universität wien

Masterarbeit

Titel der Masterarbeit

On Semantic Timecard Based Project Portfolio Management

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

Verfasser:	Dmitry Diyachenko
Matrikel-Nummer:	0300077
Studienkennzahl:	A 066 926
Studienrichtung:	Wirtschaftinformatik
Betreuer:	Dipl. Ing. Dr. ao. Univ.-Prof. Renate Motschnig
	Dr. Peter Trimmel
	Ing. Günter Baumgartner

Wien, im April 2006

Table of contents

Executive summary

In der heutigen komplexen Geschäftswelt werden in den Unternehmen viele Projekte, welche meistens Abhängigkeiten und Auswirkungen aufeinander aufweisen, parallel durchgeführt. Die Hauptaufgaben des IT-Projekt-Portfolio-Managements sind die Komplexität des projektübergreifenden Managements in den Griff zu bekommen und das Management in die Auswahl und die Abwicklung der IT Projekte mehr zu integrieren.

Das Ziel dieser Diplomarbeit war es, zu überlegen, wie das IT-Projekt-Portfolio-Management von Software Lösungen unterstützt werden könnte. Für diesen Zweck wurde eine Ontologienwissensbasis entwickelt, die ausgewählte Begriffe der IT Domäne und deren Zusammenhänge standardisiert definiert und zuordnet. Diese Ontologie wird anschließend mit Hilfe von Web-Services in einen Timecard Prototypen integriert, welcher ebenfalls im Rahmen dieser Diplomarbeit entwickelt wurde, um dem Management erweiterte Technologien- und Dienstleistungs-portfolioberichte zur Verfügung stellen zu können. Diese Portfolioberichte basieren auf den Zeitaufzeichnungsdaten, welche von den Mitarbeitern in deren Timecards eingetragen werden und auf den Ontologienkomponenten, die anschließend diese eingetragenen Daten zu bestimmten Technologien- und Dienstleistungskategorien zuordnen. Aus diesen Portfolioberichten kann das Management dann ableiten, auf welche Technologien und Dienstleistungsarten ihr Unternehmen vorwiegend setzt, um dementsprechend passende IT Projekte für die weitere Realisierung auszuwählen. Aufgrund dieser Daten können ebenfalls Umschulungen rechtzeitig angeordnet werden, um bestimmten technologischen Trends und Herausforderungen zu genügen.

Im Zuge dieser Diplomarbeit konnte festgestellt werden, dass wohl definierte und aussagekräftige Portfolioberichte nur sehr schwierig zu realisieren sind. Die größten Herausforderungen stellen dabei das eindeutige Zuordnen von IT Technologien und ein standardisiertes Befüllen von Timecard Formularen dar.

Executive summary

The project portfolio management is a very important concept, first of all, for the organizations dealing with a large number of projects simultaneously. [PR04] A lot of tasks, like control and overview of all projects, comprehensive resource allocation and, consequently, well-arranged and cost-cutting expertise and components reuse, ensuring business strategy alignment, reporting to business executives etc., can be accurately accomplished by the effective deployment of the project portfolio management.

The main focus of the current diploma thesis was to consider how the IT project portfolio management could be effectively supported by the software solutions. For that purpose a semantic timecard prototype and ontology knowledge base were designed. The semantic timecard prototype was implemented primarily to support technology and service portfolios. The ontology knowledge base was thereby built to define the standardized terms of the IT domain and their mutual relations, as well as interconnections. The ontology components were integrated via web-services into the timecard prototype to provide extended portfolio reports, considering technology structures and interrelations. On the basis of the report data provided, the management becomes able to discover which technologies and services their organizations are specialised in and how those trends and tendencies actually change over time. That information enables management to take customer orders, dealing exclusively with similar technologies to assure expertise and technologies reuse. On the basis of the well-investigated trends, essential educational trainings can be afterwards initiated, too.

While developing the semantic timecard application, one can notice that some challenges, such as clear IT technology classification and standardized timecard form completion, are to be faced to provide high quality and accurate portfolio reports.

1 Introduction

Nowadays the complexity of business processes is increasing rather rapidly because of the extremely challenging and ambitious customer requirements and business interconnectedness. Considering globalization aspects and the development of international trading, the organizations should be able to deal with much more competitors than several years ago. Those competitors might even have certain country-specific advantages that allow them to offer similar products at smaller prices. Customers' extensive demands and requirements are changing constantly and have to be met as fast as possible by organizations to survive in that highly competitive environment. Customer demands not only have to be met fast, but also with the required quality and within a certain budget. External environment and trends are changing faster and faster as well. Thus, the business cycles become shorter and have to be accomplished faster, too.

Due to these facts, the organizations have to be able to adapt to the changing business environment quickly to ensure their competitiveness in future and to expand their business activities. It means that organizations have to act flexibly, to be able to respond to customers' demands in time and up to standard, to carry out their business processes and resource allocation proceedings effectively, have control of all business activities and conform to clearly defined business strategies, goals, responsibilities and corporate guidelines, as well as standards. To achieve the acceptable flexibility and effectiveness levels, general concepts have to be reconsidered and improved. Companies have to switch from hierarchical, fixed line organizations to cooperative project- and process-oriented corporate structures. Project-oriented corporate structures allow to perform complex risky tasks and to satisfy customer demands in flexible and effective ways. [PR04] If such customer demands need to be satisfied, projects with temporary teams have to be launched. Projects are led by project managers, have clearly defined goals, responsibilities and requirements. Due to the high level of complexity of the tasks, comprehensive project management has to be absolutely essentially initiated to ensure the successful completion of projects by organizations and the resulting satisfaction of customers. More precisely, project

management is actually responsible for the planning, scheduling and monitoring of projects. [PR04]

In today's business structures projects have dependencies and impacts on each other, some projects are interrelated or redundant. It is not enough to control and manage single projects to achieve maximal overall return. Thus, comprehensive multi project management must be applied additionally. Multi project management is a very important concept, first of all, for the organizations dealing with a large number of projects simultaneously. [PR04] A lot of tasks mentioned above, like control and overview over all projects, comprehensive resource allocation, ensuring of the business strategy alignment, reporting to the business executives etc. can be accurately accomplished by the effective deployment of multi project management. [D03]

The importance of multi project management, that helps to maximize overall return considerably, has already been realized by many organizations. This thesis deals with the subject of portfolio management of IT projects. Portfolio management is a specific concept of multi project management. All the terms will be specified and explained in detail in **chapter 2 "Project Portfolio management"**. Another point that has to be considered is how project portfolio management could be improved and supported by software applications. For this purpose timecard application will be implemented. The main part of this diploma thesis deals with considerations about how implementing timecard could be extended to the semantic component through specified ontology knowledge base. Building of ontology knowledge base and its integration is thereby the most challenging procedure.

1.1 Motivation

As it has been mentioned above, project portfolio management is an essential issue for the organizations dealing with a large number of projects simultaneously. In fact a lot of organizations do not have a full control over their project portfolios and make use of poorly organized, chaotic planning processes. [D03] According to the recent AMR research report, approximately 75% of IT organizations apply project portfolio management poorly or do not apply it at all. Those companies do not have any clear idea of what is happening in their project portfolios, which projects are currently running in their organizations, and there is no clear understanding of the actual value of those projects for the business. [AMR] In these conditions project portfolio

management and its adequate use still represent noticeable challenge for many organizations. It happens, in the first place, because of the lack of understanding which business advantages could be achieved by project portfolio management and how it should be applied correctly and effectively. On the other hand, there is no comprehensive software solution on the market that would completely support portfolio management activities in an appropriate way. Several independent software solutions have to be integrated and combined. Generally, it is a very complex and costly process that can not always be successfully realized. [D03]

Due to these facts, it has to be analyzed how project portfolio management could be applied within an organization in the best possible way, how it could add value to the business and help to gain control over the internal projects and business activities and finally how its deployment could be supported by applying software in an adequate way. A very important part of this diploma thesis is to implement semantic timecard application supporting several project portfolio management activities. The semantic of the timecard application mentioned will be provided by ontology knowledge base that will be developed within the present thesis, too.

1.2 Content of the thesis

The content of this diploma thesis is restricted to the IT project portfolio management. It has to be considered how its processes could be improved by creating a guideline defining a well considered project portfolio management procedure.

Afterwards the timecard application has to be implemented in order to support IT portfolio management activities. That timecard application should facilitate project billing and invoicing, and also generate personalized reports concerning single projects and project portfolios for all roles possibly interacting with our timecard tool. Therefore it has to be considered which roles will actually interact with the timecard application and which information could be valuable or interesting for them at all. In order to deliver extended and high quality reports to all participating roles, certain semantic components will be added to our timecard application.

This semantic will be provided by the ontology knowledge base including certain IT technology components, as well as business service categories. The complex process of the development of the intended ontology will thereby make up the main part of this diploma thesis. Afterwards the ontology knowledge base will be tested for its adequate expressiveness level and consistency by the Reasoner server application.

It also has to be tested and proved that the modelled ontology is able to deliver sufficient and required information to our semantic timecard application. After all the tests are executed successfully, the ontology knowledge base will be integrated into our timecard application via web-services. It has to be mentioned that the core timecard functionality will be covered by Coldfusion web-services. Java web-services, in turn, will be used to integrate semantic components provided by the ontology knowledge base. All those web-services and their technology architectures will also be implemented and presented in detail in this diploma thesis.

Finally, the functionality of our timecard application, as well as generated personalized high quality reports, will be demonstrated and explained in the circumstantial way.

2 Project portfolio management

Project portfolio management is a part of multi project management concept. All the projects in organization can be grouped on the basis of certain criteria (for example, based on their type or business sector). Projects operating in the same business area might be classified as follows:

- Procurement projects
- IT projects
- Greenfield projects
- Logistic projects
- Marketing projects
- Investment projects
- Construction projects
- Research projects [PR04]

Each project category represents a project portfolio. That project portfolio includes projects that operate in the same business area and therefore may use many resources in common, but might possibly have different complexity levels, budgets, duration, goals and priorities. [PR04 p. 403] The task of project portfolio management is to select, manage and monitor projects of a specific portfolio. [PR04] This diploma thesis only deals with the issue of the IT project portfolio management.

Many different project management definitions are available in various existing publications. To avoid misinterpretations, the most important project management terms are going to be defined for our thesis. For our purposes the most applicable definitions of terms will be cited from the literature references. The significant differences between them will be demonstrated and explained, too.

To speak about management of multiple simultaneously executed projects, the actual meaning of a single project and its management has to be understood and defined at first. Many definitions exist in various project management books explaining what a project actually means. In my opinion, the most comprehensive and complete definition of the term **project** is as follows:

A project

- is time restricted
- has definite goals
- is a one-time intention
- is a complex task including subprojects and multiple activities
- includes risky tasks and challenges
- requires comprehensive department collaboration; experts and professionals from different departments have to collaborate and to cooperate to achieve common project goals and objectives
- requires a project manager, coordinator and a team [K00 p. 3]

Project management is actually responsible for planning, scheduling and controlling of those activities that must be performed to achieve project goals and objectives. [L00 p. 7]

In large organizations dealing with multiple projects simultaneously, it is not enough just to implement project management and manage tasks, as well as various activities, in single projects. Management of multiple projects has to be applied in order to allocate restricted resources effectively, to set up projects aligning with business objectives, to maximize overall return. [H02] Several multi project management terms often used in literature references are presented below:

Multi project management – is responsible for management of all the starting and already running projects within an organization. All the projects and their objectives have to be considered to ensure comprehensive and qualitative multiple project management. [H02 p. 25]

Program management – manages multiple projects with the common goals and business objectives in order to fulfil a higher-level intention (e. g. build an airport complex) [CK83 p. 159]

Project portfolio management – manages portfolio of similar projects (projects operating in the same business area) using many resources in common, but with possibly different goals and objectives [PR04 p. 403]

As already mentioned above, this thesis only deals with the IT project portfolio management issue. Other multi project management terms were presented to demonstrate the limits of project portfolio management tasks and responsibility areas precisely.

2.1 Increasing value trough project portfolio management

Nowadays a lot of business leaders do not have a clear idea of what projects are running in their companies and what is going on in their organizations at all. More and more companies tend to ask questions, which IT projects are going to start soon, which ones are actually running and which ones have been already completed, why in fact those projects have been chosen, which profit they brought to the business and whether these concrete projects align with the corporate objectives? Which current IT projects have a high degree of importance and are the necessary resources for their implementation available?

These questions are coming up because of the lack of adequate management integration during the execution of IT projects. Because of this lack of information the stakeholders and business executives are often not able to understand and to follow the importance of the IT projects and their impact on the organizational structures and development. The IT officers, on the other hand, are not always well informed about the status and changes of the business strategy. Thus, they are not really able to select projects aligning with the business objectives and strategies in a satisfactory manner. Looking at these facts and conditions it is quite easy to understand why many IT projects failed or were not started: their strategic importance for the future development of the organization was not recognized or resources for the execution were not available in time.

Failed projects may cause high additional costs, customer dissatisfaction and notable reputation damage. Not recognizing the importance of certain strategic projects can lead to strong business disadvantages resulting in considerable losses of market share. The consequences might be fatal and can lead to serious financial losses or, at worst, even to the bankruptcy of the organization.

Consistent and accurate deployment of project portfolio management does not only help to avoid the situations described above. It also gains overall return of the projects in certain portfolios through comprehensive resource allocation among the projects, and also knowledge and components reuse and transparency assurance.

One important task of project portfolio management is to integrate management into the IT projects in an appropriate way and to improve interaction between the business leaders and IT officers.

The IT project portfolio management also:

- provides an abstract overview of the starting IT projects [H02]

- provides an abstract overview of the currently running IT projects and what is actually happening in these projects [H02]
- ensures the selection of the projects aligning with the business objectives [PR04]
- grants an overview of all available resources [PR04]
- guarantees handling and selecting projects based on their importance and priority for the organization, while low priority projects have to wait in queue or get dropped at all [PR04]
- supports effective allocation of the project comprehensive resources; important resource sets have to be assigned to the projects with the highest strategic priority as first. Some resources might be used by several projects at a time. [PR04]
- enables effective reuse of knowledge and existing components, functionalities already implemented or designed in other projects might be reused and integrated too [PR04]
- helps to reduce the number of redundant projects and to kill problematic or hopeless projects [PR04]
- facilitates project accounting, billing and invoicing [PR04]
- supports project tracking and monitoring [PR04]
- demonstrates project dependencies and impacts on the organization or on each other [PR04]
- generates reports and provides the project stakeholders with transparent and personalized information, like project costs, effort, duration, development status, profits for the organization etc. [H02]

Project portfolio management is an essential issue for the organizations dealing with a large number of projects simultaneously. In fact a lot of organizations don't have control and full overview over their project portfolios and conduct poorly organized, chaotic planning processes. [D03] As AMR research reports, approximately 75% of IT organizations apply project portfolio management poorly or do not apply it at all. These companies do not have any clear idea of what is happening in their project portfolios, which projects are currently running in their organizations and there is no clear understanding which profit those projects actually bring to the business. [AMR]

2.2 Guideline for portfolio management deployment

There is no single absolutely right way to do the IT project portfolio management. There are a lot of methodologies developed by academic institutions, different consulting companies and large corporate groups. [D03] This part of diploma thesis will try to provide a possible guideline of how IT project portfolio management could be done well.

The following steps are crucial for the successful project portfolio management:

- Gather relevant information, establish project inventory
- Evaluate and prioritize the projects
- Manage the project portfolio actively, using, among other things, portfolio monitoring and adaptation to the regularly changing business priorities and external environment [PR04]

2.2.1 Project inventory preparation

First of all, it is necessary to get a holistic overview of all the activities and projects running within an organization. All the projects (projects scheduled to start soon, projects starting, currently running projects, already finished projects) and the detailed information about them, like duration, funding source, approximate costs, ROI, business objective and benefits etc., have to be listed and documented in the project inventory. [D03] The project inventory provides business leaders an outstanding opportunity to look at all the projects in the IT portfolio and to understand which goals and business objectives the projects are following, which resources are required for the implementation etc. Based on well established project inventory, the project evaluation and prioritization can be applied next.

2.2.2 Project evaluation and prioritization

After the project inventory is established in an appropriate way, it is absolutely necessary to evaluate and to prioritize projects included in that portfolio systematically. Business executives have to check to what extent projects listed in IT project inventory are aligning with the defined business strategies and objectives, how important single projects are for the business, how certain projects are interrelated and dependent on each other etc.

Usually there are much more projects on master schedule than the organization is actually able to apply. Thus, on the basis of specified criteria the most important projects have to be selected and funded. Those defined prioritization criteria are also used to build the project queue and to determine which projects will be applied first. [CK83]

If some running projects don't align with the business strategy at all and also do not provide acceptable value to the organization, potential consequences, as well as further proceedings, have to be considered. Those problematic projects might even get cut off or assigned as low priority projects. Detected redundant or overlapping projects should be reconsolidated. The comprehensive resource allocation among projects using similar resources has to be optimized, too. [PR04]

The prioritization criteria can depend on the following items:

- Alignment to business strategy – the project has to achieve business objectives and strategies
- Project dependencies – dependencies between the projects, impacts on other projects and organization, e. g. some projects have to be done to set up next important projects
- Strategic importance - the project is important to ensure the future competitiveness in a specific business sector
- Urgency – e. g. extra funds might be received from the government or other financial institution, if the project meets specially defined requirements (e. g. has to start or to end before certain date)
- Realization probability - what is the probability that the project might include risky tasks, unsolvable challenges etc.
- Profitability – does the project bring high profits to the business
- Cost and duration risks – it is difficult to calculate the expected costs and duration of the project, potential cost explosions have to be considered etc.
- Impacts on the environment – how intensive the impacts of the project on external environment, on employees, on relationships with the customers, on the organization itself (management of change) are
- Resource availability – whether the resources necessary for the project are available, whether there are similar projects running to optimize comprehensive resource allocation, which software components already exist and might be reused [K06; PR04]

The level of importance of prioritization criteria might differ in various organizations because of their varying business strategies. According to their business strategies and objectives, the business executives have to evaluate the above-mentioned criteria and their significance within an organization. [CK83] After that criteria evaluation the prioritization of the projects can be performed. Some organizations are willing to accept more risky projects in order to achieve higher outcome. Organizations with conservative business strategies prefer to invest in less risky projects. The final project priority might also be manipulated by other criteria like its urgency, strategic importance, realization probability etc. [PR04]

Due to the fact that organisations usually have more projects on schedule than resources available to implement them, the projects most relevant for the business need to be selected for immediate realization. The remaining projects form a queue based on their prioritization and wait for their turn. The irrelevant projects with unacceptable profit expectations even have to be kicked out of the master schedule at all. This proceeding is called project funnelling. [K06]

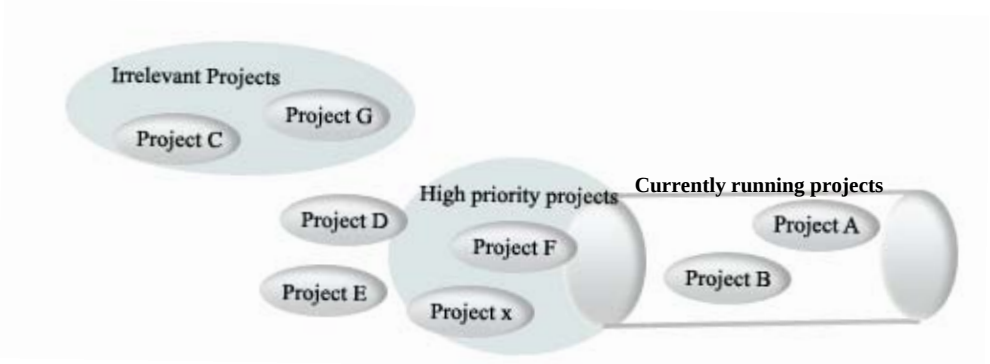

Figure 1: Project funnelling

The currently running projects are also handled on the basis of their prioritization. The projects most important and crucial for the organization have priority in comprehensive resource allocation process, too. [CK83]

2.2.3 Portfolio monitoring and adaptation

Business strategies may change and be adapted in organizations. These changes have an impact on the prioritization criteria and therefore also on the project

priorities. All these changes have to be recognized in time and project priorities need to be updated regularly. Some projects may be killed as a result of new priority definitions. [PR04] The status (cost, deadlines etc.) of the projects have to be tracked continuously and reported to the participating stakeholders (e. g. top management) in definite intervals. [PR04] The so called **project management office** is responsible for the monitoring of project portfolios based on information received from the project managers [PR04]. The project management office is also responsible for:

- tracking project portfolios
- the development of potential project managers
- support and coaching of project manager in critical situations
- reporting to the top management
- assurance of project management quality
- introduction of project management standards and guidelines [K06]

2.2.4 Portfolio management barriers

Project portfolio management is able to deliver high value to the business by doing it well, but there are some barriers which may cause complications in portfolio management process execution.

- Democracy is not always accepted. The decisions are made by group consensus and not only by business leaders anymore. Often it is very hard for business executives to share the power with other participants. However this group decision-making is crucial to ensure the quality of the portfolio. [D03]
- There is no software that completely supports portfolio management and is able to solve all the required tasks. To apply portfolio management optimally, several selected software suites have to be merged and integrated (which is rather costly). [D03]
- It is really difficult to get accurate information (project status, resource requirements, technology costs etc.) [D03]
- It is always hard to make tough decisions and kill useless projects, especially projects with high investment costs already running for a long time. [D03]
- Business executives often don’t have enough time to participate in portfolio management process. [D03]

These barriers strongly interfere with the accurate project management execution. They might be resolved through adequate corporate culture and the openness of management's attitudes.

2.3 Improving project portfolio management trough the semantic timecard application

Project portfolio management can be supported by various applications and tools. There are tools for single project planning, monitoring and tracking, resource management tools, document management tools, financial management components, risk management components, project evaluation tools by its prioritization, executive dashboard components etc.

The task of this diploma thesis is to develop semantic timecard tool supporting project portfolio management as follows:

- Control over the whole project portfolio - a list of all the projects including detailed project information
- Project progress and status monitoring
- Comprehensive resource allocation support
- Project billing and invoicing support – transparent cash flows within projects, facilitating understanding what the money was spent on (project expenditures), supporting automatic employees billing
- Personalized reports for project staff, project managers, project management office, and for top management (technology used, performed services reports, capacity utilization etc.)

The most important part of this thesis is extending the above-mentioned timecard application with a semantic component. To fulfil this purpose the ontology based knowledge base prototype has to be designed and implemented. This knowledge base prototype will consist of different IT technologies (java technology, PHP, XML etc.) and service categories (software architect, programmer, consultant etc.). The challenge is to acquire all these technologies, group them, create hierarchies and relations of the listed technology groups, store designed ontology and adequately integrate it in timecard application. It is very important to keep in mind that the knowledge base design has to guarantee its easy extensibility. The integration of the

ontology should be realized via web-services to enable flexible component reuse in various other applications.

The relations between technology components should ensure extended, high quality management reports and facilitate the completion of timecard forms for project collaborators.

If a project collaborator wants to report completed tasks, he has to fill in a timecard form. This timecard form contains input fields like date, project name, task name, task description, technology and software used, work duration in hours, service type. The tasks to be fulfilled can be imported from project management application like MS project. It is very important that task names in the timecard are given according to the names defined in project planning application to deliver accurate project progress and project status information. The project collaborators should know in the knowledge base acquired technology and service categories components. Ideally, timecard application should import personalized technology and service types from the ontology based on project data (collaborator participates in certain projects, only technologies assigned to those projects should be imported to facilitate selection of technology used to fulfil certain project task).

If specific technology is assigned to the project that is not specified in the knowledge base, the ontology has to be extended to this new component. The employees only have to select technology they used as well as service type they performed in order to fulfil certain task. They don’t need to describe these technologies and fill in more fields to specify relations to other technology categories explicitly. The relations between all these technologies have to be covered by the ontology knowledge base.

On the basis of these relations between different technologies and hierarchical structures, extended management reports can be generated. Management might be interested in which services (highly paid, routine services) the organization has been selling and which technologies are dominating internal IT projects. Through ontology it becomes possible to extract certain information even if that information is not asserted explicitly. If the query has to find out, how many hours the employees worked with specific technology like JAVA, all the technologies belonging to JAVA technology or are interrelated with JAVA in any way need to be considered. It is only possible to perform such kind of queries if the hierarchies and relations between the technology components are accurately defined. The building of ontology is absolutely necessary to describe the IT technologies used to accomplish project tasks. In this way the collaborator only has to enter the technology terms explicitly. Other

information that might be useful for certain management reports can be derived by ontology knowledge base inference mechanisms. The ontology database can be used for other purposes and be integrated in other applications, too.

