Application of Semantic Matching in Enterprise Application Integration

Bas van Gils
b.vangils@kub.nl
s184314

Tilburg, 15th February 2002

For my grandfather, A. C. M. van Gils

Preface

A journey is easier when you do not have to travel alone. This is one of the things I learned during the last half year, while writing this thesis. During the project, I received help –in many different forms– from a lot of people to whom I wish to express my gratitude.

First of all I'd like to thank my girlfriend Eva who supported me in all possible ways. With patience, enthusiasm and calmth she kept me on track. My parents always stimulated me to do 'enjoyable things' and always encouraged me to make my own choices. Their enthusiasm and support for anything I undertook helped me to understand the things that matter in life.

As for the rest of the family, I feel gratitude for their interest in my studies, and the many cheerful talks we've had over the last few years. I especially want to thank Hans and Lucie van Gils for the long talks, loud music, and for being who they are, and Jos Bertens for the many nice dinners we've had together. Inge and Caro Beekman I thank for helping me getting settled in Deventer and their cheerful view on life. I also thank my late grandfather A.C.M. van Gils who is in my thoughts always.

The second group of people who I wish to thank are my colleagues at Tilburg University / CentER-AR. I want to thank my supervisor, Willem-Jan, for guiding me during the last six months. His vision, belief in me and comments were invaluable. The people at the Infolab made me feel 'at home' from the start; I couldn't have wished for a better working environment. Cristina, Alice, Jeroen, Kees, Frans, Manfred, and Hans thanks for your humorous remarks, for introducing many useful tools (such as LaTeX and linux) and for making me feel at home in general. Cristina deserves an extra thank you for correcting the English grammar and spelling in my thesis, and Hans for participating in the experiment (see Chapter 6).

I also want to thank several people at the faculty of Arts: Hans Paijmans and Antal vdn Bosch for introducing me into the field of Information Retrieval and Machine Learning respectively, and for introducing me into the exciting field of Natural Language Processing. The input from, and talks with Bertjan Busser have helped me through some of the difficult steps of writing this thesis.

This list would be incomplete if I forgot to mention everyone else who helped me –either directly or indirectly– finish this this. The list seems endless; Marc Graafmans, Maschinka Poiesz, Thomas Wouters, Martin Schapendonk, Jaqueline Dake and all the people that I haven't mentioned just yet: thanks for your support and friendship.

Bas van Gils
Deventer, 15th February 2002

Contents

Preface **i**

1 Introduction **1**

1.1 Background 1

1.2 Research Questions and Methodology 4

2 Business Objects and Frameworks **7**

2.1 Business Objects 7

2.2 Business Object Frameworks 9

2.3 Example 11

2.3.1 Context 11

2.3.2 Checking Process 12

3 Using UML **14**

3.1 The Unified Modelling Language 14

3.1.1 The Class Diagram 15

3.1.2 Relations Between Classes 16

3.1.3 Object Constraint Language 17

3.2 Business Modelling with UML 18

3.3 Modelling BOF with UML: an example 20

4 From UML to XML **22**

4.1 XML Overview 22
4.2 Modelling XML 25
4.2.1 Document Type Definition 25
4.2.2 XML Schema 26
4.3 Representing UML using XML 28
4.3.1 XML Metadata Interchange Format 28
4.3.2 UML eXchange Format 29
4.4 Approach in this thesis 30
4.5 Example 31

5 Semantic Matching **33**
5.1 Semantics 33
5.2 Deriviation of the Algorithm 35
5.2.1 WordNet 37
5.2.2 Document Vector Model 38
5.2.3 Proposed Algorithm 42
5.3 Limitations 44

6 Proof of Concept **46**
6.1 The Experiment 46
6.2 Examples 47
6.2.1 Order Taking 48
6.2.2 Making Backups 49
6.3 Software 49
6.4 Results 51
6.5 Conclusions and Suggested Improvements 53

7 Conclusion **55**

7.1 Overview 55

7.2 Discussion 57

7.3 Future work 58

A DTD for mapping UML to XML **59**

B XML example **61**

C Python source code of the algorithm **64**

List of Figures

1.1 Dimensions in the field of EAI 2

1.2 Overview of the theoretical part 5

2.1 Business Object Architecture 10

2.2 Line-box representation of the Business Object Architecture . . 12

3.1 UML notation for a class 16

3.2 UML notation for association and generalisation 17

3.3 UML notation for composition 17

3.4 OCL example of a post-condition 18

3.5 Example from Section 2.3 in UML notation 21

4.1 XML-RPC example 24

4.2 DOCBOOK example 24

4.3 DTD example 26

4.4 XML Schema example 27

4.5 XMI example 28

4.6 Comparison of UML model elements and UXF elements 29

4.7 UML to XML conversion (partial) 32

5.1 Lexical Matrix 38

5.2 Classical model of IR (Paijmans, 1999) 39

5.3 Graphical representation of the document vector model 40

5.4 Document database in the document vector model 41

5.5 Semantic Matching Algorithm 43

6.1 UML representation of the taking order process 48

6.2 UML representation of backup process 49

6.3 Implementation of the algorithm 50

Chapter 1

Introduction

1.1 Background

An enterprise is a complex system that has a specific purpose or goal. Business systems attempt to support the enterprise in achieving these goals. The concepts that are used to define business systems are goals, resources, processes and rules (Penker and Eriksson, 2000). Traditionally, enterprises have been functionally divided (Douma and Schreuder, 1998). This functional division was the cause of many problems, such as a huge administration and difficulty to handle cross-functional problems (Johanesson and Perjons, 2001). In order to overcome these problems, many enterprises have been focusing (and re-structuring their processes) on business processes that create value for customers.

Enterprise Application Integration (EAI) is a business computing term for the plans, methods, and tools aimed at modernising, integrating, and coordinating the computer applications in an enterprise. A well-known example of EAI is the integration of legacy systems with modern Enterprise Resource Planing (ERP) software, which is an industry term for the broad set of activities supported by multi-module application software that helps a manufacturer or other business to manage the important parts of its business. Implementing these systems is considered an important issue, for they promise to improve efficiency in organisations (Laudon and Laudon, 1996).

Several dimensions play a role in examining the field of EAI. A short overview of these dimensions is given here. More details follow in the upcoming chapters. The first dimension is the *viewpoint*. EAI can be examined at the implementation level software, or at a conceptual level using models. The second dimension concerns *time*. Integrating software has both static and dynamic aspects (OMG, 1997b). Static aspects deal with what is

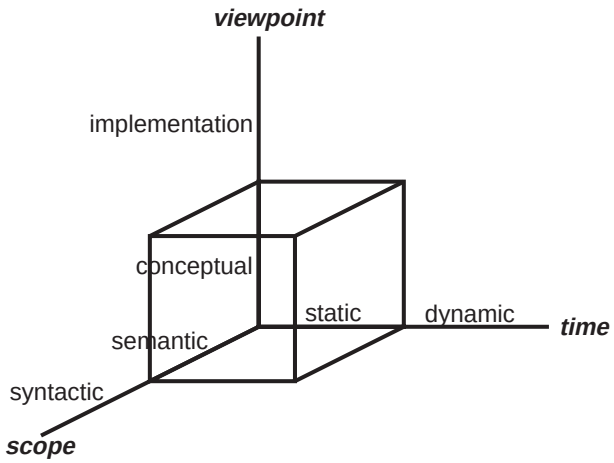

FIGURE 1.1: DIMENSIONS IN THE FIELD OF EAI

actually stored in a system. Dynamic aspects deal with the services that a system offers. The third dimension considered is *scope*: syntaxis versus semantics. The syntaxis level deals with the format (of the files/ messages/ commands) that a system accepts. At the semantic level the 'meaning' of a system is considered. Figure 1.1 is a graphical representation of these three dimensions.

The research question, which is defined in the next section, reflects the choices at the three defined dimensions. The focus is at the conceptual, static and semantic levels respectively. This corresponds to the highlighted area in Figure 1.1.

Focusing on the conceptual, static and semantic levels of EAI, several different classes of EAI can be identified. A classification of EAI can be found in (Wangler and Paheerathan, 2000):

- *Horizontal intra-organisational integration*
  An example is supply-chain management: an organisation tries to optimise the complete set of activities of a business process (order entry/ production/ shipment). For example, suppose a telemarketing company contacts people at home and tries to sell newspaper subscriptions. At dinner time (most people are at home at that time), an employee of the company calls someone at home, explains the benefits of a particular newspaper and –if the person accepts the offer– enters the customers name and address information in the computer system. This system must be (tightly) integrated with the production and shipping systems, because the new customer wants to receive his newspaper the next morning already.

- *Vertical intra-organisational integration*
  An example is the integration of a production monitoring system

(also called TPS, see (Laudon and Laudon, 1996) for details) with Enterprise Resource Planning packages. Building computer-chips is a very costly, and complex process, which must be carefully monitored. Even the slightest deviation in the setup of machines can disrupt the production process. Hence, several computer systems (fail-safe) are monitoring every detail of the production process. This results in enormous amounts of raw data. This data is important (and therefore valuable) for the organisation: it is the basis for decisions on pricing the chips, amount of maintenance and many other things. Therefore, the monitoring systems must be integrated with higher-level applications (e.g. a Decision Support System, see (Laudon and Laudon, 1996) for details) in order to optimise the decision-making process.

- *Inter-organisational integration*
  An example is the integration of the computer systems in a hospital with the systems of a supplier in order to minimise overhead and try just in time (JIT) delivery. Enormous amounts of money are involved in (national) healthcare (see e.g. (Times, 2001)). A big part of this money is used for 'administrative purposes'. These overhead-costs should be minimised; resources should be spent on healing patients. With a close cooperation between external stakeholders (the case study in (Times, 2001) mentions insurance companies, care companies and suppliers) and internal stakeholders (physicians, nurses) through Electronic Data Interchange (EDI)[1] the current situation was improved: the "members send each other requests for information over the Internet using their Web browsers. The answers are sent back on a private network.". That way a significant reduction in administrative cost (associated with gathering, securing and maintaining information) is achieved.

These three categories have in common that two (or more) systems are integrated. According to (King, 2002) this is a complex process: "Unfortunately, many of these EAI projects turn out to be more than difficult, and take more than a little longer – while chewing up more resources then expected". Because of the high costs and risks associated with this integration process, it makes sense to study and plan this process carefully before starting it. In this thesis the focus is on improving it by closely examining the systems that are to be integrated.

The BALES methodology (binding Business Applications to LEgacy Systems) deals with the integration of (new) Enterprise Applications with existing (legacy) applications (*vertical intra-organisational integration* in the classification) (van den Heuvel, 2002). The author adopts the notion that business systems can be modelled using Business Objects (the representation

[1] EDI works by providing a collection of standard message formats and element dictionary in a simple way for businesses to exchange data via any electronic messaging service (EDI, 2001)

of a business concept or process in an information system). Hence, the goal of the methodology is to construct Business Objects in such a way that they "might be used to seamlessly glue business and legacy objects together in terms of their interface definitions". This is achieved by finding out which Business Objects can be mapped to legacy objects.

This approach can be extended to the EAI-approach in general: finding out which parts of both systems can be mapped to another is an important step in the process. The goal of this thesis is to find a metric for the similarity of two Enterprise Systems. In agreement with the approach taken in (van den Heuvel and Papazoglou, 1999a), Business Objects and Business Object Frameworks (a framework in which individual Business Objects are combined to model the business (Shelton, 1995)) are used as high-level (hence, conceptual) representations of the Enterprise Systems. The choice for Business Objects and Business Object Frameworks as a representation for Enterprise Systems, as well as the importance of finding out which parts of two systems can be mapped to another, is reflected in the research question defined in the next section.

1.2 Research Questions and Methodology

This thesis consists of two parts (the first part deals with theory, the second part with a proof of concept), and tries to answer the following research question:

> Given the specifications of two systems in terms of Business Objects, how can the similarity (at the semantic level), between these two systems be *calculated* in order to determine how easy these systems can be integrated?

The first part is an extensive literature study, and is summarised in Figure 1.2. The figure shows that several steps must be taken before the research question can be answered. Hence, the theoretical part of this thesis consists of four steps.

Before these steps can be taken, however, a deeper understanding of Business Objects and Business Object Frameworks must be achieved. To this end, several definitions of Business Objects- and Frameworks are discussed and compared (e.g. (Digre, 1995), (Hung and Patel, 1997), (OMG, 1997a), (Herzum and Sims, 1998), (Persson, 2000), (Shelton, 1995), (van den Heuvel and Papazoglou, 1999b) and (van den Heuvel and Weigand, 2000)).

The first step concerns the modelling of a business system, in terms of Business Objects, using the *Unified Modeling Language* (UML). These models

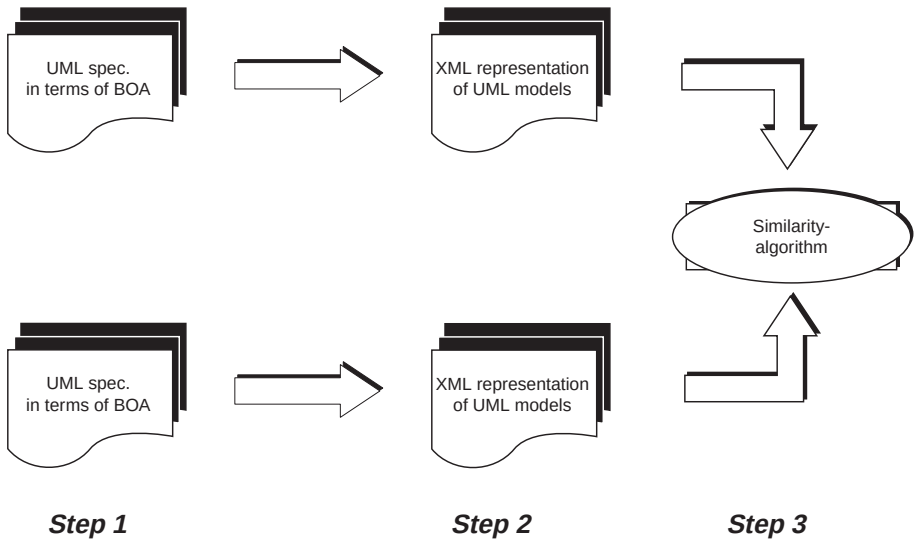

FIGURE 1.2: OVERVIEW OF THE THEORETICAL PART

will be the basis for the calculation mentioned in the research question. UML defines a standard notation for object-oriented systems (Warmer and Kleppe, 1999). It is a notation that can be used for the modelling of real-world concepts, and is therefore well suited to model Business Object Frameworks (Hruby, 1998).

UML, however, is a 'graphical' format, which is inconvenient for calculus. To be able to *compute* the similarity between two specifications, at the semantic level, a textual format is needed. The *eXtensible Markup Language* (XML) provides a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. It is therefore usable to represent the UML models (Carlson, 2001). A 'complicating' factor is that there are many XML-based technologies available, such as XML-*schema* and the XML *Metadata Interchange Format* (XMI). Some vendors also have an XML implemenation to represent UML models (Booch et al., 1999). The theory will be discussed, as well as a small example.

The last step is to find an algorithm which calculates the *semantic* similarity between two models (which are represented as XML-documents). There are two fields of research that are relevant in this part.

First of all, semantic information must be incorporated in the model. Several approaches (borrowed from the field of software engineering in general) are discussed, most notably, the semantic integration of conceptual schemas (Mirbel, 1997), semantic clustering and transformation rules (Missaoui and Sahraoui, 1998), query formulation through relationship semantics in databases (Owei and Navathe, 2001) and the lexicographic reference system called WordNet (Miller et al., 1993).

The second research field concerns the actual algorithm. The field of *information retrieval* (IR) is that part of computer science which studies the retrieval of information (not data) from a collection of written documents. In other words, the IR-theory is used to find an algorithm that calculates the semantic similarity between two models.

The derived algorithm computes the similarity between two systems. The results of this algorithm must be interpreted in such a way that they answer the research question. Hence, a discussion on intepreting the results of the algorithm is included.

After deriving -and discussing the results of- the matching algorithm, the second part of this thesis is a 'proof of concept'. Firstly, the algorithm is implemented (in the programming language *Python*). Secondly, some (partial) specifications of systems are constructed. These will be used in a small experiment where the results of the derived algorithm are compared with the opinion of a human expert. Based on this experiment, conclusions and suggestions for future research are derived.

Chapter 2

Business Objects and Frameworks

This chapter deals with the first step in answering the research question. The goal is to describe what business objects (BO's) are, and how enterprise systems can be modelled using them. In Section 2.1 a general definition of a BO is derived. This definition is the basis for Section 2.2, which explains how BO's can be used to model a business by means of Business Object Frameworks (BOF). Finally, Section 2.3 gives an example of Business Objects and Frameworks.

2.1 Business Objects

Many definitions of what a *Business Object* is are available in literature, which makes the term polysemous (Persson, 2000). These definitions differ mostly because they are used in different contexts. However, they overlap a great deal. In this section, several definitions from the available literature are discussed, and a general definition is derived.

