convert this data into an XML subschema that is a part of the data warehouse master schema. On top of each web service there is an integration engine that will collect all the data that is retrieved by the web services. This data is then sent to a web service that inserts/updates/deletes the collected data into the constructed data warehouse. This it the main task in this system. Besides the integration part there is a GUI on top of the data warehouse that will show the collected information. The GUI communicates via a web service providing functionality against the data warehouse. This web service also acts as an API to the constructed data warehouse.

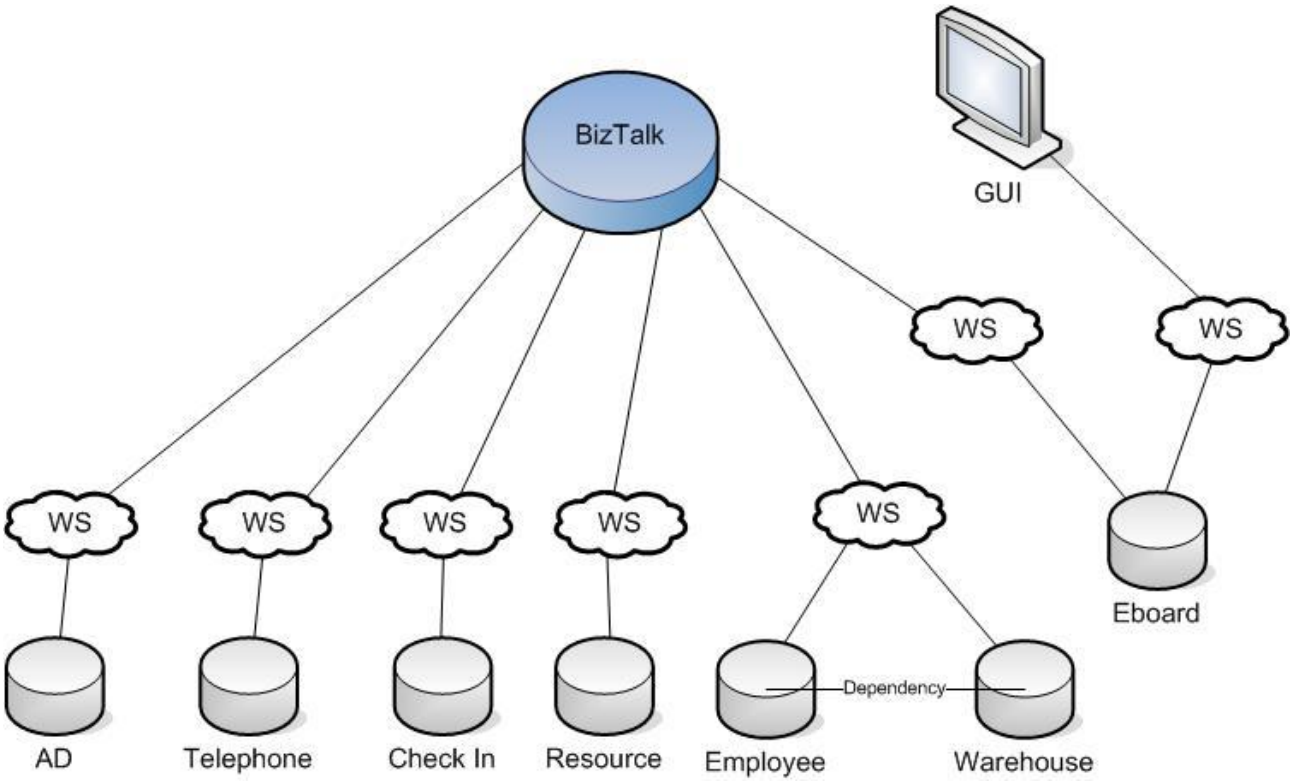

Figure 2: Picture showing all parts of the system.

3. Data integration and Data warehousing

This Chapter focuses on an in depth study on data integration, data warehouses and materialized views. At first the meaning of the word *"data integration"* must be clarified. Data integration is about the capability to link together different datasets with information and thereby enables the users to interact with them as if they were a single, unified and homogenous data source.

3.1 Why Data Integration?

Why does the need of data integration arise? Most of the companies these days acquire different datasets under an assumption that many people in the organization will use them. However as time passes users may need access to new datasets acquired by others or that are stored under a different system. If that is the situation, they may be faced with a range of barriers. To overcome these barriers it requires that the datasets must be described in detail for other users to understand their meaning. Another important issue is agreement on standards for description and for adoption of compatible computing infrastructures. It also requires that the organization holding the datasets to agree on common policies for sharing and access.

3.2 Challenge

The main reason that makes data integration a difficult task is that is has to deal with problems and conflicts arising from heterogeneity, distribution and autonomy. For example there are many different formats in which data can be encoded. Other issues that arise are that there exist different types of database software and data may be held on many different computers where the mechanisms for user authentication and authorization may differ.

3.3 Forms of Data Integration

There are four forms of Data Integration[DI]. *Copy Definition, Copy Values, Shared Definition* and *Shared Values*.

In *Copy Definition* the data definitions are copied into another system for local use. There are no real time changes between the original definitions and the copied definitions. There is no means for ensuring that there is a consistent definition between the two systems.

In *Copy Values* the data values are copied into another system for local use. There are no real time changes between the original values and the copied values. There is no means for ensuring that there is a consistent value between the two systems. In addition there is no means for ensuring that the original definition of the data is the some across the component systems.

In *Share definition* the data definitions of a component system are the same data definitions used by all other component systems. There is only one set of data used by all integrated systems. If one of the systems implements a change in the definition it must be reflected in all other component systems.

In Shared Values the data values produced or/and consumed by other systems are the same values that are used by all other component systems. There is one and only one set of instances used by the integration system. Immediately when changes are made on values it is automatically available in all other component systems. [DI]

3.4 Querying Data

The basic idea behind data mapping is to bind a global master schema by combining different local schemas, which provides access to multiple data sources with a single uniform interface. The query to the global schema is then reformulated over the different local schemas to retrieve the correct data from the different local schemas, this is called mapping.
The data transformation from the global schema to the local schema is usually executed in a number of steps; the first step is query reformulation which reformulates the query in the most efficient way. After this step there might be an optimization step, and then the optimized query is sent to the query execution engine which handles the execution on each query against the local schemas provided. Before the query is sent to the local schemas they are sent to a wrapper which translates the query into the local schema specific querying language. The execution engine then joins the result and sends the complete result back to the caller. [LBT]

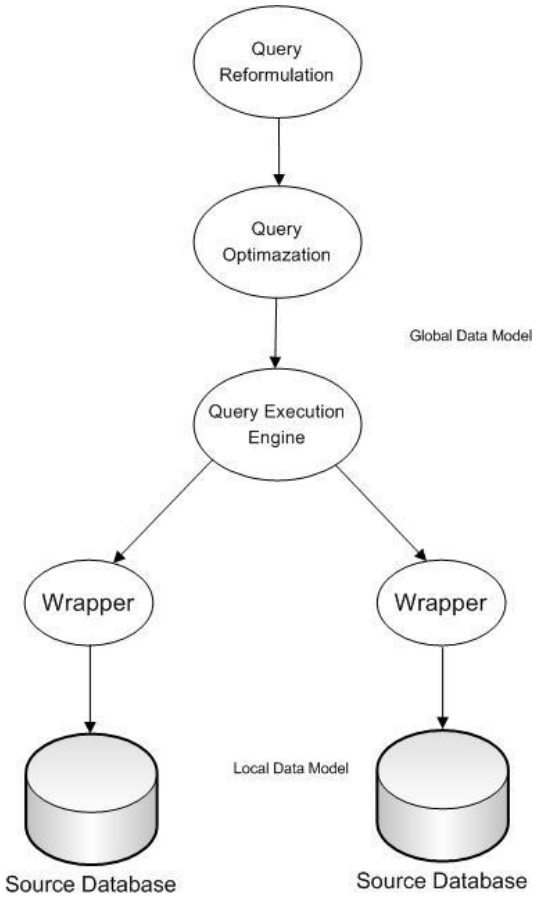

Figure 3: Architecture of a data integration system

3.4.1 Global schema

The global schema can be described as a sum of all the schemas residing at different sources in a data integration system. The schema describes the information, constraints, relations published to the user. The user can query this schema and then the underlying technology is supposed to take care of reformulation, querying, mapping to the underlying data sources. [BIS][LGAV1]

3.4.2 Local schema

The local schema is the schema of the real data at the different sources. The schema describes a part of the global schema (Reformulation, Wrapping e.g. might be needed). [BIS][LGAV1]

3.4.3 Mappings

The mapping approach can be divided into four big parts as described earlier.

3.4.3.1 Query reformulation

In the case of querying data in the data integration system, the query is posted against the global schema, the integration system then needs to reformulate the query into one or more real data sources in the data integration system. In a system like this, one would like this query to be sound (The reformulated query should give the correct answer to the input query) and complete (all the answers that can be extracted from the data sources should be in the result applying the reformulated query) and give the real answer to the query given by the user. In addition, one wants the query to be efficient and don't access irrelevant sources. [LBT][EQR][LGAV1]

3.4.3.2 Query optimization

The optimization of a query is typically to formulate a query execution plan which is a description how to query the different sources and get the result in the best and most efficient way. This plan specifies in which order to perform the different queries against e.g. sub sources, and in which order to perform operations on the result e.g. joins, selections. [LGAV1]

3.4.3.3 Query execution

The execution of the queries is basically a query against the actual data source in a general query language used by the wrapper. [LGAV1][AAQE]

3.4.3.4 Wrappers.

The wrapper layers only task is to transform data from the data source to a more general data representation used by the query execution manager. E.g. transform HTML-pages to an XML-document. [LGAV1]

3.5 Views

A view can be described as a union of subparts from different data sources. The view should have the same properties as the original data, e.g. when original data is changed the view is also changed. The purpose of a view is to increase performance on data that is commonly accessed. The picture below describes the basic functionality of a view. [MMV]

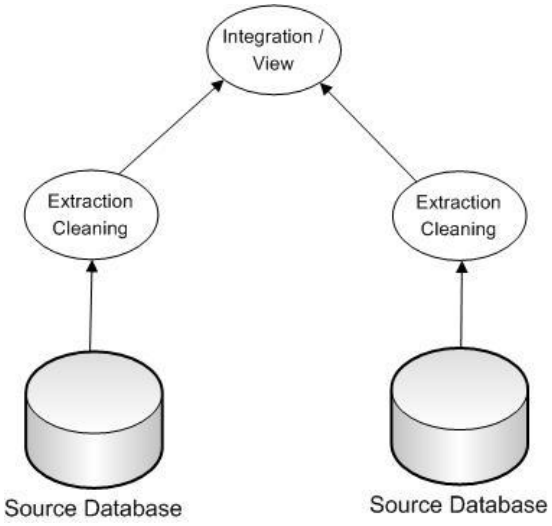

Figure 4: Description of a view

3.6 Mapping Approaches

The biggest problem in a data integration system is that the data is stored in the data sources, but the queries are made against the global schema. In order for this to work there must be some description in the mediated schema about the relations to the local data sources. The integration system must therefore provide a query reformulation engine so the user can post a query against the mediated schema and the system then reformulates the query against the different data sources needed. The easiest but not so efficient way is to describe this is to use first order logic. Hence, several approaches have been made to solve this problem, Global as view, Local as view, Both Global Local as view and lots more. These methods are described below. [MMDI][LBT]

3.6.1 Global As View

The Global As View (GAV) approach all the tuples in the mediated schema has a specific description how to obtain the specified data in the source relations.
An example how to obtain the data from the local sources can look like this, there are three databases containing data about movie actors. The three databases can then be unified to the mediated schema with a rewriting:

$DB_1(id1, title, actor, year) \rightarrow MovieActor(title, actor)$
$DB_2(id2, title, actor, year) \rightarrow MovieActor(title, actor)$
$DB_3(id3, title, actor, year) \rightarrow MovieActor(title, actor)$

If there are a fourth database containing reviews of movies which shares data with the first and the second database we can describe this with the following relation:

$(DB_1(id1, title, actor, year) \wedge DB_4(title, review))$ UNION
$(DB_2(id2, title, actor, year) \wedge DB_4(title, review)) \rightarrow MovieReview(title, review)$

In general the GAV approach is a set of horn rules that describes the relation between the mediated schema and the data sources. [LBT] [LGAV1]
The query reformulation step described earlier is fairly simple in the GAV approach since the relations in the mediated schema are described in terms of the source relations.

To make a query against the mediated schema just simply use the composed rules above, e.g. you want to find movie reviews on movies with the actor 'Nils Johansson' Simply combine the rules above to obtain the answer:

Review(title, review) :- MovieReview(title, review) ^ MovieActor(title, 'Nils Johansson')

3.6.2 Local As View

The Local As View (LAV) approach the descriptions are given in the opposite direction. The content of the data sources are described as a query over the global schema relation.