2.3.1 Roles participating in the timecard application

To create the design of the semantic timecard and ontology knowledge base that would be acceptable and would satisfy all stakeholders, it is crucial to understand the environment around the application, as well as roles participating and interacting with our semantic timecard. The issue of this section is to discover who is interacting with the timecard tool and in what way. It is also very important to visualize the environment around the intended application and its dependencies.

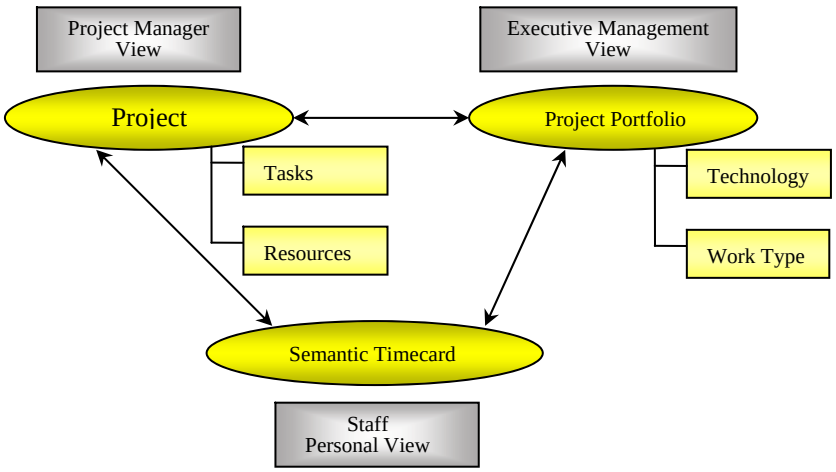

Figure 2: Semantic timecard environment

The following roles might interact with the semantic timecard application:

- Project manager
- Project staff
- Management

The project team members are interacting directly with the timecard application. They fill in a timecard form and document their solved tasks. These project team

members can also benefit from timecard application. The timecard tool is documenting all the activities of the employees and project staff: how much they worked, in which projects they participate and which projects have been already settled by them, for what tasks and activities the project members are responsible, what type of work has been done by a specific employee. The timecard reports enable the project members and employees to observe their career development and their progress in the hierarchical structures of organization. They can keep track of their personal development (e. g. project member -> project manager) and the development of their work activities (e. g. programmer -> design architect -> consultant).

Project manager is interested in completing the project he is responsible for in time, in budget and up to quality standards. The timecard application might help the project manager to track and to monitor the progress of his project. After the project staff have filled in their finished tasks, the project manager is able to perform target/actual comparison. In this way project manager might discover deviations from the schedule and take corrective measures in time.

For management it is very important to observe the project portfolio in the organization. Management wants to ensure that projects initiated in the company are aligning with business strategy and objectives. For management the following information is important:

- Overview of all projects and their status
- Service categories sold to the customers (programming, design, consulting)
- Technologies used in current projects

It might be valuable information for the management, which technologies are dominating in current projects and becoming more and more important, which services and at which ratio the business actually offers (e. g. consulting, design, programming), which projects are going on schedule and where delays might occur. All these information units are stored in the timecard application and could be delivered on request. Extended, not explicitly defined information might be extracted from the ontology knowledge base in turn.

3 Building the ontology knowledge base

In this section the complex process of the ontology knowledge base building will be documented and explained in detail. The knowledge (terms) of IT technology and IT service categories domains will be explicitly represented in the form of the ontology knowledge base prototype. The ontology will not cover all the terms of above mentioned domains. It should just represent a prototype where selected concepts, properties, relationships and hierarchies will be defined. This chapter rather provides a kind of guideline how the ontology knowledge bases could be conceptualized, modelled, stored, documented, maintained and integrated in various applications. The quality of the knowledge base will increase through its use. The knowledge base has to be maintained during its entire life cycle carefully. Maintaining the knowledge base means that previously incorrect modelled concepts are to be revised and remodelled, the ontology knowledge base needs to be extended as new concepts are coming up during the daily business operations or through the integration of other external ontologies. The ontology also might be adapted or extended to suit certain application requirements. This section will analyse ontology reasoning and querying mechanisms and their opportunities in detail, too.

To show all the steps of how to build the ontology knowledge base, that is to be considered and mentioned, a generic ontology building process will be defined below using Adonis business process modelling tool. This process visualizes all the steps that will be mentioned in this chapter and are necessary for successful ontology creation and maintenance. This generic process can also be seen as a guideline showing the structure of this chapter. Afterwards all the process activities and their possible dependencies, as well as interrelations, will be described in detail.

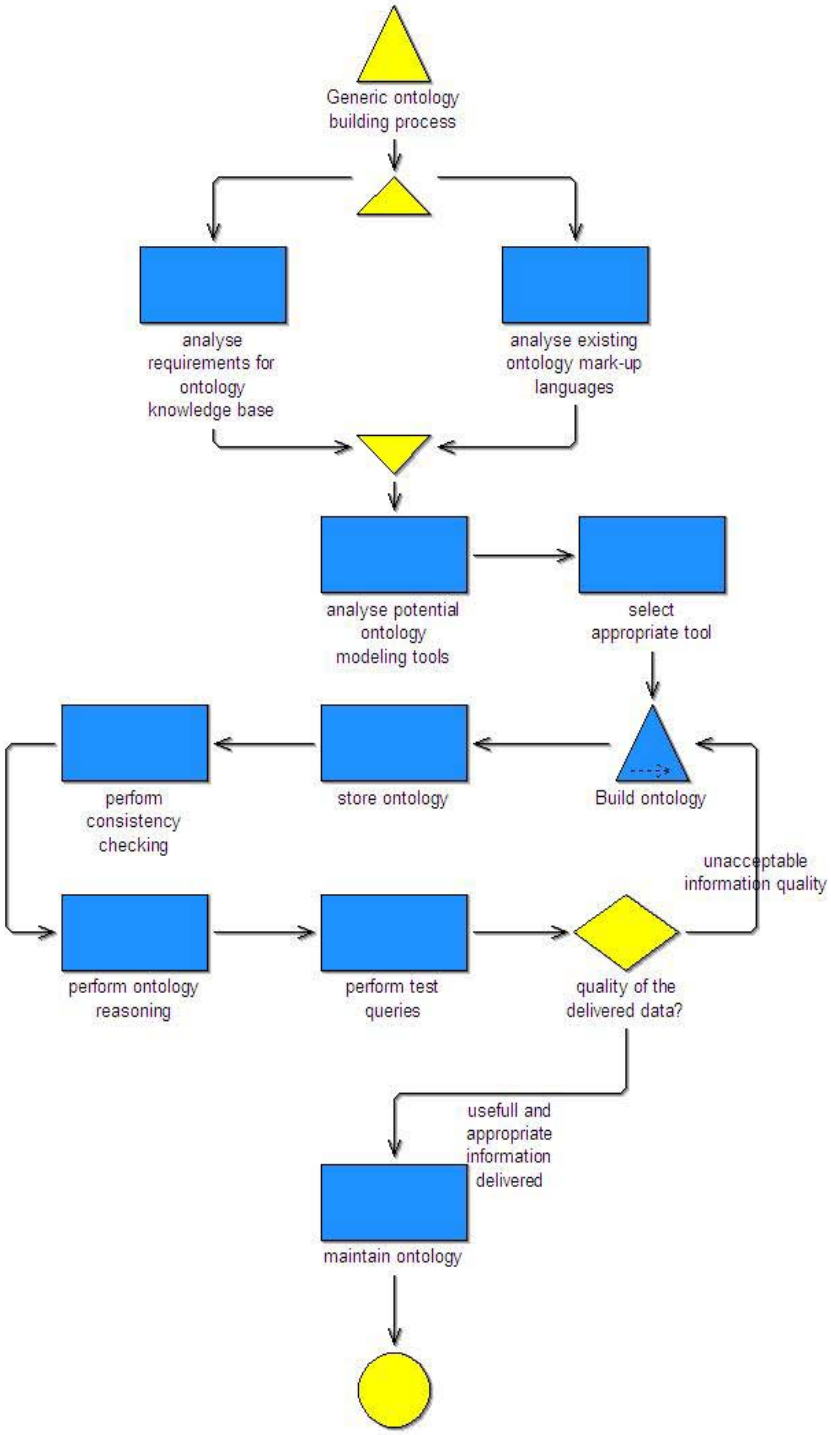

Figure 3: Generic ontology building process

At first it is very important to analyse the requirements for the ontology knowledge base. It is necessary to find out which roles are interacting with the semantic timecard application, which information might be important for those roles, in which way those roles interact with the semantic timecard prototype, which information in turn might or should be provided by the ontology knowledge base. In order to specify the detailed requirements for the ontology knowledge base the use case diagram is going to be drawn in order to visualize all the roles that might have something to do with future ontology and the information that might be interesting for those participating roles. Gathering information appropriate for the ontology requirements is crucial to ensure the adequate and correct conceptual ontology design. If the goals and needs of the interacting users and the applications, that will use this ontology in future, are understood well and, therefore, the ontology conceptual design could be developed in the way requested, then the ontology might considerably improve the information quality processed by the application, and thus deliver substantial value to the business.

In this thesis the ontology mark-up languages necessary to encode the ontology have to be analysed for their expressiveness level, computational speed and facilities. Afterwards the adequate ontology mark-up language that best fits our ontology requirements has to be selected.

The next step is to check the existing comprehensive ontology editors supporting the ontology modelling, visualization, querying and integration of other software components. The ontology modelling editor that best fits our needs has to be selected. Afterwards the conceptual ontology design and its building process have to be considered and realized. This activity is modelled as subprocess call in our generic ontology modelling process and will be described in the chapter 3.4. “Interactive ontology building process” in detail.

After the ontology knowledge base is built it is very important to consider how the created ontology could be stored. Ontology repositories enable storage of large ontologies. They provide better ontology integration, maintenance and querying options as well as improved scalability and performance facilities. Simple ontologies with low integration and maintenance requirements might be stored in files, too.

Afterwards the ontology consistence, querying and inference tests have to be performed by selected reasoner server application. Possible concept inconsistencies or modelling mistakes (it is not possible to indicate the necessary information by querying mechanisms) have to be corrected and adapted by modifying and improving

the conceptual modelling design. This cycle has to be run through till satisfactory data delivery results are achieved.

It is very important to note that the maintenance plays a very important role in the ontology knowledge base development lifecycle. The quality of the knowledge base increases through its use. All new terms coming up in the business activities and the concept improvements have to be considered in the maintenance phase regularly. All the extended constructs, new improved expertise as well as best practice experiences have to be integrated in the ontology knowledge base during its use.

3.1 Introduction to the ontology concept and the need for it

It is very important to understand what ontology actually is, its differences to other concepts like taxonomy, thesaurus etc, what purposes the ontology could be used for and how the ontology could improve knowledge management facilities and deliver value to the business. This section will provide general information about ontologies, their possible usage fields and opportunities as well as out of it resulting information processing advantages.

The definition of the ontology:

> „An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects to the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.” [SBF98, p. 185]

In other words, ontology is used to represent the information about real world concepts with all their relations and constraints explicitly and formally in order to provide machine understandable constructs accepted by a group.

Ontology generally includes the following constructs:

- classes (i.e. concepts)
- attributes (i. e. properties)
- relations (described through properties)

- instances (i.e. individuals). [NM01]

Before explaining the constructs listed above in detail, I would like to compare ontology with other knowledge representation approaches to clarify the advantages of ontologies in their expressiveness power compared to other concepts with similar purpose, namely explicit knowledge representation.

The figure below demonstrates different approaches to knowledge representation and their expressiveness power facilities. They are ordered in accordance with the rising level of their expressiveness power.

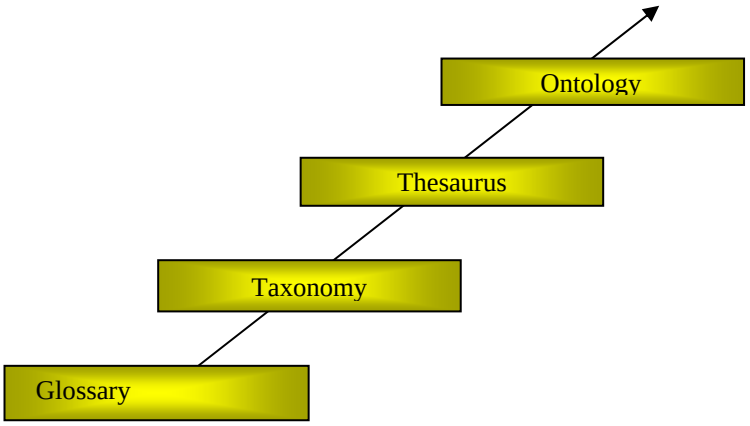

Figure 4: Evolution of knowledge representation approaches [adapted according to AUM04]

The figure shows four possible approaches for representing information about the real world or parts of it. Controlled vocabulary approach has thereby the lowest expressive power, ontology the highest.

Glossary just represents a limited number of specific domain terms. The meaning of these terms is described in natural language. Glossaries can be created by the information acquisition to define which possible terms exist in certain domains and what those terms mean.

Taxonomy builds hierarchies of certain domain terms. There are generic concepts (super classes) and sub concepts (subclasses). The subclasses have to be assigned to appropriate super classes. Generally it’s not that easy to perform these assignments because of different views in certain domains. [AUM04]

Thesaurus contains not only hierarchies of limited domain terms but also simple relations between them. However these relations can only define the similarity between the domain terms, but they cannot describe themselves. [AC04]

Ontology allows defining hierarchies of concepts, their attributes, individuals (class instances), arbitrary complex relationships, cardinalities and constraints. Ontology approach provides the most expressive power for the real world information modelling. [VK05] If the ontology is well defined and its powerful concepts are used, new knowledge can be inferred and provided by ontology reasoners. Reasoners are also able to perform complex ontology queries to derive necessary information. [HM03]

The process of the ontology creation is rather complex, time- and resource-consuming. In order to accept this considerable effort, is rather important to understand major reasons of the ontology building and how ontology might add value to the business. One important task of ontologies is to define common vocabulary for data sharing and exchange in certain domains. [M92] Also there might be following advantages achieved by using ontologies:

Sharing common information structures among people and intelligent software agents is one of the most important tasks in developing ontologies. The ontology enables people and software agents to communicate in the same domain language. All the terms, interpretations and relations are well defined and understood by participating roles. It becomes easier for people and intelligent agents to process, aggregate, extract, integrate or share the information necessary for different purposes. [M92] [G93]

Reuse of domain knowledge is a common goal of the ontology development, too. It is very important to provide ontologies publicly to improve their quality and acceptance. As mentioned above, formal knowledge definition is a very costly and time-consuming process. A lot of time and money could be saved by integrating already existing domain specific ontologies (medical ontologies) or common ontologies like UNSPSC. Several small ontologies might also be integrated to describe portions of a large domain. [M92]

Explicit representation of domain assumptions also represents a very important issue of the ontology development. It’s crucial to document all the domain knowledge and expertise to ensure its further processing, improvement and use. It has already been a considerable challenge of artificial intelligence discipline to extract experts’ knowledge in certain domain and make it accessible for other people or intelligent

agents. The documented domain knowledge allows the participating groups to understand the domain specification and its refinements. [NM01]

Domain knowledge analysis can be performed once all terms, definitions and interrelations of the real world concepts are well defined. All the structures, concepts, dependencies and interrelations can be analyzed and checked for their consistency by machines; new knowledge and assumptions might be derived and inferred. Those facts might lead to better domain understanding, knowledge gaining, as well as to more effective and efficient processes. [MFRW00]

Of course, software agents might also use domain **independent** ontologies for certain purposes by integrating several different knowledge bases. [NM01]

3.1.1 Adding semantics to the timecard application

The ontology knowledge base is created in order to provide additional semantic to our timecard application. The ontology prototype will contain specially selected IT technology components and in the organization presented IT service categories. All the relations, dependencies and restrictions between these terms have to be considered and implemented.

The conceptual design phase is crucial for the successful ontology development and its future use. The domain knowledge acquisition is one important task. Another important task is to consider how this knowledge should be represented. Hierarchical structures and concept interrelations might differ depending on the ontology's purpose and use. [HKRSW04] The primary task of our ontology is to add meaningful semantic to the data used by the timecard tool and to provide high quality information for the timecard end-users.

It's very important to analyze which information is required by the timecard tool and what kind of data is essentially important for the timecard users. To indicate and to visualize user interactions with the ontology knowledge base and their expectations the use-case diagram will be drawn below. Afterwards all the activities and dependencies will be explained in detail.

Figure 5: Ontology use-case diagram

As visualized in the figure above the requirements for the ontology knowledge base have to be gathered first. The ontology developer (author) has to consider and to include all the needs of the participating roles. [GFC03] In this case project staff, PM office and management are interacting with the semantic timecard application. After collecting and evaluating data provided from our participating roles the developer has to specify requirements definitions for the further ontology development.

The requirements specification is used by ontology conceptual design development afterwards. To model the ontology concepts in required way, the requirements data always has to be kept in mind. The ontology concept should suit users' needs in the best way. In the figure is specified that management might be interested in currently used technologies. It is also very important for management to keep track of services currently sold and their relative proportions to each other. This information allows management to identify some kind of trends (which services are in demand at the moment and which technologies become more important or are new on the market). For the project managers and PM office this information might help to recognize the service and technology trends as well. Therefore they could react on changing market situations and perform educational programs and seminars for employees in time. The project managers also might shift the main focus of their projects based on this information (the importance of certain technology increases rapidly, it has to be integrated or excessively forced).

The employees might be interested in their career development too. The career development information could show the employees how their responsibilities, functions or positions changed during certain period of time, where are they placed in the career hierarchy, how fast did they improve and increase their career status in the organization.

The project manager might also be interested in the status of his project or projects and the PM office in the status of the project portfolio (is the project status according to schedule, where are delays, resource deficits or other difficulties). This information is able to be provided by the timecard application; however the quality of that information cannot be improved by the ontology knowledge base.

But the ontology could improve the quality of information on technologies that are currently in use and on sold service categories (programming, consulting, architect etc.). The ontology knowledge base could provide extended information for all participants considering not only the domain terms but also their interrelations, dependencies, sub categories and properties. Summarizing, the ontology would enable more complex and expressive queries providing for users detailed high quality

information as well as new inferred knowledge. However the ontology could only deliver all these advantages if its requirements were specified carefully and integrated into the conceptual phase in appropriate way.

The ontology framework was designed to provide a generic idea of the ontology knowledge base environment and its components. This framework consisting of certain layers visualizes how the components interact and depend on each other.

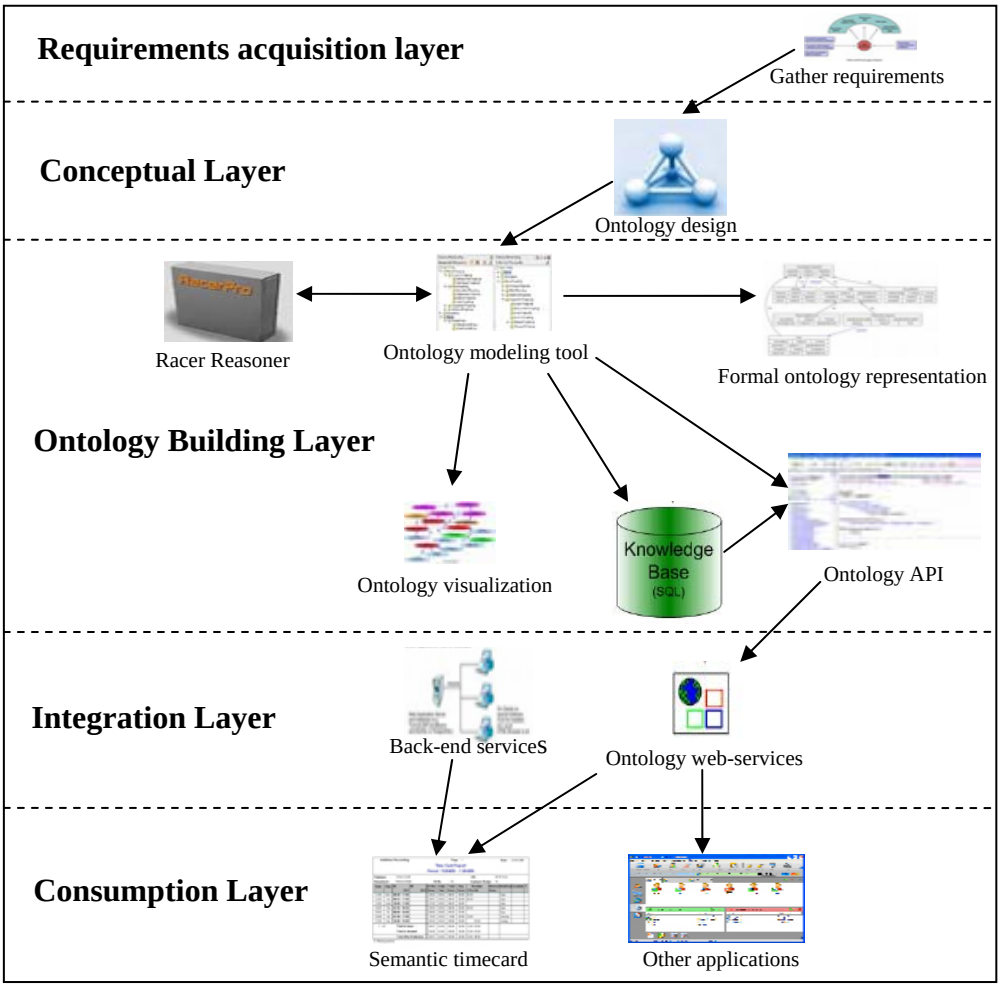

Figure 6: Ontology development framework

This framework shows how the entire life cycle of an ontology could look like. It shows, in which way the ontology's development, visualization, storage, modification, integration and consumption components interact with each other.

As first the design of the ontology derived from its requirements (identified in the requirements acquisition layer) has to be realized in the conceptual layer.

Afterwards the ontology has to be defined in a formal way with the help of the necessary constructs provided by ontology mark-up languages. That modeling process could be supported by the ontology modeling tool including comfortable graphical interface and powerful modeling and integration constructs. Some tools can also be used for ontology visualization for certain documentation purposes. After defining ontology in a formal way the ontology consistency and expressiveness power should be inspected by the reasoner (ontology querying). Ontology might be stored in repository to improve its scalability, performance, querying and modification facilities.

Most part of ontology modeling tools and repositories provide API for ontology querying, modification and integration purposes. That API could be used by applications or web-services to process the ontology knowledge base. In our case the web-services for ontology integration will be implemented to ensure interoperability and to enable flexible as well as extensible ontology integration in various other applications. That means that the developed ontology also might be used for other purposes (not only for semantic timecard application) in the organization. Among other things, the ontology designed could gain and improve understanding of IT technology domain, too, assuming that the ontology maintenance process is performed in an appropriate way. It is also very important to mention that the ontology knowledge base archives and documents all the best practice experiences and expertise of organizations and enables their further use for different purposes.

3.2 Ontology mark-up languages

As has been mentioned above, the ontology is necessary to structure and to describe different terms of a specific domain. That information described has to be machine readable and processible. There are some specifications (languages) enabling formal ontology modelling. Those languages differ in their degree of expressiveness, design and syntax constructs.

The most widespread ontology languages for web applications are RDF/S and OWL. In the present section these mark-up ontology languages will be analyzed for their construct facilities, concepts, simplicity and expressiveness level. After that comprehensive analysis for our purposes appropriate ontology language will be selected.

3.2.1 RDF/S

RDF (Resource Description Framework) is a model using XML based syntax and was developed to describe resources granting them machine-understandable semantic. The RDF metadata model is based upon the general idea of making statements about resources in order to describe themselves, their dependencies and relations. In the World Wide Web Consortium (W3C) specification those statements represent subject-predicate-object expressions, also called RDF triples. [B04]

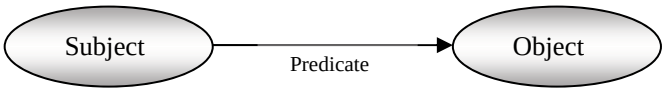

Figure 7: RDF triple [B04]

This graph could be interpreted as follows: the subject can be interpreted as resource to be described, the predicate represents a property (defined relationship between the subject and object), the object is the second resource describing subject in some way. [B04]

As has been said above, RDF represents some kind of model explaining how the resources could be meaningfully described trough their attributes and relations to each other, but RDF does not provide possibilities to describe those attributes and relationships between resources. RDF does not provide classes, subclasses and super-classes concepts either. The building of taxonomies and hierarchical structures is therefore not possible. [BG04]

For the purpose of simple ontology creation the RDF schema using RDF syntax was specified by W3C consortium. The RDF schema is providing some concepts to describe simple ontologies. It enables to build and to describe hierarchical structures and interrelations between them by using classes and properties. [BG04] RDF's vocabulary description language, RDF Schema can be also considered as semantic extension of RDF providing more expressive power. [BG04] RDF schema is

applicable for web ontologies development because of its easy integration, extensibility and simplicity. [BG04]

The unique namespaces are used to declare the vocabulary of RDF and RDFS specifications. RDFS is using rdfs namespace to define its core vocabulary and rdf namespace to include RDF resource definitions. [BG04]

RDF schema defines a considerable amount of concepts; these concepts can be generally divided into Classes and Properties.