There are three types of objects: business, technology and application objects (Shelton, 1995). Knowing this, a BO is defined as "Business Objects are abstracts that represent a person, place, thing or concept in the business domain. They package business procedures, policy and controls around business data. Business objects serve as a storage place for business policy and data, ensuring that data is only used in a manner semantically consistent with business intent.".

The author also states that BO's are specialised in two subcategories, namely *business entity objects* (representing people, places and things in much

the same manner as data-modeling entities) and *business process objects* (representing business verbs/ business processes).

The *Object Management Group* (OMG)[1] aims at creating a component-based software marketplace by hastening the introduction of standardised object software. The organisation's charter includes the establishment of industry guidelines and detailed object management specifications to provide a common framework for application development. OMG is a major player in the field of Business Objects. In (OMG, 1997a) they define the concept of BO (from an implementation point of view) as "Business Objects are specialised CORBA objects –they are network accessible through an object request broker. A CORBA 'object reference' uniquely identifies an active business object within the distributed objects environment for purposes of communication of requests through an object request broker.".

The definition of a BO in (Hung and Patel, 1997) is less implementation oriented. The context is the *Dynamic Systems Development Method* (DSDM) life-cycle environment. The paper defines a Dynamic Business Object Architecture (see also Section 2.2) with a strong emphasis on business enterprise activities and processes as well as feedback from end-users: "A Business Object is a coarse-grained object abstraction that encapsulates a typical, generic business task, adapted from a particular business domain. A Business Object is used to capture and define a model of the user's business and its information requirements.".

In (Herzum and Sims, 1998) the business component approach to distribute system development is introduced. The concept of BO is named *Business Component* and is defined as "the information system's representation, from requirements analysis through deployment and run-time, of an 'autonomous' business concept or process. It consists of all the software artifacts necessary to express, implement and deploy the given autonomous business concepts as an equally autonomous, reusable element of a larger information system.".

From the above-mentioned definitions, several facts can be derived. These facts must be relevant for the research question and the dimensions specified in Section 1.1 (see Figure 1.1 for details). This means that implementation details, as well as syntactic and dynamic aspects can safely be ignored. These facts, summarised below, are the basis for the definition of the BO-concept used in this thesis.

- A Business Object is a person, place, thing or concept in the business domain.
- A Business Object stores business policy and data.

[1] `http://www.omg.org`

- Development of Business Objects is aimed at providing a common framework for application development.
- A Business Object is a coarse-grained object abstraction that represents a business task from a particular business domain.
- A Business Object consists of all artefacts needed to represent a given business concept.

These facts lead to the following definition of the concept of a Business Object:

Definition 2.1 (Business Object)
*A Business Object (*BO*) is a coarse grained object abstraction that consists of all artefacts needed to represent a person, place, thing or process in a given business domain. Furthermore, the aim of* BO *development is to provide a common framework for application development.*

This definition is in agreement with the problem definition of this thesis, and the dimensions defined in Figure 1.1. First of all, the definition states that BO's have to do with an abstraction, which positions them at the conceptual level. Furthermore, BO's do incorporate semantic and dynamic information since they represent *all* artefacts to represent a person, place, thing or process.

2.2 Business Object Frameworks

A business is a complex system, consisting of a hierarchical organisation of departments and their functions (Penker and Eriksson, 2000). A good business model contains information on business processes, resources business rules, goals, etc.. The knowledge of domain experts ('business people' as opposed to 'technical people') is very important in business modelling (Bonar, 1997).

The aim of BO-development was to provide a common framework for application development (see Definition 2.1). Thus, it makes sense to use BO's to model business systems. In (Sutherland, 1995), Jeff Sutherland recognises that software architectures must be transformed as business models are renewed. A *Business Object Framework* (BOF) is an effective solution for dynamic automation of a rapidly evolving business environment.

As with Business Objects, there are many definitions available of what a BOF is in literature. An overview of these definitions is given here, as a basis for deriving a general definition.

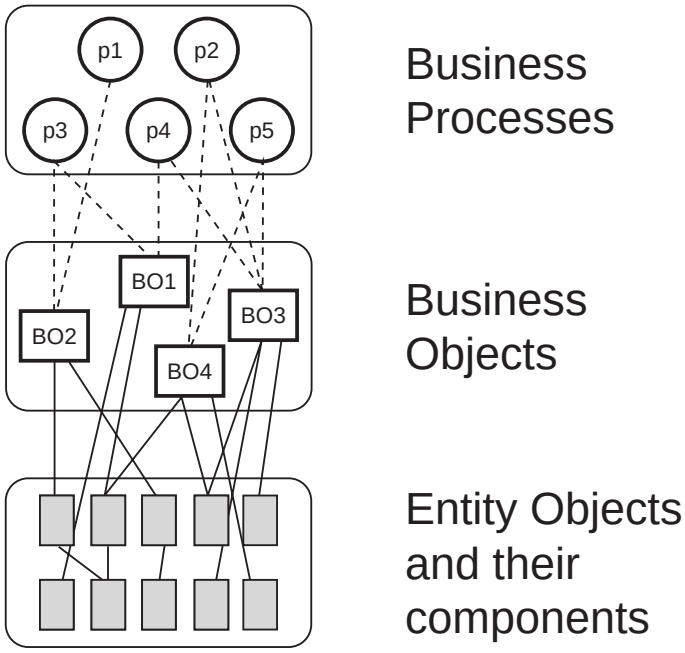

FIGURE 2.1: BUSINESS OBJECT ARCHITECTURE
(based on (Hung and Patel, 1997))

In (Herzum and Sims, 1998) a BOF is named '*business component system*', and is defined as "the whole of the business concept being developed in terms of the set of business components needed to provide autonomy. It is the highest level view, in which the system can be seen as a list of business components.". This definition stresses that business components can be used to model the business at a high level.

It is in agreement with (Casanave, 1995), which uses the name '*Business Object Architecture*' for BOF, and is defined as "an architecture that represents the components that are used to model business problems and build the system.". This definition is extended to a framework in (Hung and Patel, 1997). This framework is depicted in Figure 2.1.

There are three levels to be defined. The first is the *business process layer*, depicting the fact that BOF is used to model the business. The middle-layer is the *business object layer*, and the bottom-layer consists of *entity objects* and their components. Entity objects are objects holding information that is typically stored in a database, and are also called 'passive objects' (Penker and Eriksson, 2000)[2].

The strong emphasis on the business architecture is also present in (Penker and Eriksson, 2000). A well-designed architecture makes it possible to thoroughly understand the structure being built, to plan the actual

[2]When using Definition 2.1 for Business Objects, the two lower layers in Figure 2.1 would be merged.

construction and to estimate costs. It defines a business architecture as "an organised set of elements with clear relationships to one another, which together form a whole defined by its functionality. The elements represent the organisational and behavioural structure of a business system, and show abstractions of the key processes and structures in the business.".

In (Shelton, 1995), a framework is defined as a collection of abstract and concrete classes, and the interfaces between them. The interactions between these classes are called *patterns*. A pattern is an established generalised solution that solves a problem that is common to different business situations (Penker and Eriksson, 2000). There are three types of patterns:

1. Business patterns address problems within the business domain
2. Architectural patterns address problems that occur in the area of the architectural design of information system
3. Design patterns are used for situations in which the analysis is already mapped and described, and the focus is on producing technical solutions that are flexible and adaptable

Combined, these two concepts (framework and pattern) form a definition of BOF: "A Business Object Framework is a set of abstract and concrete Business Object classes with a set of built-in business patterns.".

The above mentioned definitions combined with the dimensions specified in Section 1.1 (see Figure 1.1 for details), lead to the following definition of a BOF:

Definition 2.2 (Business Object Framework)
*A Business Object Framework (*BOF*) represents a business architecture. Business Objects and Business Patterns are the building blocks for this architecture.*

2.3 Example

This section contains an example to clarify the concepts introduced in the previous section. The example deals with checking parcels before they are shipped to an online bookstore (e.g. `http://www.amazon.com`).

2.3.1 Context

The company used in this example, called *OnlineBooks.com*, sells books and CD's over the internet. Customers can browse through the online catalog and

add items to the virtual cart. Once the customer is done, he/she is redirected to a secure area of the website where payment via creditcard is dealt with. *OnlineBooks.com* has integrated their online webshop with back-end systems. This means that as soon as a customer has finished the payment-procedure, the order is inserted in a database, and a packing list is printed out.

This packing list is given to the packer, who collects all the goods on the list and puts them in a parcel. The packing list is stapled to the parcel for future reference. Once this is done, the parcel is stored in a storehouse, and the packer marks the parcel as 'done' in the information system of *OnlineBooks.com*. This results in a message to the storekeeper, who needs to check whether the parcel is, indeed, ready for shipment.

In the storehouse, the isles are arranged and sorted using a country code (e.g. NL for the Netherlands and UK for the Unighted Kingdom). This enables the storekeeper to find the parcels quickly; walk to the corresponding isle and find the parcel. Once it is found, the storekeeper checks whether the content matches the packing list. If it corresponds, he/she marks the parcel as 'ready for shipment' in the information system. If something is wrong, a message is sent to the packer.

2.3.2 Checking Process

The actual process examined here is the process where the storekeeper checks whether parcels are actually complete and, hence, ready for shipment. Several Business Objects, and a single Business Pattern can be identified based on the given context.

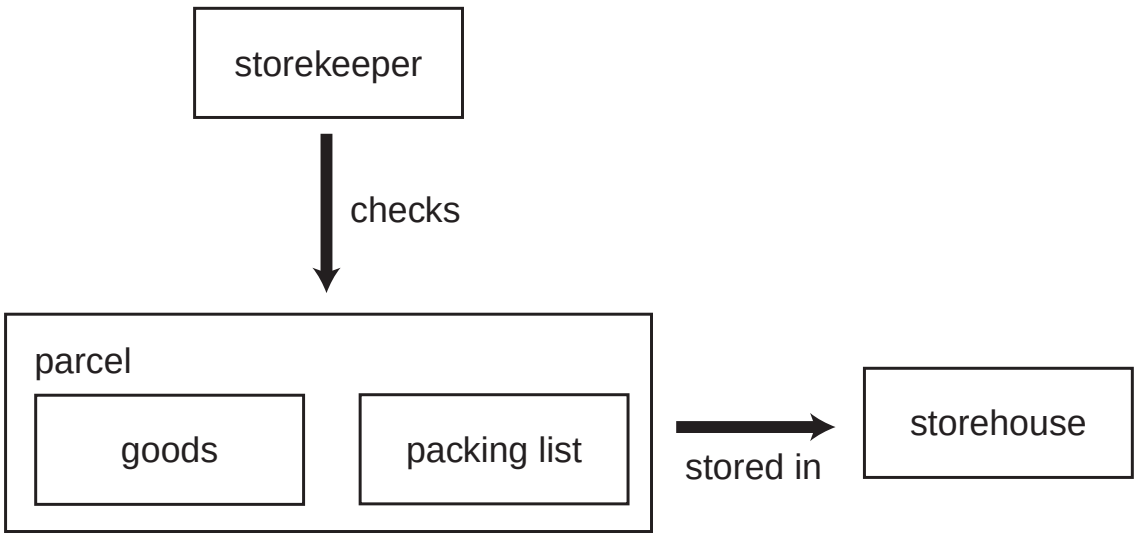

FIGURE 2.2: LINE-BOX REPRESENTATION OF THE BUSINESS OBJECT ARCHITECTURE

- The *storekeeper* is a BO of type 'person'. The BO must represent the fact that the storekeeper has the responsibility to check whether parcels are complete and ready for shipment. Thus, the BO stores data on which packages still need to be checked. Furthermore, the BO must have a 'policy' on how to check whether a parcel is complete.
- A *parcel* is a BO of type 'thing'. The BO must represent that a parcel is *composed* of goods and a packing list.
- Goods can be either 'CD' or 'book'; which are BO's of type 'thing'. Data such as title, date etc. must be stored in the BO.
- A Packing list is a BO of type 'thing', and stores things as shipping information, price and information on when it was checked.
- A *storehouse* is a BO of type 'place'. The BO must 'know' which parcels are stored in the storehouse. The policy that isles are organised using a country-code must be available in the BO.

The pattern is that the storekeeper uses the organisation of the storehouse to track parcels. Figure 2.2 shows the 'Business Object Architecture', using a simple line-box diagram style[3].

[3]In Section 3.3 this example is displayed using the UML notation

Chapter 3

Using UML

A model is a miniature representation of a part of reality. Models are used in many contexts, such as information systems and energy plants. Business Modelling, hence, can be thought of as making a 'picture' of the business, and Information Systems modelling deals with describing how an information system works. There are many ways of modelling information systems (e.g. 'Structured Analysis' and OMT). Recognising the need for standardisation, major players in the modelling field came up with a *Unified Modelling Language* (UML).

Chapter 2 described what Business Objects are, and how they can be combined in a Business Object Architecture. This chapter describes how a Business Object Architecture can be modelled using UML. Section 3.1 gives a short introduction to the Unified Modelling Language. Section 3.2 shows examples from literature on how UML can be used to model businesses. Section 3.3 describes how UML can be used to model a Business Object Architecture.

3.1 The Unified Modelling Language

The Unified Modelling Language (UML) is a language for specifying, visualising, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems (Group, 2001). There are many texts that describe what UML is, and how it can be used[1]. The purpose of this section is *not* to give a full overview of UML, but to give a short introduction. Attention will be paid to aspects that are relevant for business modelling. This introduction is based on (Warmer and Kleppe, 1999).

[1] For an overview of UML-resources see `http://www.omg.org/uml/`.

UML is a visual modelling language, not a methodology. It standardises several diagram-types that, together, describe the model of a system. They are categorised as follows:

- ***use-case diagram*** (also called *requirements diagram*): shows how a system can be used by external entities (e.g. users)
- ***static diagrams***:
  - *class diagram*: shows the static structure of a system using classes and their relationships
  - *object diagram*: shows the static structure of a system in terms of objects and their relationships
- ***dynamic diagrams***:
  - *sequence diagram*: shows when (in which order) messages are sent and received within a system
  - *collaboration diagram*: shows how objects cooperate to achieve a goal
  - *state diagram*: depicts the states an object can be in, as well as which state-transitions are possible, during the object's lifetime
  - *activity diagram*: shows which activities are executed by parts of a system
- ***implementation diagrams***:
  - *component diagram*: shows the partitioning of the entire system in components, and the relationships between these components
  - *deployment diagrams*: depicts how (software) components are used in a specific system

Given the problem definition, the static diagrams are of interest for this thesis[2], in particular the class diagram. This diagram type is described in the next section.

3.1.1 The Class Diagram

To be able to talk about a class diagram, first a definition of the concept *class* must be given. The concept of a class stems from the field of object oriented programming. Classes are a means of categorising objects. These can be either real life objects or software objects, depending on the domain. Put more formally:

[2]The other diagram types are not described in detail here. See the UML documentation for further details on those.

Definition 3.1 (class)
A class is a set of objects which share a common structure and behaviour. The structure of a class is determined by the attributes *which represent the state of an object of that class and the behaviour is given by a set of* methods *associated with the class.*
(based on: `http://foldoc.doc.ic.ac.uk/foldoc/index.html`*)*

A class, thus, has attributes. An attribute is 'information' that is stored in an object. A class also has methods, which describe the services that an object can perform. Another relevant concept is *stereotype*, which enables classification that does not originate from the conceptual domain (Warmer and Kleppe, 1999). For example, the class `Employee` has as its stereotype `Person`, attributes called `name` and `phone number`, and methods called `speak` and `walk`. Figure 3.1 visualises this example in UML-notation.

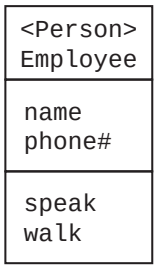

FIGURE 3.1: UML NOTATION FOR A CLASS

3.1.2 Relations Between Classes

A class diagram is more than a summing of a number of classes. This section describes several ways of connecting individual classes.

An *association* is a structural relationship between two classes (Warmer and Kleppe, 1999; Penker and Eriksson, 2000). Objects can have several associations with several classes; that is, they can play different *roles* (also called 'association end') in several associations. In UML, an association is shown as a solid line between two classes. Roles are written at the end of the relevant association. An example is the fact that all companies must register at the chamber of commerce (depicted in Figure 3.2).