In this case suppose there are two different data sources V_1 containing titles, years, directors of films produced after 1960. V_2 containing reviews of movies produced after 1990. In the LAV approach the descriptions of these relations would be:

S_1: $V_1(title, year, director)$ è $Movie(title, year, director) \wedge year \geq 1960$
S_2: $V_2(title, review)$ è $Movie(title, year, director) \wedge year \geq 1990 \wedge MovieReview(title, review)$

The query in the LAV approach is harder since it's not that simple to unfold the definitions of the relation in the mediated schema. Suppose asking for reviews made after 1950.

q(title, review) :- Movie(title, year, director) ^ Year ≥ 1950 ^ MovieReview(title,review)

The reformulated query would be

q'(title, review) :- $V_1(title, year, director) \wedge V_2(title,review)$.

In this case the reformulated query is not equivalent to the original query, because it only returns movies produced after 1990, though this is the best reformulation that is possible. [LBT] [LGAV1]

3.6.3 Both As View

The both as view (BAV) has the advantage over LAV and GAV that it both support global and local schemas. It also supports removal and addition of local schemas, so called extensions in the already existing schema. New view definitions can then be regenerated from the new pathways as needed for query processing. BAV is very well suited for integration in P2P systems because of the easy regeneration of schemas and pathways.

The basics of the BAV approach are that it uses a low level hyper graph and a set of primitives for describing schema transformations. Facilities are provided for defining higher level modeling languages and primitive schema transformations. Previous work has shown that for example ER, UML, XML etc. can define these transformations. For each type of modeling construct of each modeling language (e.g. Primary key, Foreign key, Element, Parent-Child relationship) there will be a set of primitive schema transformations for adding such a construct, removing such a construct and in some cases renaming such a construct. Schemas are incrementally transformed by adding a sequence of such incremental schemas $t_1 \ldots t_n$. [BAV]

3.7 Data Warehouses

Definition: *A data warehouse is a database geared towards the business intelligence requirements of an organisation. The data warehouse integrates data from the various operational systems and is typically loaded from these systems at regular intervals. Data warehouses contain historical information that enables analysis of business performance over time.* [DWD]

3.7.1 Introduction

A data warehouse is a set of materialized views over the operational information source of an organisation, designed to provide support for data analysis and management decisions. A data warehouse is mainly used with data integration. The data is passed from an application oriented environment to the data warehouse. The data is inconsistence and redundancy checked so that the warehouse is able to provide an integrated and reconciled view of data from the organisation.

Generally speaking, information integration is the problem of acquiring data from a set of sources that are available for the application of interest. This problem is a central issue in several contexts, including Data Warehousing, Interoperable and Cooperable systems, Multi database systems, and Web information systems. The main two modules that are needed to integrate data are wrappers and mediators. The goal of a wrapper is to collect data from a source, filter the correct information and return the data in a specified format. The role of a mediator is to collect, clean, combine data produced by different wrappers (or mediators), so as to meet the specific information needs of the integration system.

According to the above considerations the Local as view approach for information retrieval is the most relevant for data integration to achieve data warehouse quality. [DIW][LGAV1]

3.7.2 General framework

A data warehouse can be seen as a database which maintains an integrated and reconciled materialized view of information residing in several data sources. A basic framework is divided into three different levels of abstraction:

- Conceptual level - contains a conceptual representation of the data.
- Logic level – contains a representation in terms of a logical data model of the sources and of the data materialized in the data warehouse
- Physical level – contains a specification of the stored data, the wrappers for the sources, and the mediators for loading the data sources.

The relation between the different levels is explicitly specified with object mappings. [DIW]

3.7.2.1 Conceptual level

The conceptual level represents the data represented by the enterprise, including a conceptual representation of the data in the sources. The model also includes the global concepts and relationships that are of interest to the data warehouse application. The conceptual model is independent from any system consideration, and is designed to describe the semantics of the application. The picture below shows a basic conceptual layer. [DIW]

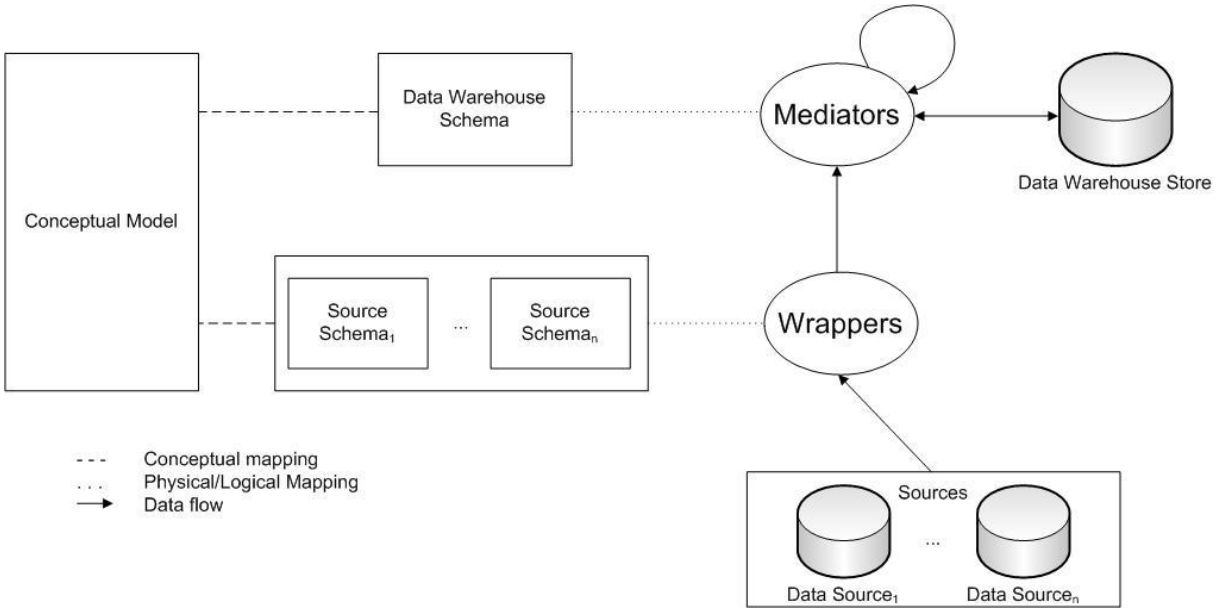

Figure 5: Architecture for data integration

3.7.2.2 Logic level

The logic level provides a description of the logical content of each source, called the source schema, and the logical content of the materialized views constituting the data warehouse, called the data warehouse schema (see figure above). This schema is intended to provide a structural description of the content of both the sources and the materialized views in the data warehouse. The logical content of a source relation is described in terms of a view over the virtual database represented by the conceptual model, adopting the local-as-view approach. To map physical structures to logical structures wrappers are used. The wrappers hide the way the sources store its data, the data model it adopts, etc. and presents the sources as a set of relations.

The data warehouse schema, which expresses the logical content of the materialized views constituting the data warehouse, is provided in terms of a set of relations. Each relation of the data warehouse schema is described in terms of a query over the conceptual model. [DIW]

3.7.2.3 Physical level

The physical level is where the actual data is stored. The data on this level is mapped from local schemas to the data warehouse schema. This is done with wrappers. According to the used technique the data is either transformed or stored in the data warehouse.

3.7.3 Keeping ER

According to the defined conceptual model above there are some properties that can be automatically verified. The consistency of the database satisfies all the constraints specified in the conceptual model. There exists a database that satisfies the conceptual model in which a certain entity has a nonempty extension. Entity relationships are where the extension of an entity is a subset of an extension of another entity in every database satisfying the conceptual model. Constraints in the conceptual model hold for all databases.

3.8 Materialized Views

Definition: A materialized view (MV) is similar to a view but the data is actually stored on disk (view that materializes). Materialized views are often used for summary and pre-joined tables, or just to make a snapshot of a table available on a remote system. A MV must be refreshed when the data in the underlying tables is changed. [MV]

Materialized views can be compared to a cache, they provide fast access to data and since query speed is a critical issue in some applications it is not effective to recomputed the view for every query. Materialized views are frequently used in application such as data warehousing, replication servers and also for query optimization. [MMV].

3.8.1 View Maintenance

Since a materialized view is a copy of data, it can contain dirty data[1] if the original data is updated. The progress for updating a materialized view with the correct information is called view maintenance. [MMV]

3.8.2 Incremental view maintenance

Incremental view maintenance is a technique used to maintain a view without recomputing it from scratch if changes have occurred in data. It is often cheaper to compute and propagate the changes to the view for the update of the materialization. This, since the size of the base relation and the view, compared to the changes are very small. There have been many proposals of strategies to support incremental view maintenance. [MMV]

3.8.3 Dimensions

The problem with view maintenance can be studied in four dimensions, Information-, Modification-, Language- and Instance-Dimension.

The Information-Dimension is about the amount of information that is available for the view maintenance. Questions like: Do you have access to all/some of the base relations? Do you have access to the materialized view? Do you know about integrity constraints and keys?

The Modification-Dimension is about what type of modification the view maintenance algorithm can handle. Questions like: Can it handle Insertions/Deletions of tuples on the base relation, Are updates of tuples handled directly or are they deletions followed by insertions?

The Language-Dimension is about questions like: Is the view expressed as a select-project-join query, or in some other subset of relational algebra? Can it use recursion? Can it use aggregation? Etc.

The Instance-Dimension can be divided into two types; database instance and modification instance and is about questions like: Does the view maintenance work for all instances of the database, or just for some? Or does it work for all instances of the modification, or only for some?

[1] E.g. incomplete or outdated information, incorrect data associated with fields, duplicated data, spelling mistakes and punctuation.

Gupta and Mumick stated in their article [MMV], that these four dimensions define the problem space for view maintenance. Figure 6 shows the problem space in a 3D model without the fourth instance dimension to keep figure manageable. For each point in the 3D space we may get algorithms that apply to all databases and modification, or that only works for some instances of each, the fourth dimension.

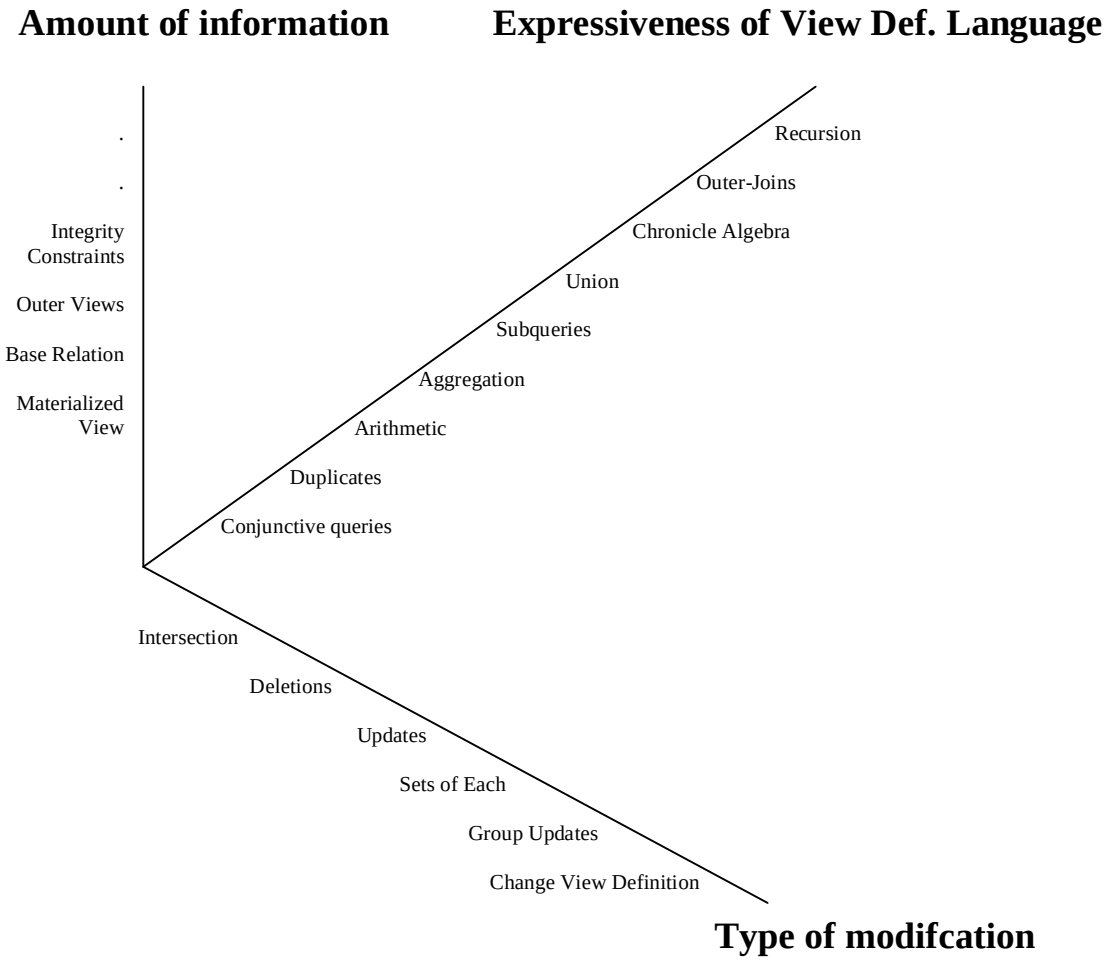

Figure 6: View maintenance dimensions [MMV]

3.8.4 Full information algorithms

Most research on view maintenance has focused on that all base relations and materialized views are available during the maintenance process. The techniques that have been developed out of theses researches have focused mostly on efficiency in different languages. All from select-project-join views to relational algebra like SQL and Datalog with features like aggregation, duplicates, recursion and outer-joins.
The techniques can be divided along the language dimension into non-recursive views, outer-join views and recursive views. [MMV]

3.8.4.1 Non-recursive Views

The most common algorithm for view maintenance with full access to information is the counting algorithm. Several counting algorithms have been developed supporting one or more languages but they all have in common that they keep a count of the number of derivations for

each tuple as extra information in the view. One that applies to SQL views is [GMS93]. It can handle duplicates, may be defined using union, negation and aggregation.