The most important RDF and RDFS **classes** are listed and explained below:

- rdfs:Resource - the class resource is used to design everything. All other classes are subclasses of Resource class.
- rdfs:Literal – this class is representing literal values, e.g. textual strings, integers, dates
- rdfs:Class – with this concept it is possible to define classes (i.e. group of resources) of certain domain, it is also possible to build hierarchies of classes using property (relationship) rdfs:subClassOf. The classes can be interrelated to each other through their properties.
- rdf:Property – use of rdf:Property class enables modeling and describing different relationships between specified Classes, using the tag rdfl:subClassOf also enables creating of hierarchies between the properties. It is important to keep in mind, that subclasses inherit all the attributes and instances of their superclasses. Properties are also used to describe classes and, in turn, other properties.
- rdfs:Datatype – this tag is used to define the data types of the classes (e. g. strings, integers, dates, floating point numbers), the predefined XML-schema data types are used for that purpose
- rdf:Statement - the class of RDF statements is necessary to model the above-mentioned RDF statements containing subject (instance of rdf:Resource)-predicate (instance of rdf:Property)-object (instance of rdfs:Resource), also called RDF triples
- rdf:Bag – this tag is used to define unordered enumerations, rdf:Bag class is subclass of rdf:Container class (opened collection)
- rdf:Seq – this tag is employed while modeling ordered sequences, rdf: Seq class is subclass of rdf:Container class (opened collection)
- rdf:Alt – this class can be used for declaring possible alternatives, rdf: Alt class is subclass of rdf:Container class (opened collection)
- rdf Collections – are the opposite of container classes (seq, bag, alt), namely, they represent closed collections (i.e. restricted amount of members) [BG04]

The properties in RDF schema are not only used to describe classes and their relationships, but also other properties and their dependencies. To define relationships between resources two main concepts - rdfs:domain and rdfs:range - are used. These two concepts define from which resource (domain) to which restricted resources (range) the relationship is set. [BG04]

In RDF Schema some properties are specified to create and to define hierarchical structures:

- rdfs:subClassOf
- rdfs:subPropertyOf [BG04]

Properties responsible for resource description are listed below:

- rdfs:comment
- rdfs:label
- rdfs:seeAlso
- rdfs:idDefined
- rdf:value
- rdf:type [B04]

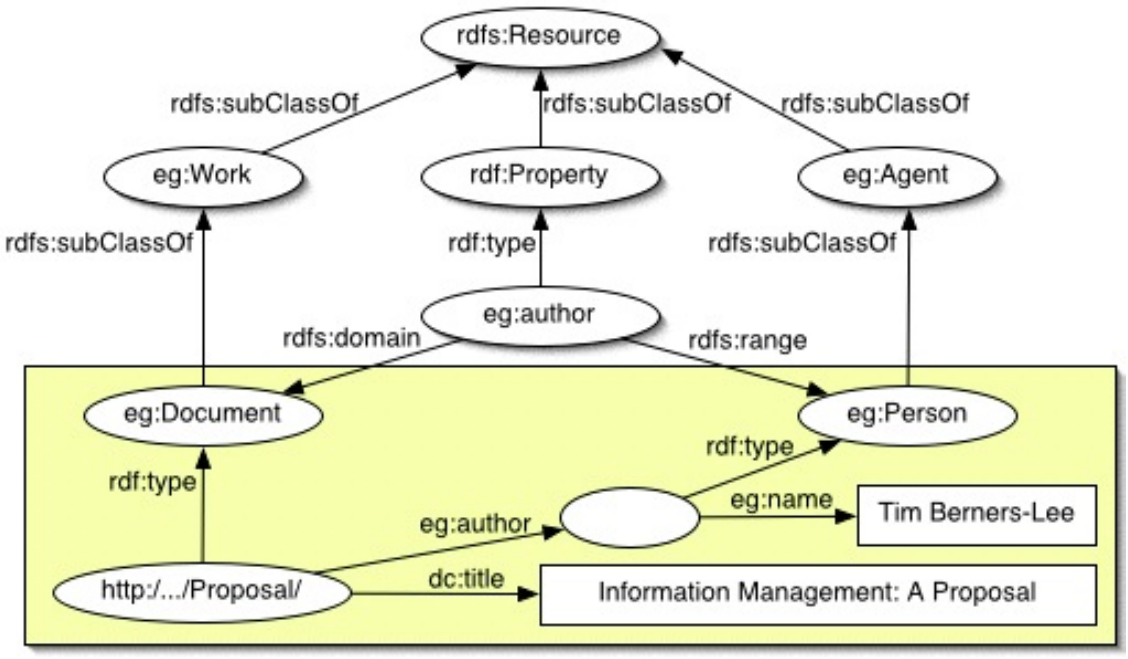

Figure 8: Simplified sample of RDFS constructs [BG02]

RDFS provides vocabulary for describing properties and classes of RDF resources. However, the expressiveness power of those description concepts is too low (no strong cardinality restrictions, no disjoint classes, no symmetric or recursive properties etc.). [MH04] Therefore they can only be applied for modelling simple ontologies, where extended reasoning aspects are not absolutely necessary.

To model complex ontology structures and to define interrelations between different terms in more expressive way, the web ontology language OWL was developed by W3C consortium. OWL also fits the requirements for an entire WEB ontology language and represents W3C recommendation since February 2004. [MH04]

3.2.2 OWL

OWL was designed by W3C consortium to provide semantic meaning to specific content of information. The described information becomes therefore machine-

understandable and can be processed by applications and not only by humans. [MH04]

The main task of this web ontology language is to provide expressive vocabulary in order to describe resources on the web and their interrelations to each other (provide explicit meaning to the content). This could also be expressed as mapping of real world concepts to machine-understandable language that should facilitate automatic information integrating and processing available on the web. [MH04] To enable extensible reasoning facilities web ontology language has to fulfil numerous modelling criteria and design goals. In the following subsections those criteria necessary for appropriate web ontology language will be listed and explained. Three in W3C consortium defined owl sublanguages (OWL Lite, OWL DL and OWL Full) will be described in detail as well.

3.2.2.1 Web ontology language design goals and requirements

W3C community specified several design goals for the web ontology language OWL and also in which areas and sectors this concept might be used and may play an important role. In fact the ontology is very important for all sectors, where large amounts of data and information need to be processed and exchanged. Ontology facilitates content structuring, interrelating and integrating that can afterwards be processed by applications for certain purposes. Actually, well-engineered ontology language should fit the following concepts:

- **shared ontologies** – all the ontologies should be publicly available. It is very important to be able to extend existing ontologies and to integrate several external sources in one ontology knowledge base. One important task of the ontology language is to support the concept of knowledge reuse effectively. For this purpose every ontology has to include unique id and has to be described through meta information tags. It should be possible to identify all the resources within a specific ontology only by using URI reference. [H04]
- **ontology evolution** – application, technology and therefore knowledge base requirements are changing frequently. There can also be some changes in ontology design preferences, and mistakes in prior versions might be indicated. Therefore the ontology structure has to be adapted and modified to follow the changes of external environment. The task of ontology evolution concept is to provide ontology versioning, relations between the revised versions and to make compatibility possible between resources that are committed to different ontology versions. [H04]

- **ontology interoperability** – there are different possible ways of modelling domain knowledge. Different organizations and domains have different ideas and concepts of how knowledge units have to be represented and how terminologies should look like. This fact might lead to the same information being represented in different ways structurally. The issue of ontology interoperability is to ensure that ontology languages provide primitives for relating different representation types. The following proceeding allows to transform data into the appropriate representation format used in the required ontology (i.e. concept mapping). To fulfill this task effectively and accurately classes and their properties have to be described in an adequate way (e. g. subclasses relations, complement relations, transitive or symmetric relations etc.). Class, property and individual equivalencies have to be defined, it should be possible to attach specific information to defined statements and treat classes as instances in specific cases. All these facilities should be provided by the entire ontology language. [H04]
- **inconsistency detection** – while ontologies with different concept views are imported and combined, there might be some inconsistencies between different ontologies or their conceptual views. Even false or incorrect information or relations might be provided. The task of inconsistency detection concept is to ensure completely automated ontology inconsistency detection through extended reasoning components. To enable adequate persistence checking facilities it should be possible to define class and property definition primitives (e. g. unionOf, complementOf, intersectionOf etc.), as well as arbitrary cardinality restrictions. [H04]
- **balance of expressivity and scalability** – generally there are two conflicting requirements every well-engineered ontology language should fulfill, namely it has to provide a wide variety of knowledge modelling concepts and guarantee their fast computation at the same time. It has to be considered which concepts are absolutely vital and therefore have to be included as well as which concepts, providing extended expressiveness power but dramatically slowing ontology computation, could be included optionally. For this purpose there are three types of OWL mark-up languages providing different levels of expressiveness and computational speed that can be chosen by organizations for their special needs and requirements. [H04]
- **easy of use** – the language should be natural and easy understandable by humans working with the syntax directly to enable easier querying as well as reasoning of ontology knowledge bases. The language should have clear concept and meaning

definitions. It should be easy to learn as well. For this purpose easy-to-understand data types have to be used or defined and multiple alternative user-displayable labels (e. g. displaying of concepts in different natural languages) need to be supported by the ontology language. [H04]

- **compatibility with other standards** – compatibility with other industrial standards facilitates tool and language development. The web ontology language should be especially compatible with commonly used web standards like XML and other XML related standards (RDF, RDF schema etc.). Compatibility with widespread modeling standards like UML is also desirable. [H04]
- **internationalization** – the ontology language has to provide concepts enabling ontology modeling in the multilingual mode. It should be possible to define different ontology views that optimally fulfill requirements of different cultures with different knowledge expression techniques. Thus, the ontology language should necessarily support user-displayable labels, a character model as well as uniqueness of Unicode strings. [H04]

3.2.2.2 OWL Lite

As mentioned above the OWL consists of three sublanguages providing different expressiveness power. Organizations have to consider which sublanguage best suits their needs before they actually start to develop their ontology knowledge bases. OWL Lite provides less expressive concepts than OWL DL and OWL Full. However the processing speed of querying and reasoning can be considerably improved by using this sublanguage for modelling knowledge bases. OWL Lite also provides quick integration paths and has lower formal complexity than OWL DL and Full. [MH04]

OWL Lite includes numerous features and constructs, which will be explained in detail below.

The following OWL Lite features related to RDF schema are included:

- **Class** – defines a group of individuals that belong together based on their similar properties
- **rdfs:subClassOf** – makes it possible to build class hierarchy
- **rdf:Property** – enables modelling of relations. There are DatatypeProperties (e. g. relation to data type Integer) and ObjecttypeProperties (relations between instances)
- **rdfs:subPropertyOf** – for creation of property hierarchies
- **rdfs:domain** – limits the individuals to which the property can be applied
- **rdfs:range** – limits the individuals that may be represented in the property value

- **Individual** – instances of the classes

OWL Lite equality and inequality constructs:

- **equivalentClass** – is used to declare two equivalent classes that have the same instances and is helpful for ontology integration and reasoning.
- **equivalentProperty** – necessary to declare equal properties that interrelate same individual sets
- **sameAs** – allows to create many names actually belonging to the same individual
- **differentFrom** – allows to declare that one individual is different from another, improves extended ontology consistency checking and reasoning facilities
- **AllDifferent** or **distinctMembers** – enable to create a set of individuals and to declare that these individuals are different from each other [MH04]

OWL Lite property characteristics and restrictions:

- **inverseOf** – useful to define inverse properties, e. g. hasChild is inverse to hasParent
- **TransitiveProperty** – defines transitiveness of properties. If e. g. ancestor property is declared as transitive, than reasoner can deduce following: a ancestorOf b, b ancestorOf c → a ancestorOf c
- **SymmetricProperty** – allows to declare properties' symmetry (if a isFriendOf b → b isFriendOf a)
- **FunctionalProperty** – property can have for each individual at most one value or even be empty
- **InverseFunctionalProperty** – meaning that the inverse property may have at most one value; enables additional reasoner deduction options, too
- **allValuesFrom** – restricts values of the property to be instances of the same class (i. e. if this property comes upon within a relation, certain class is only allowed to be related to the instances of one specific class through the specified relation, other classes are not allowed to be represented in this relation)
- **someValuesFrom** – doesn't restrict all the values of the property to be instances of the same class, at least one value or more of the property has to be instance of the specified class (i. e. certain class has to be related to the specific class. However this class is also allowed to be related with the instances of other classes in arbitrary way) [MH04]

OWL Lite provides several features for ontology versioning (versionInfo, priorVersion, incompatibleWith etc.) and notation (rdfs:label, rdfs:comment etc.). It also has to be mentioned that OWL Lite just provides restricted cardinality constructs (only 0..1). Modeling of arbitrary cardinalities is only possible in OWL DL and OWL Full. [MH04]

3.2.2.3 OWL DL and OWL Full

OWL DL and OWL Full provide additional features for ontology reasoning like:

- **oneOf** – enables enumeration of the class members; there are exactly as many members in the class as there are enumerated individuals (no more, no less)
- **hasValue** – a property can be required to have a specific individual as a value
- **disjontWith** – set of classes may be stated to be disjoint from each other, disjoint classes cannot have any instances in common
- **unionOf** – allows to create classes containing things from several classes (logical or operator)
- **complementOf** – allows to create classes containing things that are not included in specified classes (e. g. class “nonVegetarianPizza” can be created; it is only allowed to contain pizzas that don’t belong to the “vegetarianPizza” class)
- **intersectionOf** - allows intersections of named classes and restrictions (logical and operator)
- **unrestricted cardinality constructs** – the cardinality constructs are allowed to be defined in an arbitrary manner (0..1, 0..n, 1..5 etc.) [MH04]

While selecting adequate OWL sublanguage for ontology building, it needs to be mentioned that OWL Full allowing maximum expressiveness and syntactic freedom of RDF (arbitrary complex class description, Boolean combinations, property restrictions, allows to treat classes as instances, too, etc.) has no computational guarantees, i.e. it is not guaranteed that the ontology knowledge base including all OWL Full constructs can be processed in finite time. [MH04]

For users who need maximum expressive power while retaining computational completeness OWL DL might best suit. [MH04] **OWL DL** will be used in this thesis to define our IT technology and service categories ontology knowledge base in an expressive way. Computational completeness is vital for our timecard application as well.

3.3 Evaluation of ontology development tools

Ontology modeling tools should facilitate ontology development process by hiding the complexity of ontology mark-up languages. Ideally the user should be able to use and define complex ontology concepts by using convenient and intuitive graphical interface. In this chapter for the purpose of ontology knowledge base development potential development tools will be evaluated. The appropriate tool for development of our ontology will be selected afterwards. Finally, the functionality and features of that tool are going to be described in detail.

The following criteria and features are going to be analyzed while looking for the intuitive and powerful modelling tool that is best suitable for the development of our ontology base:

- **Usability and convenient user interface:** one important criterion for ontology development tool is usability and its simple deployment. It should be easy to install and to use. Intuitive graphical interface for different modeling or modification purposes should be provided by the tool. The graphical ontology representation for its navigation, editing and documentation should be enabled as well. Detailed tool documentation, comprehensive tutorials and circumstantial demonstrative examples may have considerable impacts on the ontology modeling tool selection process. [GF02] All these features necessarily have to be included in the selected modeling tool.
- **Integration and merging of external ontologies:** the modeling tool should provide components supporting integration of external ontologies. Ideally it should be possible to convert, to merge and to adapt existing ontologies in order to build composed ontology knowledge base. The ontology development tool should be able to import and export ontology in various ontology formats (OWL, OIL+DAML, RDF/s) too.
- **Extensibility of the development tool:** ontology modeling tool has to be extensible to additional functionalities. Those new functionalities or already existing external components have to be integrated in the form of plug-ins easily. At best, the ontology modeling tool has to be developed and maintained by a large community providing a considerable amount of different plug-ins for certain purposes and functionalities.
- **Interoperability with other ontology tools and languages:** the ontology modelling tool should be able to interact with other relevant tools supporting the

ontology development process (ontology reasoning tools, storage tools, querying tools, ontology merging and converting tools, evaluation tools etc.) [GF02]

- **Ontology storage and querying:** the ontology modelling tool should provide scalable ontology storage capabilities. Ideally, powerful querying engines have to be integrated to extract needed knowledge from the ontology knowledge base, too. [GF02]
- **Inference services attached to the tool:** for consistency checking and knowledge evaluation purposes the inference services should be provided by the ontology modeling tool. Through these extended inference services new implicit knowledge can be derived and analyzed. Usually those inference services are provided by the so-called reasoner applications. Therefore the ontology modeling tool has to be at least interoperable with the relevant reasoner applications. The inference and consistency checking functions are very important in the ontology development and usage life cycle. [GF02]
- **Integration of the ontology knowledge base in applications:** the ontology modeling tool or ontology repositories have to provide certain API to make the ontology knowledge base accessible to applications or web-services. That API should necessarily offer ontology modification, querying and maintenance options.

The following candidates have been selected for the ontology modeling tools evaluation and selection process:

- Apollo
- Ontolingua
- WebODE
- OntoEdit
- Protégé 2000 [GF02]

All these tools are widespread in the ontology design and development sector and are accepted by relatively large communities. These tools also provide the minimum necessary functionality supporting the ontology development process. Tools that are irrelevant or not accepted enough will not be considered in the present diploma thesis at all.

The functionalities and main features of these five modeling tools must be analyzed in short. Afterwards the features of the most appropriate and therefore selected tool will be presented and explained in detail.

3.3.1 Apollo

Apollo is an ontology modelling tool with user-friendly interface. This tool was developed in cooperation with several industry partners to support the modelling of simple ontology concepts. It was very important for the developers to create tool that would support basic ontology modelling techniques and would provide easily useable and understandable syntax, as well as ontology development environment. [Apollo]

The internal model is built as a frame system according to the internal model of the OKBC protocol. This frame based modeling system enables definition of classes, properties, instances, hierarchies, functions, rules and simple relations. Modelling, navigation, editing and definition processes are supported by the convenient graphical user interface with different possible views. Apollo performs full consistency checking while modeling the ontology knowledge base. [Apollo]

Apollo is an extensible tool. Additional functionalities might be implemented and integrated as plug-in components. However the Apollo community is not large enough and there are not many existing plug-ins that might be integrated for the further use. There is no detailed documentation, demonstrative tutorials and samples either. Apollo is a frame based ontology modeling tool and therefore does not support strongly expressive OWL constructs. The expressive power of the knowledge representation is not strong enough and extended querying facilities are therefore not available.

Apollo has its own internal language for storing the ontologies (files only), but can also export the ontology into different representation languages, as required by the user. Apollo is implemented in Java and provides specific API enabling ontology access and integration. [Apollo]

Apollo provides restricted ontology modeling constructs (OWL is not supported), it is not interoperable with other ontology development tools, it does not provide inference engines; the ontology might get exported in limited formats, Apollo does not have strong community maintaining and improving this application; there are no satisfied documentation and tutorials available as well. [Apollo] Due to these facts Apollo cannot be considered as a potential modeling tool for development of our ontology knowledge base.

3.3.2 WebODE

WebODE is a service that strongly supports ontology development, usage and integration processes. WebODE has been built using 3-tier architecture (client tier-application server-database tier). [WebODE]

Figure 9: WebODE three-tier architecture [WebODE]

The application server (middle tier) provides high extensibility and usability by allowing easy addition of new services and the use of already existing services. WebODE provides well-defined service-oriented API to access and to integrate the ontology knowledge bases into different applications. The ontologies are stored in the relational database (database tier). WebODE ontology development tool supports exports and imports in many different formats (OWL, RDF/s, DAML+OIL, WebODE's XML). Thus, the ontology bases might be integrated and merged easily. [ACFG01]

A convenient and intuitive graphical user interface is provided to define term structures and relations. WebODE supports not only hierarchy definitions and simple relationship concepts but also expressive and powerful modelling constructs (among other things, reflexive and symmetric properties, predefined relations like disjoint classes, unionOf or complement relationships, multiple inheritance, rule definitions etc.). [WebODE]

WebODE also offers consistency checking, inference, reasoning, merge and comprehensive documentation services. It additionally supports collaborative ontology development environment. Synchronization mechanisms allow parallel ontology editing by multiple users. [ACFG01]

However there are not many tutorials, wikis and demonstrative ontology modelling samples available and discussed. This tool is also not freely available (temporal free web access only). [WebODE]

3.3.3 OntoStudio

OntoStudio successor of OntoEdit supports the ontology development process in a comprehensive manner by using graphical means and various extended features. The tool is based on a flexible plug-in framework and is interoperable with other ontology development tools. Additional functionality and features can be integrated for certain purposes. A lot of plug-in based components (inference engines or reasoners, collaborative multi-user ontology editing facilities, import and export plug-ins etc.) are publicly available and might be used in the OntoStudio development application in order to customize this tool for required scenarios and purposes. Ontologies might be stored in files or in relational databases. [OntoStudio]

Short time ago a powerful OWL reasoner OntoBroker was implemented. The main task of the OntoBroker engine is to process expressive OWL DL and RDF/s mark up languages. OntoBroker checks OWL ontology's consistency, infers new implicit knowledge, integrates ontologies originated from different sources as well as provides API for ontology access, modification and integration. [OntoStudio]

There are also comprehensive documentations of most features, extended customer support, as well as tutorials available. [OntoStudio]

However, OntoStudio and OntoBroker server are commercial software releases and are not freely available. [OntoStudio]

3.3.4 WebOnto

WebOnto is a JAVA-Applet supporting collaborative browsing, creation and editing of ontologies. The ontologies created are represented in the knowledge modelling language OCML. The collaborative development is supported by the convenient graphical user interface. A lot of ontologies are provided by WebOnto service and are publicly accessible. [D98]

WebOnto does not support the OWL mark up language constructs. [D98] The modelling of complex relations and expressions is therefore not possible. This tool is not extensible and not interoperable with other ontology development or reasoning

engines. Documentation, demonstrative samples and tutorials are poorly described or not provided at all.

Due to its restricted functionality facilities WebOnto won't be used for the ontology development in this thesis.

3.3.5 Protégé 2000

Protégé provides a powerful graphical and interactive environment for the ontology and knowledge base development. Protégé has a very large community around the world. A lot of industries (e. g. medical sector) are using this tool for ontology conceptual design and development. Protégé has the component-based architecture. Additional functionality can be integrated in the form of plug-in components. Community members all over the world have implemented a considerable number of certain plug-ins that are publicly available and might be integrated fast and unproblematic. Large Protégé community provides many detailed and intuitive described tutorials. All the features of this tool are carefully documented and demonstrated, too. There are also numerous wikis, mailing lists for questions and support, as well as forums for discussions available. There are a lot of ontologies that were created by using this tool. Most of them are publicly accessible and might be used for one's own purposes. [Protégé]

Protégé is used by large communities that build ontologies containing considerable amounts of data. Very important advantages of Protégé are its scalability and extensibility. Therefore, Protégé allows to build and to process large ontologies in an efficient manner. Through its extensibility Protégé might be adopted and customized to suit users' requirements and needs. [Protégé]

Protégé provides powerful constructs facilitating building of large ontologies. It includes outstanding graphical tree navigation as well as extended zoom facilities that allow seeing the ontology in an abstract or detailed manner. Protégé also provides back-end plug-ins for storage of the large ontology knowledge bases, as well as API libraries for ontology modification, reasoning and integration. [Protégé]

Protégé provides powerful graphical and ontology merging plug-ins as well. It supports all established and relevant ontology import and export formats (OWL, RDF/s, XML etc.) and is interoperable with many other tools considerably contributing to the ontology design and development processes. [Protégé]

3.3.6 Direct tool comparison

In the previous part of the thesis five ontology development tools were evaluated and their capabilities were described in general. The point of this chapter is to compare the tools evaluated and their functionalities directly to show their advantages and disadvantages in certain areas. On the basis of this comparison, ontology development tool appropriate for our purposes is going to be selected.