A second type of relationship between classes is *generalisation*; it is the relationship between a class and one or more refined versions of it. The class being refined is called the superclass and each refined version is called a subclass (Rumbaugh et al., 1991). For example: for a wholesale trader, both a supplier and a customer are companies. They have some things in common (for example, the fact that both must have a registration at the chamber of commerce. However, they are 'specialized' in the sence that they differ

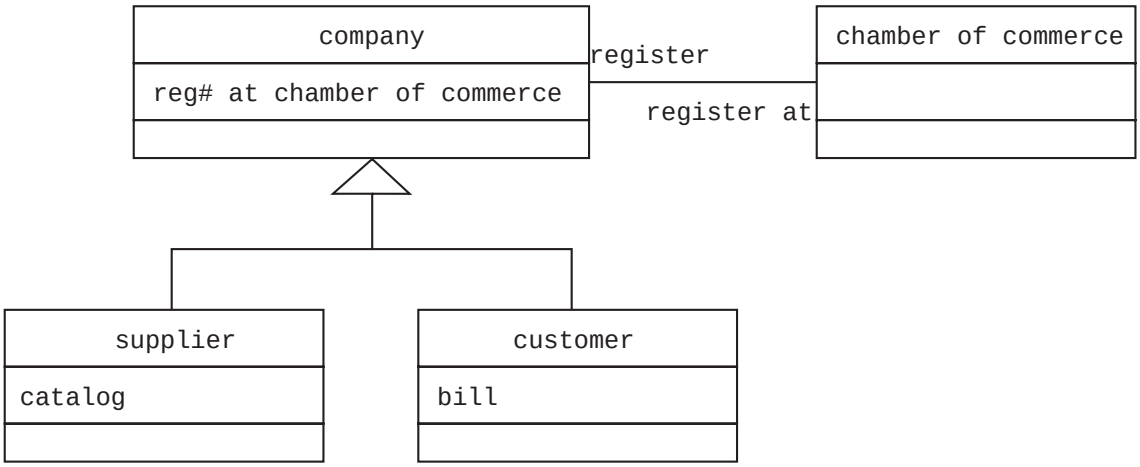

FIGURE 3.2: UML NOTATION FOR ASSOCIATION AND GENERALISATION

in some things (e.g., a supplier can have `catalog` as an attribute, and a customer can have `bills` as an attribute). In UML, generalisation is depicted as a big arrow pointing at the superclass. The above mentioned example is depicted in Figure 3.2.

Finally, a third type of relationship is *composition*. Aggregation is a relationship that indicates which classes are "part of" another class. A part-object can only exist as long as the 'whole' exists (lifetime dependency). An example from the business domain is that an invoice must *always* be composed of a picklist and a bill. In UML, composition is depicted as an association with a black, filled diamond at the 'whole'-end. Figure 3.3 depicts the above-mentioned example.

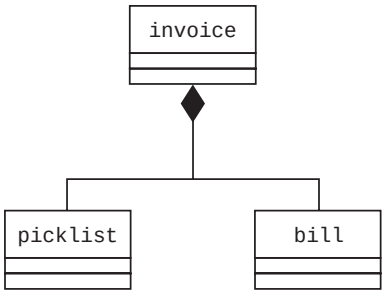

FIGURE 3.3: UML NOTATION FOR COMPOSITION

3.1.3 Object Constraint Language

Recently it became apparent that, although visual modelling gives a lot of insight, they have multiple interpretations (Warmer and Kleppe, 1999). For example, the relation between a 'person' and an 'information system' can be interpreted in different ways. One interpretation is: information about a

person can be stored in an information system. A second interpretation is that a person manages/uses the information system. The *Object Constraint Language* (OCL) is a textual language that can be used to specify constraints on objects in UML. So, by using OCL, a model gets a single interpretation. There are four types of constraints (Warmer and Kleppe, 1999):

- INVARIANT : a constraint on a class which must always hold for *every* instance of that class
- PRE-CONDITION : a constraint that must always hold directly *before* the execution of a method
- POST-CONDITION : a constraint that must always hold directly *after* the execution of a method
- GUARD : a constraint on a state transition of an object

OCL has a formal syntax which will not be discussed here[3]. An example of how OCL can be used, however, is the following: suppose the class[4] `Article` has a method `deliverable` and there is a business rule saying that the 'availability' of an article is the amount in stock of this article. Figure 3.4 shows this rule (a post-condition) in OCL.

```
context Article::deliverable(): Boolean
pre:  --
post: result = (availability = #inStock)
```

FIGURE 3.4: OCL EXAMPLE OF A POST-CONDITION

This example shows how a post-condition can be expressed in OCL. This expression should be in a UML-note, attached to the `Article` class. The other constraint-types are added to UML-models in pretty much same way.

3.2 Business Modelling with UML

This section discusses how UML can be used to model business (systems). First, several approaches from literature are discussed. After that, the approach used in this thesis is derived.

In (Penker and Eriksson, 2000), the concepts *resource*, *process*, *goal* and *rule* are the building blocks for business systems. Standard UML diagrams

[3]For a discussion on OCL see the book by Jos Warmer and Anneke Kleppe: *The Object Constraint Language: precise modelling with* UML

[4]See Section 3.1.1

and symbols can be used to model the business: "Static diagrams as well as dynamic diagrams are valid and appropriate for describing ... a business system". The static models (using stereotypes for the four 'building blocks') are used to capture structural aspects of business model. The dynamic model types are used to capture temporal aspects.

A similar approach is followed in (Heumann, 2001); it is stressed that a visual model of the business "can provide important insights" and that "the Unified Modelling language ... can be used effectively to create such a model". In this approach *Use-Case* models are used to describe business processes. The Use-Case model consists of a digram, showing a high-level overview of a business process, and a use-case specification. The latter documents the details (name, description, performance goals, benefits, requirements, etc.) associated with the diagram. Furthermore, the approach uses UML *activity diagrams* to describe the structure of different workflows. Finally, a *Business Object Model* is used to model how processes work: "it serves as an abstraction of how business workers and business entities need to be related and to collaborate in order to perform the business".

In (Popkin, 1998) this approach is also adopted. *Use-case diagrams* and *activity diagrams* capture how processes / scenarios execute; they show "what objects interrelate to make ... behaviour happen". *Class diagrams* are the central analysis diagrams, and capture the static structure of a system. An informal techique called *CRC Cards*[5] is used for a 'responsibility driven analysis'. Finally, *state diagrams* are used to model real-time behaviour of the system.

The approach in this thesis is a subset of the mentioned approaches, and is based on the notion that UML can be used to model the Business Object Framework (Hruby, 1998). The building blocks of BOA are business objects and business patterns (see Definition 2.2 on page 11).

Since a BO represents a person, place, thing or process in a given business domain, UML must be used to describe these. The emphasis in this thesis is on static aspects, which results in using the UML-class diagram. Classes in the class model map to business objects; the 'properties' of Business Objects are represented by the *attributes* and the 'behaviour' is represented by the *methods*. Furthermore, each class is stereotyped as either a person, place, thing or process to indicate which 'type' of BO it is. Finally, OCL-constraints are used to express aditional constraints (such as business rules and patterns) to the model.

[5]CRC Cards are not discussed in this thesis. For an introduction see e.g. `http://c2.com/doc/oopsla89/paper.html`.

3.3 Modelling BOF with UML: an example

Based on the approach derived in the previous section, the example presented in Section 2.3 can be modelled using the UML notation. Figure 3.5 shows the class diagram.

The class diagram has eight classes. Each class represents a Business Object, and is sterotyped according to its 'type'[6].

The first class is *storekeeper*. Obviously, this is a person. The storekeeper has three attributes and two methods which are relevant for this domain. First of all, different storekeepers have different names and social security numbers to distinguish them. Furthermore, each storekeeper has a list of parcels to check. The two methods are used to add goods to this list, and to remove them respectively.

The storekeeper participates in several *checking processes*, which is the second class. The checking process deals with checking whether a parcel is complete, and ready for shipment. This check is executed by the storekeeper and returns either *true* or *false*. The OCL constraint attached to the class represents this post-condition.

Parcel is the third class. One parcel is checked in exactly one checking process, and consists of a *packing list* and one or more *goods*. In this case, goods can be either *books* or CD*'s*.

A packing list stores data such as shipping info, price and checking info (who packed the parcel and when). The CD class has the artist(s), the tracks and a date as its attributes. The book class stores the name of the author, the title and the ISBN.

The last class is *storehouse*. Many parcels are stored in one storehouse. The isles of the storehouse are sorted using a country code (represented by the note attached to the storehouse class). Finally, the storehouse class has methods to represent the fact that parcels can be stored in, or removed from the storehouse.

[6]The definition of a BO has four types: person, place, thing and process. See Definition 2.1.

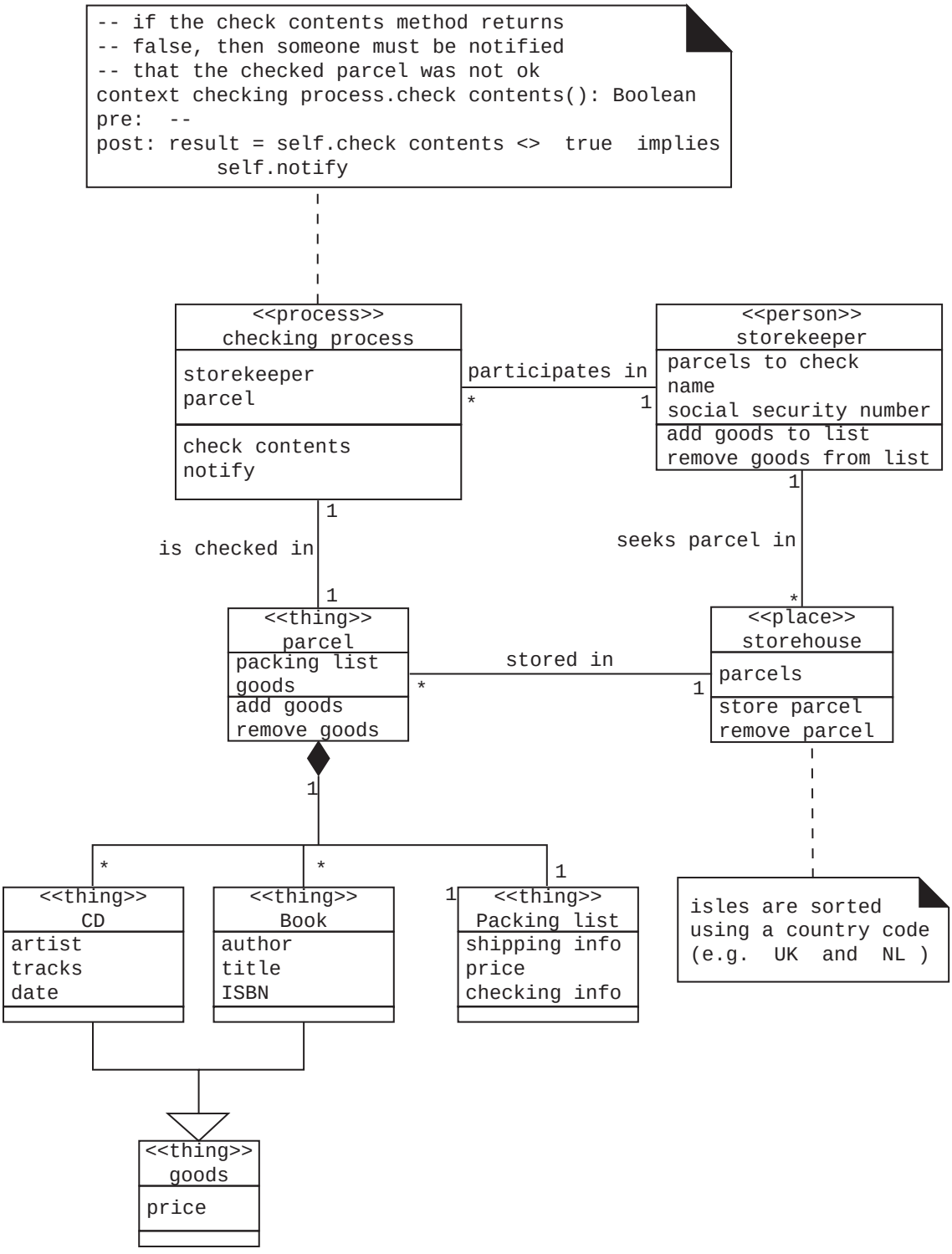

FIGURE 3.5: EXAMPLE FROM SECTION 2.3 IN UML NOTATION

Chapter 4

From UML to XML

Many different ways of representing 'structured data' exist. For structured data one should think of such things as spreadsheets, address books and databases. An essential characteristic of structured markup is that it explicitly distinguishes (and accordingly 'marks up' within a document) the structure and semantic content of a document (Walsh and Muellner, 2001).

Structured data can be stored as either a binary, or a textual format (w3c, 2000). Binary representations of data are machine readable only. However, textual formats are characterised by the fact that they can also be read by humans. Performing calculations to a UML model is difficult since UML is a graphical notation (See also Figure 1.2). This chapter discusses how the eXtensible Markup Language (XML), a textual format, can be used to represent UML.

First, a short overview of what XML is -and where it is (often) used- is presented. Two examples (XML-RPC and DOCBOOK) of XML vocabularies are covered here. These are also the basis for the section on XML modelling. After this, several mappings between UML and XMLare covered (most notably the XML metadata interchange format (XMI) and the UML exchange format (UXF)). Finally, a methodology for mapping UML to XML used in this thesis is derived.

4.1 XML Overview

The eXtensible Markup Language (XML) is a meta-markup language standardised by the World Wide Web Consortium (W3C). It defines a syntax to markup textual data with simple, human-readable tags[1], and is flexible enough to be customised for many different application domains.

[1]Tags are sometimes referred to as *tokens*

The tags of XML, look very much like the tags that are used in HTML. Start tags begin with a < and end tags begin with a </. Both of these are followed by the name of the element and are closed by a >. XML also has empty elements which have the form `<elementname />`.

Documents that conform to a simple set of rules are said to be *well-formed* (Harold and Means, 2001) The simple, though strict, syntax rules of XML are:

1. All XML elements must have a closing tag
2. XML tags are case sensitive
3. All XML elements must be properly nested; overlapping elements are not allowed
4. All XML documents must have a root tag
5. Attribute values must always be quoted

Data is included in XML documents as plain text surrounded by markup *tags*. A unit of data, its tags inclusive, is referred to as an *element*. There is no fixed list of tags that are always supposed to work for everyone and in every application domain. XML is a *meta language* – a language for describing other languages. This means that people can define their own sets of tags (also called *vocabularies* (Flynn, 2001) or *applications* (Harold and Means, 2001)) that fit their particular application domain (e.g. physics, chemistry, business modelling, etc.).

Many of these vocabularies are readily available today. Two well-known examples are XML-RPC and DOCBOOK. The first is a specification and a set of implementation that allows software running on different machines and / or operating systems to make procedure calls over the internet, using XML as the encoding (XML-RPC, 2001). Using XML makes sense because of the fact that, nowadays, most machines have an XML-parser available (Winer, 1998). Applications, regardless of the platform they are running on, can use this parser to separate the actual data from the XML encoding[2].

Figure 4.1 holds a small example of a XML-RPC-request taken from (XML-RPC, 2001). In the example, a methodcall `examples.getStateName` is sent to `betty.userland.com`. The methodcall has one parameter named `i4`, which has the value `41`.

The second example – DOCBOOK – is a popular XML vocabulary for writing (technical) documentation (Walsh, 2001). The rationale behind DOCBOOK

[2]An example on how to use XML-RPC within the *Python* programming language can be found at
`http://www.onlamp.com/pub/a/python/2000/11/22/xmlrpcclient.html`

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>

  <methodName>examples.getStateName</methodName>
  <params>

    <param>
      <value> <i4> 41 </i4> </value>
    </param>

  </params>
</methodCall>
```

FIGURE 4.1: XML-RPC EXAMPLE

is to promote the interchange and delivery of documents. Using XML, the semantics of a document could be defined, without -much- worrying about how the document will be rendered in the end. For example, one specifies that something is a *emphasized* rather then 'this is 11pt, italic, in a Times new Roman font' (Walsh and Muellner, 2001).

Figure 4.2 has a small DOCBOOK example. This document/ excerpt is an `article`, written by the honorific `Dr Emilio Lizardo` and currently has one – empty – paragraph.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
  <artheader>
    <title>My Article</title>
    <author>
      <honorific>Dr</honorific>
      <firstname>Emilio</firstname>
      <surname>Lizardo</surname>
    </author>
  </artheader>

  <para> . . . </para>
</article>
```

FIGURE 4.2: DOCBOOK EXAMPLE

4.2 Modelling XML

It is often useful to check whether an XML document (also called *instances* (Ray, 2001)) conforms to a specific vocabulary. The process of formally defining a language in XML is called *document modelling*. For example, a document model answers such questions as 'Can this element have price?' or 'What kind of data does a person-element hold?'.

Definition 4.1 (Document Model)
A document model describes a document type and defines the documents that can be produced with a language. Conforming documents are said to be valid within the context of the language; other documents are invalid.

The document model is a document, written in a specific syntax designed to describe XML languages and explicitly defines the markup of a vocabulary. Two document modelling types exist: *Document Type Definition* (DTD) and *Schema*.

4.2.1 Document Type Definition

According to (Ray, 2001), the Document Type Definition (DTD) is the most popular type of document model. A DTD declares which elements are allowed in a document that should be conform to the DTD (e.g. a 'book' has a 'title'). It also defines what elements or data can go inside an element, in what order, and in what number. Together, these are the *vocabulary* and the 'grammar' of the language (Ray, 2001; Harold and Means, 2001).

The syntax for DTD's stems from SGML[3]. A DTD encompasses a set of rules, or *declarations*, which add elements, sets of attributes or entities. A declaration of an element has the following syntax: `<!ELEMENT name content-model>`. The `name` is case-sensitive. Furthermore, there are five different content models (empty elements, elements with no content restrictions, elements containing only character data, elements containing only elements and elements with mixed content), which are not further discussed here[4].