An example that illustrates the counting algorithm is shown below. The example is written in SQL syntax.

```
CREATE VIEW hop(S, D) AS
(
     SELECT DISTINCT l1.S, l2.D
     FROM link l1, link l2
     WHERE l1.D = l2.S
)
```

The view `hop` is defined such that `hop(x,y)` is true if `x` is connected to `y` via an intermediate node. It is given that link consist of the tuples `{(a,b), (b,c), (b,e), (a,d), (d,c)}`. According to the view definition hop is then evaluated to `{(a,c), (a,e)}`. The tuple `hop(a,c)` has two derivations and `hop(a,e)` has one unique derivation. The counting algorithm stores these numbers of derivations in the view.
Now suppose that `link(a,b)` is deleted, then hop can be recomputed to `{(a,c)}`. The counting algorithm tells that one derivation of each of the tuples `hop(a,c)`, `hop(a,e)` is deleted. Since the algorithm uses the stored counts it results in that `hop(a,c)` has one derivation left and `hop(a,e)` has no more derivation left and therefore is deleted. [MMV] [GMS93]

3.8.4.2 Outer-Join Views

In domains like data integration and extended relational systems is outer joins very important. Outer joins can be subdivided further into left outer join, right outer join and full outer join. Outer joins are very different from inner joins. Instead of limiting results to those in both tables, it limits results for those in left table for left outer join and is reversed for right outer joins. Full outer joins combines the result of both left and right outer joins.

For example in left outer joins:

```
SELECT *
FROM employee
     LEFT OUTER JOIN
     department
       ON employee.DepartmentID = department.DepartmentID
```

If the tuple `employee.DepartmentID` doesn't have a match with a department, the row will still be returned, but with `NULL`-values for each column from department. The result for a right outer join is reversed and for the full outer join the result is combined both left and right outer join.

An algorithm for maintaining an incrementally full outer join view has been developed by Gupta and Mumick. [GJM94]. The following SQL syntax defines the view `V` as a full outer join of relation `R` and `S`.

```
CREATE VIEW V AS
     SELECT X1, …, Xn
     FROM R
     FULL OUTER JOIN S
     ON g(Y1, …, Ym)
```

X and Y are attributes from the relations R and S. g(Y1, …, Ym) is a conjunction of predicates that represent the outer-join condition. Modifications of the two sets is denoted as △(R) and △(S) witch consist of insertions (△+) and deletions (△-). The view maintenance algorithm rewrites the view definition to compute △(V) as follows.

a)
```
SELECT X1, …, Xn
FROM △(R)
LEFT OUTER JOIN S
ON g(Y1, …, Ym)
```

b)
```
SELECT X1, …, Xn
FROM Rv
RIGHT OUTER JOIN △(S)
ON g(Y1, …, Ym)
```

Rv represents relation R after modification. All the other references in queries a and b refer to either the pre-modified extents or to the modifications themselves. Query a) computes the changes to relation R and query b) computes changes to relation S. Both of the queries however may compute possible side effects that are explained below. The algorithm developed by Gupta and Mumick handles these side effects [GJM94]. Since query a) computes the effects on view V according to changes in relation R. Say that a tuple r+ is inserted to R. It will imply an effect on the view. If r+ does not join with any tuple in S then r+ padded with nulls has to be inserted into the view V. Another scenario is if r+ does join with some tuple s in S. Then r+ joined with s is inserted into the view V. However, this may cause a side effect. Say if r+ does join with a tuple s. Lets say s padded with nulls, may have to be deleted from the view V if it already exist in the view. This will be the result if the previous tuple s didn't join with any tuple in relation R. Similarly, a deletion r- from R does not only remove a tuple from view V, but it may also imply a insertion of a tuple s padded with nulls, if before the deletion r- is the only tuple that joined with s. Query b) handles changes to relation S similar to that the query a) handles changes to relation R. The side effects also appear with query b). But as mentioned these side effects is handled with the algorithm that Gupta and Mumick has developed [MMV] [GJM94].

3.8.4.3 Recursive Views

Recursive views are often expressed in Datalog. All the work of maintaining a view were made in this context. Several techniques have been proposed for maintaining materialized views in this area. The DRed algorithm that applies to Datalog or SQL views and is explained in the article [GMS93]. Other algorithms are The Propagation/Filtration algorithm [HD92], The Kuchenhoff algorithm [Kuc91] and The Urpi-Olive algorithm [UO92]. It is also possible to use counting based algorithms but most of them are limited in for example that every tuple is guaranteed to have finite number of derivations.

3.8.5 Partial Information

It should be mentioned that there also exist partial information algorithms. These techniques are using only a subset of the underlying relation involved in the view. Unlike full information algorithms, partial information algorithms does not always work for maintaining a view if changes has been made. The maintaining process for partial information algorithms

is also dependent of what type of modification it is about. If it is an Insert, Delete or an Update. In this way the algorithms must check if it is possible for the view to be maintained, and then how to do it. At this moment there only exist algorithms that handles updates modifications since algorithms that handles deletions and insertions becomes very complex. [MMV]

4. Overview of used techniques

This chapter describes used techniques and tools during the implementation of the system. It only provides a quick overview of the concepts.

4.1 Web Services

Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[2]). Other systems interact with the Web service in a manner prescribed by its description using SOAP[3] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [WS3]

Functionality of a web service:

- The web service's functionality is publicly described in a WSDL file.
- Web services communicate with other applications through XML messages, typically in SOAP format.
- Web services generally use a standard network protocol, such as HTTP.
- Most common web services uses Service oriented architecture

4.1.1 Architecture

The most common architecture in web services is service oriented architecture. This architecture has a service provider, a service requestor and a discovery agent. The service publisher publish it's functionality to the discovery agent. The requestor then asks the discovery agent for available services and gets the response that there are a service at a specific location. The service requestor and the service provider can then interact independently according to the functionality described by the discovery agent. The communication between the provider and the requestor is platform independent. The communication is done in XML via e.g. SOAP, MIME or HTTP. [WS1][WS2]

[2] Web Service Description Language
[3] Simple Object Access Protocol

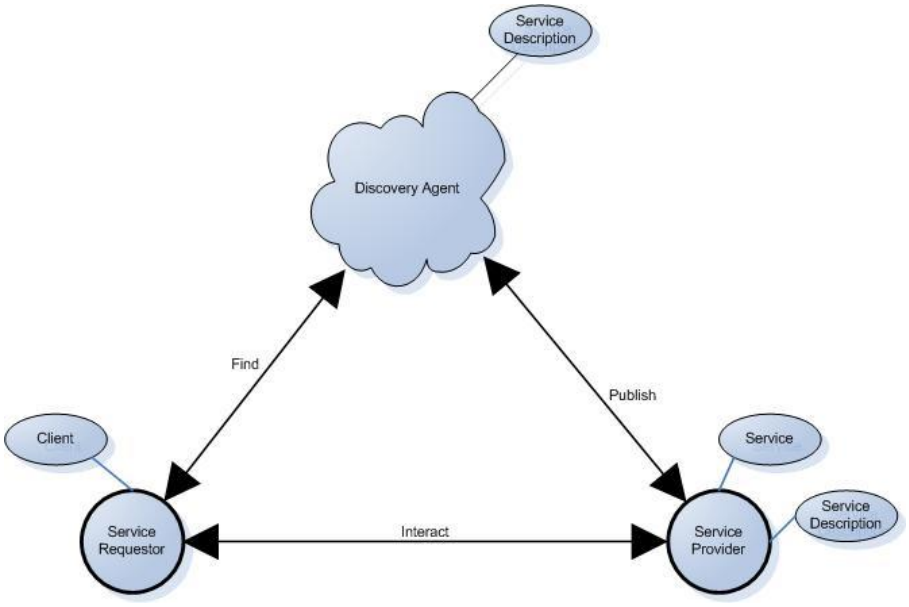

Figure 7: Overview of service oriented architecture

4.1.2 WSDL (Web Service Description Language)

WSDL defines the functionality of a web service. It defines the endpoint operation messages in a communication, this can contain either procedure oriented information or document oriented information. This definition is described abstractly.

The communication besides the WSDL definition is not described. It can be e.g. SOAP, MIME or HTTP. WSDL describes the network service in an XML grammar file. A basic grammar file contains a request and a response specifying which types that are expected in a request and a response message. Every element in a request/response is specified as a complex type, e.g.

```
<element name="parameterElement">
  <complexType>
       <all>
              <element name="outputParameter" type="string"/>
       </all>
  </complexType>
</element>
```

Where an element named `outputParameter` is mapped to the element `parameterElement`. Then a message has to be specified to describe the communication between the server and the client, this is done via an abstract definition, in this example. E.g.

```
<message name="GetOutputParameter">
   <part name="body" element="xsd1:parameterElement"/>
</message>
```

Then a message flow has to be described. E.g. if there are inputs or output parameters, in this example, there is no input parameter, but one output parameter. [WSDL]

4.1.3 SOAP

SOAP describes the exchange of typed information between decentralized hosts, this is done in XML. The SOAP structure also describes the content of a message sent over a decentralized network.

SOAP is fundamentally a stateless one way exchange paradigm, but if it takes advantage of underlying protocols it can crate more complex interactions, e.g. request/response, request/multiple response.

SOAP is mostly used with web services to transport silent data over the network. It can mainly be described as XML over HTTP. [SOAP]

4.1.4 XML Extensible Markup Language

XML is a very simple format mainly used to describe data on the web. Every XML-document is built as a structure, all element must have a root element. Under the root element there can be any element, and under the any element there can be any element, a simple example can look like this:

```
<root>
      <car>
            <color>Blue</color>
            <engine>V8</engine>
            <tires>4</tires>
      </car>
      <car>
            <color>Black</color>
            <engine>4 cylinder</engine>
            <tires>3</tires>
      </car>
</root>
```

This document describes two cars that have different specifications. [XML]

4.1.5 HTTP Hypertext Transfer Protocol

HTTP (Hypertext Transfer Protocol) is an application-level protocol for distributed, collaborative and hypermedia information systems. It's a generic, stateless protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. [HTTP]

4.2 BizTalk

BizTalk is a product from Microsoft and is known as a Business Process Manager (BPM). The aim of this product is that it enables companies to automate and optimize their business processes by several powerful integration tools. BizTalk 2004 has become a well known product and over 6000 organizations world wide uses it. It is fully integrated in Microsoft Visual Studio .NET 2003. [BT]

4.2.1. Messaging

BizTalk uses a technique called messaging. It is a popular technique to reduce coupling between applications. To explain what loose coupling is, take the example where an application A wants information from application B. B was developed before A and therefore it doesn't know the existence of A. When integrating the two applications, A has to make some assumptions about B. For example:

- B is running while A is making the call.
- B runs on the same platform as A.
- B runs on the same network.

The problem with making assumptions about B can be if, we want to replace application B with one that is better or if B is down while A needs it. Loose coupling is about to make less assumptions between applications when integrating them. The fewer assumptions made, the more loosely coupled they are.

Messaging relies on a middle-ware to transmit messages between the applications. In BizTalk this is made by message queues. It's a kind of channel that on it input side receives messages and on the output delivers the messages. The advantage with queues is that they are asynchronous. This means that the sending application can put messages in the queue while the application that receives the data can pull the data as soon as it is ready.

Messaging reduces coupling in the way that both application don't need to be running at the same time, the middleware is not coupled to a particular OS. Messaging is unaware of the programming language used when implementing the applications. Queues can easily be rerouted when application is moved or replaced by a new one.

4.2.2 Orchestrations

Another technique that makes BizTalk a quality product is the use of orchestrations which is the core part of Business Process Management (BPM). Orchestrations were invented to overcome the problems having point to point connections between applications. Instead of connecting each application to each other and the result quickly turns in to spaghetti-code, orchestration allows to make a message flow. A message flow defines a route that the message travels. This makes it easy to track each message and get a good manageable overview that enables even more functionality like Business Activity Monitoring (BAM). The message flow can consist of several parallel branches, decision makings and transformations of the messages etc

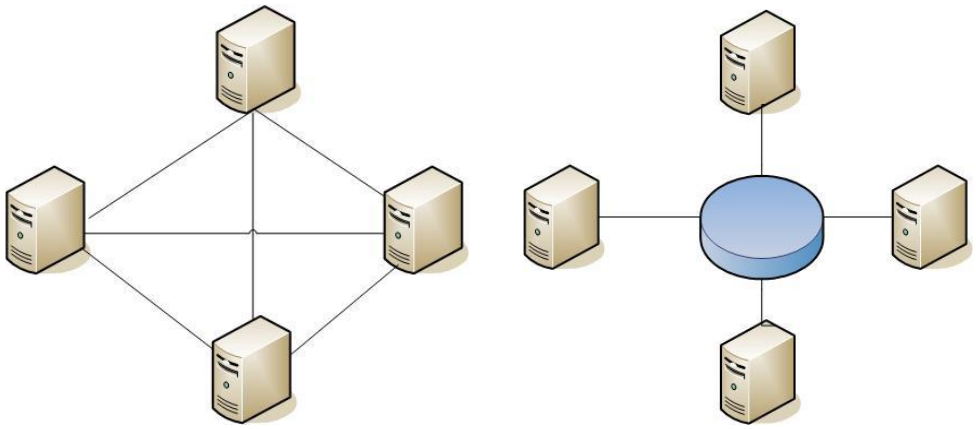

Figure 8: Spagetticode vs. Data Integration

The pictures above show an example how the links between four integrated systems are reduced when using orchestrations. When other systems is added it is a matter of configuration in the Orchestration instead of hardcode it into every other systems.