The criteria of the comparison and ontology development tool features are shown in the table below:

Feature	**Protégé**	**OntoStudio**	**WebODE**	**Apollo**	**WebOnto**
Developers	SMI (Stanford University	Ontoprise	Ontology group (UPM)	KMI (Open University)	KMI (Open University)
Availability	Open source	Software license	Software license, temporal free web access	Open source	Free web access
SW architecture	Standalone, Client/Server	Standalone, Client/Server	3-tier	Standalone	Client/Server
Extensibility	Plug-ins	Plug-ins	Plug-ins	Plug-ins	No
Ontology storage	Files, DBMS	Files, DBMS	DBML	Files	Files
Import formats	XML-Schema, XML, RDF(s), OWL	XML-Schema, XML, RDF(s) OWL FLogic	XML, RDF(s) CARIN	Apollo meta-language	OCML
Export formats	XML-Schema, XML, RDF(s) OWL, HTML, Java, Clips, FLogic,	XML-Schema, XML, RDF(s), OWL, FLogic, SQL-3,	XML-Schema, XML, RDF(s) OWL, HTML, Java, Clips, FLogic, Prolog, CARIN	OCML CLOS	OCML, Ontolingua, RDF(s), OIL

Axiom language	PAL	FLogic	WAB	Unrestricted	OCML
Inference engine	PAL	OntoBroker	Prolog	No	Yes
Consistency checking	Yes	Yes	Yes	Yes	Yes
Graphical support	Yes	Yes	Yes	Yes	Yes
Zooms	Yes	Yes	No	No	No
Collaborative working	Yes	Yes	Yes	No	Yes
Ontology libraries	Yes	Yes	No	Yes	Yes

Table 1: Ontology development tool comparison [GF02]

While comparing the ontology tools, two powerful and well documented ontology development environments were identified: Protégé and OntoStudio. These tools provide powerful and interactive ontology modeling concepts, are interoperable with other important ontology development tools, have large community behind them, are well maintained and documented, have support and mailing lists available, provide powerful ontology reasoning engines and support OWL ontology modeling language. OWL support is one of the most important criteria in this tool selection process, because the mark up language mentioned was selected for the modelling of complex ontology constructs and relations.

These two tools provide similar functionality and scalability level. Their functionality facilities may be extended in an arbitrary way and also customized to the developers' needs and requirements easily. Both tools provide convenient and intuitive but also powerful graphical user interface with different views. However Protégé is open source software and is used in many large projects. Therefore Protégé has larger community maintaining its labs regularly and providing considerable number of useful and powerful plug-ins. Protégé community also offers many useful experiences and guidelines for ontology conceptual design and its development as well as its later integration. Due to these facts the **Protégé** ontology editor will be selected to develop our ontology conceptual design and afterwards to build the ontology knowledge base required for our timecard application.

3.3.7 Detailed description of the selected ontology development tool

As has been mentioned above, Protégé is an extensible and powerful application supporting complex ontology development process in a comprehensive way. Protégé-Frame editor (compatible with the Open Knowledge Base Connectivity protocol OKBC) represents the core component of this ontology development tool. In the course of time Protégé OWL plug-in was developed by Stanford University enabling usage of complex OWL mark up language (semantic web standard) constructs. The OWL editor represents a complex extension of Protégé that fits the semantic web requirements and needs. [KFNM04] This OWL editor plug-in is going to be used to specify our ontology concepts in expressive way. In this chapter the OWL plug-in and other relevant plug-ins supporting ontology visualization and merge as well as reasoner server are going to be described in detail.

3.3.7.1 OWL plug-in

Ontology development is a very time-consuming and complex process. The ontology development tool should provide intelligent assistance for developers and facilitate the development process through convenient user views hiding complex OWL syntax, consistency checking, ontology design inspections and visualization facilities. All these features are provided by the OWL plug-in. Furthermore, Owl editor is able to integrate already existing plug-ins (ontology testing, querying, integrating services) to customize the OWL ontology development tool and its power to the developers' needs. Highly scalable Protégé might be used to store large ontologies, multi-user mode based on client/server architecture might be reused for collaborative working. Other applications for the ontology processing (Jena) and reasoning (Racer server) can be included, too. [KFNM04]

To build simple ontology concepts like classes, properties, simple relations, individuals the Protégé core system components might be reused. However the core Protégé API enabling ontology access and manipulation should be extended to the OWL API supporting OWL ontology development. This extended Protégé OWL API implements OWL Lite and OWL DL constructs completely and OWL Full constructs (including meta-classes) partly. [KFNM04]

The OWL plug-in extension and its interaction with other Protégé components are shown in the figure **10** below:

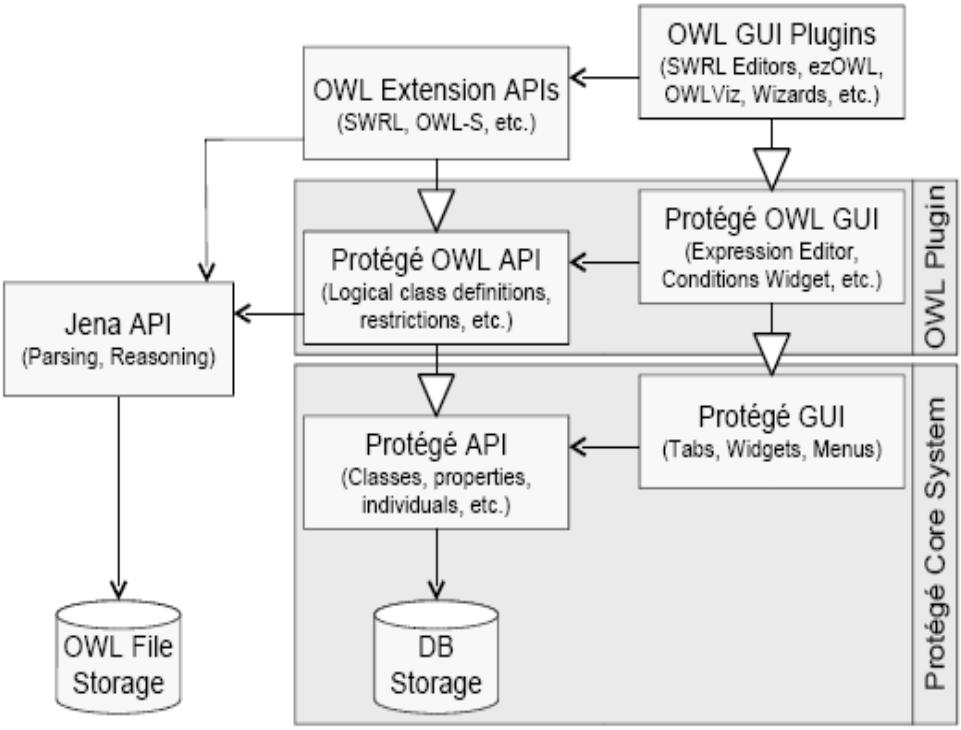

Figure 10: OWL plug-in as extension of Protégé core system [KFNM04]

The OWL plug-in extends the core system components and API to the classes that can implement OWL specification.

The Protégé core API is used to implement the core Protégé user interface (ontology access and manipulation). The Protégé OWL API inherits the functionality of the Protégé core API and extends this API with custom-tailored Java classes for the various OWL class types (unionOf, complementOf, subsectionOf constructs etc.). Afterwards this extended OWL API is used to implement the OWL editor user interface. It's possible for developers to use extended OWL APIs for developing customized plug-ins and therefore provide additional functionality for the OWL editor. [KFNM04] Numerous plug-ins for OWL visualization, OWL ontology merge, querying etc. were already implemented and are publicly available.

OWL plug-in provides comprehensive mapping between its extended API and the standard OWL parsing library Jena. Loaded Jena model is synchronized with all the

changes performed in Protégé OWL plug-in. All the objects defined in OWL plug-in are presented in Jena terms, too. Therefore the Protégé user might integrate arbitrary Jena parsing, querying, and reasoning services. Defined ontologies (Jena objects) are able to be serialized to file in OWL format by using Jena API. There were also several mechanisms implemented to maintain traditional semantics of frame-based Protégé in spite of using OWL syntax. [KFNM04]

Several important plug-ins for OWL editor are presented and superficially described in the following chapters.

3.3.7.2 OWLViz plug-in

OWLViz plug-in was designed by Mathew Horridge at the University of Manchester for further use in Protégé OWL plug-in. The main task of OWLViz is to visualize OWL ontology class hierarchies and its relations (asserted class hierarchies as well as inferred class hierarchy). It enables an overview of generic classes, navigation in the tree hierarchy, quick structure changes and detailed views provided through outstanding zoom facilities. OWLViz provides an easy comparison of the both class hierarchies (asserted class structure and inferred class structure), too. [OWLViz]

In OWLViz certain class hierarchy trees might be exported to various graphics formats like jpeg, png and svg too. This feature is rather useful for certain project specific documentation purposes. [OWLViz]

3.3.7.3 Promt plug-in

Ontology is changed through the constant evaluation process regularly. Ontologies sometimes have to be adapted to different application requirements, modeling concepts or domain interpretations might change over time, design mistakes must be eliminated. Due to these facts appropriate tools for efficient ontology evaluation management are required. Promt plug-in designed for Protégé OWL editor should provide mechanisms facilitating ontology evaluation management. [SAS05]

Protégé provides concrete mechanisms for ontology versioning, comparison of different ontologies and their versions, definition of relations between certain ontology versions. These functions allow developers to store different ontology versions, define relationships, extract differences between certain versions and compare different ontologies. On the basis of that well-structured information ontology incompatibilities can be resolved, and ontology integrations, partly extractions, merges might be performed by **Promt** plug-in automatically. [SAS05]

3.3.7.4 Racer reasoner

OWL plug-in provides direct access to description logic (DL) reasoners such as Racer server [HM03]. Following functionalities are provided by this OWL DL ontology reasoning engine that will be used to infer domain knowledge in our ontology project, too:

- **Consistency checking:** on the basis of the defined relations and restrictions the DL reasoner checks if instances and classes were asserted logically correctly [HM03]
- **Classification (inference):** on the basis of the described relations and term definitions the reasoner derives new knowledge, adds appropriate instances to certain classes satisfying their restrictions, restructures class and property hierarchies in an appropriate way (new classification and concept reorganization according to the defined rules and restrictions) [HM03]

Ontology reasoning tools perform one of the most important tasks in ontology usage process, namely, generating new assumptions and cognitions that are not defined in the ontology knowledge bases explicitly. [HM03]

3.4 Interactive ontology building process

An important part of this diploma thesis is to create the ontology knowledge base that is to be integrated in an already existing timecard application. That ontology knowledge base will contain partly selected terminology of IT sector (IT technologies, programming languages, developers' software, as well as IT software engineering and consulting services – provided within Siemens Austria AG). The actual ontology development process, numerous modeling decisions and ontology's structure as well as its content are going to be documented and explained in the current chapter in detail.

The comprehensive ontology building process will be presented below. It should visualize all the steps that are necessary to build a useful ontology knowledge base successfully. Afterwards single steps and activities are going to be explained in a more detailed way.

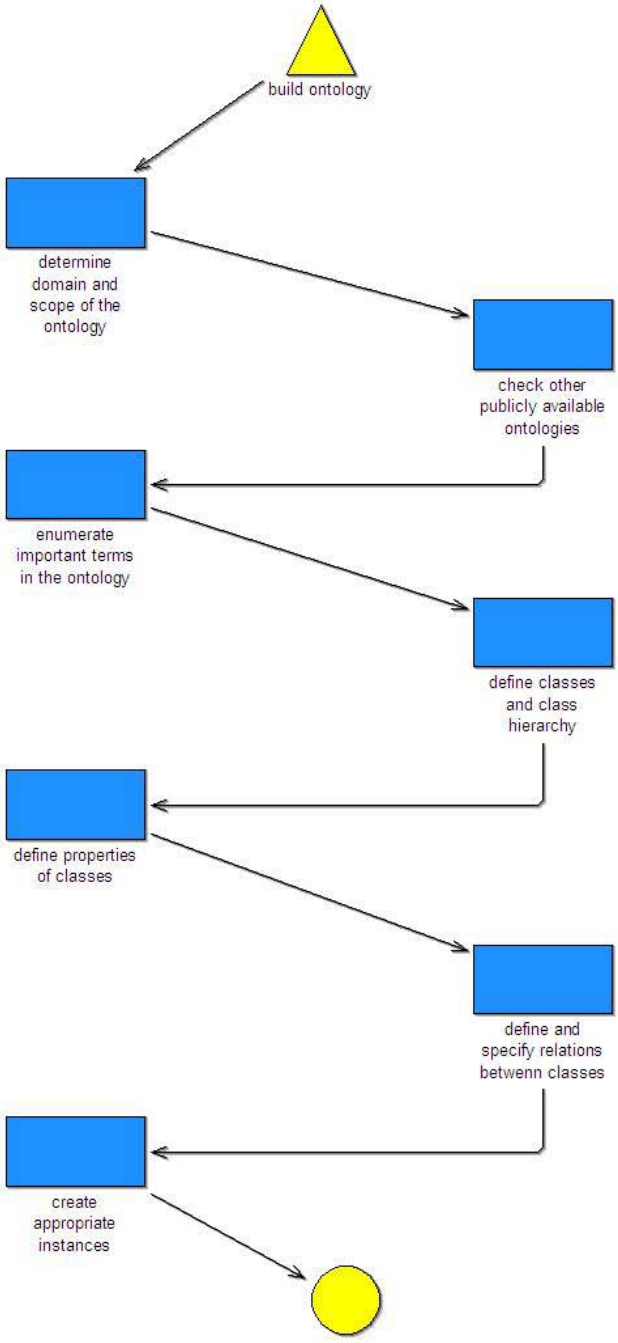

Figure 11: Ontology building process [HKRSW04]

The first important step is to understand what our domain of interest is, i.e. it has to be considered which information is required by the timecard application. In our case IT domain represents a high level of importance. The timecard application has to be able to process terminology of IT sector such as IT technologies, as well as IT engineering and consulting services. Programming languages, different kinds of developers' software, databases, internet technologies etc. and their interrelations are going to be modeled in our IT technology knowledge base. The software engineering and consulting service categories will be included in our ontology, too.

It is also very helpful to inspect and analyze the already existing ontologies to get the feeling about general ontology structures and modeling patterns, as well as constructs. Ontologies covering similar domains of interest might be imported and reused. However, for our needs ontology integration and adaptation processes are rather complex and time-consuming. It is possible to get advantages of external ontology imports if really considerable ontology fragments might be reused. For the building of our IT technology knowledge base concepts and content of following ontologies will be included and adapted to the needs of the timecard application:

- DMOZ ontology (computers section) – provides internet links for all possible IT tools, technologies, methodologies, architectures, tutorials etc. All the IT components are grouped in classes and hierarchies. A lot of dependencies and interrelations are defined and publicly maintained. Ontology editors may register and publish their links (represent instances) for appropriate categories. They are also allowed to add new IT components and categories. Therefore DMOZ ontology is regularly extended and maintained by registered ontology editors contributing their expertise. The consistency checking might be regularly performed by DMOZ ontology providers. [ODP]
- ITEC categorisation – IT engineering and consulting service categorisation internally used in Siemens Austria AG. That standard defines various software engineering and consulting services provided within Siemens Austria AG. The paper mentioned includes service categories such as Software engineer, consultant, design or solution architect, project manager, program manager, principal etc.

In order to get a general idea about the ontology scope, it is very helpful to list important terms and concepts. That list helps to understand the structure and elements of the ontology more clearly, too. After the ontology scope and composition are mostly clear, classes and their hierarchies have to be specified as entirely as possible.

Some properties might be defined to describe specified classes accurately and to enable advanced querying facilities during further ontology use and processing. One of the most important tasks is to consider and to specify interrelations as well as dependencies between the classes represented. Extensible rules have to be defined that assign class restrictions, interrelations, cardinalities and dependencies. According to those rules the reasoner server applications can infer new facts, perform taxonomy classification and provide qualified, as well as logically proved, assumptions.

Classes represent a set of instances that might be created in the final ontology building phase and described through the already defined properties.

3.4.1 Taxonomy definition

As first it is very important to mention that there are many alternatives to model ontology knowledge bases describing certain domains. There is no completely correct alternative because of the different views, interests, interpretations and needs. [HKRSW04] You are on the best way to model the ontology successfully if you periodically have application, further using the ontology mentioned, in mind, as well as its needs and specified requirements. If the ontology delivers required data sets and can fit the needs of the application, then the ontology model was specified in the best way. [HKRSW04]

Ontology modelling is an iterative process. The conceptual shortcomings become visible in the course of the actual use of a certain ontology by applications. Those defects are to be corrected, and the conceptual decisions must be adapted carefully and deliberately. [HKRSW04] In this way the ontology knowledge base is going to be extended and updated regularly during its use.

To make the ontology processing by humans, as well as by machines, easier, it is necessary to name classes, their properties, interrelations and instances close to the real world objects (close to the domain of interest terms and relations). [HKRSW04]

The top-down approach is going to be used to classify our taxonomy. As first the most general concepts are to be considered, as well as their siblings. These siblings have to be inspected and, if necessary, grouped into a certain general concept. This way they become subclasses. It is very important to keep balance. On the one hand, the ontology is not defined concretely enough if generic classes have too many subclasses. Thus, detailed information might not be extracted by application. On the other hand, it is counterproductive and not really meaningful to have only one

subclass. It also has to be considered that the ontology might be extended during its use, and additional subclasses or siblings might be inserted.

While modelling the ontology knowledge base, it is very important to follow a consistent naming strategy for the further maintaining and processing reasons. [HKRSW04] All the classes will start with capital letters, and properties - with lower case characters (e. g. hasStatus). Spaces are going to be represented as underlines (e. g. Programming_Languages). A constant naming strategy should facilitate the ontology processing, querying and navigation by humans. [HKRSW04]

The IT ontology will include three top level classes. The ITTechnology class is going to include the IT concepts important and useful in our case. Business_Services class will include the IT engineering and consulting service categories, according to the ITEC standard. The service categories mentioned determine the charging level for performed services. Professionals might be engaged in complex and responsible tasks, and therefore ask for higher hourly earnings. Management might be interested to follow labor market trends and analyse proportion changes of cleared complex tasks to the routine tasks. If complex tasks requiring highly qualified experts gain importance, additional educational seminars and programs are to be initiated to adapt employees' qualification level urgently or external labor forces must be contracted in time. The third top level class Employee_Status is irrelevant and is just used to determine the employee classification and their charging amounts indirectly.

These top level classes include various subcategories that are going to be shown in the figure below. This figure also gives a look at the Protégé ontology editor used to model our ontology knowledge base.

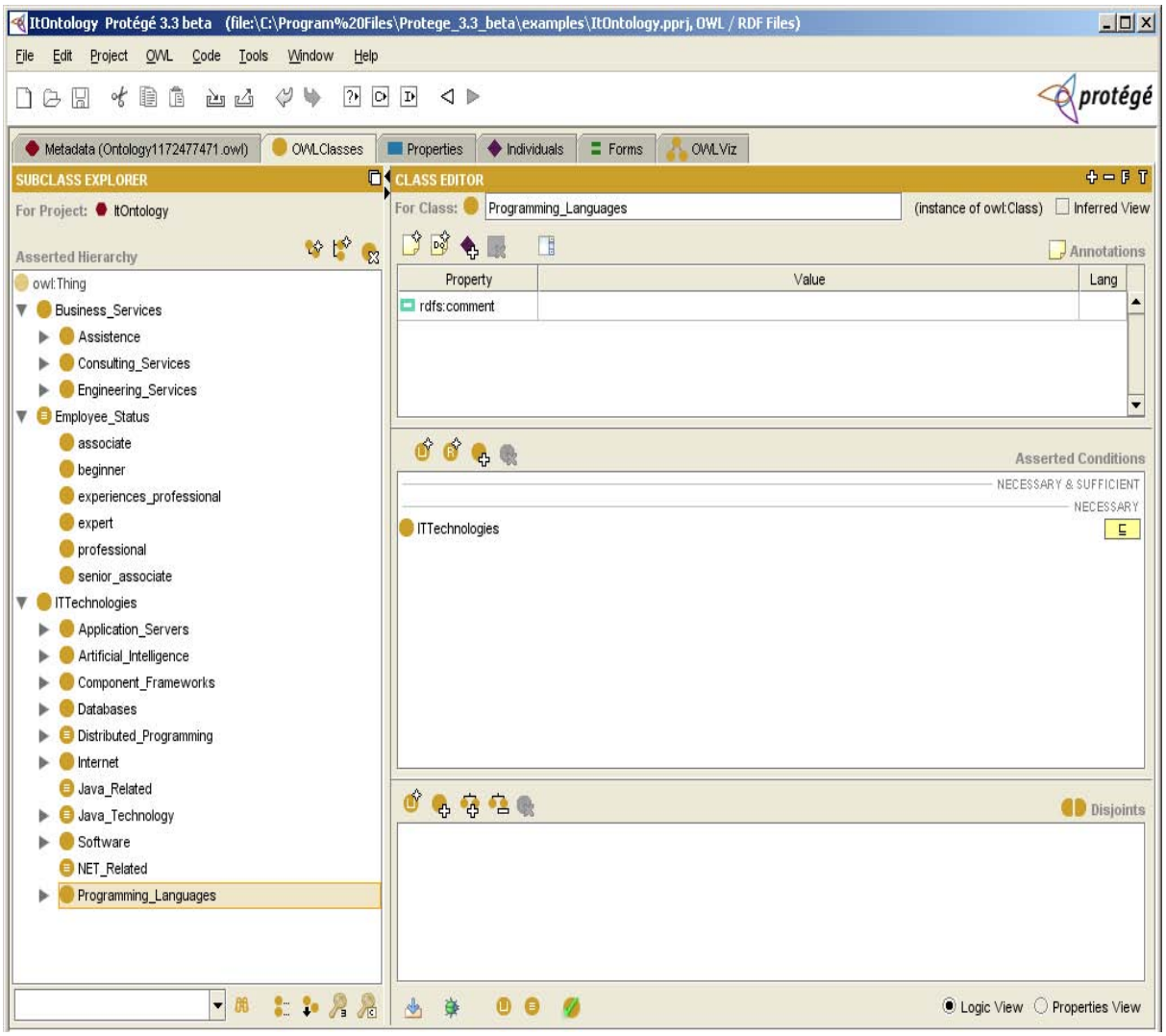

Figure 12: Generic IT ontology structure shown in Protégé editor

The core class of this ontology is ITTechnology concept. This class, its subclasses, as well as dependencies and interrelations, will be described in detail. OWLViz plug-in will be used to visualize those concepts and reasoner activities.

In order to facilitate ontology maintenance and avoid redundancies class lists are going to be declared and described (e. g. lists of programming languages, developers' or users' software, various IT components etc.). Afterwards data categories, necessary

or relevant for the timecard application, are going to be created (e. g. Scripting_Languages, ObjectOriented_Languages, ServerSide_Languages, LogicBased_Languages, Commercial_Application_Servers or databases etc.). According to the logic-based concept descriptions, the reasoner will be able to classify the defined classes and move them to the appropriate categories in the class hierarchy. Used approach allows adding new IT components, software or programming languages easily. After a new class has been added, logic-based concept description has to be accomplished. Automatic concept classification performed by the reasoner will integrate the classes added into appropriate tree hierarchies. In this way arbitrary additional properties might be defined to describe classes in a more expressive way and therefore infer new arbitrary knowledge, if required by the timecard application.

In this section the most important concepts will be picked up to show the general ontology building constructs and patterns, reasoner power, as well as logic-based concept description facilities and further automated ontology classification.

Firstly, it is very helpful to determine the general concept restrictions. [HKRSW04] Through these restrictions reasoner server becomes able to perform basic ontology consistency checking and reasoning activities. The following basic restrictions might get defined in the initial modeling phase:

- Define disjoint classes of certain concept – the concept can not share the same individuals with its disjoint classes
- Define possible Union restrictions – a concept may include individuals of several classes defined in the union property (logical or)
- Define possible intersection restrictions – a concept may include individuals representing intersection of several classes (logical and)

Union restrictions and intersection restrictions might be declared through logic-based rules or by using owl:intersectionOf as well as owl:unionOf commands. Protégé OWL plug in provides direct constructs to define disjoint classes that are shown in the figure below.