Elements can have attributes. For each element, thus, an attribute declaration list can be included in the document model. Such an attribute declaration takes the following form: `<!ATTLIST elementname attname1 atttype1 attdescname1 >`.

[3] SGML is *Standard Generic Markup Language*, and is not covered in this thesis.
[4] See e.g. (Harold and Means, 2001)

```
<?xml version=''1.0''?>
<!DOCTYPE person[
  <!ELEMENT first_name (#PCDATA)>
  <!ELEMENT last_name  (#PCDATA)>
  <!ELEMENT profession (#PCDATA)>
  <!ELEMENT name       (first_name, last_name)>
  <!ELEMENT person     (name,profession*)>
]>

<person>
  <name>
    <first_name> Alan </first_name>
    <last_name> Turing </last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

FIGURE 4.3: DTD EXAMPLE

Each attribute list belongs to one single element (attname). Further, every attribute has a type (atttype) and a description of the attribute behaviour (attdescname). There are several attribute types (CDATA, NMTOKEN, NMTOKENS, ID, IDREF, IDREFS, ENTITY, ENTITIES and NOTATION) and different kinds of behaviour for attributes (default value assigned, #IMPLIED, #REQUIRED or #FIXED), which are not covered here.

Figure 4.3 has a small XML-example, consisting of a DTD and an XML-document that is *valid* according to this DTD. The DTD specifies that a person must have a name, and may have *one or more* (indicated by the *) professions. Furthermore, a name consists of a first_name and a last_name. The XML-document has only one person.

4.2.2 XML Schema

For some uses, DTD's turned out to be insufficient; the syntax is not flexible/ expressive enough for some needs. For example, expressing multiplicity-constraints (a la UML) are nearly impossible to implement using a DTD. Also, the fact that documents follow XML syntax and DTD's follow a whole different syntax is 'odd' to some people (Ray, 2001). XML Schema (sometimes referred to as *XSchema*) provides an alternative document modelling technique.

XSchema is a standard provided by the Schema Working Group at the World Wide Web Consortium (XSchema, 2001). Its syntax is well-formed XML and provides more control over datatypes and patterns. An XSchema identifies a document as a schema, and associates it with the XSchema namespace.

```
<xsd:schema xmlns:xsd=''http://www.w3.org/1999/XMLSchema''>
  <xsd:element name=''person'' type=''PersonType''/>

  <xsd:complexType name=''PersonType''>
    <xsd:element name=''name''       type=''name''/>
    <xsd:element name=''profession'' type=''xsd:string''
      minOccurs=''0'' maxOccurs=''*''/>
  </xsd:complexType>

  <xsd:complexType name=''name''>
    <xsd:element name=''first_name'' type=''xsd:nme''/>
    <xsd:element name=''last_name''  type=''xsd:nme''/>
  </xsd:complexType>

  <xsd:simpleType name=''nme'' base=''xsd:string''>
    <xsd:pattern value=''[a-zA-Z]+''/>
  </xsd:simpleType>
</xsd:schema>

<person>
  <name>
    <first_name> Alan </first_name>
    <last_name> Turing </last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

FIGURE 4.4: XML SCHEMA EXAMPLE

Next the constraints on XML elements are included. An element can be either *simple* (one that has no attributes or elements for content) or *complex*. Restrictions on the content model are made by using attributes such as `minOccurs` and `maxOccurs`.

Furthermore, every element has a `type` attribute. There are several predefined datatypes available (e.g. `float`, `boolean` and `binary`). Using a `<pattern>` declaration, additional pattern restrictions on the model can be made as well. These patterns follow the syntax of *regular expressions*[5].

Figure 4.4 has a small example of XSchema (actually, the same as the DTD-example in Figure 4.3). The first line of the example defines the namespace `xsd` for the rest of the schema-definition. The top-level element is `name`, which is a `complexType` consisting of `name` and `profession`. The profession must occur between 0 and 'infinite' times. A `name` is a `complexType` consisting of a `first_name` and a `last_name`; both of which are of type `nme`. Finally, `nme` is a `simpleType` based on `string`. The 'extension' is a pattern saying

[5] See e.g. `http://www.devshed.com/Server_Side/Administration/RegExp/` for an introduction on regular expressions

```
<!-- partial, header not included -->

<Model xmi.id="a1">
  <Class xmi. id="a7">
    <name>Customer</name>
    <feature>
      <Attribute>
        <name>id</name>
        <multiplicity>
          <XMI.field> 1</ XMI.field>
          <XMI.field> 1</ XMI.field>
        </multiplicity>
        <type> <DataType href="|a247"/> </type>
      </Attribute>
      <Operation>
        <name>update</name>
        <scope xmi.value="instance"/>
      </Operation>
    </feature>
  </Class>
</Model>
```

FIGURE 4.5: XMI EXAMPLE

that anything that is of type nme must conform to the pattern 'one of more occurrences of a lower- or uppercase letter'.

4.3 Representing UML using XML

Section 1.2 explains that it is necessary to represent UML models in a textual format in order to *calculate* the similarity, at the semantic level, between the two models. In this section, two approaches that use XML to represent UML models are discussed, most notably XML *Metadata Interchange Format* (XMI) and the UML *eXchange Format* (UXF). The methodology used in this thesis is derived after that.

4.3.1 XML Metadata Interchange Format

The XML Metadata Interchange Format (XMI) provides, among other things, a way for creating an open interchange format from a domain using UML (Brodsky, 1999). The XMI technology was originally developed by OMG and is based on the XML standard from the World Wide Web Consortium (W3C) (Brodsky, 1999).

UML model element	UXF Representation
Association	<Association>
AssociationEnd	<AssocRole>
Attribute	<Attr>
Class	<Class>
Generalization	<Generalization>
Operation	<Operation>
Refinement	<Refinement>

FIGURE 4.6: COMPARISON OF UML MODEL ELEMENTS AND UXF ELEMENTS

In November 1997, both MOF[6] and UML were adopted as OMG standards. However, an interchange format was *not* included at that time (Iyengar and Brodsky, 1998). XMI uses XML to enable the representation of UML based models, using a specified UML DTD[7].

The following example is taken from (Iyengar and Brodsky, 1998). Suppose a very simple model comprised of only one class `Customer`. This class has one attribute `id:CustId` and one method `update`. Figure 4.5 shows the XMI-representation of this model. It makes use of namespaces[8], which are not discussed in this thesis. However, the figure clearly demonstrates how UML constructs described in Chapter 3 are represented.

4.3.2 UML eXchange Format

The authors of (Suzuki and Yamamoto, 1998) recognised the need for a format to make UML models exchangeable on the Internet and/or between development tools. The solution presented in this paper uses the UML eXchange Format (UXF). UXF provides a mapping from UML to XML. The choice for XML is based on the fact that it focuses on the description of the information structure and content as distinct from its presentation.

Each construct in the UML specification is mapped directly to an XML tag[9]. Figure 4.6 shows a part of this mapping. Furthermore, the attributes of each UML construct are mapped to attributes in the XML model.

[6] MOF stands for 'Meta Object Facility' and is not explained in this thesis. See e.g. `http://www.omg.org/technology/documents/formal/mof.htm`.

[7] The DTD can be found at `ftp://ftp.omg.org/pub/docs/ad/98-10-16.dtd`

[8] For an explanation of XML namespaces see e.g. (Harold and Means, 2001)

[9] Not all UML elements were discussed in Chapter 3; this overview is an integral copy from (Suzuki and Yamamoto, 1998)

4.4 Approach in this thesis

The approach in this thesis is a combination of the two previously mentioned methods. The basis for the mapping is the definition of a Business Object (Definition 2.1 on page 9) and Business Object Framework (Definition 2.2 on page 11). The goal is to create a mapping from UML to XML that enables *semantic comparison* of two UML models. This mapping is dealt with in the next chapter.

The first choice to be made is between a DTD and XSchema. This choice comes down to deciding whether a DTD has enough expressive power to represent the semantics of a UML model representing a Business Object Architecture. Both the UXF- and the XMI use a DTD. The approach followed in this thesis is based mainly on UXF [10]. Therefore, the choice for a DTD seems apparent. The remainder of this section explains how the DTD -which can be found in Appendix A- was constructed.

The first step in generating this DTD is to create a top-level node in the XML document called `model`. This node has an optional attributed `name`, which may contain the name of the model.

In this thesis there are four types of BO's: a *person*, *place*, *thing* and *process*, represented by the *stereotypes* in the UML model. The second step in representing the UML model in XML therefore, is to create tags for each class. The format for this is `<class type=''sterotype'' name=''name_of_the_class'' id=''classid''>`. `stereotype` is one of the four BO-types, and `name_of_the_class` is a mandatory attribute representing the class name in the UML-model. Furthermore, all classes have a mandatory attribute `id`. These id's are unique.

The attributes of each class are represented by `attr` XML tags. These tags have no further attributes. The same applies to methods, which are represented by `method` tags.

Mapping associations is less straightforward. This is done using a `role` tag inside the tag holding the class-name. This tag holds the role-name, and has a mandatory attribute `peer` to indicate the id of the related class. Composition is mapped using a `composition` tag in the class. The 'composites' are defined completely (starting with a class-tag) within this composition-tag. Specialisation is mapped using a `specialisation` tag within a class. The format for this is `specialisation ref=''id'' />`. `id` refers to the definition of a specialised class.

Finally, the UML-model can have OCL-constraints, which must also be mapped to XML. This is achieved through a `ocl` tag. The `ocl` tag contains tags

[10] actually, the DTD developed in this chapter is a stripped down version of the UXF format.

for doc's, pre- and post-conditions, context, and types. These tags have no attributes.

4.5 Example

The example presented in this section is a continuation of Section 3.3, particularly Figure 3.5 on page 21. One of the classes in this figure is `checking process`. Using the strategy and the DTD of the previous section, this class can be mapped to XML.

- The name of the class is `checking_process`. Thus, the top-level tag for this class is `class`, with attributes `type=''process''` and `name=''checking_process''`. Furthermore, the `id` parameter must be provided.
- The class has two attributes; storekeeper and parcel. Therefore, there must be two tags `attr` nested in the `class` tag with these attributes as values.
- The class also has two methods; check contents and notify. Thus, there must be two `method` tags nested in the `class` tag with these methods as values.
- The class has an OCL-constraint, represented by an `ocl` tag. Nested in this tag are the `doc-`, `context-`, `type-`, `pre-` and `post` tags.

Figure 4.7 shows the XML representation of this class. The entire XML-representation of the UML document can be found in Appendix B.

```
<class type="process" name="checking_process" id="pro1">
    <attr>storekeeper</attr>
    <attr>parcel</attr>
    <method>check_contents</method>
    <method>notify</method>
    <role peer="th1">is_checked_in</role>
    <ocl>
        <doc>
            if the check contents method returns
            false, then someone must be notified
            that the checked parcel was not ok
        </doc>
        <context>checking process.check contents()</context>
        <type>Boolean</type>
        <pre />
        <post>
            result=self.check contents != true implies self.notify
        </post>
    </ocl>
</class>
```

FIGURE 4.7: UML TO XML CONVERSION (PARTIAL)

Chapter 5

Semantic Matching

In the previous chapters, a framework for describing and representing Business Objects (BO's) was presented. These corresponded to the first two steps in Figure 1.2. This framework was based mainly on literature on BO-technology, UML and XML. In this chapter, a different approach is needed, for it deals with the *Similarity/matching algorithm* from Figure 1.2. An algorithm for *semantically* matching two specifications (Business Object Frameworks), expressed in XML needs to be derived in this chapter.

Section 5.1 describes what is meant with *semantics*. A definition is derived based on examples from literature. Also a clearer formulation for *semantic match* between two specifications is presented. These definitions are the basis for Section 5.2, which describes the algorithm. It turns out that literature from the field of *Machine Learning* and *Information Retrieval* help in deriving this algorithm.

5.1 Semantics

This section attempts to give a definition of the concept *semantic matching*, more specifically: semantically matching two specifications (Business Object Frameworks, expressed in XML). First, an overview of literature on semantics (what is meant by the term 'semantics'?) is given. It turns out that many definitions of this concept are in use. Based on this discussion, a working definition is derived. This definition is the basis for explaining semantic matching in the context of this thesis.

Even though many authors use the concept 'semantics', a clear description is hard to find in literature. Several examples of how semantics are used in literature illustrate this. In (Owei and Navathe, 2001) 'semantic datamodels' and 'relationship semantics' are used to define a Concept-based

Query Language (CQL). CQL uses concepts that are as close as possible to those in the end-users' mental model to query databases. Semantic clustering techniques are used in (Missaoui and Sahraoui, 1998) as part of a methodology for handling database migration. The authors briefly mention that semantic modelling uses 'high-level constructs for structuring data', but no more than that.

A more detailed discussion about semantic modelling is found in (Mirbel, 1997), where semantics are used to define a methodology for integration of conceptual schemas at a semantic level (which is similar in many respects to the approach taken in this thesis). The authors state that "In order to integrate (design) schemas, their elements (attributes, methods, classes and links) must be compared.". The criteria for this comparison -at the semantic level- are *names* and -at the structural level- *roles*. Two measurements are used in this approach:

- *semantic likeness*: words are no more than *tokens* to represent a certain *concept*. Several words (synonyms) can be mapped on the same concept.
- *semantic ambiguity*: one word can have several meanings (homonyms). These words are said to be ambiguous.

This approach makes sense, because in (Hoppenbrouwers, 1997) it is shown –amongst other things– that natural language (and therefore 'words') play an important role when examining a system.

Since no single, clear definition of semantics was found in literature, a dictionary was used. According to WordNet[1], semantic means *of or relating to the study of meaning and changes of meaning; "semantic analysis"*. This definition is in agreement with definitions taken from other dictionaries (e.g. the online dictionary wordsmyth[2]).

In other words, the semantics of some concept x deals with what x means, rather then what its structure is. This distinction is important, for many authors define semantics as the opposite of structure (e.g. (Mirbel, 1997)). For example: the semantics of the concept 'car' could be 'a machine for transportation, also called automobile'. Note that no remarks about the structure of a car (it consists of four wheels, a motor, etc.) are included here. This leads to the following definition of a semantic match:

Definition 5.1 (Semantic match)
A semantic match between two concepts is a measurement of how similar the meaning of two concepts is. It is based on semantic likeness (synonyms) and semantic ambiguity (homonyms), and is considered to be the counter part of the structural similarity of these concepts.

[1] http://www.cogsci.princeton.edu/~wn/
[2] http://www.wordsmyth.net/

Hence, the semantic matching activity deals with finding out how similar the *meaning* of two (or more) things are. In the context of this thesis these things are, of course, Business Object Frameworks. Suppose, for example, that one has two computer-systems. The first system is a tradition Enterprise Resource Planning (ERP) system which is used to support the entire business from purchase to sales. The second system is a newly developed webshop. Ideally these two systems should be coupled. However, this is currently not the case. In order to find out whether integration is possible, the two systems should be compared (e.g. (Mirbel, 1997)). Following the approach in this thesis, both systems are modelled as Business Object Frameworks, and an algorithm for finding the semantic match (derived in the next section) is used. The result of this algorithm must be interpreted in order to decide whether integration is possible or not.

5.2 Deriviation of the Algorithm

The algorithm in this thesis is based on approaches described in (van den Heuvel, 2002) and (Zaremski and Wing, 1995). In (van den Heuvel, 2002), the BALES methodology (binding Business Applications to LEgacy Systems) is discussed. The methodology aims at "constructing parameterisable business objects to link (off-the-shelf) business components to legacy system components on the basis of their interface definitions". In this paper, an algorithm is derived that retrieves the "semantically relevant" legacy specifications. To achieve this, both the legacy system, and the business components are represented in a *common ontology*; currently WordNet. The calculated distance between these specifications[3] is calculated. All legacy specifications whose distance to the business components are within a specified range (smaller than a predefined threshold value) are retrieved. Hence, the algorithm can also deal with 'partial matches'.

In (Zaremski and Wing, 1995) a methodology for finding software components that conform to a (formal) specification at the semantic level is discussed (the context is "software reuse and library retrieval"). In this methodology the behaviour of software components are described using a formal notation, in terms of pre- and postconditions[4]. The difference between an *exact match* and a *relaxed match* are discussed. The approach in this thesis uses the setup (shared ontology) from the approach in (van den Heuvel, 2002), and uses the notion of *partial match* as introduced in (Zaremski and Wing, 1995).

The goal of this section is to derive an algorithm that takes two Business Object Frameworks (BOF's) as input. It calculates how similar they are and returns this similarity. Recall that a Business Object (BO) is a representation

[3]The details on how this is calculated is not covered in this thesis.
[4]Details on what pre- and postconditions can be found in (Warmer and Kleppe, 1999).

of a person, place, thing or process in a given business domain (See Definition 2.1 for details), and that it is represented using a UML-class diagram. A BOF uses BO's to represent the (computer) systems in an organisation (Definition 2.2).