4.2.3. Publish Subscribe

Both Messaging and Orchestrations solves many problems but one problem that arises is that both Application A and B has to agree what type of queue to use. This is a shared piece of information and it increases the coupling between the systems. Publish/Subscribe is a solutions to this problem. No shared knowledge is needed besides an agreement of what type of message format that is used. The producer publishes the output using messaging to the publish/subscribe engine and the consumer subscribe itself to the engine using a filter.

BizTalk builds on the following concepts. In these the techniques described above is implemented.

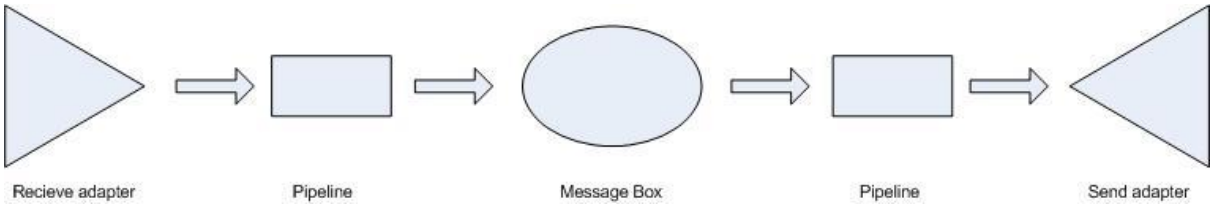

Figure 9: Receive adapter's → Pipeline → Message box → Pipeline → Send adapters

The receive adapter takes the input data and pushes it into BizTalk. The input data is then passed on to the Pipeline that transforms the bytes of data to XML. The message box receives the transformed input message and two possible actions can be made. Either forward it directly to an external location using one of its supported protocols (adapters) or forward it to an Orchestration. When the message has passed the Message box a pipeline is used to transform the XML back to bytes. The collection of bytes is then passed to the send adapter.

4.2.4. Describing data

BizTalk need to know what kind of data that is passed and the structure of it in order to make decisions based on the information in the data. Since BizTalk uses XML it is the XML schema that describes the structure of the message. Based on this information on the structure

BizTalk can make routing decisions, flow decisions in orchestrations, validate messages, transform messages between two different data structures etc.

4.2.5. Receive Adapters

In one end of a BizTalk application is the Receive adapter located that handles incoming traffic and on the other end is the Send adapter that handles outgoing traffic. Their task is to listen for incoming data and transfer it to a specified location. The receive adapters or receive ports are basically a combination of a transport protocol and properties that are specific for that protocol, for example where the messages is coming from. The property setting for protocol SOAP is then the http-address to the Web service.
Some examples of protocols that are supported by BizTalk are FILE, SOAP, FTP, HTTP, SQL etc. It is also possible to develop custom adapters.

4.2.6. Pipelines

When messages are received from, or sent from BizTalk server some pre-processing on data can occur. This is done with so called pipelines. Pipelines are implemented as .NET components and can easily be implemented using the "pipeline" item in a Visual Studio .NET BizTalk Server project. The meaning of pre-processing on data is that it makes it possible for example to decrypt messages, signing and verifying messages, adding additional data on the message and splitting the message into smaller parts before the data is passed on to the "Message Box".

4.2.7. Message box

As mentioned above there are two alternative tasks that the message box can do. One is to send the message to an Orchestration or directly to the send adapter. The message box is basically a couple of SQL Server tables. It is the heart of BizTalk and it has several functions.

- It provides a persistent and transactional storage for messages that is in some way processed by the server. This includes
  - Messages inside orchestration
  - Messages outside an orchestration that is waiting for to be processed.
  - Messages that an orchestration that are waiting to be transferred to an external location.
  - Messages waiting to be transferred to a tracking database
- It provides queuing
  - For Orchestrations when there are resources available to start a new orchestration instance, another message will be picked up from the current orchestration queue.
  - For outgoing transport protocols when a protocol adapter is ready to transfer new messages, it picks it up from its queue.
- It manages subscriptions
  - Some tables in the Message box contain all subscription related data.
- Routing happens inside the Message box.

In figure 9 all component described above is shown and how they interact with each other's.

4.3 Database

The database used in this system is Microsoft SQL server 2000, a relational database with primary query language SQL(Structured Query Language). This database is mostly used to small and medium sized databases, but is also suited for larger databases. [SQL]

4.3.1 Relational Database

SQL server is a relational database, which means it follows the terms of a relational model. The term refers to the structure of the data and the actual data.

A Relational database is a collection of relations (tables) and a set of other items (rules, keys, constraints etc.) to help structuring and organizing the data to force the database conforming to a set of requirements. [RD]

4.3.2 Stored procedures

A Stored procedure is executable code within a database. This code describes operations on the database and how to perform them. They are frequently used as an API to simplify and secure the database. Stored procedures are not always considered as a part of the relational database, mostly because they are not essential to the functioning of the database.

4.4 Graphical user interface

This section describes techniques used in the implementation of the Graphical User Interface.

4.4.1 CSS

Cascading Style Sheets enables to separate design and the context of a webpage. Things like font types, colors, etc can be modified. The settings are written in a separate file which enables changing the looks of multiple web pages at once. This also makes it easier to change the looks in the future. [CSS]

4.4.2 JavaScript

JavaScript is a programming language frequently used in web pages. It is used for field validation, events, manipulation on the content of the web page, etc. There are many limitation of this script language due to security reasons. For example it is not allowed to read and write files on the client's computer or make database calls. [JS]

4.4.3 XML

XML (Extensive Markup Language) is used to structure data. In a very simple way structured data can be transformed into XML which makes it easy to read in a programmatic way. XML is very similar to HTML but XML uses "<" and ">" to encapsulate data. The pros of using a text based protocol are that is much easier to understand and extend. The cons of the XML protocol is that it takes more space to specify data, but now days the requirements of small and efficient protocols are not that high because of higher computer power.
[XML2]

4.4.4 Integrated Windows Authentication

Integrated Windows authentication is used to reduce the number of logins needed to access for example a webpage in a Windows network. This is called *single sign on* and is a function supported by Microsoft Internet Information Server. It enables the web browser to

automatically log on to a webpage with the information about the user that is logged on to the current computer. [IWA]

5. Implementation

This chapter describes the implementation part of the project. Some of the information is confidential and therefore not explained in detail.

5.1 Preliminaries

The implementation part of this thesis can be divided into two large tasks, a Pre-study and the main task of constructing the system. The workflow is described below.

- *Pre-study*: Can be divided into several tasks.
  - Find all systems that the new system is dependent on.
  - Obtain how to retrieve information from the dependent systems.
  - Decide which functionality the GUI will have.
  - Study the implementation guidelines used at LSH.
- *Main task*: The implementation of the system
  - Construct a prototype for evaluation.
    - Collect information from the different business systems.
    - Create web services that will retrieve information from the different data sources.
    - Create a data warehouse that will hold this information.
    - Use an integration engine to collect information from the different data sources.
    - Create a GUI that will show the collected information.
  - Evaluate prototype and construct the final system.
    - Make changes to the GUI.
    - Check functionality of the system/add desired functionality.
    - Document the constructed product.

5.2 Web Services

Each system that will be monitored will have a web service that will collect the desired information. The web service will act as a small API against the data source. The web service will support information retrieval in an operating system independent manner (XML over SOAP).

The procedure when retrieving information from a business system is that it should build on the three layer structure specified earlier. The first layer is a public layer accessible for all users that will support information retrieval from the system in XML. The second layer is a business layer that will convert the data retrieved from the data access layer. The data access layer performs the communication with the actual data source.

5.3 BizTalk

The task that BizTalk is performing in this system is to pass on messages between web services connected to different business systems. On one side it receives data from web services connected to different status/information systems and on the other side it passes on the messages to the web service that handles updates of the *eboard* database.

Since BizTalk is event based it has to be triggered in some way. This is easiest to do with some kind of start file. BizTalk is listening on a port connected to a folder on the server. When the start file is put in the folder BizTalk grabs hold of it and the orchestration is triggered.

Functional requirements

The functional requirements of the BizTalk application is presented below

- Pass on messages
- Input and output messages are in XML format.
- Handles the update interval for each system that it communicates with
- The goal is to make this process self maintaining, error reporting is handled from the web services.

5.4 The Database

The database that's going to hold the collected data retrieved from the different web services has two main relations. The first one is Eboard which is a status board that has a number of attributes, and also has links to department, mail lists, resources and employees that the board consists of. The relations mail lists and department also consists of employees. These groups are regularly updated from the web services in the system. This to allow dynamic updates to the status boards without changing the links to the members of a board. The other main relation is an employee, this relation has links to the different status systems and to the Eboard, either via another relation (department or mail list) or directly.

5.4.1 Requirements

The requirements of the database were the following.

- One user can have several meetings (0 - ∞)
- One user can have a(0-1) telephone status
- One User can have multiple(0 - ∞) entries for check in status
- A user belongs(1) to a department
- A User can belong to multiple(0 - ∞) mail lists
- A board can have multiple(0 - ∞), employees, mail lists, departments and resources
- A user may add several(0 - ∞) boards as favorites
- A resource booking can have several(0 - ∞) bookings on a resource
- A mail list can have one or more(0 - ∞) sub mail lists
- A department can have one or more (0 - ∞) sub departments.

According to the above specification and requirements the constructed ER-diagram looks like this:

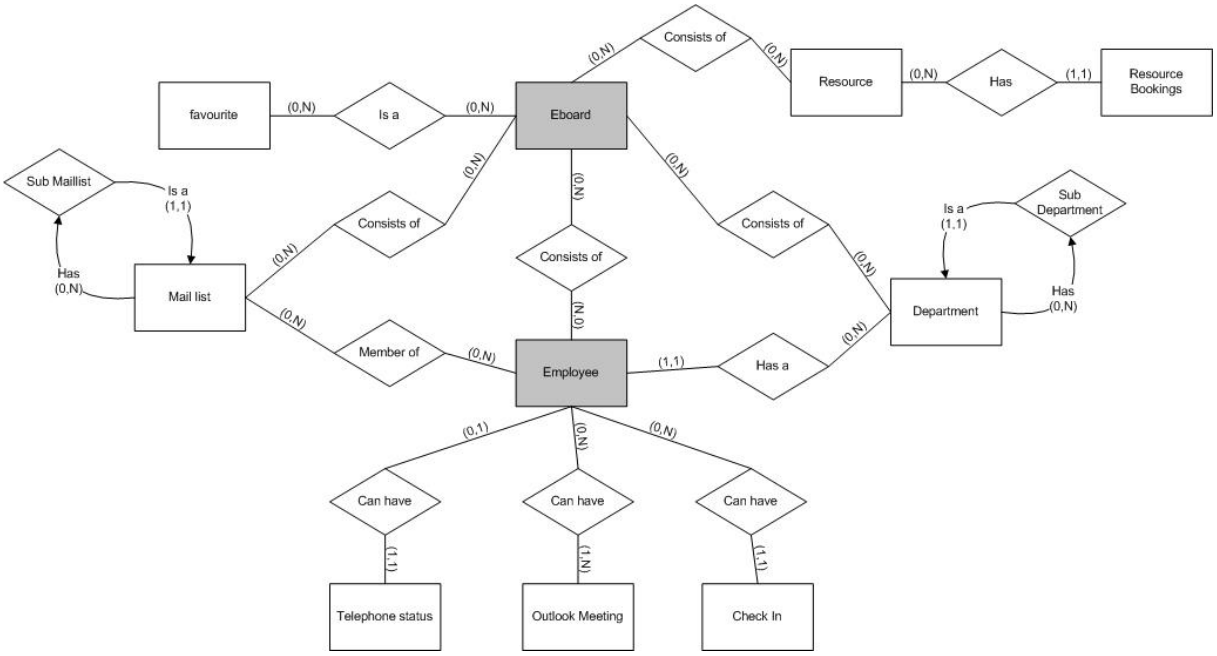

Figure 10: ER diagram of data warehouse.

There are two main relations, the employee and the Eboard relation. The employee can be a member of an Eboard, a department and a mail list. It can also have a telephone, a calendar and a check in account attached to itself. The Eboard can have mail lists, departments and employees attached; it also has links to resources and favorites.