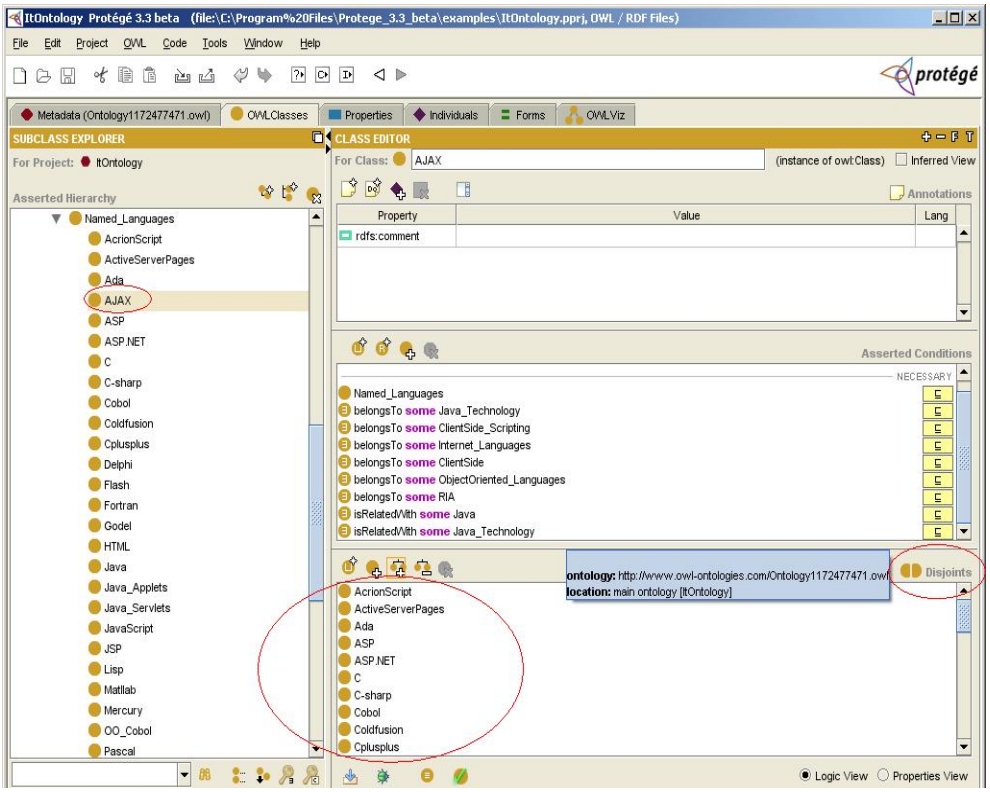

Figure 13: Defining disjoint classes

Category Named_Languages is a subcategory of Programming_Languages. It includes all programming languages defined in our ontology knowledge base. All these languages are described through the already defined properties:

- belongsTo – this property defines how concepts belong to each other, to which groups, super categories, vendors, producers etc.
- isRelatedWith – this property is symmetric and transitive. It defines which technologies are related with each other in any way.

The property belongsTo defines to which technologies and concepts Ajax belongs. This information can be used by Racer Server Reasoner to move Ajax concept to the appropriate defined categories and technology groups (e. g. Racer can add Ajax to internet client side languages, scripting languages, Racer can assume that Ajax is part of Java technology etc.). Depending on the application needs, rule definitions and

properties quantity might be extended to express certain concepts in more detail in order to extract additional information.

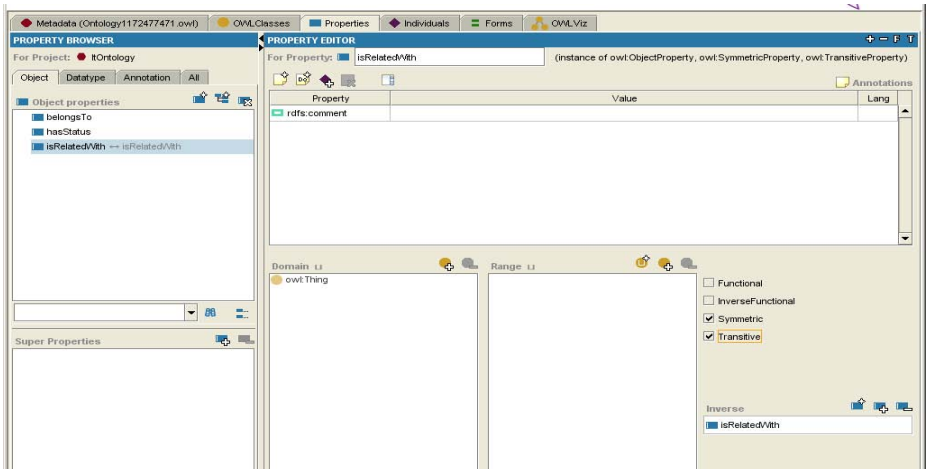

Figure 14: Protégé properties view

After the technologies and certain programming languages, as well as software units, were defined the reasoner server might be integrated to check the consistency and to classify the ontology in an adequate way. All the logical rules were defined as the **necessary** conditions to describe the explicit asserted technologies. Ajax is, for example, a subclass of Named_Languages, that belongs to certain technology groups **or** vendors **or** is necessarily related with several technologies. Afterwards it is necessary to consider which information is required by the timecard application. Management is interested in the technology portfolio. Regarding programming languages, it would be interesting to group them in relevant categories. It could help us to extract some information about usage proportions of certain programming language categories within an organization. The following concepts might be created in our ontology:

- Internet languages
- Object-oriented languages
- Procedural languages
- Client-side languages
- Server-side languages
- Logic based languages
- Open source languages
- Scripting languages

- Distributed programming languages etc.

The above-mentioned programming language categories might be used to classify the programming languages explicitly entered into the timecard. These categories could be integrated in management portfolio for technologies to illustrate their importance and how intensively they are used in IT projects within an organization. In the figure below explicit asserted programming language categories are partly shown.

Figure 15: Asserted programming language categories

To add appropriate programming languages to the asserted categories, **necessary & sufficient** conditions must be defined for every specified category. For example the category “internet languages” includes two subclasses ClientSide and ServerSide programming languages. Specific rules are to be defined, giving the reasoner server information that only internet or (internet and client side) or (internet and server side) languages are allowed to represent the sub classes of given internet languages category. As already mentioned above the necessary conditions of all explicit defined programming languages must declared before that. Inferred ontology classification after defining **necessary & sufficient** rules for each programming language category:

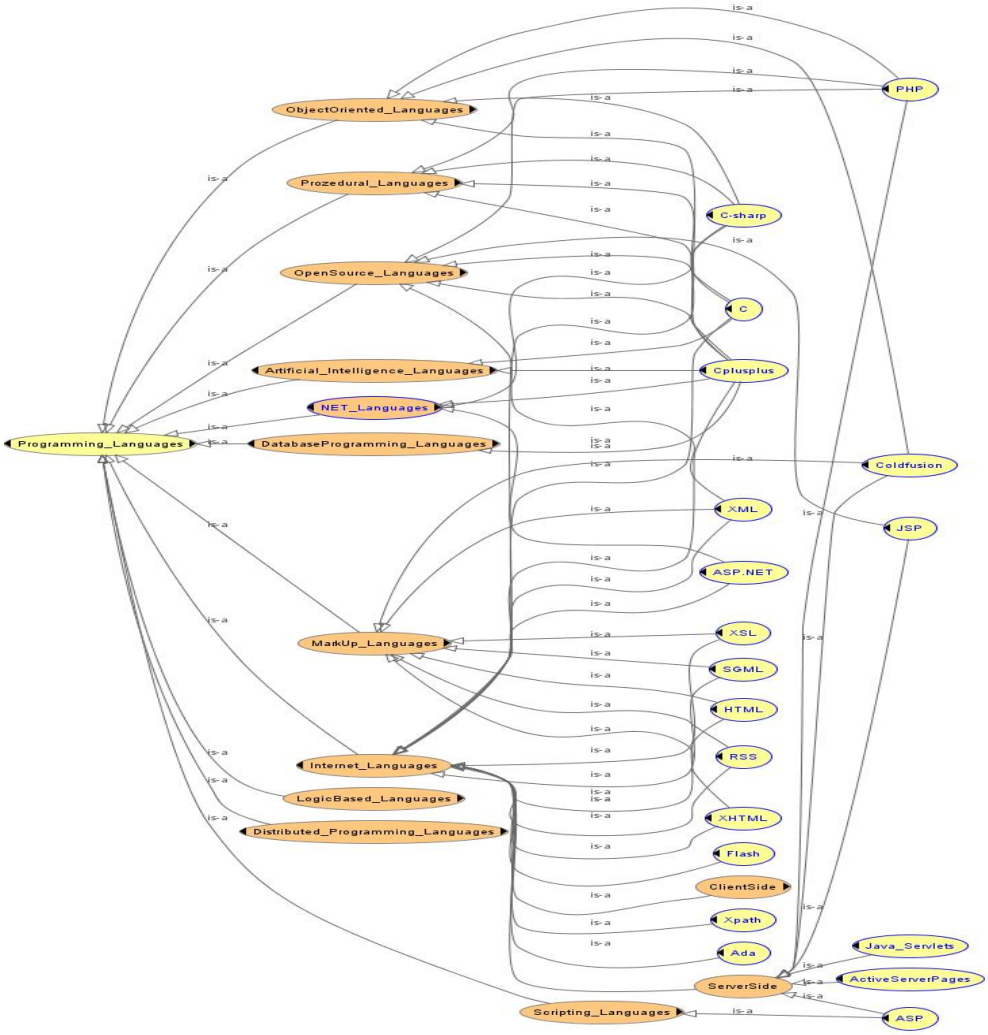

Figure 16: Partial cutout of the inferred programming language categories

Another very important subcategory of the IT technology class is software. In the subclass *Named_Software* many different software products were defined and clearly described through given properties. For technology portfolio inspection it might be interesting, if commercial or open source software products are mostly preferred, and also which kind of web servers, databases, application servers or developers' software are intensively used within departments. The software units can be described through logic-based definitions arbitrarily. On the basis of those rules, the required information might be inferred in the following way. In our ontology it is clearly defined through the **necessary** conditions, if the software units belong to the databases, browsers, application servers, web servers, commercial or open source software type, editors of different kinds etc. These conditions also might be combined to infer an additional knowledge. For example, specific software might be described as database, commercial, belonging to certain vendor, used in server side applications etc. to enable reasoner classification to move the described software unit to appropriate category groups. Editors might also represent commercial software, rich internet application development environments, belong to certain vendor, be used to develop internet applications, support distributed programming etc.

After the software products are described through logic-based rules, they become able to get distributed to defined categories by the Racer server reasoner. The category classifications were created according to the timecard application requirements. Through those categories it becomes possible to group technologies explicitly inserted in the timecard and developers' software products. Thus, the following categories were defined in the ontology knowledge base to group software products:

- Application Servers – commercial and open source application servers
- Databases – commercial and open source databases
- Artificial intelligence
- Distributed programming
- Internet software – clients as browser, ftp, ssh, utilities. Servers as web servers, but also ftp or ssh servers etc.
- Rich internet software – also represents sub category of internet class
- Web-Service enabling software – is also classified as part of internet concept
- Java technology – java commercial or open source editors, J2EE application server implementations, Java micro edition software, java software products etc.
- Component frameworks – software used to support and realize component-based programming

- .NET software products – web servers, application servers, databases, web-service enabling products, remote programming enabling software, commercial or open source editors etc.
  Partial cutout of the asserted software products is shown in the figure below:

Figure 17: Partial cutout of the asserted software products

After reasoner reorganization and classification the defined software products were added to the specified categories. You can partially see the inferred commercial software concept in the figure below.

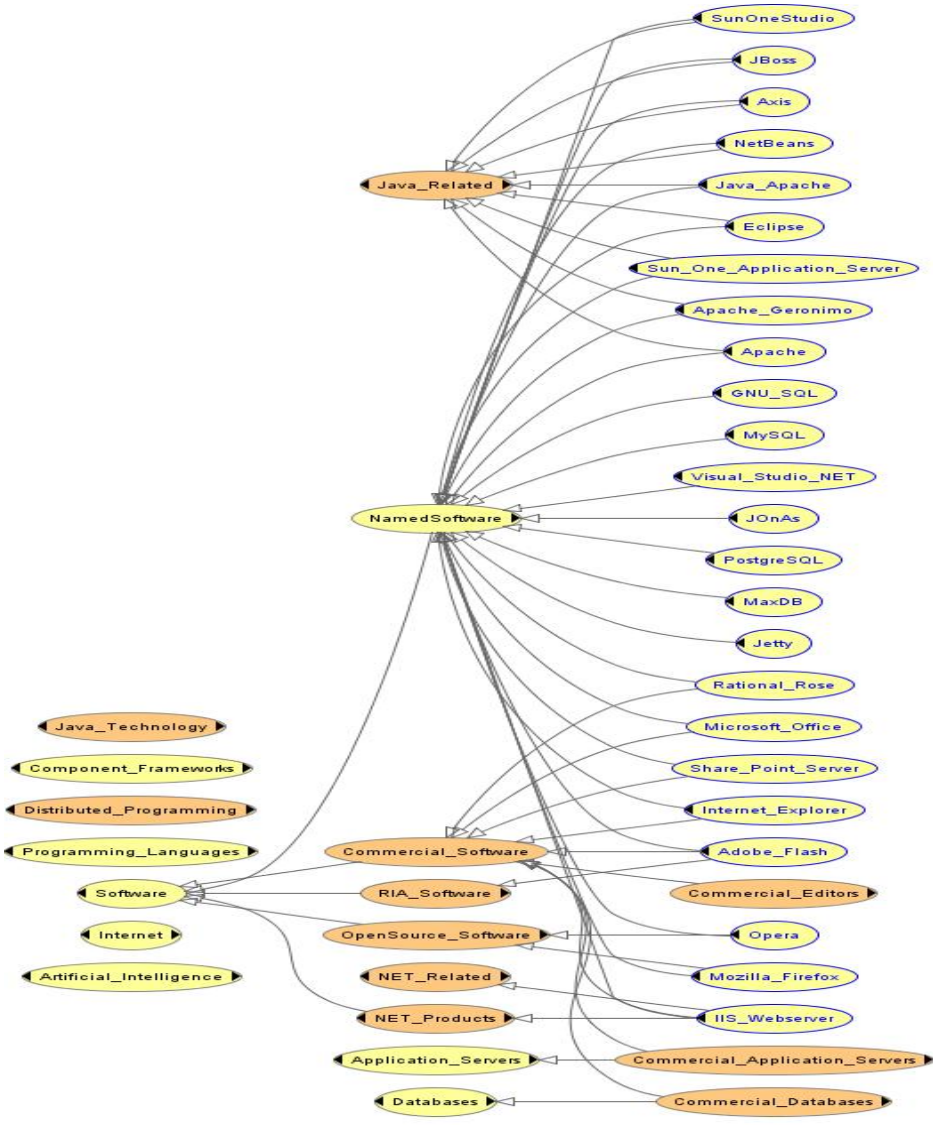

Figure 18: Partial cutout of the inferred commercial software products

In the screenshot above only a small cutout of the inferred ontology, as well as commercial software products, is presented. The commercial software category consists of commercial application servers, commercial editors and commercial databases with their subcategories. They, in turn, also have their instances that were added from the named software concept during the reasoning process.

After specified software products are grouped to certain concepts, the following questions could be analyzed for the technology management portfolio:

- To what extent is commercial software used in the IT projects?
- To what extent is open source software used in our IT projects?
- Which kind of databases is commonly used?
- Which kind of the application servers is mostly used (commercial, open source)?
- What are the popular software vendors in our IT projects?
- Which IT technologies are offered by certain software vendors?
- Which software vendors offer open source technologies?
- How many hours were the rich internet technologies used?
- How intensive are the Java technologies used in internal and external IT projects?

After all that information is defined in our ontology knowledge base, the timecard user ideally just has to enter technology name or software product used to accomplish a certain task. Through technology description the timecard application will be able to generate management reports demonstrating e.g. usage proportions of open source software to commercial software, even if this information was not inserted in the timecard application explicitly. If other information for the management reports is required, additional concepts and logic-based descriptions must be specified to extract necessary information. In this way the ontology might be extended during its life cycle regularly.

It has to be mentioned that our top level ontology categories also include other subcategories with appropriate content. That content is partly asserted by the ontology editor directly and partly imported during the above-mentioned reasoning process.

The internet top level category includes its clients and servers. The content of client and server subcategories partly represents the software products that are automatically added during the reasoning process. The internet category also has internet languages (divided in server-side and client-side languages) subcategory that is filled with the programming languages already defined in the category programming_languages. Web-services category (sub category of internet concept) becomes filled while reasoning process with the web-services enabling software, web-

service programming languages. Rich internet application category will be filled with the already defined RIA software, RIA programming languages and other RIA technologies that are partly inferred by the reasoner but also explicitly asserted by ontology editor. Of course, all the rules must be defined in the ontology knowledge base beforehand. It also has to be defined what kind of content should be included in certain categories and its sub categories. On the other hand, the structure of internet concept and information like web-service specifications have to be asserted by the ontology editor explicitly.

The databases category consists of commercial databases, open source data bases and database programming languages subcategories. All the data are already defined in other categories (software, programming languages) and will be added to the databases concept during the reasoning process.

Remaining IT technology top level concepts are defined in the similar way and partially consist of the asserted or inferred knowledge (knowledge imported from other categories through certain rules and constraint definitions).

Business services represent the second important part of our ontology. In that concept the IT engineering and consulting service categories are defined. Specified categories are used to determine the professional status of employees and therefore their charging level for the services provided. Experienced professionals generally handle complex tasks with high responsibility level and are, therefore, usually highly paid for the services performed.

The Employee status class is used in our ontology to define employees' status and, consequently, their charging level. The employee status is defined through *hasStatus* property. After all general employee information and rules are defined, the following information might be extracted for the management reports in the timecard application:

- How many highly paid services were provided in the actually running or already finished projects (in hours)?
- How many cheap services were performed during the projects?
- What is the proportion of certain services?
- How many consulting services were charged (in hours)?
- How many design services were charged?
- How many software engineering services were performed etc.?
- Arbitrary service proportions and their changes might be derived to forecast future service trends

All those answers and reports may be provided by the timecard application. The employee does not need to insert additional information in the timecard application. Ideally he only has to enter service category listed in the business service concept. The remaining information will be inferred by the ontology knowledge base, where all those service categories are accordingly described. Structure of the business service category is partially visualized in the figure below:

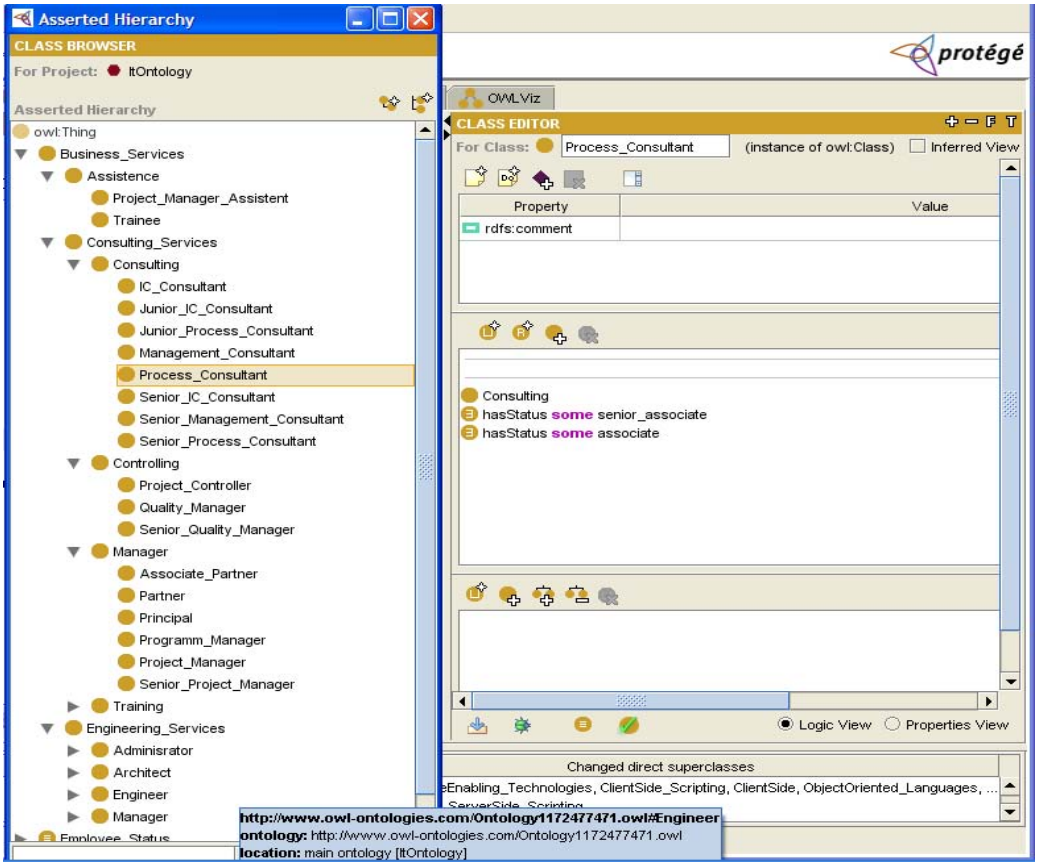

Figure 19: Subcategories of the business service category

In our IT ontology approximately 250 named classes were explicitly asserted by the ontology editor. Many additional classes were inferred through logic-based rules by the Racer server reasoner. About 450 restrictions and rule definitions were specified to build adequate taxonomy and concept interrelations. The Racer server

was used to check the ontology consistency and to perform its taxonomy classification.

3.4.2 Ontology reasoning

In this section the already mentioned reasoning process and its facilities will be shortly demonstrated and explained on the basis of a small example. This example should provide basic idea of how the information processed by the reasoner must be declared and, therefore, which possible ontology modeling constructs or patterns could be implemented.

The short example will show how **necessary** constraints must be defined to describe certain concepts. It is very important to mention that necessary constraints might be declared in arbitrary complexity. Firstly it has to be considered which information is necessarily required by the application further using the ontology knowledge base. Afterwards certain constraints must be defined to express the concepts in the required way. On the basis of these constraints, the relevant information might be extracted by querying the ontology knowledge base. The role of **necessary & sufficient** constraints is going to be explained, too. These constraints provide crucial information for the reasoner tool that will be used during the ontology processing activity.

The Racer server reasoner is used in the thesis to perform ontology consistency checking and classification processes. Racer server is available for download from http://www.sts.tu-harburg.de/~r.f.moeller/racer/ and can process the OWL DL ontologies. [MH04] Racer provides TCP service on port 8088 and HTTP service on port 8080. The HTTP service is enabled by default. [HM03] This way the ontology could be sent via HTTP service to the RACER server reasoner for the further processing. Manually constructed class hierarchy is called *asserted class hierarchy* in Protégé OWL editor. The asserted class hierarchy is going to be sent to the reasoner. Racer performs consistency checking of the asserted ontology and infers new class hierarchy based on the defined constraints and rules. So called *inferred class hierarchy* is sent back to the Protégé OWL editor via HTTP service. [HKRSW04]

In our example certain software products will be picked up to demonstrate basic rule definitions and constraints provided by the OWL DL specification. The necessary constraints are used to define concepts and to specify which members or subclasses are allowed to belong to the specific concept. If some classes or subclasses fulfill the constraints defined in a certain concept, then they become allowed to represent

concept's member and to be placed in its subcategories. Simple rule definition sample is visualized in the figure below:

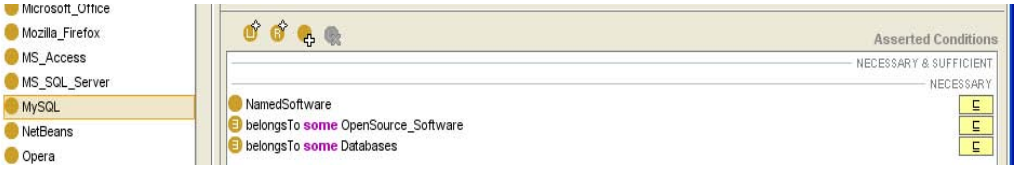

Figure 20: Necessary constraints of MySQL database concept

MySQL database necessarily has NamedSoftware as its superclass, belongs to open source software class (some ->belongs at least to one open source software) and to databases class. *Some* restrictions mean that certain class might belong to open source class, databases class and to other arbitrary classes that might be defined later. [HKRSW04] It is possible to perform so-called closure axiom. This axiom enables to define the completeness of a certain concept. [HKRSW04] In case of MySQL concept closure axiom would look as follows -> `MySQL belongsTo only (open source software or databases).` In this case MySQL might only belong to open source software or to databases class. The class is completely defined and is not allowed to represent other classes at all. Closure axioms are rather useful for the so-called excluding querying mechanisms, [HKRSW04] e. g. show me all software products **not** representing commercial software products. Reasoner will include MySQL in the query result, because of its clear definition (only belongs to open source or only to databases classes, and nothing else). Reasoner will not add MySQL to the query results if just *some* constraints are defined because of the incompleteness of the concept. MySQL might also belong to commercial software products if it is not excluded explicitly.