Two high-level approaches must be considered for this algorithm. In the first approach, comparing two BOF's is done by comparing individual BO's and later combining the results of these (many) comparisons. The advantage of this approach is that it is very intuitive; first one finds BO's in both specifications that are similar by examining attributes and methods, and later combine these results into a single metric. The disadvantage is that computers do not *understand* the data that they are processing (e.g. (Cover, 1998)). This results in the fact that all BO's from one BOF must be compared to all BO's from the other BOF. In other words: the overhead would be significant. A second problem arises when the results of the individual comparisons must be combined into a single metric. This would involve a good deal of math, with a high risk of blurring the interpretation of the results.

The second approach is more direct: the two BOF's are compared in an integral manner; that is, they are treated as *holistic entities*. The downside of this approach is that it is less intuitive then the first approach: there is no obvious way of how to do this comparison (the next paragraphs will show that *feature selection techniques* and the *document vector model* can be used to this end). The advantage of this approach is that it fits in an entire step, and that -hopefully- the output is easy to interpret.

The second approach is used in this thesis, which brings up the question: 'How can two BOF's be matched in an integral manner?'. The first step to be taken is the extraction the (semantically) relevant information from the two BOF's. Similar problems are tackled in the research-field called *Machine Learning* (ML). The ML research field deals with the question of how to construct computer programs that automatically improve with experience (Mitchell, 1997). An example of an application is a program that leans to detect fraudulent creditcard transactions by examining a set of training-material. In order for the program to learn efficiently, the correct *features* must be selected.

In ML, objects are called *instances*. These instances represent real-world things (e.g. creditcard-transactions, or words) with a set of features. These features must be selected carefully. In ML this process of feature selection is based on statistics such as *Information Gain*, which measures the quantitative worth of a feature (Mitchell, 1997). In this thesis, the features must be selected from the BOF. This BOF was originally expressed in UML, but later converted to XML so that a program can read- and work with it.

In (Mirbel, 1997) it is shown that *synonyms* and *homonyms* play a great role in semantic matching (see also the discussion in Section 5.1). Therefore, the

feature selection in this thesis is based on the *nouns* in the BOF's. This is done with a so called *tagger*[5]. Hence, all nouns must be extracted from the BOF'. This means that:

1. All non-nouns are currently ignored, even though they might have some (relevant) semantic information. Incorporating these wordcategories in the algorithm is left for future research.
2. All structural aspects such as the 'structure' and 'complexity' of the model (that is, the XML-tree) are not in the algorithm. Currently it is unclear how this data is related to the problem definition (semantic matching). Future insights in this field could result in incorporation in the algorithm.

In short, the first step comes down to extracting a list of nouns from each BOF. These two lists are matched in the second step. This seems simple: compare all the words one by one and count the number of 'matches'. However, the reality is that the matching algorithm is not so straightforward. Problems occur with homonyms and synonyms.

Suppose two BOF's have a homonym (e.g. *note*) with different meanings in each BOF (e.g. "musical note" and "a paper note with something noteworthy"). Then, using our "simple algorithm", both words would match incorrectly. Now, suppose two BOF's have synonyms in them (e.g. "client" and "customer"). These two synonyms would not match, even though they should.

Obviously, the problem is that the matching algorithm has no understanding of the words (similar to the fact that computer programs do not *understand* the data they are processing (Cover, 1998)). The solution for these problems is twofold. First, the two sets of nouns must be *enriched*; semantically relevant data must be added to deal with different meanings of the words (tokens). A system called *WordNet* (Secton 5.2.1) is used for this. Secondly, the matching algorithm must be improved/extended. The *document vector model* (Section 5.2.2) -from the field of *information retrieval*- is applied.

5.2.1 WordNet

WordNet is an on-line lexical reference system, inspired by psycholinguistic theories (Miller et al., 1993). It started as a proposal for a more effective combination of traditional lexicographic information and modern, high-speed computation. Pyshcolinguistics is an interdisciplinary field of research,

[5]A tagger is a Machine Learning program that predicts which word-category (e.g. 'noun' or 'verb') a word has. The tagger that is used in this thesis can be found at `http://ilk.kub.nl/~zavrel/tagtest.html`

concerned with the cognitive bases of the linguistic research field. Research in this field resulted in the notion that information in a lexicon must be more then an alphabetised list of words and their meanings (such as the respected and well known dictionary *Oxford English Dictionary*) (Miller et al., 1993). Lexical information -such as *word associations*- must be incorporated as well. WordNet is the result of this research, and is organised into word meanings (*synsets*).

WordNet divides the lexicon into five categories: nouns, verbs, adjectives, adverbs and function words. For this thesis, only the nouns are of interest. According to (Miller et al., 1993), lexical semantics begins with the recognition that a word is a conventional association between a lexicalised concept and an utterance that plays a syntactic role. That is, word tokens are associated with word meanings. These can be represented in a Lexical Matrix (see Figure 5.1).

Word Meanings	Word Forms				
	F_1	F_2	F_3	...	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
$\vdots$				$\ddots$	
M_m					$E_{m,n}$

FIGURE 5.1: LEXICAL MATRIX
(taken from (Miller et al., 1993))

In Figure 5.1, word forms are listed as headings for the columns; word meanings as headings for the rows. An entry in a cell of the matrix implies that the form in that column can be used to express the meaning in that row. This results in word forms F_1 and F_2 sharing word meaning M_1: they are *synonyms*. Word meanings M_1 and M_2 share word form F_2: word form F_2 is a *homonym* (it is polysemous).

Other relations (antonymy, meronymy and morphological relations) are represented in WordNet as well. However, they are not of interest for this thesis. Section 5.2.3 will show how the *synonymy*- and *homonymy* relations are used in the matching algorithm.

5.2.2 Document Vector Model

IR is a science that is concerned with finding the relevant parts in a (large) set of documents, based on some query. A formal definition of IR is given in (Fuhr, 1995). The definition is quite technical and formal, but it comes down to the following: documents and queries can be translated into compatible

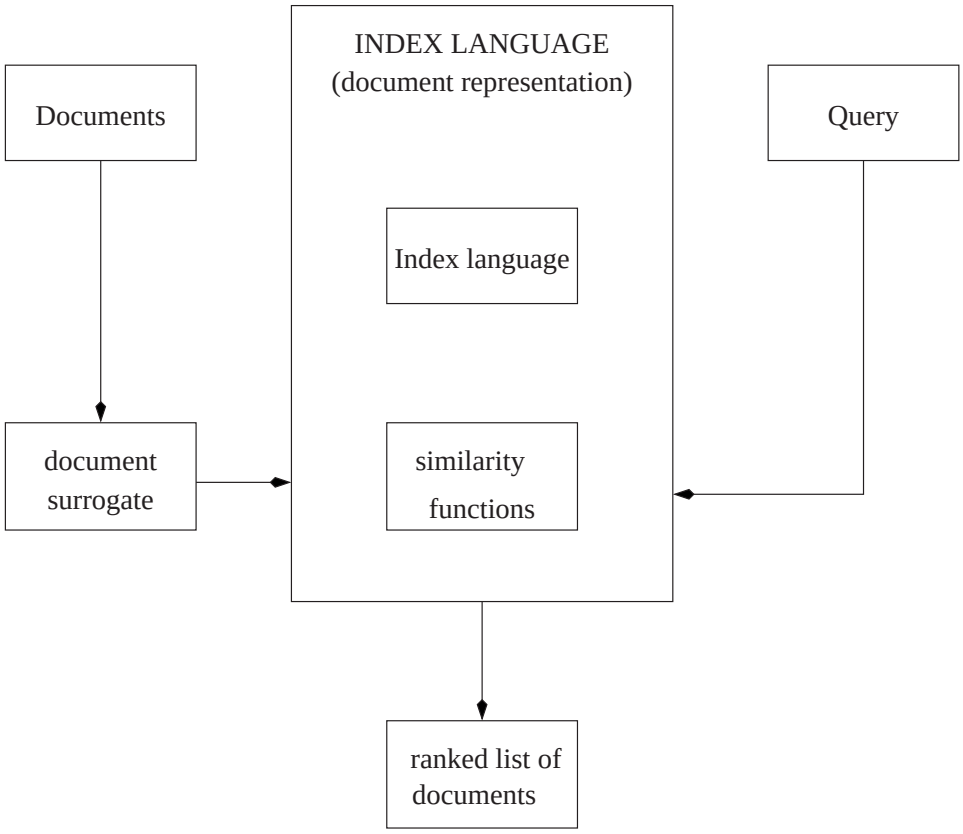

FIGURE 5.2: CLASSICAL MODEL OF IR (PAIJMANS, 1999)

representations in such a way that the semantic representations of these two have a relevance relationship.

Paijmans (Paijmans, 1999) defines that "The Function of any IR system is to extract relevant items from texts; translate them into the symbols of an index language; arrange these symbols to improve accessibility and offer them to the prospective user.". Figure 5.2 is a graphical representation of the classical model of IR. This figure explains the general concepts of IR mentioned above.

As Figure 5.2 shows, several aspects of IR should be taken into account. These aspects are represented by boxes in the figure. Their meaning can be summarised as follows:

- The *document surrogate* is the part of the document that is the input to the IR system. For example, one may choose to enter only a part of the document into the system (the abstract, the conclusions, the captions) or choose to enter the full text into the system.
- The *document representations* are constructed by the system from the document surrogate. They are stored as the representation of the

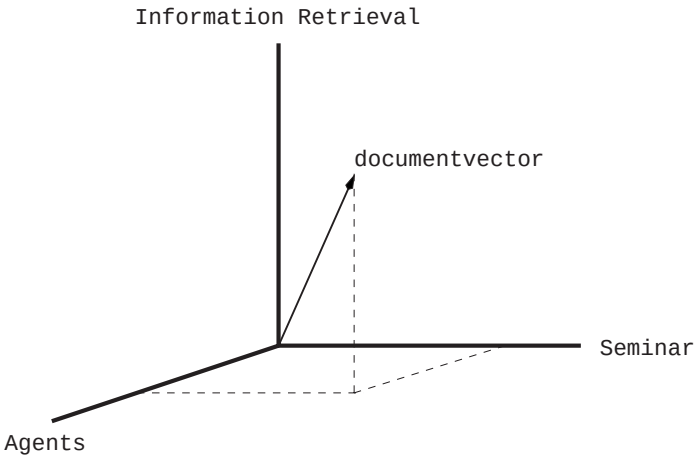

FIGURE 5.3: GRAPHICAL REPRESENTATION OF THE DOCUMENT VECTOR MODEL

original documents in the symbols of the index language. They usually are a set of keywords.

- The *query* is the question to which the user of an IR system would like to see an answer. Before it is checked with the document representations, the system also converts it to the index language.
- The *similarity functions* are (usually mathematical) functions to check which documents are relevant and which are not, for the selected query.
- The *ranked list of documents* is the result of the system, for that is the purpose of it!

One thing that needs to be looked at more closely is the moment of creating the index (document surrogate). Two situations are possible. First of all, it is possible to combine the terms in an index prior to searching. This system is called *pre-coordinative*. A second possibility is to only allow short terms, and leave it until the moment that a search is done to create the index. Such a system is called *post-coordinative*.

Another aspect that is relevant in this context, is the way the translation to the index language is done. A distinction must be made between *assigned* versus *derived* indexing. The difference between the two is whether the keywords that are chosen for use in the index language are taken from the document itself (derived indexing) or from an independent list of terms (assigned indexing). The former can be done by computer, but the latter is usually done manually.

A related aspect is the manner in which words are weighed in an index language. These wordweights come in two flavours. In (Paijmans, 1999) these are introduced as *plain wordweights and word-document weights*.

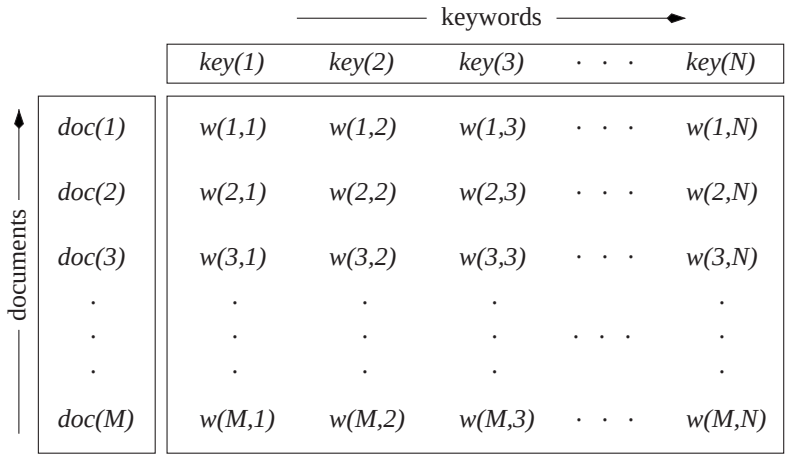

FIGURE 5.4: DOCUMENT DATABASE IN THE DOCUMENT VECTOR MODEL

Plain wordweights are characterised by the fact that the weight is related only to the keyword itself. Simply counting how often a keyword occurs, or a binary representation (1 means that the word occurs, 0 means that it doesn't) are examples of this type of weights. The word-document weights, however, have another characteristic. They take into account that large texts contain more words than smaller texts. Therefore, the *tf.idf* of weights distinguishes between words that occur frequently in *all* documents, and words that occur only a few times in (a small number of) documents. The latter are –often– a good indicator for the contents of these documents. For technical details see (Salton, 1989).

The document vector model is based on the notion that documents can be represented by vectors of keywords. For example, suppose a language exists with only three keywords: "seminar", "agents" and "Information Retrieval". A document vector has, in this situation, three dimensions. To be more specific, in Figure 5.3 the three dimensions are represented by the axis, and a document dealing with all three keywords (e.g. this paper) is represented by the arrow.

Several types of weights can be used for this. The document database can be represented as a term-document matrix with documents as rows and keywords as columns. Figure 5.4 depicts this type of document database.

With the document vector model, similarities with a query can be calculated in several ways. In (Norealt et al., 1981) several of these functions are discussed. The basic principle on which these functions are based, is that vectors (documents) that are related are *closer* than unrelated vectors. Three examples (taken from (Paijmans, 1999)) of similarity functions on documents d_j and d_k are:

- The *matching coefficient* counts the number of dimensions on which both documents have a non-zero entry:

$$sim(d_j, d_k) = |d_j \cap d_k|$$

- The *Jaccard coefficient* is the matching coefficient, penalising for a small number of shared entries:

$$sim(d_j, d_k) = \frac{|d_j \cap d_k|}{|d_j \cup d_k|}$$

- The *Euclidian distance* is the well known angle between two n-dimensional vectors:

$$sim(d_j, d_k) = \sqrt{\sum_{i=1}^{m} (d_{ji} - d_{ki})^2}$$

These similarity functions measure -on a scale from 0 to 1- how similar two vectors (and thus, two *documents*) are. Section 5.2.3 will show how the document vector model is used in the matching algorithm.

5.2.3 Proposed Algorithm

The theoretical basis for this algorithm was described in previous chapters. Furthermore, Section 5.2.1 describes the working of WordNet and Section 5.2.2 describes the document vector model. This section, finally, describes the matching algorithm by combining these.

The algorithm –which takes the XML version of the BOF's as input– is shown in Figure 5.5. The first step shown in the figure is the extraction of all words in both XML-specification. The names of the XML-tags are not included. The result of this step is two lists of words.

The second step is the selection of all nouns from both lists. For this end a *tagger* (see page 35) is used. This step results in two lists with the nouns from each XML-specification.

The third step uses WordNet to add semantic information to the list with nouns. For each word, all word meanings are selected. In turn, all synonyms for each word meaning are taken from WordNet, and added to the noun-lists. Hence, the result of this step are two (longer) lists with nouns.