5.5 Graphical user interface

The graphical user interface of the system was a very important task to solve. According to the pre study of future users, they can be divided into two groups.

- Advanced users
- Novice user

Since it not only advanced users like administrators that will use the system, the interface must be attractive to ordinary users as well. Additional to the functional requirements, information from future users were gathered to decide how the interface would look like. The conclusion that could be drawn from all information gathered was that the user interface must both suit the needs for advanced users as well as novice users. It should be easy for a novice user to create a status board with default values and at the same time support advanced users the ability to adjust the settings of the status board.

An important task in the implementation was how to prensent information for a normal user and while using the board as a presentation screen outside a department. Therefore two versions were made, one for internal use (user who uses a browser to get a status for a specific user or a board) and one for external use (full screen board that updates regularly).

To make decisions about the design a prototype were sent out for testing. One was sent out in the beginning of the user interface implementation phase to get reflections about the design and additional functionality. After considering all feedbacks from users, changes were implemented and a final prototype was sent out for the last test and the last calibration was made. A total of about 40 people were a part of the evaluation.

6. Result

This chapter presents the result of the implementation of the practical part of the Master Thesis.

6.1 Web services

On top of each business system there is a web service that act's as a small API. This API supports retrieval of information in XML over the SOAP protocol. The web services are built on the guidelines specified at LSH, a three layered structure. To make the service platform independent the service pass messages via SOAP and XML. The Global As View approach specified earlier is used to map collected data to the global database schema.

6.2 Database

This database has two web services connected to it, the first web service is used from BizTalk updating the eboard database, and the other one is used to retrieve information from the database (used by the GUI).

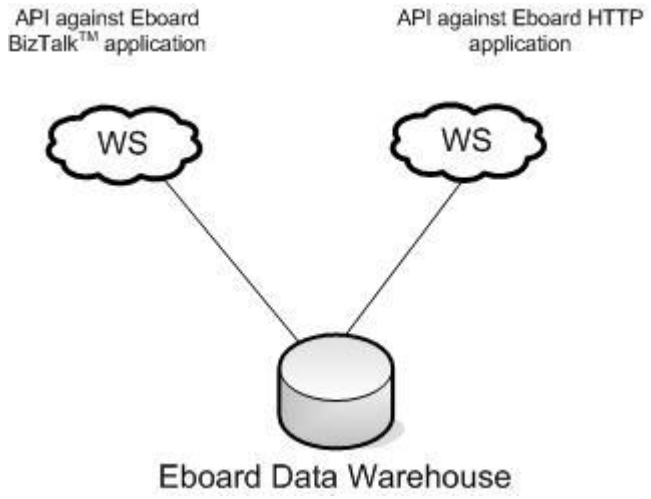

Figure 11: API's aginst the Eboard Data Warehouse

6.2.1 Design

The designed database can be seen below, the centre of the database is on the employee table, every employee is linked to the table's *eboard*, *mailgroup*, *department*, *telephonestatus*, *checkinstatus* and *calendar*. The other big table eboard is linked to the GUI. This table uses three help tables that are used to link employees, departments and mail lists to the eboard. The only table left is then favourites; this table is used for GUI users who want to add another user's eboard as a favourite.

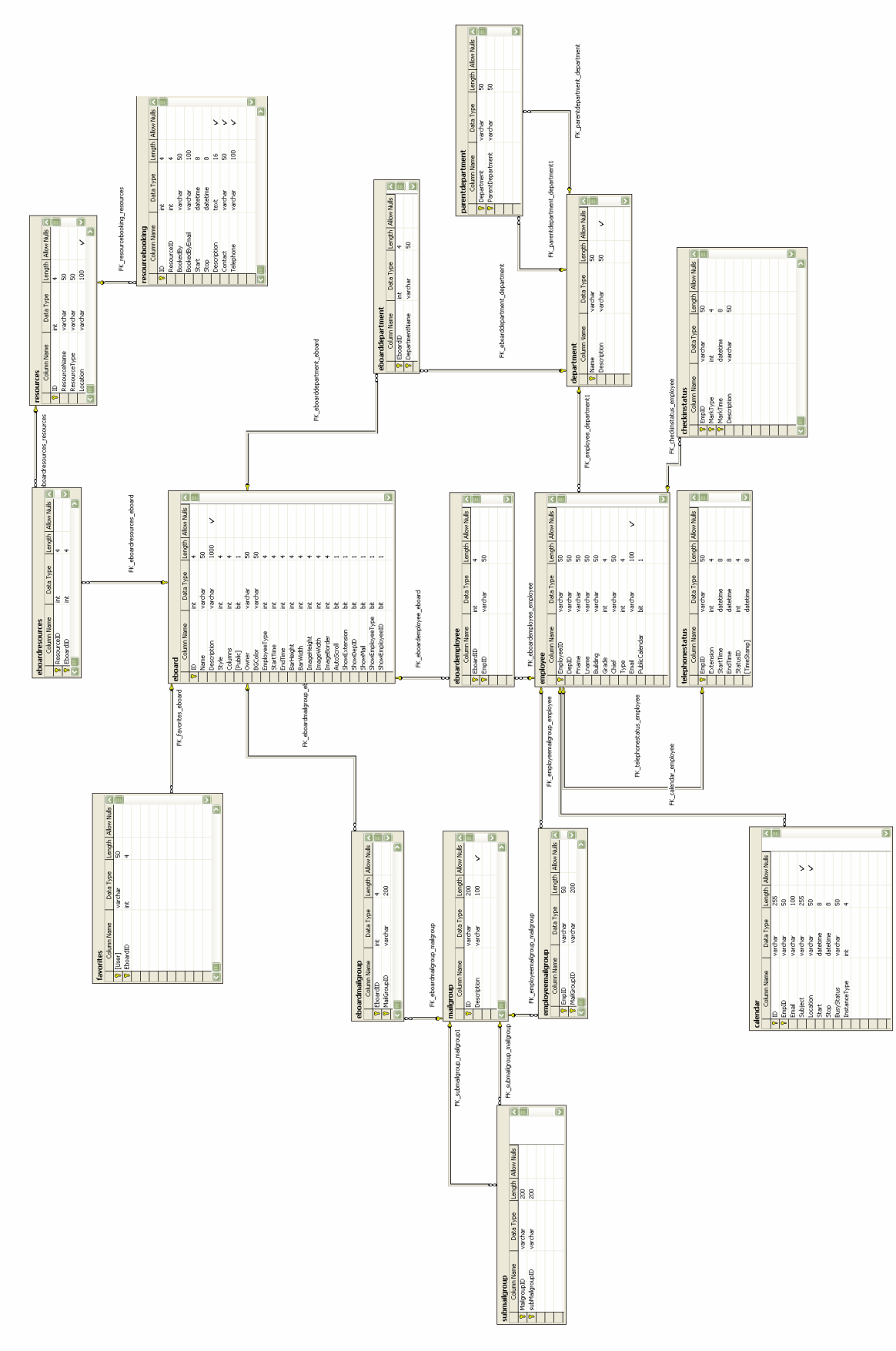

Figure 12: Constructed Database

6.2.2 Tables

Employee

This table is used for storing employee information that is collected from the company personal registry. Each employee has a unique employee number which is used for linking to other tables in the system.

Eboard

This is the main table for the board. Table contains settings for the eboard e.g. Name, Description, image settings, bar settings. Eboards can have employees, department and mail groups linked to it.

Telephonestatus

Table contains Information about user telephone status, only one record per user.

Calendar

Table contains information about user meetings, user can have multiple meetings during a day. This information is collected from the company exchange system.

Checkinstatus

Table contains information about check in status of a user. A user can have multiple records during a day. This information is collected from company check in system.

Mailgroup

Table is used for storing mail group in the company AD directory.

SubMailgroup

Table is used to handle the relation that a mail group can consist of several sub mail groups.

Department

Table is used for storing department and department description.

ParentDepartment

Table is used to handle the relation that a department can have several sub departments.

Resource

Table is used for storing information about company resources that are available for booking.

Resourcebooking

Table is used for storing bookings on the company resources.

EboardResources

Resources linked to an eboard

EboardDepartment

Departments linked to an eboard

EboardEmployee

Employees linked to an eboard.

EboardMailgroup

Mail groups linked to an eboard.

EmployeeMailgroup

Employees linked to an eboard.

Favourites

A system user linked to an existing eboard.

6.3 User interface

The user interface has been designed to both support expert and novice users in their work at Land Systems Hägglunds. An Employee has the ability to create their own status boards or subscribe to one created by other employees. It is also possible to search for individual employees for their current status without doing the procedure of creating a status board. The user interface is divided into several web pages. Each page is designed by a global theme that is used on all pages in the system.

In order to support employees in creating their own status boards there has to be some kind of login mechanism to separate the employees. This was achieved by using *Integrated Windows Authentication*. Therefore there is no need for a user to log on to this system beyond the log in sequence on the computer they are working at.

6.3.1 Main Menu

The main menu is the main page of the system. All functionality is presented in a top down list as showed in the figure below.

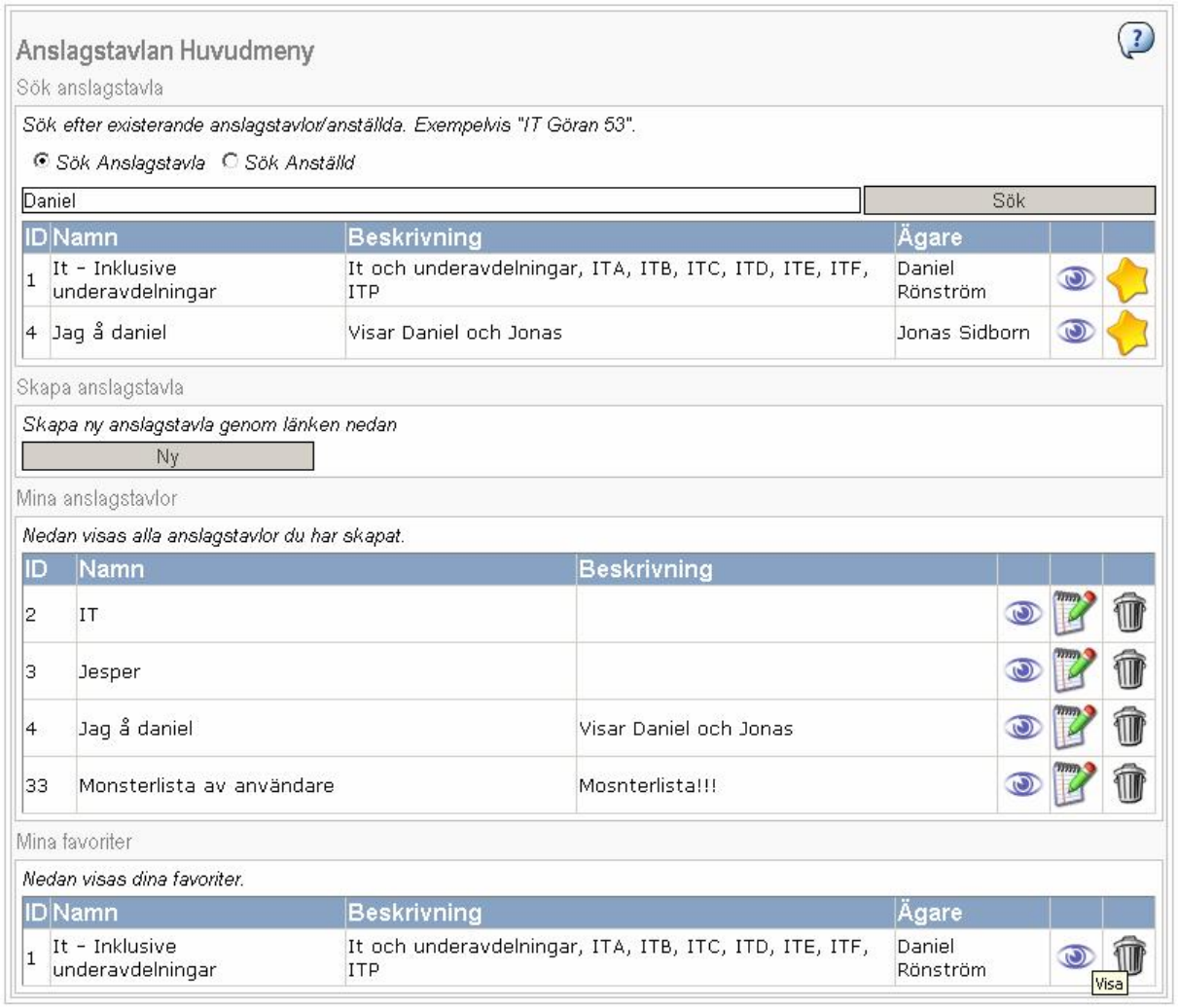

Figure 13: A snapshot of the main page of the system

At the top of this page an icon representing FAQ (Frequently Asked Questions) is presented. This icon is also presented on all other pages in the system. It provides help to users if problem or questions occur.

The page is further divided into four sections. The first section presents the search functionality of the system. A user can search for both status boards or for an individual employee. In order to view the search result the user has to click on the *eye* (view button) and a page is opened presenting the requested information. If the status board was interesting also in the future, the employee has the ability to subscribe on it by pressing the *star*.