Necessary & sufficient constraints are used to define following restrictions:

- Only classes fulfilling necessary & sufficient constraints are allowed to be members of a certain concept [HKRSW04]
- All ontology classes fulfilling those constraints must be added to a certain concept in an appropriate way (reasoning process produces inferred class hierarchy) [HKRSW04]

The figure below demonstrates the necessary & sufficient constraint in the databases concept:

Figure 21: Necessary & Sufficient constraints of the open source databases concept

The necessary & sufficient constraints shown above provide the following information for the Racer reasoner server:

- only concepts belonging to the class databases **and** belonging to the class open source software are allowed to be added to the open source databases concept (both constraints have to be fulfilled in parallel)
- all the concepts available in the IT ontology knowledge base and fulfilling these constraints in parallel have to be added to the open source databases concept
- open source databases concept must be added to open source software concept, too
- as the necessary condition open source databases have their super class called databases

After defining the rules and performing ontology classification presented above, the following changes are visible in the inferred class hierarchy compared to the asserted class hierarchy:

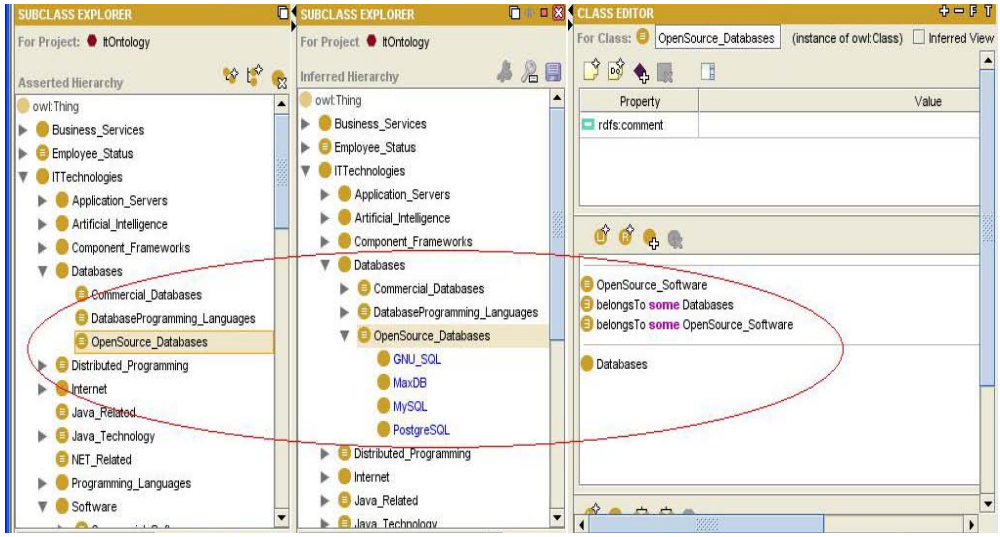

Figure 22: Asserted vs. inferred class hierarchy

As visualized in the figure above, several software products fulfilling defined constraints were added to the open source databases concept. Open source and commercial databases concepts were added to the open source and commercial software concepts, too.

In this way arbitrary constraint definitions are allowed to be performed to determine which concepts are allowed to be added or necessarily have to be added to certain categories.

3.4.3 OWL syntax

Protégé OWL editor facilitates ontology modeling process dramatically. Ontology constructed in Protégé might be exported in OWL format and then get integrated in other ontology knowledge bases or processed by arbitrary applications. Protégé is using the Jena model to provide appropriate API that allows navigating, modifying and querying the OWL ontology knowledge bases. Ontology integration and querying processes will be explained and demonstrated in detail in chapter 4 “Web-service based component integration”**.**

Small OWL ontology cutout is shown below to demonstrate the OWL syntax and how OWL ontology describing the real world concepts actually looks like:

```
<owl:Class rdf:ID="NET">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Component_Frameworks"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="EJB"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Corba"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="OpenSource_Languages">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="belongsTo"/>
            </owl:onProperty>
            <owl:someValuesFrom
rdf:resource="#OpenSource_Languages"/>
          </owl:Restriction>
          <owl:Class rdf:ID="Programming_Languages"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
<owl:Class rdf:about="#DatabaseProgramming_Languages">
    <rdfs:subClassOf rdf:resource="#Databases"/>
    <owl:equivalentClass>
```

```
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:someValuesFrom
rdf:resource="#DatabaseProgramming_Languages"/>
            <owl:onProperty rdf:resource="#belongsTo"/>
          </owl:Restriction>
          <owl:Class
rdf:about="#Programming_Languages"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

This OWL syntax describes defined .NET, open source programming languages and database programming languages concepts.

4 Web-service based component integration

In the following chapter the procedure and approaches, that are used to integrate necessary components in our timecard application, will be described and demonstrated. For improved maintenance, extensibility and flexibility reasons, the distributed software development approach will be used to implement the functionality of our timecard application. More precisely defined service-oriented architecture (SOA) has to be applied in our timecard implementation to ensure loose coupling and independency of the software components used. Functionality provided via web-services can also be reused in other applications for different purposes.

Firstly, the implementation of Coldfusion web-services was considered to integrate the back-end application logic. Due to the fact that current Coldfusion application server version (7.02) only supports java virtual machine version not higher than 1.4* and Protégé OWL java library for ontology integration not lower than 1.5*, further incompatibility problems can be definitely prognosticated. After numerous tests the incompatibilities between Protégé and Coldfusion were proven. Thus, a workaround is necessary to integrate in Protégé modelled OWL ontology knowledge base by using its java library successfully.

After in-depth research, an effective and efficient alternative to Coldfusion application server could be found for ontology integration reasons. The Apache Tomcat web server in combination with soap enabled Apache Axis application server represent a scalable and stable environment for the building and publishing of java web-services. That platform can be set up easily and facilitates web-service creation dramatically, too. The timecard back-end application logic and database queries will be provided by the Coldfusion web-services. The following strategy encapsulates the whole application back-end functionality, as well as ontology integration, totally, and therefore can be integrated in other applications for various reasons easily, too.

To provide basic understanding about distributed software development approach, the SOA (service oriented architecture) and web-services concept will be shortly explained in the following subsection. Afterwards the environments for java web-services and Coldfusion web-services, as well as their architectures, will be described

in detail. Finally, the web-services themselves and their functions are going to be described in a circumstantial way.

4.1 SOA and web-services basics

The distributed approach enables an easy and flexible functionality integration, as well as process-oriented proceeding. Simplified expressed, functions available in intranet or internet should be consumed to implement process activities. In reality complex web-service interaction procedures, security, service quality and composition issues must be considered and applied. [W04, p. 111] To improve the understanding of the web-services concept, simple service-oriented architecture constructs need to be mentioned and explained, first of all.

Basically, the service-oriented architecture concept is rather simple. There are service suppliers and consumers. The services are implemented by supplier and registered in the public directory UDDI (universal description, discovery and integration). In that public directory service type, category, vendor, functionality etc. are clearly described. On the basis of the information provided, service consumers can search for the required and relevant services, as well as integrate the found services that best fit their needs. [HL04, p. 14] The SOA concept and its role interactions can be visualized more precisely and clearly in the following way:

Figure 23: Service oriented architecture [HL04, p. 16]

The figure above shows that the public UDDI directory just provides web-service publishing and discovery opportunities. The consumer can extract service address (end point) and therefore find out where the necessary service is actually located. Service requests and responses might follow afterwards. Web-service function computing is performed by the service provider's server. The computing results are finally sent to the service requestor. [HL04, pp. 14-16] The SOA idea must also be realized through concrete specifications and approaches. The web-services concept is seen as new distributed approach that might potentially fulfil challenging SOA requirements in appropriate way. [DJMZ05, p. 25]

Web-service technologies, standards and certain specifications are already widespread and used in many organizations. Web-services enable flexible process-oriented proceeding and can be adapted to the changing environment quickly. They allow fast and uncomplicated integration of external services and functionality reuse. Through the standardization of xml, certain communication protocols, as well as web-service description, platform and programming language independent interactions became possible. The computing power can be distributed among many machines by using web-services. That fact might increase computing performance and scalability in the required way. The organizations are also not necessarily made to implement complex functionalities (e. g. maps with road works in certain regions), but just integrate the already existing services providing that functionality for certain charge. That proceeding allows the organizations to concentrate on their core business and to avoid high expenses for secondary purposes. [V05, pp. 403-405]

Many advantages of the web-services were presented above. Anyway, it is still not enough to deserve the acceptance of the business world completely. Many issues, like web-service composition, semantic machine processible description, quality of service, permanent availability, security, transaction management and controlling etc. are still inconsistently defined. The W3C consortium and industry global players are working hard on these topics. Many, or maybe even too many, standards and specifications referring to those issues have already been produced. However many of them are partially redundant or contradictory and make the potential users insecure in this way. [W04, pp. 108-111] Due to the fact that web-services don't represent the core topic of this thesis, it is not relevant to go into detail of those extended web-service standards. Just core web-service standards, that are also partly used for back-end logic components integration, will be mentioned and shortly described.

The core web-service components that are crucially necessary for web-service description, consumption, publication and messages exchange are:

- SOAP - is simple XML based message exchange protocol, basically using HTTP transport protocol. SOAP enables web-service communication (request, response) in a standardized format.
- WSDL – describes methods, for processing necessary parameters, result composition in XML format. Thus, the consumer can inspect methods provided by web-service, required arguments and expected results.
- UDDI – describes public directory and web-services listed in there (web-service category, vendor and its contact information, functionality provided or additional information concerning offered web-service etc.). It enables web-service publication and its facilitated discovery. [G03, pp. 163-177]

4.2 Java web-services with Tomcat/Axis

To provide the appropriate environment for java web-services development, as well as deployment, the following infrastructure was installed:

- JDK (java development kit) version 1.5.0_06) – can be downloaded from http://java.sun.com/
- Apache Tomcat 5.5.12 used as servlet container – open source software, downloadable from http://tomcat.apache.org/
- Apache Axis 1.3 for soap messages processing - open source software, downloadable from http://ws.apache.org/axis/
- Apache Ant build tool with Axis and Tomcat tasks – open source component, downloadable from http://ant.apache.org/
- Eclipse IDE (optional with web-service plug in) – comprehensive open source java editor and development environment, downloadable from http://www.eclipse.org [M04, p. 9]

Firstly, the home variables of Apache Tomcat, JDK and Apache Axis, as well as necessary classpaths, need to be set. The Axis soap engine must be integrated into the Apache Tomcat web server. Soap web-services provided from Axis will therefore be deployed as web applications. [Apache] All necessary java libraries as well as Protégé java libraries have to be copied into the Axis lib directory.

After Apache Server was started axis test might be performed to check installation status. The Axis happiness page http://localhost:8080/axis/happyaxis.jsp should give

ok if all necessary and optional components are found in their expected locations. [Apache]

Tomcat/Axis architecture is simplified visualized in the figure below:

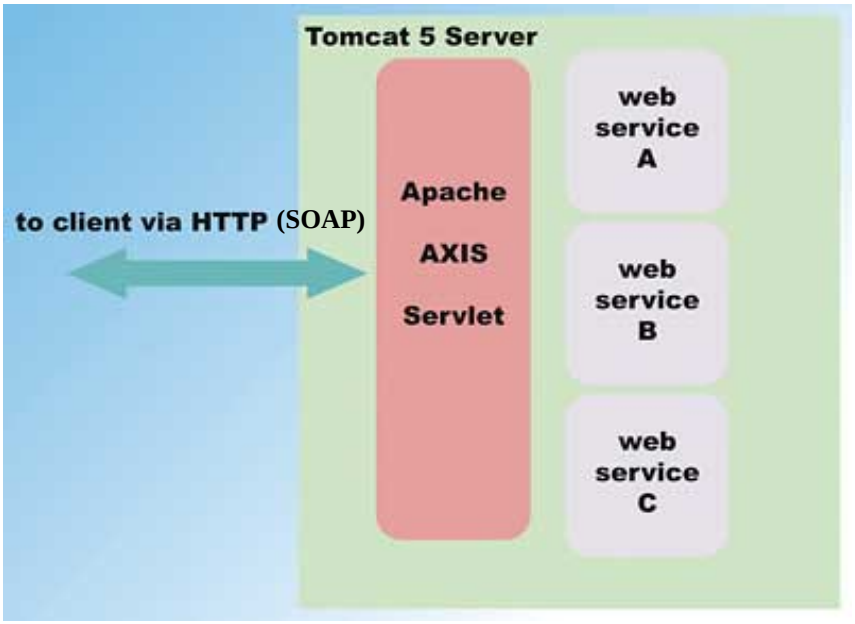

Figure 24: Tomcat/Axis web-service environment [Apache]

Apache Tomcat is a servlet container that includes build-in support for Java servlets. JSP might be transformed within Tomcat to the servlets and therefore hosted in the Tomcat servlet container too. Summarized, Apache Tomcat is responsible for hosting of web applications and indirectly of web-services, as well as their processing. [Apache] Web applications are hosted in CATALINA_HOME/webapps directory, web-services, in turn, are in CATALINA_HOME/webapps/axis directory.

Axis is, basically, a SOAP engine that is responsible for handling and routing of SOAP messages (requests and responses). For that purpose Axis needs to understand various transport protocols and standards. [Apache] The most important features and functionalities of Axis will be described in the following subtopic in detail.

4.2.1 Apache Axis

Axis is essentially a soap engine and can be used for constructing clients, servers and gateways [Axis].

Beside these features Axis also includes:

- a simple stand-alone server
- a server which plugs into servlet engines such as Tomcat
- extensive support for the *Web-Service Description Language (WSDL),* generates WSDL code on the basis of existing java classes
- emitter tooling that generates Java classes from WSDL.
- some sample programs
- a tool for monitoring TCP/IP packets
- various transport protocol support (HTTP, FTP, Mail)
- components enabling enterprise java bean access [Axis]

Axis application server is generally responsible for web-service deployment and hosting, as well as soap messages handling, but also provides various flexibly extensible components (transport listener, router, serializer/deserializer, dispatcher, and handler objects) included in the so-called message processing node. [Axis]

WSDD (web-service deployment descriptor) allows defining and adapting the chain sequence of components installed in Axis message processing node to the users' needs. [Axis]

A message flow could look at runtime as follows:

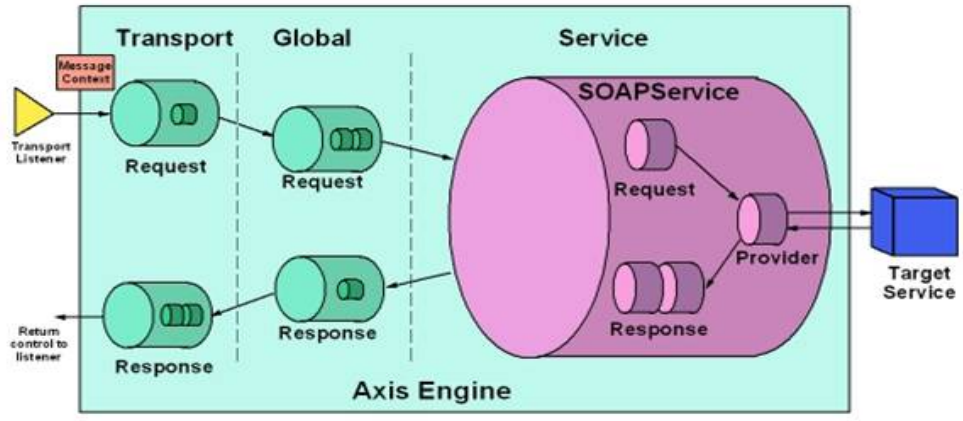

Figure 25: Axis server message path [Axis]

Transport listener receives incoming messages in a specific transport format and converts them into the format appropriate for further processing. The message flows through several chains of handlers that potentially might alter the original message construction. It has to be mentioned that those handlers might be customized and extended to the users' needs in an arbitrary way. Own handlers allowing arbitrary soap message manipulation might be included too. In this way Axis handles SOAP requests and routes them to the necessary web-service for further proceedings. Afterwards web-service is going to be invoked by a dispatcher to perform the request and to return the computed result. This result will finally be converted into soap response by Axis message processing node and sent to the service requester in a certainly requested format. [Axis]

4.2.2 Java web-service for ontology integration

As it has already been mentioned, eclipse IDE (interactive development environment) will be used to implement required java classes and methods. Afterwards Axis has to be introduced to transform the implemented java classes to our java web-services. The Axis engine will automatically create WSDL files, too. Finally, those services have to be deployed by the Axis application server for further use.

Java web-services are necessary for integrating ontology components in our timecard application in order to create extended reports, providing additional implicit information for the management and other participating roles. Thus, java web-services will be integrated into our timecard application. Adobe Flex 2.0 IDE will be used for that purpose.

Firstly, it has to be mentioned that certain java virtual machine parameters have to be set if a proxy server is used to access the internet. This action is absolutely necessary if the ontology is stored on the web and not on a local machine. Precisely, proxy host name and proxy port have to be passed as arguments to the java virtual machine to enable access to the required ontology published on the web. If all the settings are carried out successfully, Protégé API providing several ontology querying and modification functions can be invoked for our purposes. It's very important to understand how Protégé manages all the ontology components before starting the actual web-service implementation.

Protégé is tightly integrated with the Jena tool. Jena is responsible for ontology storage and querying. All the classes and concepts created in Protégé OWL are also

represented in Jena model as java objects. This Jena model is integrated in Protégé OWL and is always running in parallel. Protégé API actually allows accessing the Jena model and therefore the whole ontology represented in java objects. [KFNM04]

Implemented java web-service OntologyWS.jws contains several methods providing general information about our ontology such as its properties' number, individuals' number, classes' number, RDF resources' number etc. There are also some methods returning superclasses as well as subclasses or individuals of a certain selected class. Other methods provide information about asserted or inferred resources, their interrelations and dependencies on each other, too.

Those methods will be listed and their functionality shortly explained below:

```
getAllResources():Array – returns all RDF resources
specified in the ontology knowledge base as array

getAllProperties():Array – returns all RDF properties
specified in the ontology knowledge base as array

getAllInidividuals():Array – returns all individuals
(instances) specified in the ontology as array

getAllClasses():Array – returns all defined classes in
the ontology knowledge base as array

getDirectSublcasses(className:String):Array – returns
only direct subclasses of certain class as array

gedtAllSubClasses (className:String):Array – returns
all the subclasses of certain class (also the
subclasses of the subclasses, processing till nil)

getDirectSuperClasses (className:String):Array –
returns direct superclasses of certain class as array

getRangeOfProperty (propertyName:String):Array –
returns classes, to which certain property is allowed
to be connected
```

```
getDomainsOfProperty(propertyName:String):Array –
returns classes, from which certain property is allowed
to be defined

getDirectRelationsOfClass(class:String,property:String)
:Array – returns classes, to which certain class is
related through certainly specified property

getAllRelationsOfClass(class:String,property:String)
:Array – returns classes and all their subclasses, to
which certain class is related through certain
property. This function might be important to discover
if two classes are interrelated somehow through
certainly specified property.

getDirectInferredSubclasses(className:String):Array –
returns subclasses of certain class that were inferred
by Racer reasoner server (not asserted explicitly)

getAllInferredSubclasses (className:String):Array –
returns all subclasses of certain class that were
inferred by Racer reasoner application (also the
subclasses of the subclasses, processing till nil)

getDirectInferredSuperClasses(className:String):Array –
returns superclasses of certain class that were
inferred by Racer reasoner as array
```

To reduce the complexity of ontology processing, only basic ontology query functions were implemented in our web-service. All the modifications and ontology updates can be performed within Protégé OWL editor, if required. It has to be mentioned that just several web-service functions will be used in our timecard application. Other methods might be included for various future timecard extension reasons or used by other applications.

4.3 Web-service based back-end application logic integration

The semantic timecard tool is a rich data-driven internet application. All the timecards and their detailed information, as well as personalized user reports, will be generated on request dynamically. For this purpose the back-end logic has to be implemented to coordinate necessary database transactions, such as timecard storage and querying of necessary data. That back-end application logic will be provided by web-services to ensure interoperability and easy extensibility of our timecard functionalities. Those services might also be easily reused in other applications for various purposes.

In the current chapter it is explained in general how the web-services providing back-end application logic are actually implemented, and how they interact with the database used for the timecard objects storage.

The architecture and facilities of the Coldfusion application server used for the web-service implementation and publication will be accurately explained in this chapter, too. The proceedings of Allaire Spectra framework used to map timecard objects to the database tables, in order to facilitate their storage and maintenance dramatically, will be presented as well. Finally, the actual web-service methods and their functionalities are going to be explained in a detailed way.

4.3.1 Coldfusion application server

The Coldfusion MX application server is based on the powerful J2EE technology platform and is therefore tightly integrated with the Java technology components. The Coldfusion application server can run as standalone solution or be easily integrated into already existing Java application servers such as Bea Weblogic, IBW Webshpere etc. [Adobe]

The abstract Coldfusion MX architecture is shown in the figure below:

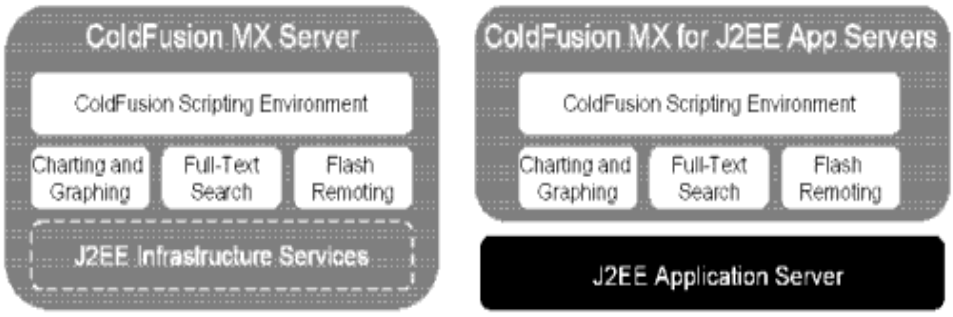

Figure 26: Coldfusion application server architecture [Adobe]

This architecture presents two Coldfusion MX editions (standalone and integrated) providing the same functionalities. For our purposes a standalone Coldfusion application server will be used and then described. The Coldfusion application server includes the entire infrastructure necessary to run the Coldfusion applications, but also embedded Java server based on Jrun technology. Due to the fact that Coldfusion is implemented on top of J2EE platform, many J2EE services are used to perform complex tasks such as database connectivity, naming and directory services and other important runtime services in the scalable way. [Adobe]

The Coldfusion applications consist of simple CFML mark-up language and object-oriented CFC components (used to create classes) that might be integrated into the CFML pages as objects. At the runtime, the CFC components are finally transformed to the JAVA objects for further processing. This fact allows integrating other JAVA objects into the Coldfusion applications in arbitrary way. [Adobe] To be more precise, the following JAVA components might be included in Coldfusion applications for further processing:

- Custom JSP (java server pages) tags might be used from the JSP libraries
- JSP pages might be included in Coldfusion applications
- Java servlets might be used
- Custom and defined java objects including J2EE Java API, JavaBeans and Enterprise JavaBeans might be used and integrated [Adobe]

Summarizing, Coldfusion application server takes many advantages from a scalable and powerful J2EE platform and hides its complexity through simple and easy-to-use scripting environment. If required, extended functionality might be

integrated through external objects (Java objects, COM objects, CORBA objects). [Adobe]

Additionally, the Coldfusion application server is capable of interacting with a number of external data sources, including databases (via JDBC and ODBC), Verity collections, LDAP directories, POP3 and SMTP mail servers, SOAP-based web-services, FTP servers, and other HTTP servers. [Adobe]

The Coldfusion application server also supports Flash remoting that might be rather relevant for our data-driven timecard application. Flash client consuming Coldfusion services might integrate remote Flash objects directly. That proceeding would increase transfer speed noticeably compared to web-service integration. In our timecard application a lot of data will be sent for personalized reports and timecard overviews. The remote objects are sent in a binary format, which is rather chunkier than bloated XML format used by SOAP based messaging. However, our back-end functionality will be implemented as web-service to provide it for other applications, too. But our timecard application client will invoke that service as remote object to speed up the data transfer and, therefore, considerably improve the performance of our timecard application.

The Allaire Spectra framework will be used to facilitate the database storage and maintenance. Operating mode of that framework will be shortly presented in the following subchapter.

4.3.2 Allaire Spectra framework

Allaire Spectra is an open source framework incredibly facilitating interactions with the databases and their management. Usually it is rather time-consuming to build an appropriate database design for a certain application. The functionalities and expectations of the applications change frequently. Thus, the database necessarily needs to be maintained and adapted, too. This fact generates high additional costs that are not accepted by the customers and developers as well. First of all, for simple data-driven and non-sophisticated web applications, frameworks, such as Allaire Spectra dramatically facilitating database storage and maintenance, could be very useful. To be more precise, Spectra generates three database tables handling all the application objects. The first table defines and stores all the user objects' types. It is defined, which structure the objects actually have (e. g. timecard type can be specified). The actual timecard instances (objects) are stored in the second database table. These objects might be accessed by their ID for the further processing. All the simple and

complex properties of those objects are stored in the third database table in turn. [Adobe]

In short, you do not need to care about database transactions. They are all handled by Allaire Spectra framework. The developer can define his objects with their customized properties in a specified format (e. g. timecard object with start date, end date, duration, technologies, work type, user properties etc.). The timecard objects will be stored in a specified format and are able to be queried by the Allaire Spectra in an arbitrary way. The querying results may be integrated into various applications. In our case, the results will be sent to Coldfusion web-services and afterwards integrated into our Flash client that finally aggregates and presents all the data in a convenient way.

The complete semantic timecard architecture will be presented and explained in the chapter 5 “Semantic timecard application”.

4.3.3 Coldfusion web-service for back-end logic integration

In the current subchapter the actual Coldfusion back-end service functionalities and methods will be explained in detail. This web-service provides basic timecard storage and update functionalities. Several querying functions are provided to extract the necessary timecard information for personalized user reports, too.

The back-end application logic web-service actually consists of the following methods:

```
getCard(id:String):Object – returns a specific timecard
based on its ID as object

getCardsMulti(year:Number, month:Number, day:Number,
username:String):Object – the arguments year, month,
day and username are optional. If these arguments are
specified, timecards for specific period of time and
for certain user will be returned. If these arguments
are empty the timecards of all users for the whole
period of time will be returned for further processing.

getWorkTypes():Object – returns all defined work types.
This function might be useful to fill combo box where
certain work types might be selected afterwards.
```

getTechnologies():Object – returns all the defined technologies as object. Ideally, it should be assigned, which technologies are used in which projects in order to facilitate technology selection process.

getProjectTasks():Array – returns all the tasks assigned for the currently running projects. Many project objects will be returned. Each project has property tasks, where all tasks assigned to that project are listed as array.

addTimecard(stData:object):Object – adds timecard filled in by the project member. If some exceptions occur while this proceeding, notification will be returned as an object.

updateTimecard(stData:object):Object – updates the data of a certain timecard. Errors will be returned as object.

deleteTimecard(objectID:String):Object – deletes a certain timecard based on its ID

getRepWorkType(year:Number, month:Number, day:Number, username:String):Object – returns individual work type report for a specific period of time (which services were performed by certain project member in certain period of time)

getRepTechnology(year:Number, month:Number, day:Number, username:String):Object – returns individual technology report for a specific period of time (which technologies were used by certain project member)

getRepProject(year:Number, month:Number, day:Number, username:String):Object – returns list of projects, in

```
which a certain employee was participating in a certain
period of time

getRepProjectTask(year:Number, month:Number,
day:Number, username:String):Object – returns tasks
that were performed by certain user in his projects
while certain period of time

validateUser(username:String, password:String):Object –
performs user authentication. The authentication result
is returned as an object.