These two noun-lists are combined in the fourth step. A 'wordspace' is generated: this is a list of all nouns that exist in both specifications. The wordspace is used to generate a vector for each specification, using *binary*

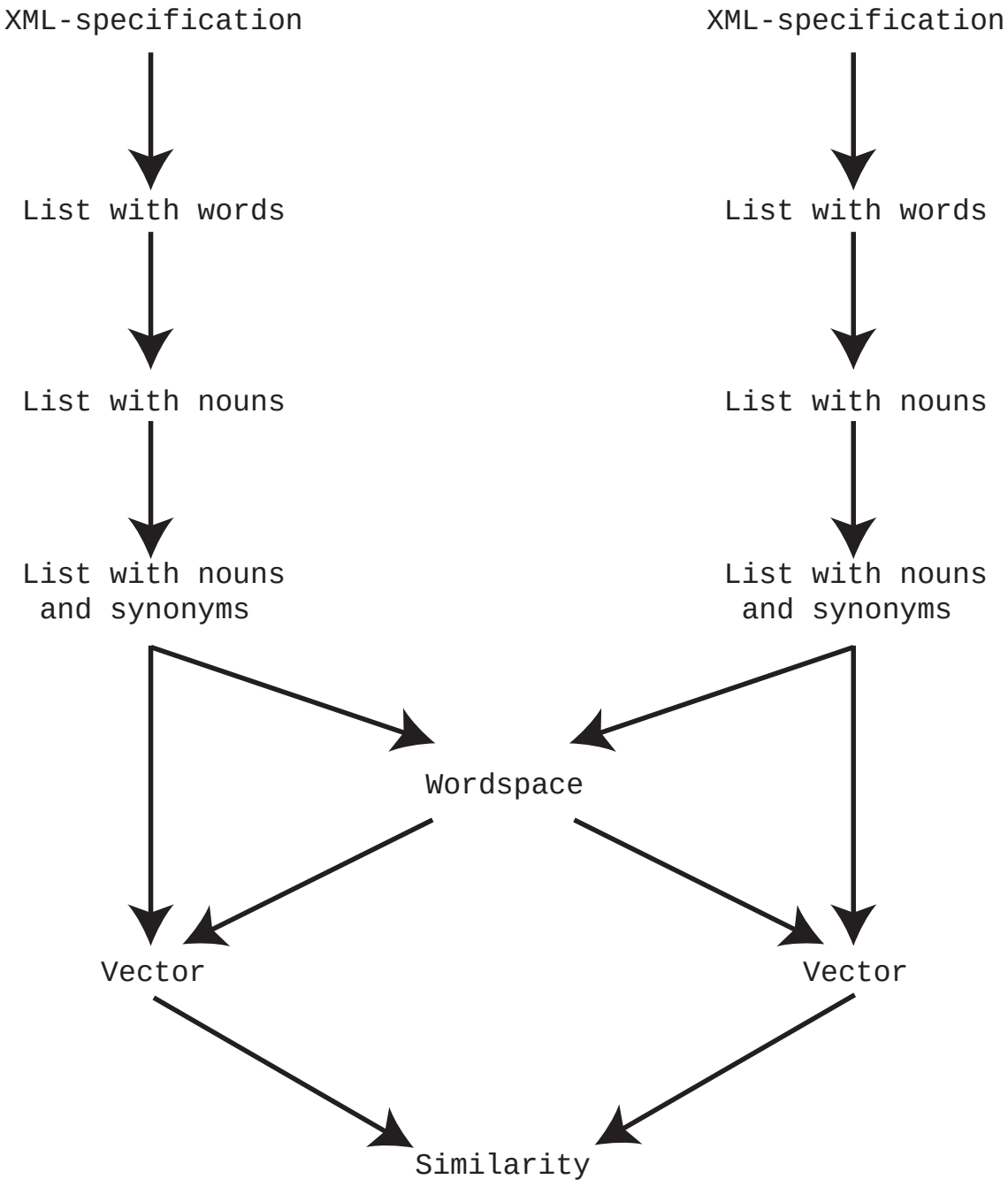

FIGURE 5.5: SEMANTIC MATCHING ALGORITHM

wordweights (see Section 5.2.2 for details). This vector has as length the amount of words in the wordspace; the words that exist in the noun-list (step three) get a 1, the nouns that do not appear in this list get a 0.

The fifth, and last, step compares the two vectors that were generated in step four. The *Jaccard coefficient* is used for this, because it penalises for a small number of shared entries: if the amount of synonyms added by WordNet (step three) is big, then there is a high risk of getting many entries that are *not* shared between specifications. This would result in a (much to) low similarity value. This is partly corrected by putting a penalty on a small number of shared entries. The result of the fifth step is a similarity-value that lies between zero (no match) and one (full match). A high value implies a better match.

The result of the algorithm must of course be interpreted. The goal was to find a metric that helps improve the process of enterprise application integration (see Section 1.2). Interpreting the results of the algorithm is fairly intuitive. If the semantic match is near zero, then the two systems have little to do with one another. Integrating those systems is expected to be very difficult. If the semantic match is near one, then the two system are expected to be nearly identical. Integrating them is probably an 'easy task'. For values in between, the simple rule "the higher the match, the better the semantic match" is applicable. Additional research can be used to identify which parts of the two systems are similar can be done. This is, however, not covered in this thesis. The algorithm is demonstrated in the next chapter.

5.3 Limitations

In the current implementation, the algorithm is highly dependent on the *tagger*, and on *WordNet*. This means that both the tagger and WordNet must be of high quality if the algorithm is to perform as expected.

The quality of the tagger is discussed in (Daelemans and Zavrel, 1996). The authors have conducted three experiments to test the tagger (which is based on *memory based learning* (MBL), see e.g. (Mitchell, 1997) for details). The experiments are based on 'common practice in Machine Learning': independent training- and test sets are used in combination with a 10-fold cross-validation approach. Based on their experiment, the authors conclude that "a memory-based approach to large-scale tagging is feasible both in terms of accuracy ... and also in terms of computational efficiency" (Daelemans and Zavrel, 1996). The following accuracies are reported:

	accuracy	*percentage*
known words	96.7	94.5
unknown words	90.6	5.5
total	96.4	100.0

The table shows that 94.5 percent of the words that were used in the experiment were 'known' by the tagger. Of these words, 96.7 percent was classified correctly. Of the 'unknown' words (5.5 percent), 90.6 percent was classified correctly. This leads to an overall performance of 96.4 percent.

Unfortunately, no papers were found on the quality / performance of WordNet. However, (Miller et al., 1993) suggests that the (psycho) linguistic framework underlying WordNet is correct. Small experiments with the WordNet system (e.g. (Leon, 2000)) suggest that it performs quite well: "WordNet has proven to be an excellent source of English words, their definitions, and their interrelations.".

Also note that the algorithm has some intrinsic limitations. First of all, it treats specifications as *holistic entities*, it is difficult to deal with 'partial matches': using the algorithm alone, it is unclear which individual BO's from one specification are similar to BO's from the other specification. Secondly, the algorithm only takes into account the *nouns* in the Business Object Frameworks (based on the results presented in (Hoppenbrouwers, 1997; Mirbel, 1997)). Other features (other word categories such as verbs and adverbs, as well as features dealing with the complexity/detaildness of the models) are important as well. In the current implementation of the algorithm, however, they are ignored.

Chapter 6

Proof of Concept

The algorithm for semantic matching that was derived in the previous chapter(s) is tested in this chapter. The ultimate goal of the algorithm was to find out how easy two systems can be integrated (Enterprise Apllication Integration, or EAI). The way this is done is through computing how similar two Business Object Frameworks are and using this as an indicator for ease of integration.

There are two strategies for checking the (results of) the algorithm: *proof of concept* and *experimentation*. Both are used in this this thesis. Firstly, the algorithm is implemented in a small computer program. The results of this program are verified by comparing its results with the opinion of a *human expert*. The comparison consists of checking if the algorithm distinguishes between systems that are similar and systems that are dissimilar. Furthermore the level of similarity (exactly *how* similar are the two systems) is checked.

This chapter is organised as follows. In Section 6.1 the details of how the experiment was conducted are described. The experiment is set up as a five-step process that starts with constructing sample data and ends with comparing the results of the human expert with the results of the algorithm. In Section 6.3 the current version of software that was developed for this thesis is discussed. Section 6.4 compares the results of the software with the opinion of the human expert, and suggests some improvements on the algorithm.

6.1 The Experiment

A small experiment was constructed in order to find out if the algorithm recognises two systems that are similar. The context of the experiment is

Enterprise Application Integration (EAI, see Section 1.1). Since the process of integrating (computer) systems turns out to be extremely difficult and costly, the goal of this thesis is to find out how easily two enterprise systems can be integrated. In this section the algorithm that measures how similar two systems are (the assumption is that a higher similarity results in an easier integration process) is tested using a (small) experiment.

The setup for the experiment is based on (Foltz, 1996). In this article, the effectiveness of a so called *Latent Semantic Analys* (LSA, a statistical model of word usage that permits comparisons of semantic similarity between pieces of textual information) is tested. This is achieved by comparing its results with predictions made by two human raters. The two human raters, who are 'highly familiar' with the field, performed their task independently of each other, and without prior knowledge of the results of LSA.

In this thesis, the results of the derived algorithm must be verified. Hence, its results will be compared with the opinion of a (single) human expert. The obvious way to do this is to take three specifications (of which two specifications are similar and one unrelated) as a basis. Independent of each other, the human expert and the algorithm estimate / calculate how similar the specifications are. In short, the experiment consists of the following steps:

1. The first step is to gather three BOF's. One BOF is already explained in Section 2.3 (the UML model is in Figure 3.5 on page 21). Two more examples are summarised in Section 6.2.
2. The second step is the creation of a program that takes two XML-representations of BOF's as input and performs the semantic match (See Figure 5.5). The details on this program are in Section 6.3.
3. With the examples and the program ready, the similarity between the three specifications can be computed. These results are summarised in Section 6.4.
4. The three specifications are shown to a *human expert*: Hans Weigand (`H.Weigand@kub.nl`). He was asked to estimate how similar the three BOF's are (a percentage), as well as a short explanation on his choice.
5. The results of the human expert are compared with the results of the algorithm in Section 6.4.

6.2 Examples

In the previous section it is explained that three examples are needed for the experiment. The first example (*Order checking*) is already discussed in

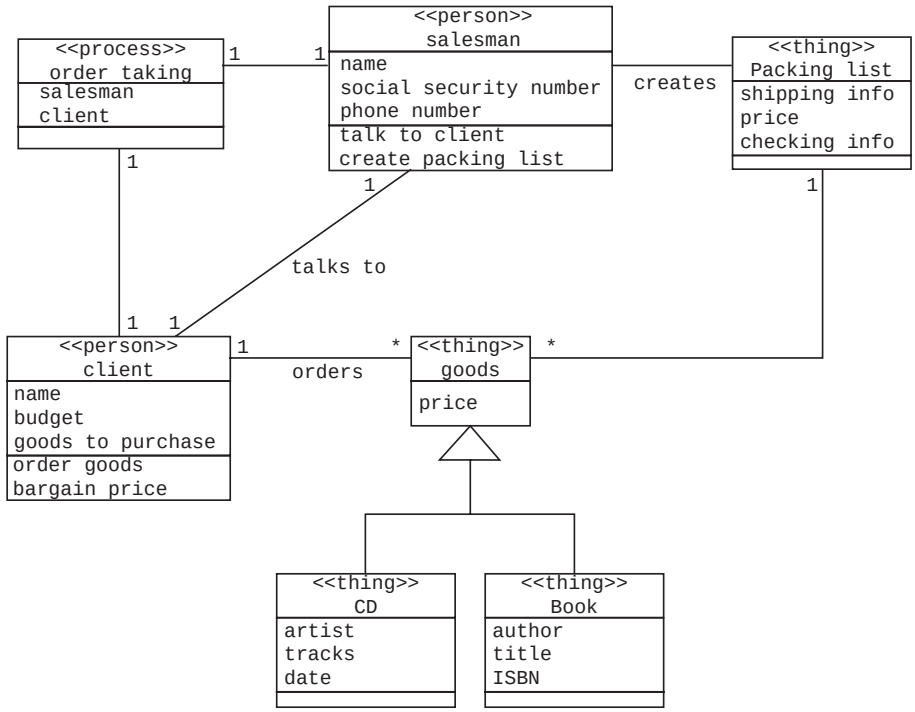

FIGURE 6.1: UML REPRESENTATION OF THE TAKING ORDER PROCESS

Section 2.3 (the UML model is in Figure 3.5 on page 21). This means that two more examples are needed. In order to verify that the algorithm differentiates between specifications that are similar and specifications that are dissimilar, two of the examples are to be similar, and that the other is to be (totally) unrelated.

In the following sections two more examples are discussed briefly: *Order taking* and *Making backups*. The order taking process is designed to be similar to the order checking process. The backup process is designed to be unrelated to the two other specifications

6.2.1 Order Taking

Clients can call to one of the salesmen from our example company. After mentioning which goods he/she might want to purchase, a price can be bargained. The salesman creates a packing list based on the goods that a client wishes to purchase. These goods can be either cd's or books. The UML representation of the model can be found in Figure 6.1.

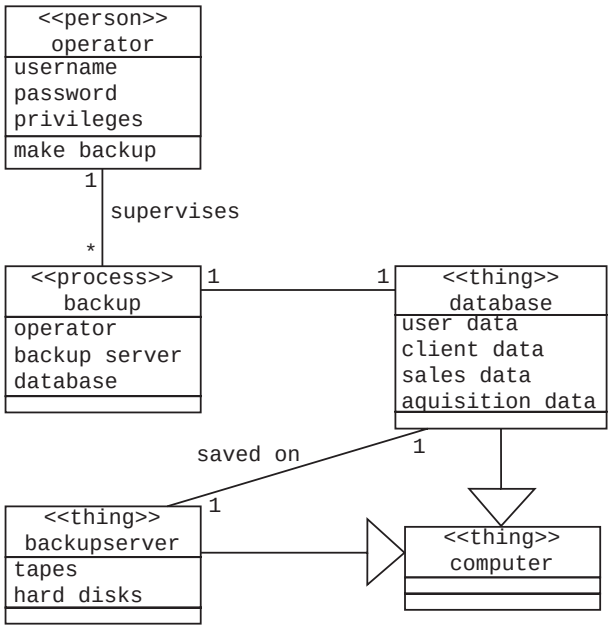

FIGURE 6.2: UML REPRESENTATION OF BACKUP PROCESS

6.2.2 Making Backups

Data in (online) companies is often very valuable; so a good backup system is needed. An operator starts the backup process. Databases with all sorts of data (users, clients, sales and aquisitions) are sent to a backupserver. Backups are stored on either a disk or a tape. The UML representation of the model can be found in Figure 6.2.

6.3 Software

Part of the experiment was implementing the algorithm described in Section 5.2.3. The algorithm must take XML representations (which are valid according to the DTD in Appendix A). The steps described in Figure 5.5 must be followed. The algorithm is implemented in the programming language *Python*[1]. In short, the program has to perform the following steps:

1. Read the XML file and parse it. That is, extract all words that are between XML-tags, and the names of the XML-classes from the XML document
2. Use a *tagger* to select only the nouns from the list with words

[1]The details on the programming language are not included in a thesis. For introductions see e.g. `http://www.python.org/doc/Intros.html`

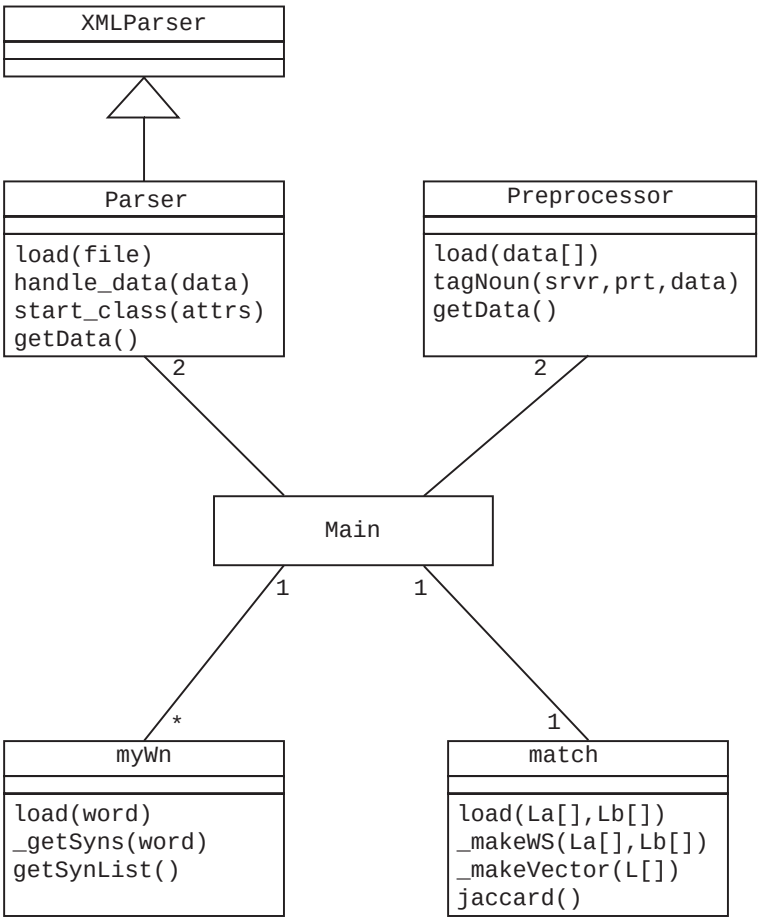

FIGURE 6.3: IMPLEMENTATION OF THE ALGORITHM
The source code of this program is in Appendix C

3. Use WordNet to add all synonyms of all word meanings for each noun in the list.

4. Match the two lists using the document vector model:

   (a) generate a wordspace

   (b) generate two vectors

   (c) calculate the distance between the two vectors using the *Jaccard coeffiecient*

In the current implementation, each step in the list corresponds to a Python class. This results in the classes `parser` (is a subclass of the default `xmllib.XMLParser`), `tagger`, `myWn` (heavilly uses the default class `wntools`) and `match`. Classes were used for two reasons. First of all, they provided a convenient way to structure the program so that the code can be built in an incremental manner. The second reason is that, using the object oriented paradigm makes it easy to adapt or extend the program. For

example, currently the algorithm uses the Jaccard coefficient to determine the similarity between two noun-lists (the last step in Figure 5.5). However, at a later point it might be interesting to try a different coefficient (e.g. the *Euclidian distance metric*, see Section 5.2.2). This can be achieved quite easily by creating a new class (and use *inheritance*) for this metric.