The second section presents a link to the procedure of creating a status board. How this is done is described later. The third section of the page presents the status boards created by the current logged on user. The logged on user has full rights on these status boards since he/she owns them. Actions like *view*, edit and *delete* are supported. Other users that subscribe on these status boards automatically sees changes that's been made on the original. The last section lists all status boards that the current user subscribes on. Actions that are supported are *view* and *delete* (It is the favorite that is deleted, not the status board).

6.3.2 Show employee status

There are two different pages that status information of employees can be viewed. One is presented in figure 12 and the other is presented in figure 13. Figure 12 presents a single employee with some additional information that is not presented in figure 13, that presents x number of employees. This page can be accessed from two places, directly from the search section on the main page, or by clicking on an employee on a status board as figure 13 shows.

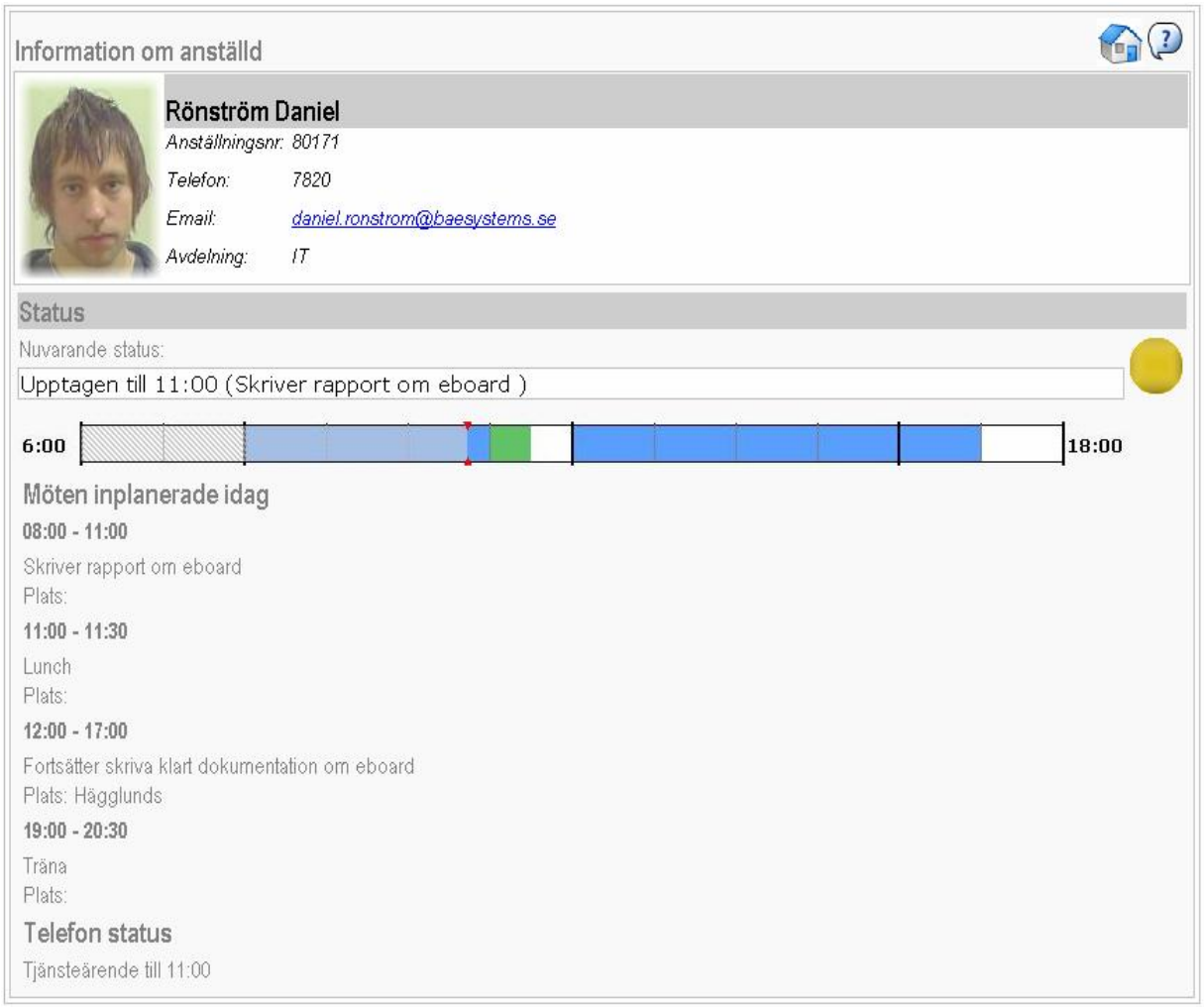

Figure 14: A snapshot of the page showing status information about an employee.

The page that is presented above is divided into 3 sections. In the top some quick links is presented. The *house*-button as a link to the main page and the *FAQ*-button is also presented there as described above. The second section presents the employee that has been selected with additional information. The last section presents the status of the employee during the current day. Different status lights (upper right corner in section 3) shows current status of the employee. *Yellow* means that the employee is busy, *Green* the user is available and *Red* that the employee is not in the office. Below a bar that shows the status during the day is presented. The status colors here is the same as in Outlook Calendar. There is also additional text information about the activity during the day.

6.3.3 Status board

The picture below shows the second page mentioned that shows status of an employee. This view is the actual status board of employees that was the main task of the system. In the top

of this page there is some quick buttons. The left button navigates to the main page. The middle one is the full screen button. It shows the same information but the layout is adapted to the screen resolution and all unnecessarily information is not shown. The last button shows the FAQ. The name of the eboard and a description is shown in the top of the page. Under these the members of the status board is presented in alphabetical order.

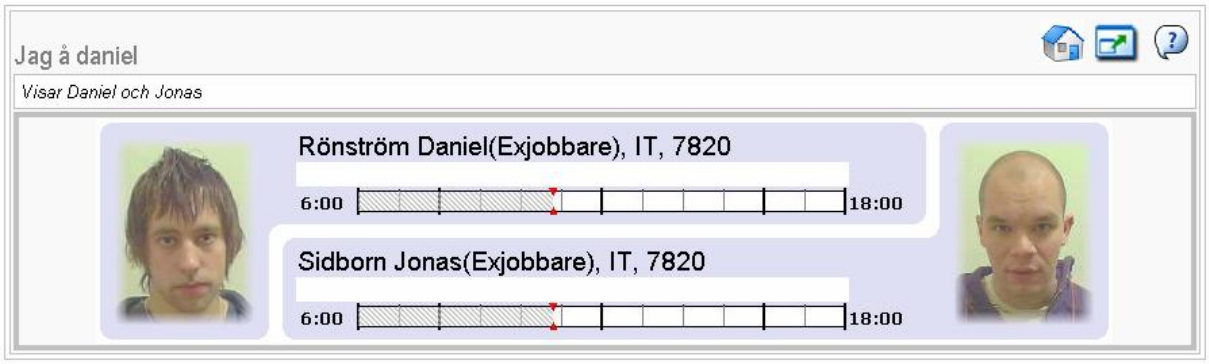

Figure 15: Snapshot showing a small status board.

6.3.4 Create status board

The task of creating a status board is a sequence of four steps. The first step is to select members of the status board. There is three possible ways of doing this. First, you can select a department. All employees in the selected department will be shown on the status board. Secondly, you can select a mail group and all members of this are presented. One big advantage with the first two approaches is that they add and delete members of the eboard automatically if changes occur in them. The last approach is to add individuals one at a time. They will be shown even if they change department or mail groups. But if they quit the job they will not be shown.

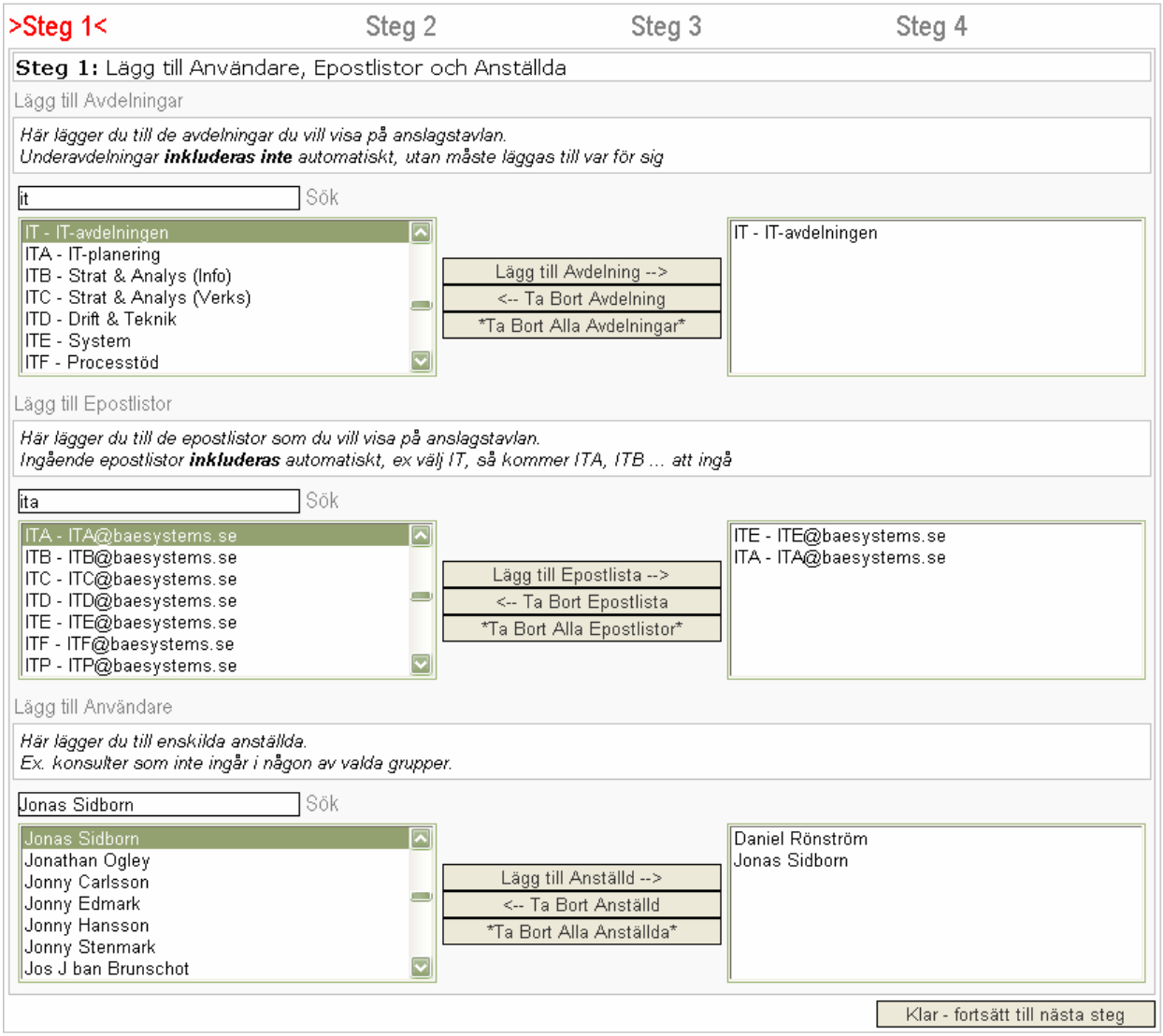

Figure 16: Screenshot of step 1 in the sequence of creating a status board.

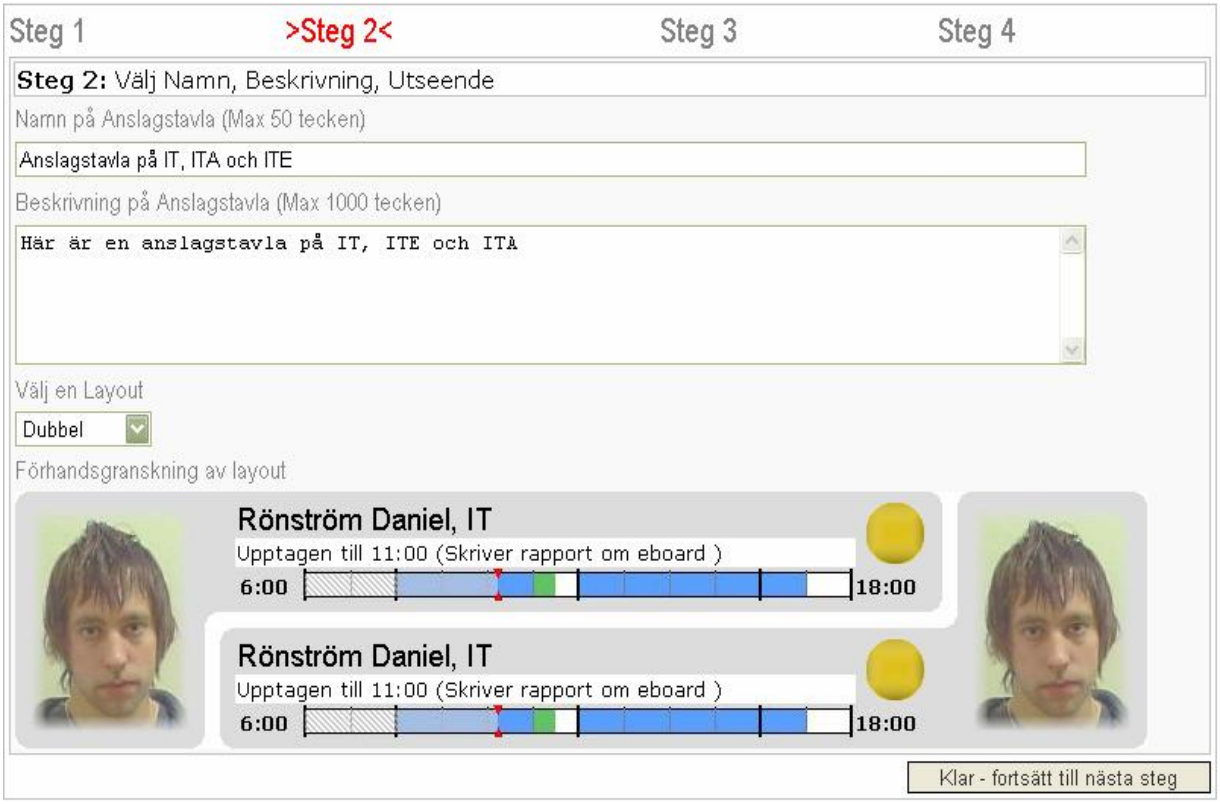

Figure 17: Screenshot of step 2 in the sequence of creating a status board.