getTechnoStats(username:String):Object – returns
comprehensive technology statistics for a specific
user. On the basis of this method it is possible to
visualize which technologies become more or less
important for certain employee and in which way
individual technology usage changed over the whole
career.
```

Functions getWorkTypes, getTechnologies and getProjectTasks are used to fill in the combo boxes. Ideally, the work types, used technologies and tasks need to be assigned to a certain project and to a certain employee in the project planning phase already. All that data asserted in those project planning tools just could be imported and reused in our timecard application. Thus, the project members might select only to the projects assigned tasks, technologies and service types. That proceeding could considerably facilitate the completion of the timecard form and increase the quality of extended personalized reports because of the performed terms standardization. If specific project is selected by a certain user, all the above-mentioned combo boxes have to be filled with the already to this project and to this user assigned data. The completion of the timecard can be performed by the user very simply in that way. The ontology can also expect entered information and perform the technology and service type matches accurately.

5 Semantic timecard application

The prototype of semantic timecard is a rich web-service based internet application. Flash client (implemented in Adobe Flex 2.0) uses several services to integrate the back-end application logic and ontology components enabling extended portfolio reports. All the integrated data will be aggregated and represented by Flash client in an appropriate way.

The semantic timecard prototype is primarily implemented to demonstrate the integration of our ontology knowledge base. It has to be shown, how developed ontology could provide extended technology and services portfolio reports and add semantic to our timecard application. The basic functionalities like timecard creation, modification, storage and querying will be provided by our timecard application, too. Personalized user and portfolio reports will be analyzed and created on the basis of the user comprehensive timecard data and ontology components. All the reports will be visualized by well-arranged diagrams.

If the services performed by employees would be quantified in terms of money, project billing and invoicing might be supported by our timecard application in an effective way, too. However, the information on charging for the performed services is not available. Thus, this functionality will not be considered furthermore.

It would be also possible to inspect the progress of currently running projects on the basis of the timecard data entered. For this purpose, MS project server data has to be integrated into our timecard application in order to perform the target/actual comparison. Due to the high complexity and missing project planning data, this functionality will not be covered by our timecard prototype, too.

The whole functionality, as well as the roles participating in our semantic timecard application, will be presented by the use-case diagram below and described in detail afterwards.

Figure 27: Semantic timecard use-case diagram

The project staff should have the ability to edit their timecards. In particular, they should be able to enter new timecards, update the already inserted timecards or delete incorrectly acquired timecards. They also should be able to present their timecards for a specific time segment in order to overview their activities for this period of time. This function could be helpful to prove faithful tasks completion in disputes with the customer or supervisors over the unexpected project delays and other irregularities, too.

Project staff is also interested in individual graphical statistics to get information on the following questions:

- In what projects did I participate within a certain period of time?
- Which projects did thereby represent my core activities?
- Which kind of tasks did I perform in the projects mentioned, how long and at which ratio?
- Which technologies did I use to handle certain tasks in specific projects during a specified period of time?
- Which service categories (consulting, software engineering, controlling etc.) did I perform to accomplish my tasks, how many hours, and at which ratio?

All the answers to the questions presented above will be generated and accurately visualized by our timecard application.

Management and project management office, but possibly also project staff might be interested in the technology and service portfolio reports. These reports could provide information on:

- Which technologies are dominating in the currently running projects and are becoming more and more important;
- Which services and at which ratio the business actually offers (e. g. consulting, design, programming etc.);
- How technology and service trends change over time.

It might be interesting for management and for project management office to have an overview over all the currently running projects within a specific department or even the whole organisation, too.

As it has been mentioned above, the project progress/status reports as well as project billing and invoicing reports will not be covered by our timecard application, in order to reduce the complexity of the timecard tool prototype. The information such as project planning data and charging structures of Siemens Austria AG is not

accessible for this diploma thesis, too. Due to these facts, it was considered to exclude the functionality highlighted red in our use-case diagram.

It is very important to show which technologies are involved in the timecard implementation process and how they interact with each other. The technologies used to implement the semantic timecard prototype were partly mentioned and described in chapter 4 "Web-service based component integration". In the following subchapter the holistic timecard architecture will be presented to show how all those technologies are actually interconnected and dependent on each other.

5.1 Semantic timecard architecture

This subchapter will present the holistic architecture of the semantic timecard prototype, where all the technologies used to implement this tool will be mentioned and provided with an explanation. It naturally has to be mentioned for what purpose each technology is actually responsible, too.

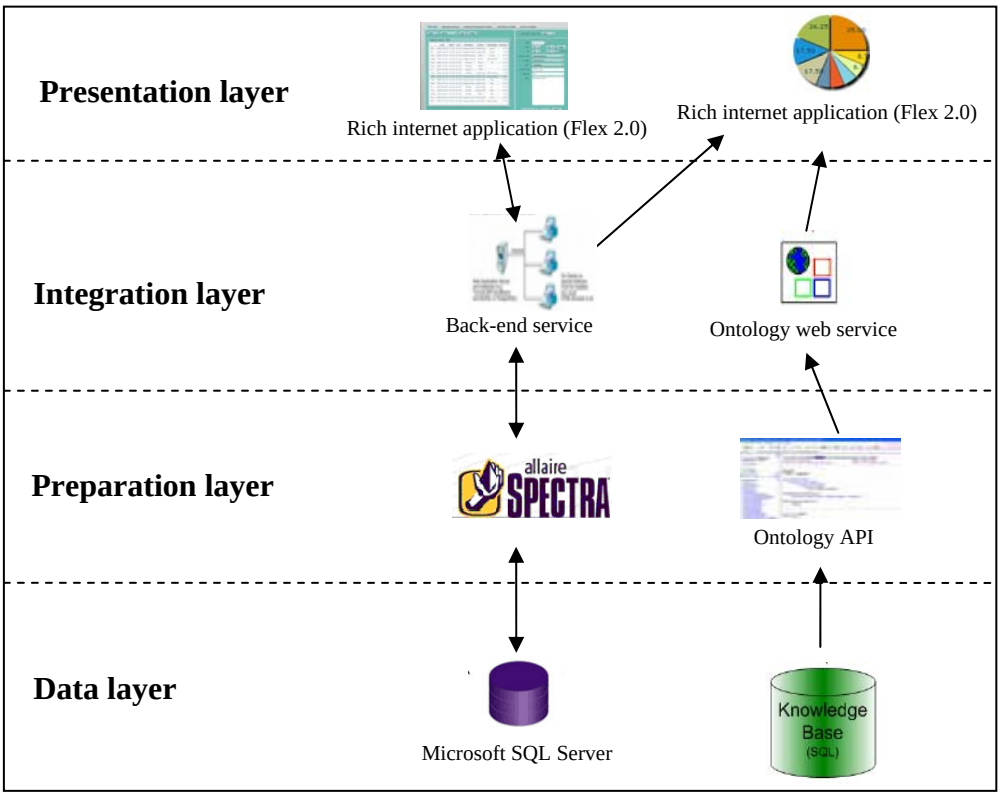

Figure 28: Semantic timecard architecture

As visualized in the figure above, data layer is responsible for the storage of the timecard objects and ontology concepts. The Microsoft SQL Server is thereby used to enable persistent storage of complex timecard objects. Protégé OWL plug-in is in turn responsible for the accurate storage and management of numerous knowledge base components.

The Allaire Spectra framework is responsible for the object mapping and database maintenance. All new properties or components attached to our timecard application will be managed by the Allaire Spectra framework. Allaire Spectra is responsible for appropriate storage and querying of those components, too. Ontology API provided by the Jena model provides comprehensive ontology querying facilities. That API will be used by the ontology java web-service in order to perform the ontology component

integration. Coldfusion web-service, as mentioned in chapter 4, was implemented to integrate complete back-end application logic, such as timecard modification, storage, querying, report data delivery etc.

All the data delivered by web-services is processed by Flash client and presented by means of tables and convenient graphs. The Flash client is implemented using Adobe Flex 2.0 IDE (integrated development environment). Flex 2.0 consists of certain already implemented components and scripting environment (actionscript). [Adobe] Flex components might be used to determine application design and user interface. Actionscript might be introduced to generate application content dynamically and to aggregate, as well as to process, data units delivered by the external web-services.

5.2 Semantic timecard functionality

The timecard application should provide the possibility to archive employees' activities within an organization. Through this timecard tool the employees are able to track their career development and observe numerous individual statistics concerning the services performed by them, the technologies used and project participations, as well as their functional responsibilities. On the basis of the timecard data and ontology concepts, management can observe project comprehensive technology and service portfolios for given periods of time. In that way, it is also possible to follow technology developments and their changes over time.

This semantic timecard prototype will just show some samples of how service or technology portfolios could look like. The timecard's data can provide more information and extend portfolio reports if required, too.

The complete timecard's functionality will be described in this chapter in detail. Some screenshots will be shown to facilitate functionality description process and to give an idea of how this rich internet application actually looks like. In the end of this chapter some problems and considerations, concerning timecard form completion process and the corresponding portfolio report quality, will be addressed and discussed, too.

The semantic timecard prototype basically consists of five tabs. The first tab provides basic timecard functionalities such as personal timecards listing, timecard deletion, saving and editing. The second and the third tabs provide numerous individual statistics for specific time segments. The forth and the fifth tabs generate

possible technology and service portfolio reports. It has to be mentioned that the portfolio reports might be expanded in order to provide additional portfolio information, if necessary.

The interface of the first tab is looking as follows:

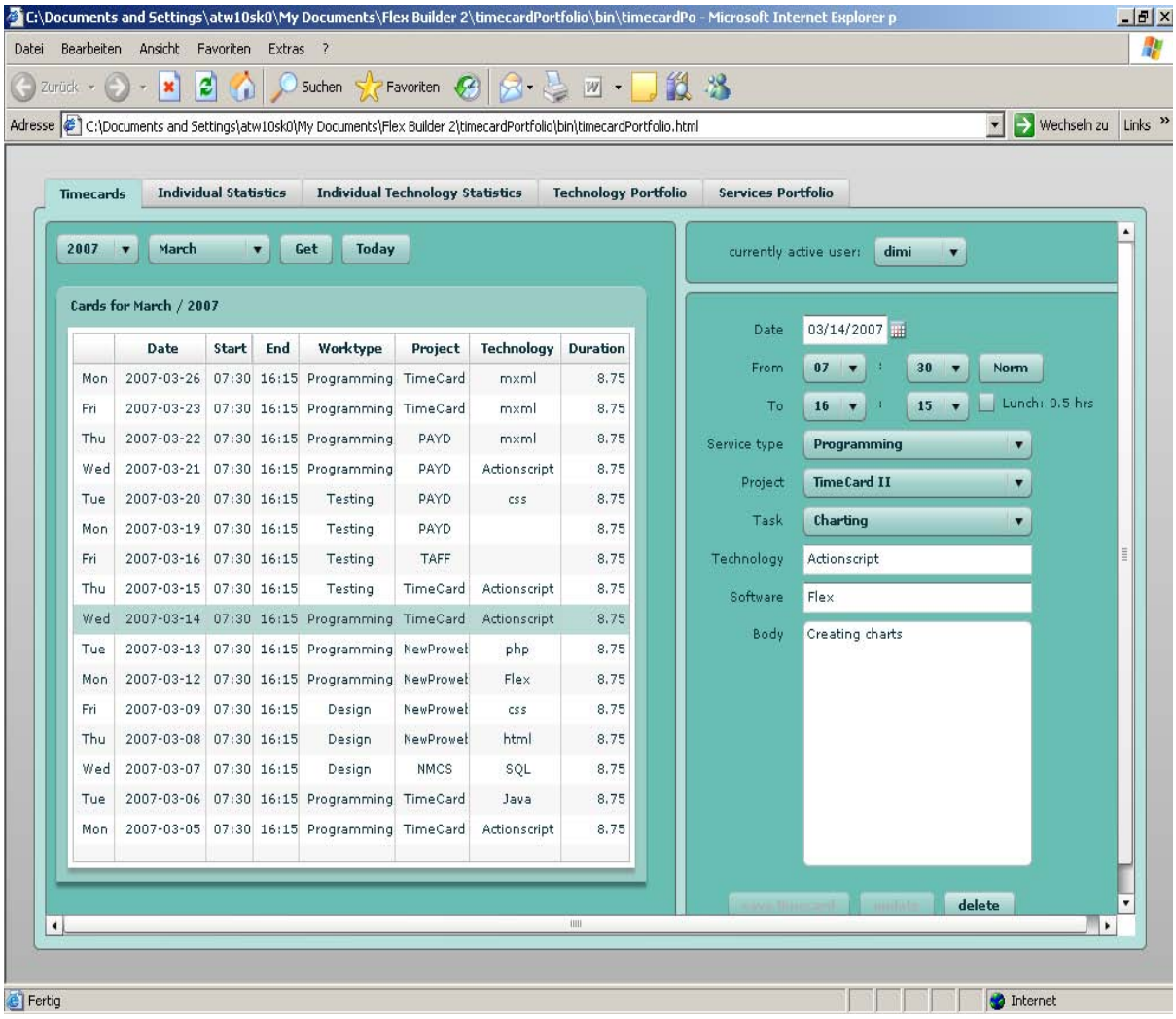

Figure 29: Semantic timecard form

After a certain user has passed the authentication process successfully, he becomes able to load his personal timecards for specific month or just for today. All the timecards are listed in the table and sorted by date. It is possible to sort the timecards by duration, work type, project and technology, too. The user can arrange his interface as he likes. Ideally, if certain project is selected in the combo box, the combo boxes task and service type should be filled with the specified data assigned to the selected project automatically. In our case only specific tasks are defined for certain project and will therefore be set if a certain project is selected in an appropriate way. It means that the user can only select the tasks assigned to his project. That proceeding enables some kind of standardized form completion. Standardized term definitions increase the accuracy and quality of reports.

A user can fill in a form. This form completion process has to be accomplished as fast as possible. For that reason some automatic settings were integrated (current date, usual work time) and can be chosen, if required. Several irregularities, such as incorrect start/end date or time inputs, are thereby intercepted by appropriate exceptions. As long as one form field is filled in or changed, the save or update button will be activated. The timecard saving or updating might therefore be performed. The timecard list and all the statistical graphs will be updated immediately. Thus, the timecard changes have an immediate impact on the whole timecard application. The timecard application must not even be reloaded manually.

The timecard data grid (table) doesn’t provide all the timecard information. Some irrelevant fields are hidden. To see the complete timecard information, specific data grid row needs to be clicked. All the form input fields will automatically be filled with the data of the selected timecard. Now, the selected timecard might be deleted, updated or just observed by the user.

In the figure above user timecards for March 2007 are shown. User naturally might change the date and look at his timecards for the past. All the individual and portfolio statistics will be adapted to the new date automatically.

The second tab “individual statistics" consists of several graphs showing certain user-specific statistics for a specified period of time:

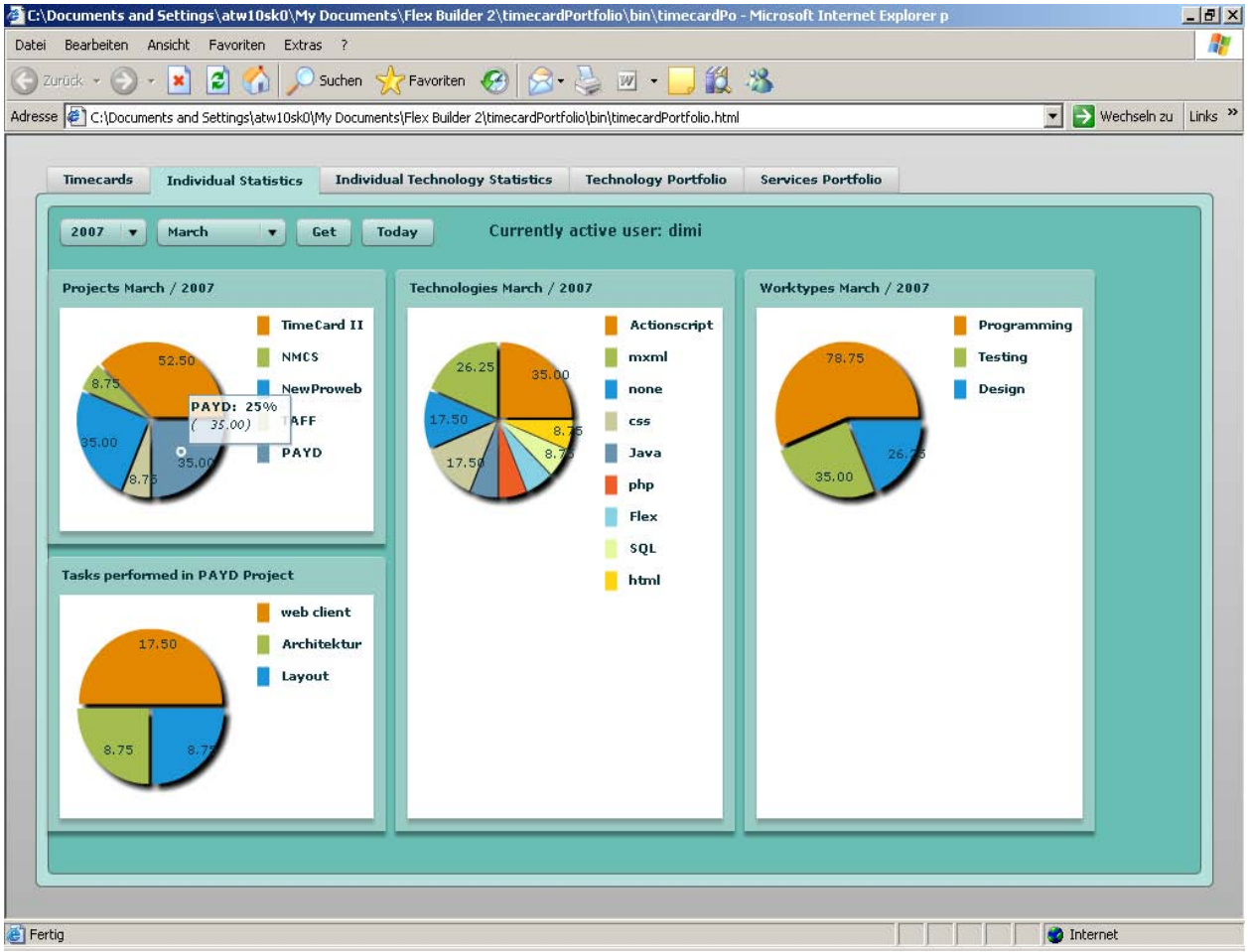

Figure 30: User-specific statistics

Individual statistics tab provides user-specific reports for a certain time segment. In this case March 2007 is selected. A user can change the date and observe reports for different time segments and their changes as well. User can extract the following information from the graphs presented above:

- How much did I work for a certain project in hours and in percent in March 2007?
- Which technologies did I use to accomplish project tasks and at which ratio?
- Which service types in hours and in percent did I perform in March 2007?
- Which tasks and at which ratio (in hours or in percent) did I accomplish within a certain project?

It also has to be mentioned that “Tasks performed” graph is generated dynamically. If user clicks a certain project within “Projects” graph, tasks data of the selected project will be extracted. The tasks performed by a certain user in the selected project will be calculated in order to fill in “Tasks performed” graph in an appropriate way. In that way, the user can see which tasks and for how long he actually performed within a certain project in the specified period of time.

The tab individual technology statistics provides a user-specific overview of how the technology usage developed and changed over his whole career. The end-user is therefore able to observe how his technology focuses and strengths actually changed over time. On the basis of that data, user's profiles concerning their technology specialisations could be created, too. An individual technology statistics graph could look as follows:

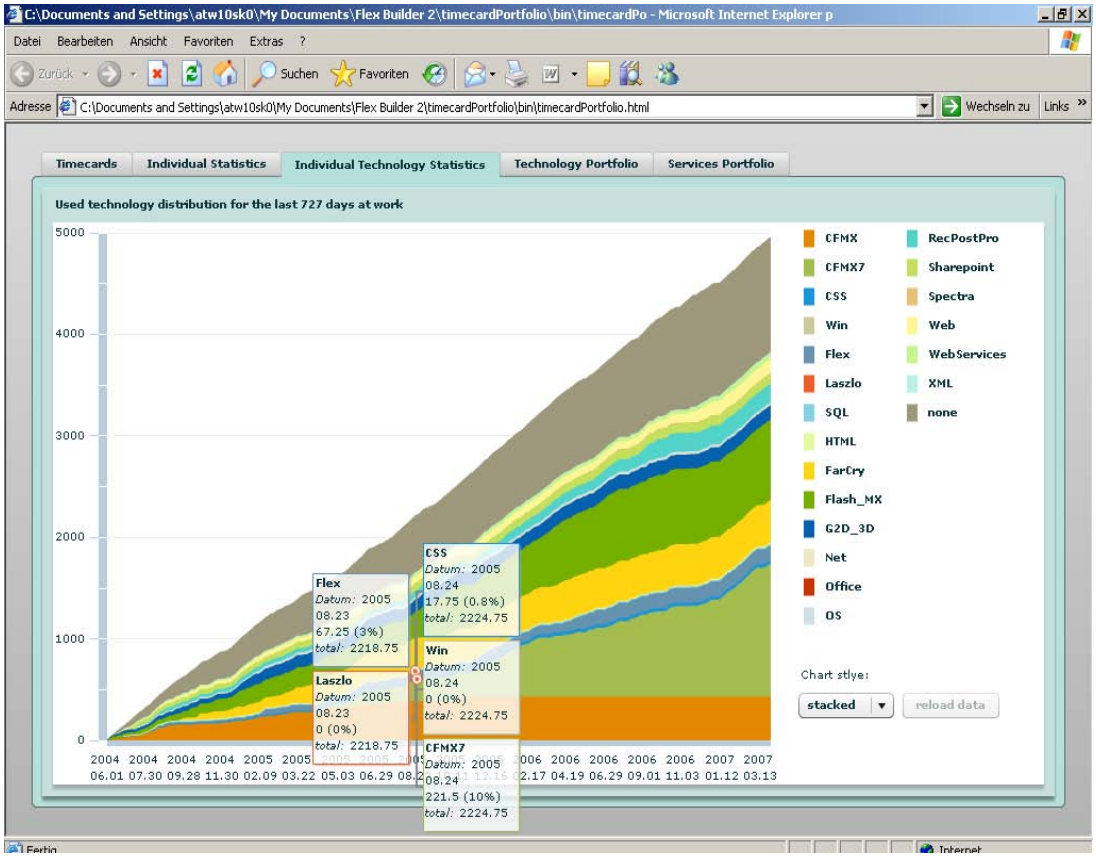

Figure 31: User-specific technology report

This graph shows how a certain employee's technology usage changed over the years of his employment. In this specific case, it is possible to observe that, for example, CFMX7 (Coldfusion MX7) was used for 220 hours (10 % of the whole work time for this period). Coldfusion MX in turn made up approx. over 400 hours (20 % of the whole work time). One year later the CFMX technology became less important for the tasks completion (10 % of the whole work time), while improved Coldfusion MX7 technology rapidly gained in importance and its deployment was therefore doubled. It is also clearly visible that Flash MX technology considerably gained in importance over years, too.

The technology and service portfolio tabs provide user comprehensive reports for a specific time segment. A lot of data sets need to be evaluated and compared to perform these portfolio reports. For performance reasons just monthly portfolio reports will be generated in our semantic timecard prototype.

For test purposes technology portfolio tab presents several figures showing the programming languages report, java technology vs. .NET technology report and software report (commercial software vs. open source software).

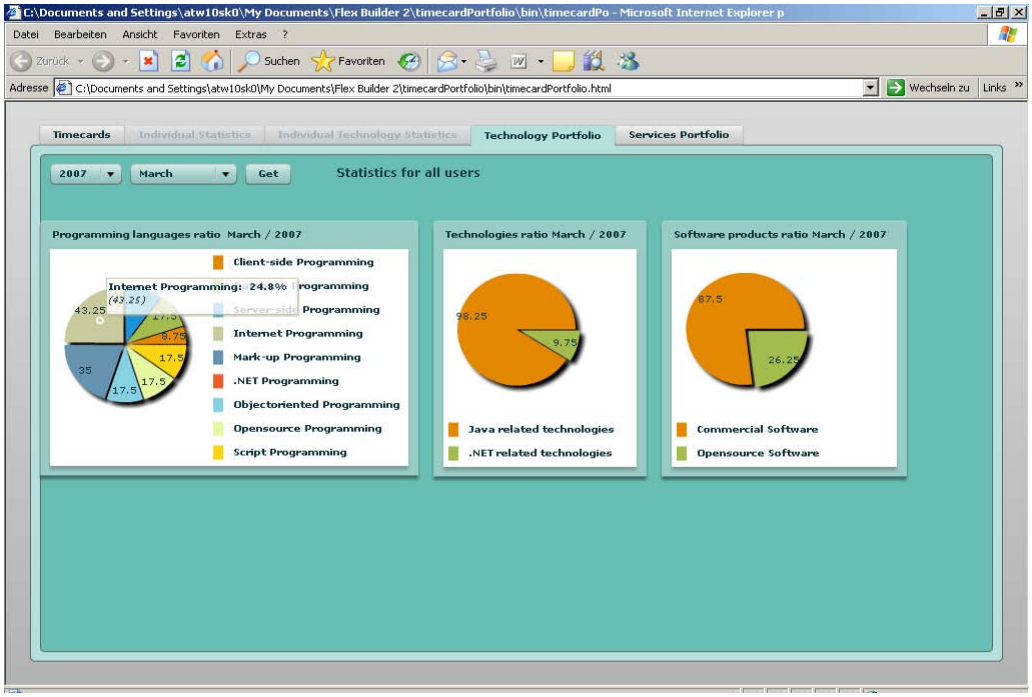

Figure 32: Technology portfolio reports

The programming language categories defined in our ontology knowledge base will be loaded by the ontology web-service. The programming languages asserted in the ontology will be compared with the technologies entered in the timecard application afterwards. That comparison procedure is based on simple match operations. Those match proceedings might cause several inaccuracies. For example, Javascript programming language was entered in the timecard application. The ontology might interpret this technology correctly and assign it to the correct programming language categories (to internet language, client-side language etc.). But Javascript would also match with Java programming language and therefore be assigned to wrong programming language categories, too. The portfolio reports just provide general information and analyse a huge amount of data. Thus, irrelevant deviations could be accepted. These and other portfolio report problems, as well as considerations, will be addressed and discussed in the chapter 5.3 “Portfolio inaccuracies and difficulties” in more detail.

The same procedure will be performed to generate technology and software comparison reports. The second figure shows the proportion of Java-related technologies to the .NET-related technologies that are used in the projects within a certain period of time.

The third figure visualizes which software type (commercial software vs. open source software) is dominating in the projects within an organization. It can be observed, how software proportions actually change over time, too.

It has to be mentioned that the technology portfolio reports might be expanded according to the management’s needs. If required, it also might be accurately visualized, which types of databases are introduced in the currently running projects, how intensive web-service enabling technologies are actually used etc.

The service portfolio report is generated in a similar way. The service portfolio tab is shown in the figure below:

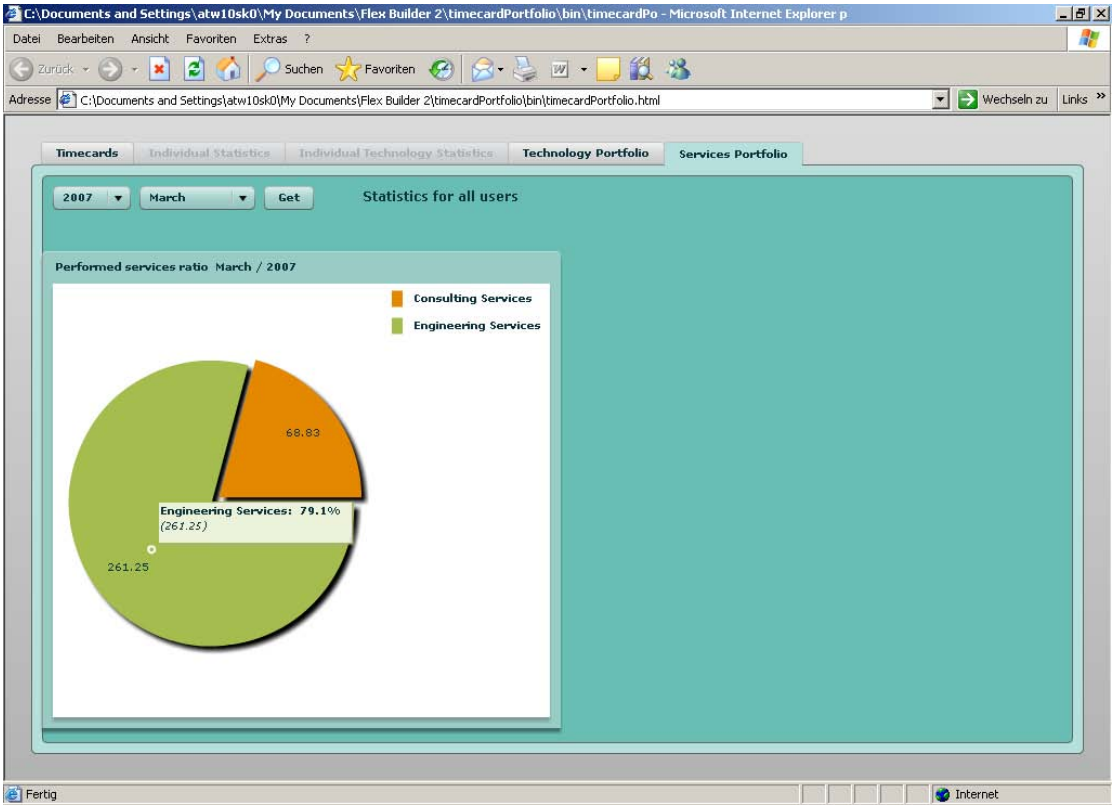

Figure 33: Service portfolio report

The above-presented figure shows the service proportions for test purposes. The management might observe, in which way the organizational service structures are changing and developing. On the basis of that service portfolio data, the changes over time and therefore future trends might be derived; systematic educational trainings can be identified and initiated in time, too.

If the services performed within an organization are quantified in terms of money, other portfolio reports regarding the service value and responsibility level could be generated and analysed.

It has to be mentioned that the ontology concepts were introduced to generate this service report, too. The ontology knowledge base asserts which services actually might belong to the engineering and to the consulting service concepts. Those services were matched with the services entered in the timecards. Services that

successfully matched with the ontology concepts were added to the appropriate service category and finally shown in the service portfolio report.

The main focus of the timecard design should be usability and intuitive user interface. Some functions were integrated into the timecard prototype to speed up the form completion process. The actual date and standard work times may be selected immediately. The timecard form provides intuitive and well-described input fields. The number of those fields was reduced to the minimum to speed up the timecard's completion process and therefore to increase user satisfaction and acceptance.

All the individual and portfolio reports are updated automatically, if some new information is entered into the timecard application. If a user selects a date for a specific individual report, all the reports (individual and portfolio reports) will be updated to the chosen date (data updates only, flash movie will be loaded once). The report switches might be performed quickly and without any difficulty. The complete flash movie is only loaded once. Thus, the whole application content may be immediately accessed without additional site downloads.

5.3 Portfolio inaccuracies and difficulties

There are two main topics that need to be discussed in this subchapter: the timecard form completion process and the technology as well as service matches for our portfolio reports.

Firstly, it has to be considered how the timecard form completion process could be optimized in the best way. On the one hand, the completion process should be fast and uncomplicated; otherwise employees will not accept the timecard application. On the other hand, the technologies entered in the timecards should be precisely defined to enable qualitative and accurate portfolio reports. It is also necessary to consider how the completion process could be standardized to allow only defined by group and expected technology entries and to avoid uncontrolled technology definitions.

The following technology portfolio evaluation problems might occur:

- Inaccurate interpretation of the technology capabilities – javascript might be used to perform the client-side script programming, but also the server-side script programming in some cases. This technology may be used in certain projects for the client-side programming reasons only. The ontology is not able to derive this information. It can see javascript as server-side or as client-side programming language. Additional fields might be added to the timecard form to specify the used

technology more precisely. However, that proceeding would cause more complexity and confusion in the timecard completion process, because many employees are not willing to fill in many form fields and even sometimes don't know all the technology finesses. Thus, they often are not able to define precisely enough which type of technology (server, client etc.) they actually used. Wrong form completions would cause portfolio inaccuracies, too. Due to these conditions, it was decided to create two form fields: technology and software product. Software product is an optional field and might be filled in, if possible. The technology used will not be specified precisely. Portfolio just provides abstract reports, and a lot of user-comprehensive data is evaluated (partial mistakes or single inaccuracies may be accepted). The employees are often inaccurate concerning their tasks and durations, too. Thus, minimal inaccuracies will always be given and have to be accepted.

- Incorrect technology matches – as mentioned in the previous chapter, the incorrect technology matches might occur as well. For example, the database Microsoft SQL Server would be handled not only as a commercial database, but also as a SQL database querying language, while ontology concepts match. Thus, Microsoft SQL Server would be added to the commercial software, commercial database software, Microsoft software, but also to database programming language categories. In fact, Microsoft SQL Server does not belong to the database programming language category. It was decided to perform simple matches (not exact matches such as Microsoft SQL Server = Microsoft SQL Server) because of the missing standardization of technology terms. The naming of IT technologies might differ among employees. In order to achieve the highest match rate, simple match operations are to be performed.
- Unexpected technology and software entries – the timecard entries might be unpredictable for the ontology knowledge if the timecard form completion process is not predefined in an adequate way. To achieve an appropriate level of the standardization certain technology and software presets should be defined and provided while timecard forms completion process. Some possible procedures concerning this issue will be discussed in the chapter 5.4 "Semantic timecard enhancement capabilities" in a more detailed way.

It has to be mentioned that IT portfolios generally provide abstract information for the stakeholders concerning certain technology and service trends. A large amount of data is analysed and evaluated for that purpose. In the most cases minimal

inaccuracies are irrelevant and should not be considered furthermore. However, noticeable portfolio deviations should be taken into consideration and further applicable actions need to be attempted.

5.4 Semantic timecard enhancement capabilities

As it was already mentioned in the previous sub chapter certain portfolio inaccuracies might occur because of possible unexpected timecard data entries and, consequently, falsified ontology matches. The functionality of the semantic timecard application might be extended to increase the quality of the portfolio reports and to support the ontology maintenance process. Following features might be included in the semantic timecard prototype:

- Standardization of the timecard completion process - the standardization of IT technology terms within an organization is a very important step towards high quality and accurate individual, as well as portfolio, reports. Ideally, specific tasks, technologies, service types and software products need to be assigned to a specific project and to a certain employee performing certain tasks in his projects. All that information might already be defined in the project planning tool. The timecard application should be able to import and to process that data in the necessary way. The complex project structures and rather deep task hierarchies, that are changing frequently, make it rather difficult and time-consuming for the organizations to realize and to maintain that project planning environment. For many organizations accurate project management environment and its maintenance still represent a considerable or sometimes unsolvable challenge. However, form input presets would enable high quality reports and well-defined technology matches if technology terms entered in the timecard application are defined in the expected way.
- Analysing of unmatched technologies – it is also rather useful to analyse unmatched technologies and their importance (measured in hours). Important expressions or even new technologies might be discovered in this way and entered into the ontology knowledge base. That proceeding would therefore considerably contribute to the ontology maintenance process and ensure its quality sustainably.

If already mentioned in the several previous chapters the portfolio reports might be easily expanded according to the management’s needs, too.

6 Summary

In today's business structures projects have dependencies and impacts on each other, some projects are interrelated or redundant. It is not enough to control and manage single projects to achieve the maximum overall return. Thus, comprehensive project portfolio management must be additionally applied. Project portfolio management is a very important concept, first of all, for the organizations dealing with a large number of projects simultaneously. [PR04] A lot of tasks, like control and overview of all projects, comprehensive resource allocation and, consequently, effective and cost-cutting expertise and components reuse, ensuring business strategy alignment, reporting to business executives etc. can be accurately accomplished by the effective deployment of project portfolio management.

The main focus of the current thesis was to consider how IT project portfolio management could be effectively supported by software solutions. For that purpose a semantic timecard prototype and ontology knowledge base were designed. The semantic timecard prototype was implemented primarily to support technology and service portfolios. The ontology knowledge base was thereby built to define the standardized terms of the IT domain and their mutual relations, as well as interconnections. Furthermore a comprehensive ontology development and integration guideline was specified and explained in a detailed way. That guideline covers ontology design, building, inference, testing, storage and integration processes. The ontology components were integrated via web-services into the timecard prototype to provide extended portfolio reports considering technology structures and interrelations. On the basis of the report data provided, management becomes able to discover which technologies and services their organizations are specialised in, and how those trends and tendencies actually change over time. That information enables management to take customer orders, dealing exclusively with similar technologies to assure expertise and technology reuse. On the basis of the well-investigated trends, essential educational trainings might be performed in time, too.

While developing the semantic timecard application, it could be noticed that some difficulties and challenges are to be faced to provide high-quality and accurate portfolio reports. On the one hand, it is rather complicated to extract the knowledge of

experts and to define the terms of the IT domain accepted by all the experts within an organization. Various experts might have different views on technology facilities and capabilities. Thus, it is sometimes rather complex to classify a certain IT technology uniquely. Some compromises have to be accepted. On the other hand, it is rather difficult to force standardized timecard form completion in order to perform accurate ontology matches and, consequently, highly qualitative portfolio reports. To achieve comprehensive standardization of the timecard form completion, all tasks, resources, technologies and service types need to be assigned to the defined projects and project staff. For that purpose a project planning environment has to be established and frequently maintained. However, for many organizations that proceeding may represent a considerable or sometimes unsolvable challenge.

7 References

[AC04] Aitchison Jean, Clarke Stella Dextre: The Thsaurus: Historical Viewpoint, with a Look to the Future; Cataloging & Classification Quarterly vol. 37 issue (3-4), 2004, pp. 5-21

[ACFG01] Arpirez Juan Carlos, Corcho Oscar, Fernandez-Lopez Mariano, Gomez-Perez Asuncion: WebODE - a Scalable Workbench for Ontological Engineering, First International Conference on Knowledge Capture (KCAP01), Victoria, Canada, 2001

[AMR] AMR research is leading advisory firm focused on the intersection of business processes with supply chain and enterprise technologies, http://www.amrresearch.com, requested on 28.01.2007

[AUM04] Angele Jürgen, Ullrich Mike, Maier Andreas: Taxonomie, Thesaurus, Topic Map, Ontologie—ein Vergleich, Ontoprise GmbH, Whitepaper, 2004

[Adobe] Adobe Systems Incorporated, further information available on http://www.adobe.com, requested on 01.03.2007

[Apache] Apache Software Foundation: http://apache.org, requested on 01.03.2007

[Apollo] Ontology building editor developed by Knowledge Media Institute, The Open University, further information available on http://apollo.open.ac.uk/

[Axis] Apache Software Foundation: Apache Axis: http://apache.org/axis, requested on 01.03.2007

[B04] Backett Dave: RDF/XML syntax specification, W3C Recommendation 2004

[BG02] Brickley Dan, Guha Ramanathan: RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 2002

[BG04] Brickley Dan, Guha Ramanathan: RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, 2004

[CK83] Cleland David, King William: Project Management Handbook, Van Nostrand Reinhold Company Inc., New York, USA, 1983

[D98] Domingue John: Tadzebao and Webonto – Discussing, Browsing and Editing Ontologies on the Web, in Proceeding of Eleventh Knowledge Acquisition Workshop (KAW98), Banff, Canada, 1998

[D03] Datz Todd: Portfolio Management How to Do It Right, CIO Magazine, issue Mai. 1, 2003

[DJMZ05] Dostal Wolfgang, Jeckle Mario, Melzer Ingo, Zengler Barbara: Service-orientierte Architektur und Web Services, Elsevier GmbH, München, Deutschland, 1. Auflage, 2005

[G93] Gruber Thomas: A translation approach to portable ontology specifications, Knowledge Acquisition (pp. 199-220), Academic Press Ltd, London, UK, 1993

[G03] Gläßer Lothar: IT-Lösungen im E-Business, Publics Corporate Publishing, Erlangen, Deutschland, 2003

[GF02] Gomez Perez Asuncion, Fensel Dieter: Ontology-based information exchange for knowledge management and e-commerce (Deliverable 1.3: A survey on ontology tools), 2002, published within OntoWeb project founded by European Union, further information available on http://www.ontoweb.org/

[GFC03] Gomez-Perez Asuncion, Fernandez-Lopez Mariano, Corcho Oscar: Ontological Engineering (Advanced Information and Knowledge Processing), Springer Verlag, London, UK, 2003

[H02] Hiller Mark: Multiprojektmanagement: Konzept zur Gestaltung, Regelung und Visualisierung einer Projektlandschaft, FBK Produktionstechnische Berichte, Universität Kaiserslautern, Deutschland, 2002

[H04] Heflin Jeff: OWL Web Ontology Language (Use Cases and Requirements), W3C Recommendation, 2004

[HKRSW04] Horridge Matthew, Knublauch Holger, Rector Alan, Stevens Robert, Wroe Chris: A Practical Guide To Building OWL Ontologies Using The Protégé Owl Plug-in and CO-ODE Tools, The University Of Manchester, Stanford University, 1st Edition, 2004

[HL04] Hauser Tobias, Löwer Ulrich: Web Services- Die Standards, Galileo Press GmbH, Bonn, Deutschland, 1. Auflage, 2004

[HM03] Haarslev Volker, Moeller Ralf: Racer - A Core Inference Engine For The Semantic Web, 2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003), Sanibel Island, FL, 2003.

[K00] Kerber-Kunow Annette: Projektmanagement und Coaching, Hüthig GmbH Heidelberg, Germany, 2000

[K06] Kagl Alexander: Multiprojektmanagement, Wien, Östrereich, 2006

[KFNM04] Knublauch Holger, Fergerson Ray, Noy Natalya, Musen Mark: The Protègè OWL Plugin – An Open Development Environment for Semantic Web Applications, Third International Semantic Web Conference (ISWC 2004), Springer Berlin/Heidelberg, Germany, 2004

[L00] James Lewis: The Project Manager's Desk Reference: A Comprehensive Guide to Project Planning, Scheduling, Evaluation and Systems, McGraw-Hill Companies, New-York, USA, 2000

[LAS05] Liang Yaozhong, Alani Harith, Shadbolt Nigel: Ontology Change Management in Protégé, AKT DTA Colloquium, Milton Keynes, UK, 2005

[M93] Musen Marc: Dimensions of Knowledge Sharing and Reuse, Computers and Biometrical Research 25 (pp. 435-467), Academic Press Professional Inc, San Diego, USA, 1992

[M04] Maffeis Silvano: Einführung in Soap Web Services anhand von Apache Axis, Java Web Services conference initiated from Java User Group Switzerland, Zürich, Switzerland 2004

[MFRW00] McGuinness Deborah, Fikes Richard, Rice James, Wilder Steve: An Environment for Merging and Testing Large Ontologies, Principals of Knowledge Representation and Reasoning: Proceedings of 7th international conference (KR 2000), Morgan Kaufmann Publishers, San Francisco, CA, 2000

[MH04] McGuinness Deborah, Harmelen Frank: OWL Web Ontology Language (Overview), W3C Recommendation, 2004

[NM01] Noy Natalya, McGuinness Deborah: Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, California, USA, 2001

[ODP] Open Directory Project administrated by Netscape Communication Corporation, provides comprehensive open source web sites categorization, further information available on http://dmoz.org/

[OntoStudio] Comprehensive ontology building environment developed by Ontoprise GmbH, further information available on http://www.ontoprise.de/content/

[OWLViz] Ontology visualization plug-in for Protégé OWL editor, developed by Matthew Horridge, The University Of Manchester, further information available on: http://www.co-ode.org/downloads/owlviz/co-ode-index.php

[PR04] Patzak Gerold, Rattay Günter: Projektmanagement: Leitfaden zum Management von Projekten, Projektportfolios und projektorientierten Unternehmen, Linde Verlag Wien GmbH, Wien, Österreich, 2004

[Protégé] Protégé ontology editor developed by Stanford Medical Informatics, Stanford University School of Medicine, further information available on: http://protege.stanford.edu/

[SBF98] Studer Rudi, Benjamins Richard, Fensel Dieter: Knowledge Engineering: Principles and Methods, Elsevier Science Publishers B. V, Amsterdam, Netherlands, 1998, pp. 161-197

[V05] Vonhoegen Helmut: Einstieg in XML, Galileo Press GmbH, Bonn, Deutschland, 3. Auflage, 2005

[VK05] Vouros George, Kotis Konstantinos: Extending HCONE-Merge by Approximating the Intended Meaning of Ontology Concepts Iteratively, 2nd European Semantic Web Conference ESWC 2005 pp. 198-210, Springer Verlag, Berlin, Deutschland, 2005

[W04] Wiehler Gerhard: Mobility, Security und Web Services, Publics Corporate Publishing, Erlangen, Deutschland, 2004

[WebODE] Ontology building editor developed by Artificial Intelligence Department, Technical University of Madrid, further information available on http://webode.dia.fi.upm.es/WebODEWeb/index.html

8 Table of Figures

9 Index of Tables