In the body of the program (the `main` loop) the classes are instantiated. The details are shown in Figure 6.3, which is a UML class diagram. The code of the algorithm can be found in Appendix C.

The application runs from the UNIX-command line[2] as follows:

```
[bas@kubstu:~/work/scripts$ ./thesis.py example.xml example2.xml
Parsing example.xml...done!
Parsing example2.xml...done!
Retrieving nouns first spec...done!
Retrieving nouns second spec...done!
Add synonyms with wordnet, first spec...done!
Add synonyms with wordnet, second spec...done!
similarity is 0.326693227092
```

The current implementation prints both verbatim status messages that show the progress of the algorithm, and the actual similarity between the two input specifications. At a later stage other useful statistics can be printed at runtime as well, such as the length of the noun-lists and the amount of words that were added by WordNet[3].

6.4 Results

Section 6.2 discusses three examples that are used in this experiment. Furthermore, Section 6.3 discusses the current implementation of the algorithm. The program was fed each possible combination of two specifications from the set of three, after which it calculated the similarity between the two selected specifications.

To be able to verify whether the output of the algorithm makes sense, a human expert (Hans Weigand) was asked to estimate the similarity between the specifications (without showing the results of the algorithm)[4].

[2] It was developed on a linux-machine running *Python* version 2.1.1

[3] Upon implementation, command-line arguments to the program can serve as 'switches' for these options

[4] From a statistical point of view, this proves nothing. An experiment with a large number of specifications and human exports is needed to be able to show –with a certain level of confidence– whether the algorithm works or not. An experiment of that magnitude is out of the scope of this thesis.

specification numbers	*the algorithm*	*the human expert*
1,2	0.326	0.40
1,3	0.008	0
2,3	0.035	0.1

Specification 1: 'Order checking' (page 21)
Specification 2: 'Order taking' (page 48)
Specification 3: ' Making backups' (page 49)

TABLE 6.1: RESULTS OF THE ALGORITHM AND A HUMAN EXPERT

Previous sections explained that two of the three specifications are designed to be fairly similar. The third specification is designed to be totally different then the other two. Hence, it was expected that both the human expert and the program agree that the *order checking*- and *order processing* specifications are similar, and that the *database backup* specification is unrelated to both. Table 6.1 shows the results of both.

The table shows that the human expert agrees with the algorithm that the first two specifications are fairly similar. He makes the following observation: "The first and second BOF match a lot. They use the same concept of 'thing' with subtypes 'cd' and 'book'. They have (at least on the level of words) the same concept of 'packing list'. The other object types are different but this is no problem, it can be considered to be complimentary. The relationship between 'packing list' and 'goods' is clear in the second model, while in the first model there is only an indirect relationship via parcel. But the models can be said to be complimentary.".

They also agree on the fact that the third specification is totally different from the other two specifications. The results are both near zero. The following observation was made by the human expert: "The third model is very different. There are no overlapping objects. The only link with the other two models is in the client data that is stored in the database. The object type 'client' appears in the second BOF. Structurally, the representations are very different. Perhaps there is also a link between acquisition data and the 'things' in the other two diagrams ('cd', 'book'), but both the structure and the naming is different.".

The observations of the human expert are, of course, true. The examples were constructed in such a way that two of them are similar, and that one is totally different. The main difference between the estimate of the human expert and the algorithm is *how similar* specifications one and two are. The estimate is a little (approximately 0.08) higher. This higher estimate is probably due to the way the algorithm works.

Because of the highly specific application domain, a lot of words might not be in WordNet. This could lead to unexpected outcomes of the algorithm. Words might have *many* different meanings; which, in turn, can have many different word meanings. Because they are *all* added, the vectors can grow very long which might lead to unexpected outcomes also. Based on the observations by the human expert, it becomes apparent that the algorithm does not *understand* what it is processing. Humans have the ability to recognise that two objects *might* be similar, or that there is an indirect relationship between objects. On the upside, the algorithm does differentiate between specifications are similar and specifications that are not similar. This suggests that it handles different naming correctly by using WordNet.

6.5 Conclusions and Suggested Improvements

Interpreting the results of the algorithm and the human expert lead to the same conclusion: the *order taking process* and the *order checking process* are -partially- similar. Integrating them, hence, should be relatively easy (see Section 5.2.3 for details). The *backup process* is, indeed, 'totally different'. Integrating this process with one of the others is probably hard / might not be possible (because they have almost nothing in common). This result leads to the conclusions that, even though the experiment is relatively small, it is fair to conclude that the algorithm works as expected.

The analysis in the previous section shows that additional research is needed to find improvements for the algorithm. The following options come to mind:

- The current implementation of the algorithm takes only the *words* (that is a list with nouns, with all synonyms) in a specification into account. Aspects such as the size and complexity of the model / XML-tree can be added, if an appropriate (semantic-based) metric can be found;
- Currently the algorithm uses a binary weighting scheme. That is: words (nouns) occur in a specification or not. Furthermore, the nouns that are added by the algorithm from WordNet also use this scheme. It is likely that the algorithm will perform better if a more elaborate weighting scheme is implemented. This scheme ought to take the following into account:
  - Words that occur in the specification must get a higher score than the synonyms that are added;
  - Words that are important must have a higher score than words that are less important. For example, one could argue that *class names* are more important than *attribute names*. In terms of *Machine*

Learning theory, words with a high *Information Gain* value must get a higher score (see Section 5.2 for details);

- The current similarity metric (*Jaccard* coefficient) penalises the *matching coefficient* for a small number of shared entries. Furthermore, it depends on a binary weighting scheme. With the 'improved weighting scheme' a different similarity metric can be tested as well. This scheme must make use of the new weighting scheme (eg by using the *Euclidian distance metric* (see Section 5.2.2).

Exploring these, and other, options is left for future research.

Chapter 7

Conclusion

Many organisations are integrating their computer systems these days; either because the old systems need replacement, because new (web-based) applications that need integration with back-end systems are introduced, or for some other reason. This process –*Enterprise Application Integration* (EAI)– is both costly and risky.

In the previous chapters an algorithm for supporting the process of EAI is developed step by step. Both the algorithm, and the theory on which it is based are summarised briefly in this chapter after which conclusions are drawn. Finally, suggestions for future research are summarised at the end of this chapter.

7.1 Overview

In Chapter 1 the context and problem definition of this thesis are described. The three main reasons for examining the process of EAI are the high reliance on computer systems, the (monetary) costs that are associated with them, and the fact that many computer systems are (in the process of getting) integrated. Unfortunately, integrating computer systems –especially if they are big and complex– is not always as easy as people think. Simply put: EAI is an important, risky and expensive activity and it is therefore important to pay attention when starting an EAI-project.

EAI is –in this thesis– examined using three dimensions: EAI is examined at the conceptual (as opposed to implementation) level, with a focus on static (as opposed to dynamic) and semantic (as opposed to syntactic) aspects. The basic assumption is that computer systems can be modelled using Business Objects (BO). A BO is a coarse grained abstraction of all artefacts that are needed to represent a person, place, thing or process in a given business domain (see

Definition 2.1). A framework representing a business (application) is called a Business Object Framework BOF (see Definition 2.2).

Given two specifications of two (complex) computer systems. The assumption underlying this thesis is that 'similarity' between these systems is a good indicator for estimating how easy they can be integrated. Hence, the goal of this thesis is to answer the following research question:

> Given the specifications of two systems in terms of Business Objects, how can the similarity (at the semantic level), between these two systems be *calculated* in order to determine how easy these systems can be integrated?

The first step (see Figure 1.2 for an overview of the steps) was to find a notation for these business object frameworks. The *unified modelling language* (UML) –which is the defacto standard for specifying, visualising, constructing and documenting the artifacts of software and business systems– was chosen for this. Since the focus of this thesis is on static aspects of (computer) systems, the class diagram was selected as a notation for BOF's. Additional constraints on the model can be added using the *Object Constraint Language* (OCL). For details on UML and OCL see Chapter 3.

Since UML is a graphical notation, it is not very well suited as input for a computer program. This is inconvenient, since the algorithm is to take two specifications as input, and calculate the similarity between them. Hence, a mapping to a textual notation is to be found. In Chapter 4 a mapping to the *eXtensible Markup Language* (XML) is described. Even though several mappings from UML to XML are in use (such as XMI and UXF) today, a new –simple– DTD is created for this thesis. The details on this approach are described in Section 4.4; the DTD itself can be found in Appendix B.

The concept 'semantics' is used for many things in literature. In Section 5.1 several examples are discussed, and a working definition for this thesis is derived. Two measurements play an important role in this definition: *semantic likeness* and *semantic ambiguity*. The working definition (see Defintion 5.1) is based on them, and basically deals with finding out how similar the *meaning* of two (or more) things are. In other words, the algorithm must find out how similar the *meaning* of two computer systems is. It is based on two pillars: *WordNet* and the *Document Vector Model*. In short, the algorithm takes the following steps (See Figure 5.5):

- Extract all *words* that are in the XML-representations of the BOF's and stores them in two lists;
- Select all nouns from the lists using a *tagger*;

- Use *WordNet* to enrich the two noun-lists by adding synonyms (words can have multiple wordmeanings; in this case all synonyms for each wordmeanings are added to the list);
- Create a *wordspace* and use this to create two vectors, using *binary wordweights*: words that exist in the noun-list get a 1; nouns that do not appear in this list get a 0.
- Use the *Jaccard coefficient* to calculate the similarity between the two noun-lists (and hence, the two specifications of the two computer systems)

Interpreting the results of the algorithm is –to a certain extent– intuitive. If the semantic match is near zero, then the two systems have little to do with one another. On the other hand, if the match is near one, then they are (expected to be) nearly identical. As stated before: it is expected that a higher similarity results in an easier integration process. If the results are unclear, then additional research can be used to identify which parts of the two systems are similar.

7.2 Discussion

A small experiment was designed to test whether the algorithm differentiates between specifications that are similar, and specifications that are dissimilar. Furthermore, the level of similarity is checked. Three BOF's (two of which are similar) were constructed to this end. They are described briefly in Section 6.2. Also, a *Python*-program was built that implements the algorithm. The three specifications were converted to an XML-representation and fed to the program. The results of the program were compared with the opinion of a human expert.

Both the Python-program and the human expert agreed on which specifications are similar, and which are not (exact results are in Table 6.1). The main difference between the estimate of the human expert and the program is exactly *how* similar the specifications are. In short, the estimate for the two similar systems is approximately eight percent higher then the result of the program; for the dissimilar specifications the algorithm and the human expert agree on the level of similarity (deviations are negligible).

As stated before, it is clear that additional research (in the form of a large scale experiment) – with more data (BOF's) and more independent human experts – is needed to verify this result. Furthermore, several improvements on the algorithm can be considered, such as a metric for the complexity / size of the XML-tree and add that to the current model, using a different weighting

scheme based on the 'importantness' of words, and finally the implementation of different similarity function (see also Section 6.5). The following section describes a possible setup for such an experiment.

7.3 Future work

The first thing that must be done is verifying the results presented in this thesis by means of an experiment. The setup of the experiment should contain several elements. First of all, the current version of the algorithm must be tested with a multitude of Business Object Framework specifications, and several independent human experts. Statistical analysis can be used to verify whether the opinion of the human experts differs significantly from the results of the algorithm. To achieve this, all the specifications are matched against one another, thus compiling a list with pairs of specifications as index, and their similarity as value. After the list is compiled it must be sorted (most similar pair first, least similar pair last). If the ranking of the lists is the same –a predefined deviation is allowed– then it is safe to conclude that the algorithm really works.

A similar setup can be used to test additions to the algorithm. Additions are preferably 'modular'; meaning that if they can be added either alone, or in combination with others. The results of the newly constructed version of the algorithm can be compared to the 'old' results to check whether the additions are a (significant) improvement or not.

Assuming that the algorithm works, it might be applicable in other domains as well. A requisite for this is that the task at hand is a *mapping* task, and that specifications / documents are represented in an XML vocabulary. An example is *E-service discovery*, where the right (online) service with the right capability is found to represent a business process.

Taking into account the boundaries of the research sketched in the previous section it is fair to conclude that the algorithm really works as expected. The main contribution of this research, hence, is using *WordNet* and the *document vector model* together for semantically matching (the models of) two enterprise systems: the document vector model –which comes from the field of Information Retrieval– uses WordNet to 'simulate' that the algorithm actually understands the data it is processing.

In conclusion: the algorithm derived in this thesis is a good first step. However, the road is long and many more steps have to be taken to get where we want: easy integration of enterprise applications.

Appendix A

DTD for mapping UML to XML

```
<!-- top-level is the model-node -->
<!-- it holds 1-or-more classes  -->
<!ELEMENT model (class)+>
<!ATTLIST model
        name CDATA #IMPLIED>

<!-- class definition                              -->
<!-- can hold 1-or-more of several uml-constructs -->
<!ELEMENT class (attr|method|role|ocl|composition|
    specialisation|note)+>
<!ATTLIST class
        type (process|person|place|thing) #REQUIRED
        name CDATA #REQUIRED
        id   CDATA #REQUIRED>


<!-- attribute definition -->
<!ELEMENT attr (#PCDATA)>

<!-- method definition -->
<!ELEMENT method (#PCDATA)>

<!-- role definition -->
<!ELEMENT role (#PCDATA)>
<!ATTLIST role
        peer CDATA #REQUIRED>

<!-- ocl definition                           -->
<!-- could contain a lot of different things -->
<!-- therefore 'ANY' applies                 -->
<!ELEMENT ocl (pre|post|doc|context|type)+>
<!ELEMENT pre (#PCDATA)>
<!ELEMENT post (#PCDATA)>
<!ELEMENT doc (#PCDATA)>
<!ELEMENT context (#PCDATA)>
<!ELEMENT type (#PCDATA)>
```

```
<!-- composition definition                          -->
<!-- a class can hold classes inside a composition -->
<!ELEMENT composition (class)+>

<!-- specialisation definition                 -->
<!-- specialisation refers to another class id -->
<!ELEMENT specialisation EMPTY>
<!ATTLIST specialisation
        ref CDATA #REQUIRED>

<!-- note definition -->
<!ELEMENT note (#PCDATA)>
```

Appendix B

XML example

```
<?xml version="1.0"  standalone="no" ?>
<!DOCTYPE model SYSTEM "file://d|scriptie/umlxml.dtd">

<!-- Author: Bas van Gils <bas.vangils@home.nl>                -->
<!-- Date  : October 17th 2001                                 -->
<!-- About : This model is an XML-representation of a UML-model -->
<!-- BEWARE: This model comes with umlxml.dtd                  -->

<model>

    <class type="process" name="checking_process" id="pro1">

        <attr>storekeeper</attr>
        <attr>parcel</attr>

        <method>check_contents</method>
        <method>notify</method>

        <role peer="th1">is_checked_in</role>

        <ocl>

            <doc>
                if the check contents method returns
                false, then someone must be notified
                that the checked parcel was not ok
            </doc>

            <context>checking process.check contents()</context>

            <type>Boolean</type>

            <pre />

            <post>
                result=self.check contents != true implies self.notify
            </post>
```

```
        </ocl>

    </class>

    <class type="thing" name="parcel" id="th1">

        <attr>packing_list</attr>
        <attr>goods</attr>

        <method>add_goods</method>
        <method>remove_goods</method>

        <role peer="pla1">is stored in</role>

        <composition>

            <class type="thing" name="goods" id="th2">

                <specialisation ref="th3" />
                <specialisation ref="th4" />

            </class>

            <class type="thing"  name="packing_list" id="th5">

                <attr>shipping_info</attr>
                <attr>price</attr>
                <attr>checking_info</attr>

            </class>

        </composition>

    </class>

    <class type="thing" name="cd" id="th3">

        <attr>artist</attr>
        <attr>tracks</attr>
        <attr>date</attr>

    </class>

    <class type="thing" name="book" id="th4">

        <attr>author</attr>
        <attr>title</attr>
        <attr>isbn</attr>

    </class>

    <class type="person" name="storekeeper" id="per1">

        <attr>parcels to check</attr>
```

```
        <attr>name</attr>
        <attr>social security number</attr>

        <method>add goods to list</method>
        <method>remove goods from list</method>

        <role peer="pro1">participates in</role>
        <role peer="pla1">seeks parcel in</role>

    </class>

    <class type="place" name="storehouse" id="pla1">

        <attr>parcels</attr>

        <method>store parcel</method>
        <method>remove parcel</method>

        <note>
            isles are sorted using a country code (e.g. UK and NL)
        </note>

    </class>

</model>
```