The second step is shown in the figure above. Here the name and a description on the status board can be given. These fields are used in the search functionality on the main page. It gives the owner possibility to specify the content of the eboard in detail. In the bottom of the page the creator can choose a layout type. In the current version there are four layout types, *Extra Small*, *Small*, *Medium* and *Double*. Depending on how many members are going to be displayed on the eboard.

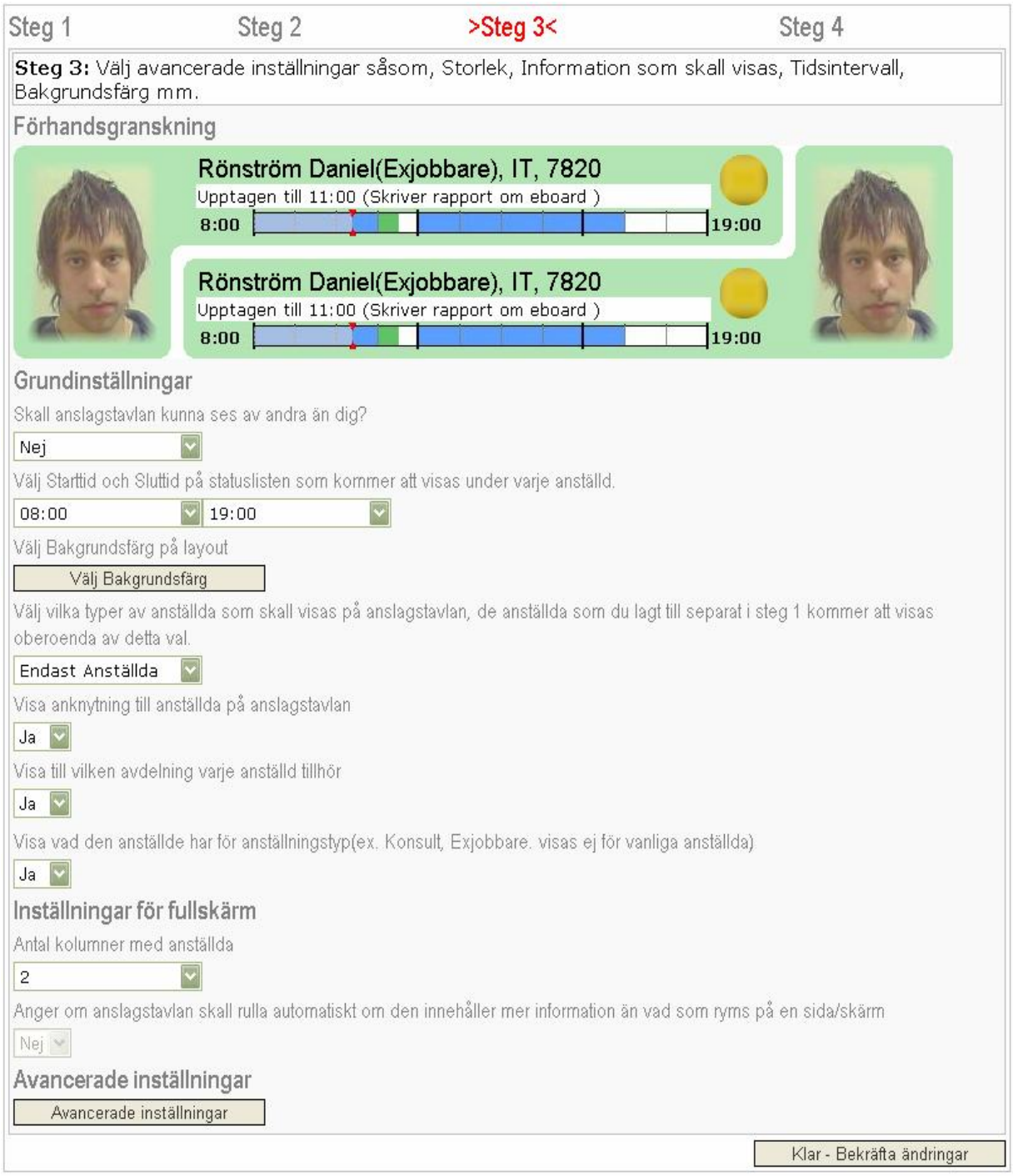

Figure 18: Screenshot of step 3 in the sequence of creating a status board.

The third step contains advanced settings of the eboard. the user wants, he/she can set colours, start and end time of the status bar, different type of additional information about the employees, number of columns for full screen presentation and width/height of images and status bar.

The last step that as shown in the figure below, the status board is saved. It provides some basic functionality of the eboard that has been created. The creator can view the eboard in standard mode or in full screen mode. It also provides links if the creator wants to make changes to it or directly navigate to the main page.

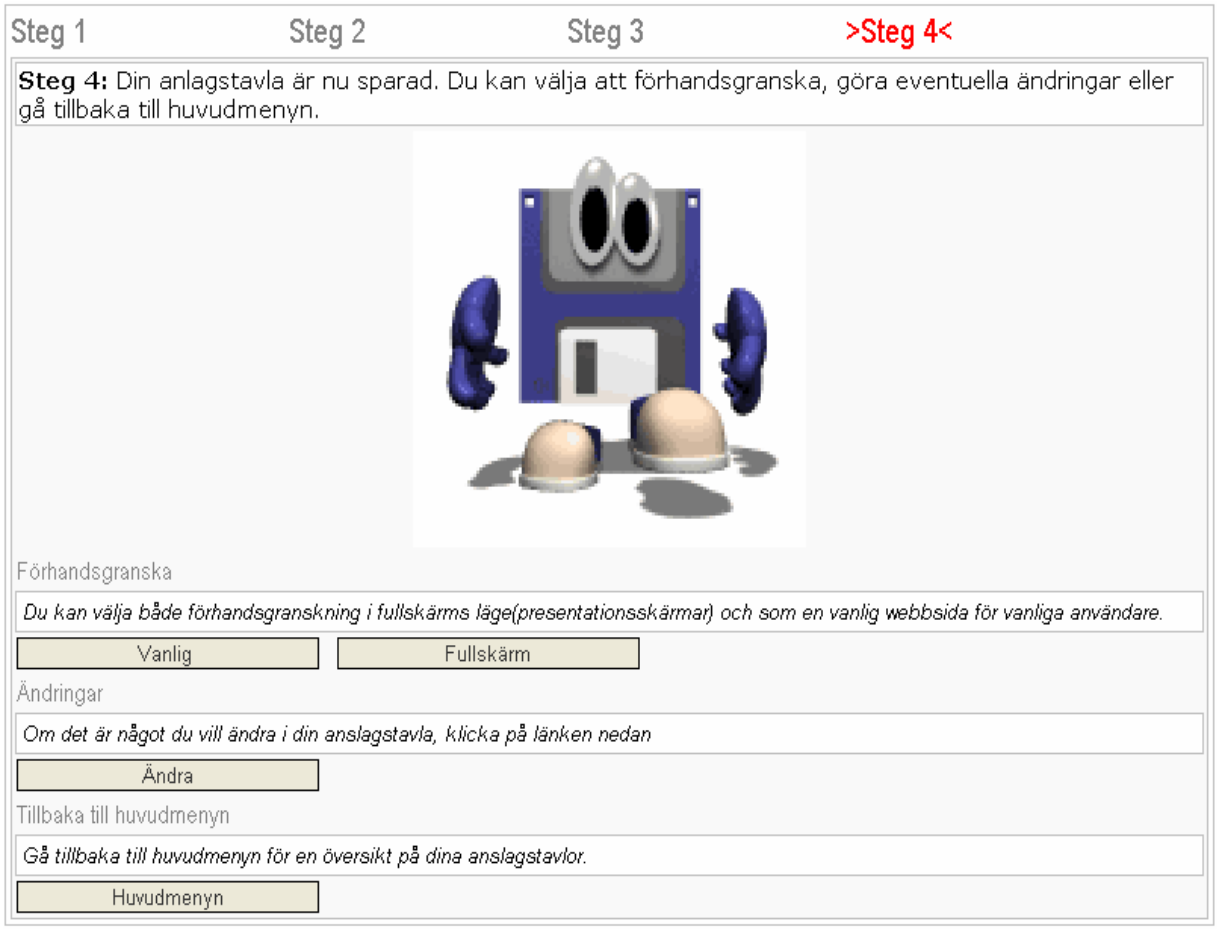

Figure 19: Screenshot of step 4 in the sequence of creating a status board.

6.4 BizTalk

The BizTalk orchestration implementation is divided into 4 parts. Each part is running in a separate thread. The different branches are listed below:

- Telephone branch
- Check In branch
- Calendar branch
- Synchronization branch

The orchestration in figure 18 and 19 shows the message flow in the BizTalk application. To start the flow, the application listens on a file port that is connected to a catalog on the BizTalk server. BizTalk grabs hold of the files that is dropped there and checks if it has the start signature that is required to start the flow. When the right signature has been captured the message flow starts. The flow is branched into four threads as described above. Each of the branches handles the communication with the different business systems that integrates the data and stores it in the new eboard database. The type of integration that is used is a so-called *Copy Value* approach [*chapter 3.3*]. Figure 19 shows three of the branches. One communicates with the telephone status system, one with the check in system and the last one communicate with the Outlook calendar system. Each one of these threads is built on the same idea. First, each of the branches send a request the web service that is connected to each of the business systems. The web service reacts on the request and gathers the information and sends back a response message. If everything went well an Xml file with changes in data is returned. The BizTalk application grabs on to the response message and continues down the flow. Next halt in the flow is to pass on the message to the web service that is connected to the eboard database. The web service updates the changes according to the data that has been send. If no exception was caught it send back a response message to tell the BizTalk orchestration that it went ok. If something didn't work as expected in the communication with the web services it throws an exception that is caught in each thread and is reported to a log file. In the bottom of each flow there is a delay that is configurable for each thread. After the delay time has expired it executes the flow again and continues in eternity.

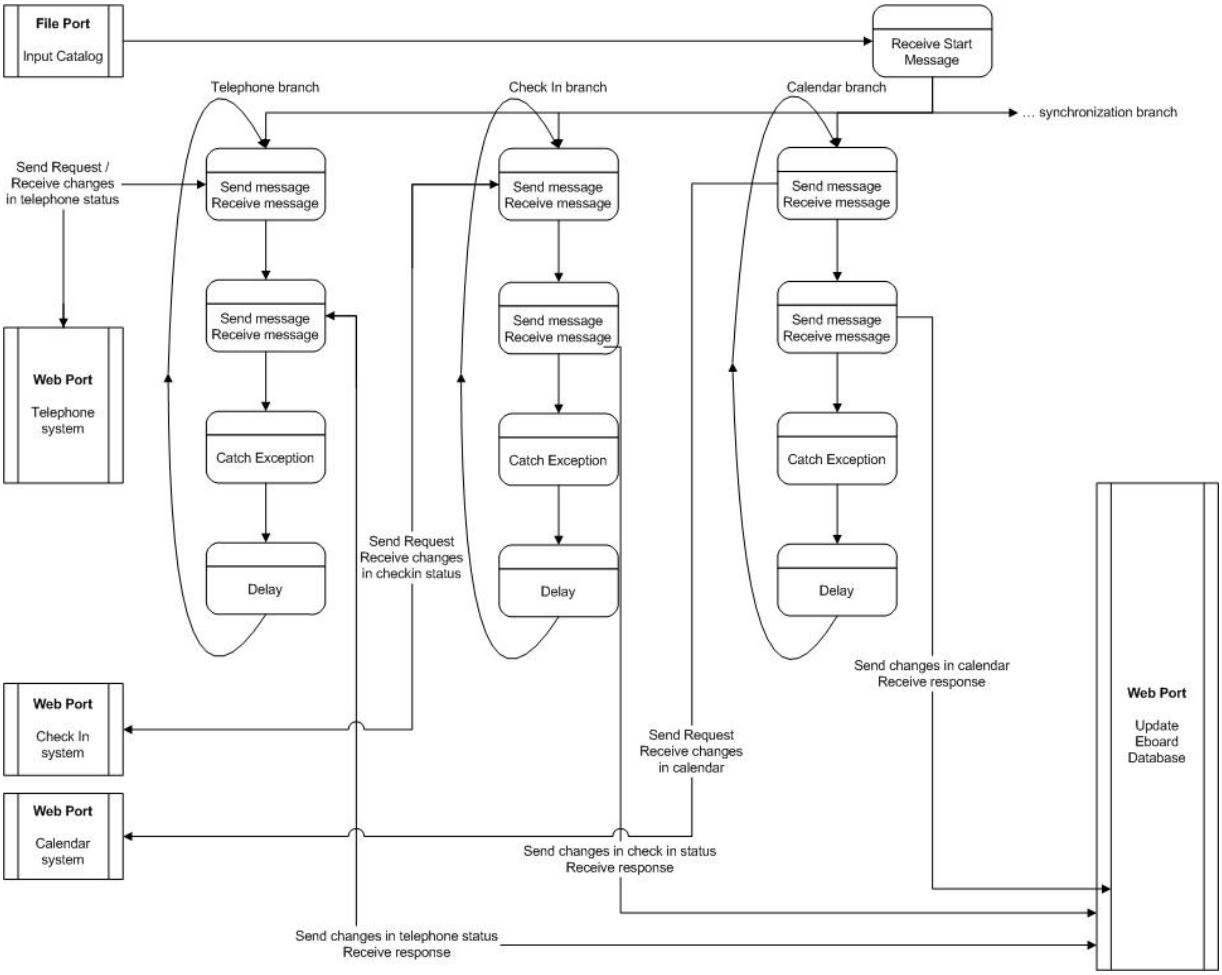

Figure 20: The three branches Telephone, Check in and Calendar is shown.

Figure 19 shows the branch that synchronizes all other data that the new system is depended on. The orchestration flow works with the same idea as the other branches. The difference is that it gathers information from more than one business system. First it gets employee information that is the base information that the new system is dependent on. After that it gets the employees calendar and in the last step mail groups that the employee is a member of. Since it don't occur changes in the data as often as the other systems the update interval has been set to be updated once a day.

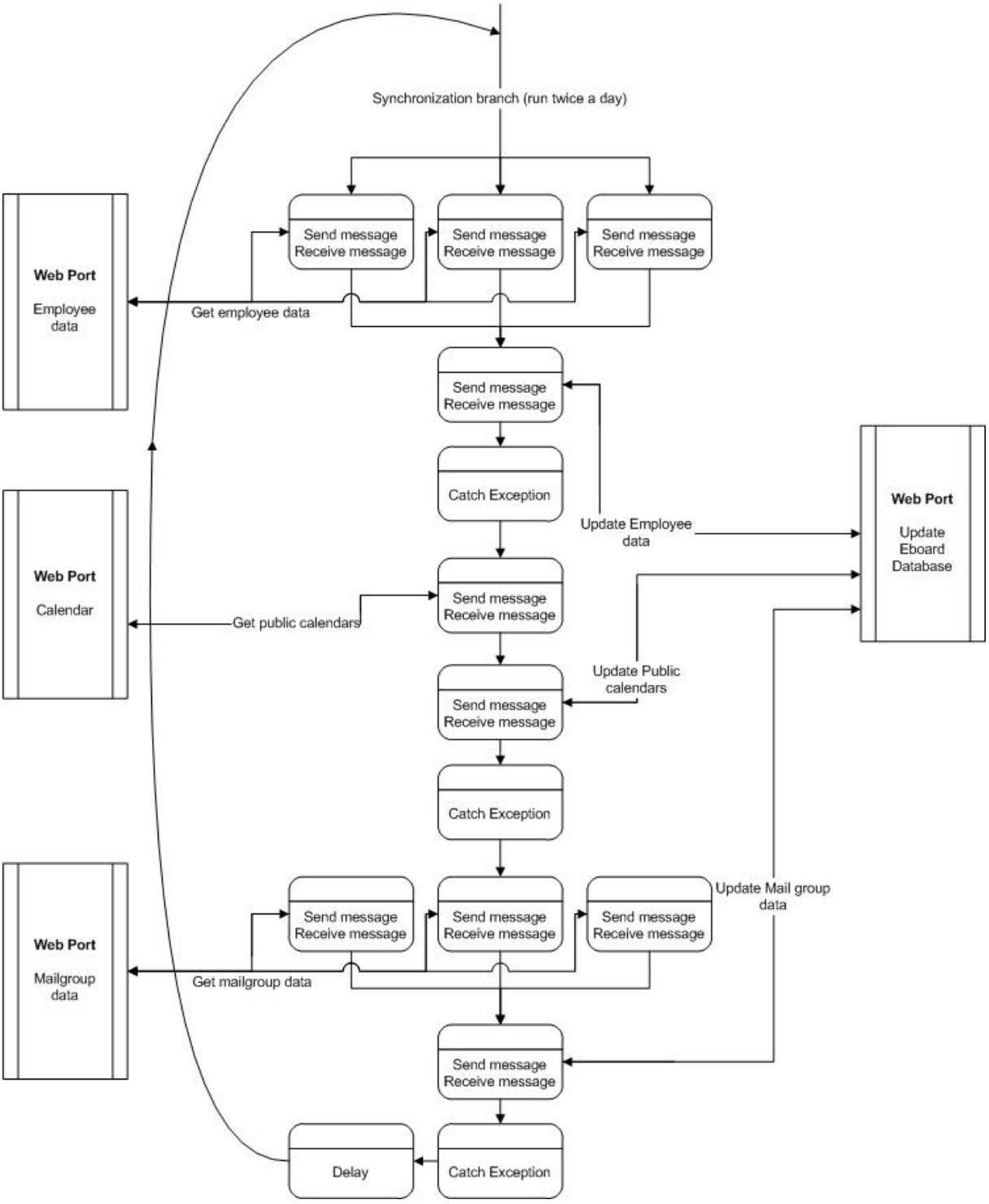

Figure 21: The branch for synchronization is shown.

7. Discussion & Conclusion

7.1 Discussion

When integrating business systems there is a lot of research on how to obtain the best and most efficient result. The choices of algorithms are big; however most of them are very easy in theory but practically impossible because of system instability, heterogeneity and distribution. Most algorithms assume system stability which is impossible in the real world. The best approach today that handles this the best way is BAV (Both As View), but this algorithm is very time consuming to implement, so the best choice besides this algorithm is the LAV or GAV approach.

A big problem when construction a system that will integrate data from different data systems is that it has to retrieve data from the actual sources in some way. Bigger business systems usually have API's supporting data retrieval in some platform independent language, but some systems you have to build this module your self, and if this system is changed you have to change this module for the system to work. In some cases there is also impossible to just change this module, and then the whole application has to be changed for the system to work, so the integration is very dependent on the underlying business systems.

A big question when integrating data is the availability of data compared to the performance. As discussed earlier there are four forms of data integration, if you copy values, the performance increases but there may be lack of information in some cases, if you instead use shared values the information is always up to date, but there is lack in performance.

7.2. Conclusions

As a summary on the work on this master thesis at LSH we can say that it was an instructive experience. One thing that didn't work as planned was our time schedule. In the beginning of the project it was meant to work in parallel with the practical implementation and the theoretical report. But different aspects occurred in the beginning of the work, particularly our in depth study didn't get approved and therefore the time schedule was dislocated. In the mean time the practical work went on and not until a couple of weeks before the twenty weeks was finished our in depth study got approved. It was just to take the consequences and work on the report during the summer holiday. But instead we had time to develop a fully operating system which had been difficult to do if we had work in parallel with the report.

The actual system is a quick and effective system that works as it was supposed to do according to the requirement stated in the beginning of the project. This procedure reduces the employee workload affected by writing his/hers status on multiple locations. One other opportunity is that employees can watch status of employees at a distance and therefore make their work at LSH more effective.

The BizTalk implementation makes the system flow in big parts self maintaining. But it requires a supervisor that gets fault messages during the integration process in order to act on faults that can be critical for the application to work properly.

7.3 Current Status of the system

The system is in use in a development environment at the moment waiting to be released in target environment. The system doesn't have a manager at the moment but it is to be decided "soon" according to our supervisor. Some hardware and software changes were to be made during our work at Land Systems Hägglunds and the web services affected during these changes needs to be fixed in order for the system to work correctly.

During the development of the system we made some smaller evaluations of the system where we received a lot of response from different employees. This information was then used to improve the quality of the constructed system (GUI part).

7.4 Performance

The performance of the system is devided into four parts that is described below.

7.4.1 Status retrieving via Web services

There are five Web Services that retrieves information from different data sources. The performance of these is very good except the Web Service connected to Exchange. This due to that it has to ask the status for each employee individually via a single HTTP request over a WebDav API. The WebDav doesn't support multiuser queries.

7.4.2 Biztalk application

This application just forwards information to the constructed data warehouse. Each Web service connected to the application is running on a separate thread making the application both secure and tolerant against delays and corrupt data. There is no data transformation in this application so the limit is on the Ethernet interface.

7.4.3 GUI

Running the webserver with the application on a small desktop computer was no problem at all when loading all employees at the company in one single eboard. Future improvements might be to introduce a better image-server that holds temporary generated images so that the images not have to be regenerated at runtime.

7.4.4 Data warehouse

All information from the database is retrieved via stored procedures. There is no direct connection to the database, all information is retrieved via Web Service API's that supports GUI functionality and Information Update.

7.5 Acknowledgement

We would like to thank our supervisor Sofie Borg for all help during the master thesis work at LSH. We would also like to thank Gunno Hamberg and the rest of the department for helping us constructing this system, giving us valuable feedback and ideas.

Also big thanks to Jonas Forsberg helping us with the BizTalk implementation.

Thanks to Michael Minock helping us with the report and the in depth study.

8. References

[AAQE] Zachary G. Ives, Daniela Florescu, Marc Friedman and Alon Levy. An Adaptive Query Execution System for Data Integration

[BAE] Bae systems homepage.
www.baesystems.se

[BAV] Peter McBrien and Alexandra poulovassilis. Defining Peer-to-Peer Data Integration using Both as View Rules.

[BIS] Peter McBrien and Alexandra poulovassilis. Data Integration by Bi-Directional Schema transformation Rules.

[BT] BizTalk Server
http://www.microsoft.com/biztalk/default.mspx

[CSS] Cascading Style Sheets
http://www.w3.org/Style/CSS/

[DI] Data Integration – What is it Anyway?
www.dmreview.com/editorial/dmreview/print_action.cfm?articleId=8232

[DIW] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi and Riccardo Rosatti. Data Integration in Data Warehousing.

[DWD] Data Warehouse definition.
www.oranz.co.uk/glossary_text.htm

[EQR] Igor Tatarinov, Alon Halevy. Efficient Query Reformulation in Peer Data Management Systems

[GJM94] A. Gupta, H.V. Jagadish, I.S.Mumick. Data integration using self-maintainable views.

[GMS] A. Gupta, I.S Mumick, V.S Subrahmanian. Maintaining Views Incrementaly. 1993, pages 157-167.

[HD92] J.V Harrison, S. Dietrich. Maintenance of Materialized Views in a Deductive Database: An Update Propagation approach. 1992

[HTTP] HTTP Hypertext Transfer Protocol
http://www.w3.org/Protocols/Specs.html

[IWA] Integrated Windows Authentication
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/523ae943-5e6a-4200-9103-9808baa00157.mspx?mfr=true

[JS] JavaScript
http://en.wikipedia.org/wiki/JavaScript

[Kuc91] V. Kuchenhoff. On the Efficient Computation of the Difference Between Consecutive Database States. 1991

[LBT] Alon Y. Levy. Logic Based Techniques in Data Integration.

[LGAV1] Andrei Lopatenk. Query Answering Under Exact View Assumption in Local As View Data Integration System.

[MMDI] Robert McCann, Bedoor AlShebli, Quoc Le, Hoa Nguyen, Long Vu, AnHai Doan. Mapping Maintenance for Data Integration Systems.

[MMV] Ashish Gupta, Inderpal Singh Mumick, Maintenance of Materialized Views: Problems, Techiques, and Applications

[MV] Materialized View - Definition.
http://orafaq.com/glossary/faqglosm.htm

[RD] Relational database
http://en.wikipedia.org/wiki/Relational_database

[SOAP] Simple Object Access Protocol (SOAP) 1.1
http://www.w3.org/TR/soap/

[SQL] Microsoft SQL Server
http://en.wikipedia.org/wiki/Microsoft_SQL_Server

[OU92] T. Urpi, A. Olive A Method for Change Computation in Deductive Databases. 1992

[WS1] Web Services
http://www.w3.org/2002/ws/

[WS2] Web Services and the Microsoft Platform
http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp

[WS3] Web Services Glossary
http://www.w3.org/TR/ws-gloss/

[WSDL] Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[XML] Extensible Markup Language (XML)
http://www.w3.org/XML/

[XML2] XML i 10 punkter
http://www.w3c.se/resources/office/translations/XML-in-10-points_sw.html