Appendix C

Python source code of the algorithm

```
import os
import re
import socket
import string
import sys
import time
import wntools
import wordnet
import xmllib

####################
# module variables #
####################
os.putenv("WMHOME","/home/users/bas/wn")
TAGSERVER = "kubsuw.kub.nl"
TAGPORT   = 7124

############
# classes #
############
class parser(xmllib.XMLParser):
    """specialised Parser for XML-files

    the load(file) method reads a file one line at a time,
    feeds it to the inherrited XMLParser.

    the handle_data() method gets rid of needless whitespace
    and punctions before storing all words in  the XML-spec

    the getData() method returns the stored words (list)"""

    def __init__(self):
        """__init__()
        constructor initialises the Parser
        and self.__data"""

        xmllib.XMLParser.__init__(self)
        self.__data = []

    def load(self,file):
        """load(file)
        load the file, read it one line at a time
        feed each line to the Parser"""

        while 1:
            s = file.readline()
            if not s:
                # no more lines was available
                break
            # feed the line to the Parser
            self.feed(s)
        self.close()

    def start_class(self,attrs):
        """handler for class start-tags
        stores the name-attribute in this start-tag"""
        try:
            self.__data.append(attrs["name"])
        except:
            print "Error in method start_tag(): \n\t no attribute 'name' found"
```

```
    def handle_data(self,data):
        """handler for data (stuff between xml-tags)
        only stores data of none-zero length
        also converts whitespace to a single space-character"""

        if len(data) <> 0:
            # get rid of needless whitespace
            # and punctuation ("\W" = whitespace)
            data = re.sub("\W"," ",data)
            self.__data.append(data)


    def getData(self):
        """getData()
        returns the class attribute self.__data"""

        return self.__data

class tagger:
    """specialised tagger: the goal is to take
    a list with word-groups as input and convert it into
    a list with nouns

    the load() method takes a list of word-groups as
    input and split it into words

    the getData() method returns the stored words (list)

    the tagNoun() method takes a list of words as input
    uses a 'tagger' to select only nouns. The noun-list
    is returned """

    def __init__(self):
        """__init__()
        constructor initializes class variables"""

        self.__data = []

    def load(self,data):
        """load(data)
        parse the lines from the passed in data (a list),
        splits each item on whitespace
        and store all unique words in a class variable"""

        for item in data:
            tlist = string.split(item)
            for i in tlist:
                self.__data.append(i)

    def tagNoun(self,SRVR,PRT,data):
        """tagNoun(SRVR,PRT,data)
        loop over the passed in data-list (words)
        use a tagger to select only the nouns
        returns a list with nouns"""

        nouns = []

        # socket to ilk-tagger (Thanks Bertjan :-)
        sockobj = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        sockobj.connect( (SRVR,PRT) )

        # DEBUGGED: server sends a welcome mesg
        sockobj.recv(1024)

        # precompile regexp for speed-optimisation
        # nouns have these tags:
        #     NN   : Noun, common, singular or mass
        #     NNP  : Noun, proper, singular
        #     NNS  : Noun, proper, plura
        patt = re.compile("/{1,2}NN[PS]?")

        for item in data:
            # make sure everything is lowercase
            # before sending it over the socket
            sitem = "%s \n" % (string.lower(item))
            sockobj.send(sitem)

            # DEBUG: keep reading untill
            # the socket returns only whitespace
            rdata = ""
            while 1:
                more = sockobj.recv(1024)
                rdata = rdata + more
                if re.search("\s+",more):
                    # no more data: break from the loop
                    break

            # search for nouns only
            if patt.search(rdata):
                noun = patt.sub("",rdata)

                # DEBUG: there is a single space
                # after each noun which is a problem
```

```
                # for the wordnet connection (class myWn)
                nouns.append( re.sub(" +","",noun) )

        sockobj.close()
        return nouns


    def getData(self):
        """getDta()
        returns the class attribute self.__data"""

        return self.__data

class myWn:
    """specialised acces to WordNet

    load(word) loads a word to the WordNet-connection
    uses the _getSyns() method to store a list with synonyms
    in the class variable synlist

    _getSyns() queries WordNet for the class variable self.word
    all synonyms from all word meanings (just assume it's
    a homonym) are retrieved and returned

    getSynList() returns the list of synonyms in the class
    variable self.synlist"""

    def __init__(self):
        """__init__()
        set class variables to empty strings"""

        self.word = ""
        self.synlist = []

    def load(self,word):
        """load(word)
        load a word and use _getSyns() to querie WordNet
        for it's synonyms. The list of synonyms is stored in
        the class variable self.synlist"""

        self.word = word
        self.synlist += self._getSyns( word )

    def _getSyns(self,word):
        """_getSyns()
        queries WordNet for the class variable self.word
        all synonyms from all word meanings (just assume it's
        a homonym) are retrieved and returned"""

        # synonym-list
        syns = []

        # try to retrieve wordmeanings via wordnet
        try:
            wmeanings = wntools.N[word].getSenses()
            found = 1
        except KeyError:
            found = 0

        # if the word was not found in wordnet
        # append only the word
        # else append all synonymns as well
        if not found:
            syns.append(word)
        else:
            # loop over all word-meanings
            for meaning in wmeanings:

                # convert meaning to string
                # get rid of annoying characters: {},'
                items = string.split( re.sub("[{},']" , "" , str(meaning)) )

                # use a boolean to check  if the string "noun:" is found
                # (after which the nouns are listed)
                found_noun = 0

                # loop over the owrds in a word-meaning-string
                for i in items:

                    # append current word to the
                    # list if the "noun:"-string was found
                    if found_noun <> 0:
                        syns.append(i)

                    # change boolean from 0 to 1 if the
                    # string "noun:" is found
                    if re.search("noun:",i):
                        found_noun = 1
        return syns

    def getSynList(self):
        """getSynList()
        returns the list in the class variable self.synlist"""
```

```
        return self.synlist

class match:
    """class match: specialised vectorspace-based matching class

    __init__(): initializes all class variables (word-lists,
    wordspace, vectors)

    load(1a,1b): loads lists 1a and 1b in class variables
    based on these two lists a wordspace is built (using
    method _makeWS() which is stored in class variable.
    Two vectors are built based on the wordspace, using
    method _makeVector(). Vectors are stored in class variables

    _makeWQS(a,b): makes a wordspace. That is, the two input
    lists are joined (a AND b).

    _makeVector(list): makes a binary vectory (0 if a word is
    not in the wordspace, 1 if it is in the wordspace)

    jaccard(): actual similarity-function. Based on
    Paai's phd-thesis: "explorations in the document
    vector model of information retrieval" """

    def __init__(self):
        """__init__(): initializes all class variables (word-lists,
        wordspace, vectors)"""

        self.match = 0
        self.la = []        # words in spec 1
        self.lb = []        # words in spec 2
        self.ws = []        # wordspace = (a OR b)
        self.v1 = []        # vector 1
        self.v2 = []        # vector 2

    def load(self,la,lb):
        """load(1a,1b):
        loads lists 1a and 1b in class variables based on these
        two lists a wordspace is built (using method _makeWS()
        which is stored in class variable.  Two vectors
        are built based on the wordspace, using method
        _makeVector(). Vectors are stored in class variables"""

        self.la = la
        self.lb = lb
        self._makeWS(self.la,self.lb)
        self.v1 = self._makeVector(la)
        self.v2 = self._makeVector(lb)

    def _makeWS(self,a,b):
        """_makeWQS(a,b):
        makes a wordspace. That is, the two input
        lists are joined (a AND b)."""

        self.ws = a + [i for i in b if i not in a]
        self.ws.sort()

    def _makeVector(self,list):
        """_makeVector(list):
        makes a binary vectory (0 if a word is not in the
        wordspace, 1 if it is in the wordspace) """

        v = []

        # loop over the wordspace
        for i in self.ws:
            if i in list:
                v.append(1)
            else:
                v.append(0)

        return v

    def jaccard(self):
        """jaccard():
        actual similarity-function. Based on
        Paai's phd-thesis: "explorations in the document
        vector model of information retrieval" """

        if len(self.v1) <> len(self.v2):
            sys.exit("Error in jaccard(): v1 and v2 must have same length")

        a_and_b = 0
        a_or_b = 0

        for i in xrange(0,len(self.v1)):

            # if both are 1, then
            # a_and_b (numerator) is increasted
            if (self.v1[i] == 1) and (self.v2[i] == 1) :
                a_and_b += 1

            # if one of the two is 1, then
            # a_or_b (denominator) is increasted
```

```
            if (self.v1[i] == 1) or (self.v2[i]  == 1):
                a_or_b += 1

        if a_or_b == 0:
            # divide through 0 is illegal
            sim = None
        else:
            sim = float(a_and_b) / float(a_or_b)

        return sim



#############
# main loop #
#############
if __name__ == "__main__":

    # two files must be passed in
    try:
        file1 = sys.argv[1]
        file2 = sys.argv[2]
    except:
        sys.exit("Error: need two files")

    # parse the two xml-files
    # returns all words in the xml-files
    sys.stdout.write("Parsing " + file1 + "...")
    Parser1 = parser()
    Parser1.load( open(file1) )
    sys.stdout.write("done!\n")

    sys.stdout.write("Parsing " + file2 + "...")
    Parser2 = parser()
    Parser2.load( open(file2) )
    sys.stdout.write("done!\n")

    # use the tagger-class to
    # extract all nouns from the lists of
    # words (previous step)
    sys.stdout.write("Retrieving nouns first spec...")
    prep1 = tagger()
    prep1.load( Parser1.getData() )
    words1 = prep1.getData()
    nouns1 = prep1.tagNoun(TAGSERVER,TAGPORT,words1)
    sys.stdout.write("done!\n")

    sys.stdout.write("Retrieving nouns second spec...")
    prep2 = tagger()
    prep2.load( Parser2.getData() )
    words2 = prep2.getData()
    nouns2 = prep2.tagNoun(TAGSERVER,TAGPORT,words2)
    sys.stdout.write("done!\n")

    # use the wordnet-class to
    # add semantically relevant nouns
    # (all synonyms of all word meanings)
    sys.stdout.write("Add synonyms with wordnet, first spec...")
    results1 = []
    for w in nouns1:
        myw1 = myWn()
        myw1.load(w)
        t1 = myw1.getSynList()
        # add only the items that
        # were not in the list just yet
        for i in t1:
            if i not in results1:
                results1.append(i)
    sys.stdout.write("done!\n")

    sys.stdout.write("Add synonyms with wordnet, second spec...")
    results2 = []
    for w in nouns2:
        myw2 = myWn()
        myw2.load(w)
        t2 = myw2.getSynList()
        # add only the items that
        # were not in the list just yet
        for i in t2:
            if i not in results2:
                results2.append(i)
    sys.stdout.write("done!\n")

    # use the match-class for matchting
    # first make vectors, then run the
    # similarity algorithm
    mm = match()
    mm.load(results1,results2)
    similarity = mm.jaccard()
    sys.stdout.write("similarity is " + str(similarity) + " \n")
```

Bibliography

Bonar, J. (1997). Business objects for front-office applications: Making domain experts full partners. *OOPSLA*.

Booch, Christerson, Fuchs, and Koistinen (1999). *UML for XML Schema Mapping Specification*. http://www.rational.com/uml/resources/documentation.

Brodsky, S. (1999). *XMI Opens Application Interchange*.

Carlson, D. (2001). *Modeling XML Applications with UML: Practical e-Business Applications*. Addison-Wesley.

Casanave, C. (1995). Business objet architectures and standards. *OOPSLA*.

Cover, R. (1998). *The XML Cover Pages, XML and Semantic Transparency*. http://www.oasis-open.org/cover/xmlAndSemantics.html.

Daelemans, W. and Zavrel, J. (1996). Mbt: A memory-based part of speech tagger-generator. *WVLC*.

Digre, T. (1995). Business application components. *OOPSLA*.

Douma, S. and Schreuder, H. (1998). *Economic Approaches to Organizations*. Prentice Hall, 2 edition.

EDI (2001). *The e-Business framework*. http://www.geocities.com/WallStreet/Floor/5815/.

Flynn, P. (2001). *The XML FAQ*. http://www.ucc.ie/xml/.

Foltz, P. W. (1996). *Latent Semantic Analysis for Text-Based Research*. http://www-psych.nmsu.edu/ pfoltz/reprints/BRMIC96.html.

Fuhr, N. (1995). *Information Retrieval, Skriptum zum vorlesung im ss 93*.

Group, O. M. (2001). What is uml and why is it important? http://www.uml.org/.

Harold, E. R. and Means, W. S. (2001). *XML in a nutshell*. O'Reilly.

Herzum, P. and Sims, O. (1998). The business component approach. *OOPSLA*.

Heumann, J. (2001). Introduction to business modeling using the unified modeling language. Technical report, The Rational Edge, `http://www.therationaledge.com/content/mar_01/`.

Hoppenbrouwers, J. (1997). *Conceptual Modeling and the Lexicon*. PhD thesis, CentER for Economic Research.

Hruby, P. (1998). Structuring specification of business systems with uml (with an emphasis on workflow management systems). *OOPSLA*.

Hung, K. and Patel, D. (1997). Modelling domain specific application frameworks with a dynamic business object architecture: An approach and implementation. *OOPSLA*.

Iyengar and Brodsky (1998). *XML Metadata Interchange Format*. `ftp://ftp.omg.org/pub/docs/ad/98-10-17.pdf`.

Johanesson, P. and Perjons, E. (2001). Design principles for process modelling in eai. *Information Systems*.

King, N. (2002). Eai directions. *intelligent enterprise*.

Laudon, K. and Laudon, J. (1996). *Management Information Systems, Organization and Technology*. Prentice Hall, 4th edition.

Leon, D. J. (2000). *Incorporating the WordNet Lexical Database in an Intelligent Tutoring System*. `http://anny.kinjo-u.ac.jp/ houser/jcp/28.html`.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). *Introduction to WordNet: An On-line Lexical Database*.

Mirbel, I. (1997). Semantic integration of conceptual schemas. *Data and knowledge engineering*, 21.

Missaoui, R. and Sahraoui, H. (1998). Migrating to an object-oriented database using semantic clustering and transformation rules. *Data and knowledgeengineering*.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Norealt, T., McGill, M., and Koll, M. (1981). *A performance evaluation of similarity measures, document term weighting schemes and representations in a Boolean environment*. Butterworths.

OMG (1997a). Business objects architecture interoperability specification. Technical report, OMG Business Objects Domain Task Force.

OMG (1997b). *Unified Modeling Language Glossary*. `www.csci.csusb.edu/dick/samples/uml.glossary.html`.

Owei, V. and Navathe, S. B. (2001). Enrichting the conceptual basis for query formulation through relationship semantics in databases. *Information Systems*.

Paijmans, H. (1999). *Explorations in the Document Vector Model of Information*. PhD thesis, Tilburg University.

Penker, M. and Eriksson, H.-E. (2000). *Business Modeling With UML: Business Patterns at Work*. John Wiley and Sons.

Persson, E. (2000). Business object components? *OOPSLA*.

Popkin (1998). *Modeling Systems with UML*. Popkin Software, `http://www.popkin.com/whitepaper/uml.pdf`.

Ray, E. T. (2001). *Learning XML*. O'Reilly.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Printice Hall.

Salton, G. (1989). *Automatic text processing: the transformation, analysis and retrieval of information by computer*. Addison Wesley.

Shelton, R. E. (1995). Enterprise re-use. *Distributed Computing Monitor*, 10(3rd).

Sutherland, D. J. (1995). The business object architecture: Business objects for corporate information systems. *OOPSLA*.

Suzuki, J. and Yamamoto, Y. (1998). Making uml models exchangeable over the internet with xml: Uxf approach. *OOPSLA*.

Times, L. A. (2001). *Case Study: New England Healthcare EDI Network*. `http://www.cisco.com/warp/public/345/hipaa/docs/`. Februari 4th issue.

van den Heuvel, W. J. (2002). *Integrating Modern Business Applications With Objectified Legacy Systems*. PhD thesis, Tilburg University. Forthcomming.

van den Heuvel, W. J. and Papazoglou, M. (1999a). *Bridging Legacy and Business Components with Parameterizable Business Objects: The BALES Methodology*. Infolab, Tilburg University.

van den Heuvel, W. J. and Papazoglou, M. (1999b). Bridging legacy and business components with parameterizable business objects: The bales methodology. *OOPSLA*.

van den Heuvel, W. J. and Weigand, H. (2000). Cross-organizational worklow integration using contracts. *OOPSLA*.

w3c (2000). *Extensible Markup Language (XML)*. World Wide Web Consortium, `http://www.w3.org/XML/`.

Walsh, N. (2001). *DocBook*. `http://nwalsh.com/docbook/`.

Walsh, N. and Muellner, L. (2001). *DocBook: The Definitive Guide*. O'Reilly.

Wangler, B. and Paheerathan, S. (2000). Horizontal and vertical integration of organizational it systems. *Information Systems Engineering*.

Warmer, J. and Kleppe, A. (1999). *Praktisch UML*. Addison Wesley.

Winer, D. (1998). *XML-RPC for newbies*. `http://davenet.userland.com/1998/07/14/xmlRpcForNewbies`.

XML-RPC (2001). *XML-RPC*. Userland Software Inc., `http://www.xmlrpc.com/`.

XSchema (2001). *XML Schema*. World Wide Web Consortium, `http://www.w3.org/XML/Schema`.

Zaremski, A. M. and Wing, J. M. (1995). *Specification Matching of Softare components*. Carnegie Mellon University.