Diplomarbeit

Ontology-based Architecture to support Software Process Improvement

Brandenburg University of Technology Cottbus
Faculty of Mathematics, Natural Science and Computing
Internet-Technologies

André Frenzel

Cottbus, 31.05.2005
Abstract

Software Process Improvement is the key factor for better software quality. Software development organizations might be interested in finding improvement areas in their current practices. By performing assessments the capability of organization’s processes is determined and analyzed in order to select several improvement steps. These improvement steps consist of requirements and criteria e.g. for good software engineering or management practices, however, defined in a general way. Professional guidance is then needed for implementing these improvements. By transferring knowledge about their implementation the need of external consulting could be diminished.

This thesis deals with the question how Software Process Improvement can be effectively supported by a software tool. A reasonable architecture is introduced that is based upon ontologies. In this thesis we investigate how Process Models used for Process Capability Determination, such as SPICE or CMMI, can be integrated into an ontology. We describe how languages intended for the Semantic Web contribute to achieve knowledge transition. By a new approach we show that knowledge transition can take place by restricting the range of possible knowledge items to allow "virtual knowledge linking". By another Semantic Web feature, we make comments on ontology elements that supply information to build a proper browsing structure through Process Models. Finally, we introduce a Model-View-Controller Architecture in which the thesis outcome in the form of a prototypical implementation has been set up.
Task

student name: André Frenzel

date of birth: 28.11.1977
matriculation number: 9801254
email: afrenzel@informatik.tu-cottbus.de
field of study: Computer Science

first assessor: Prof. Dr. Gerd Wagner,
Brandenburg University of Technology, Cottbus
second assessor: Prof. Dr. Hannu Jaakkola, Conceptual Analyst,
Tampere University of Technology, Pori
attending tutor: Timo Mäkinen, Senior Researcher, Software Engineering,
Tampere University of Technology, Pori

thesis begin: 01.12.2004
thesis end: 31.05.2005

Task description:
The thesis consists of an experiment of applying a generic software tool to a specific software development problem relating to Software Process Improvement. It also produces a software prototype with appropriate documentation. The tool in the application is to support Software Process Improvement work after assessment. It should give advices how to fulfill criteria defined by Process Reference Models and Process Assessment Models. The database of the system should include models used in Process Assessment and more practical models, that give guidance for Software Process Improvement implementation. These two types of models should be able to be linked together. The structure of the models differs from each other and that is why it is appropriate to have a system that is not fixed to the defined meta-data.
Contents

1 Introduction 1

1.1 Motivation 1

1.2 Requirements Analysis 2

1.3 Overview about the Remaining Chapters 3

2 Background 5

2.1 Definitions 5

2.2 SPICE 8

2.2.1 SPICE Process Assessment Model 8

2.3 CMMI Framework 10

2.3.1 CMMI Representation for Capability and Maturity 11

2.4 Data Model Structure of ISO/IEC 15504-5 and CMMI 13

2.4.1 SPICE Exemplar Assessment Model Structure 13

2.4.2 CMMI Structure 15

2.5 Knowledge about Software Process Improvement 17

2.6 Vision of the Semantic Web 20

2.6.1 Resource Description Framework 22

2.6.2 Ontologies 23

2.6.3 RDF Schema 24

2.6.4 Web Ontology Language 26
List of Figures

2.1 Context of Process Assessment ........................................... 6
2.2 Activities of SPICE Process Improvement Process .................. 7
2.3 Context of SPICE Process Assessment .................................. 9
2.4 Process Entities in ISO/IEC 12207 ....................................... 11
2.5 SPICE Process Assessment Model ...................................... 12
2.6 ISO/IEC 15504-5 conceptional data model ........................... 16
2.7 CMMI continuous representation ....................................... 17
2.8 CMMI conceptional data model .......................................... 18
2.9 Layers of Semantic Web pyramid ...................................... 21
2.10 Graphical representation of a RDF statement ....................... 22
2.11 Simple class hierarchy .................................................. 25
2.12 Synonymous classes created with a hasValue-Restriction ....... 27

3.1 Gnosis' search result ...................................................... 29
3.2 Gnosis' browsing applet .................................................. 30
3.3 Gnosis' knowledge item view ............................................ 31
3.4 Gnosis' maintenance page ............................................... 32
3.5 Gnosis' graphical view on relationships ............................... 33

4.1 Single data model mapping .............................................. 37
4.2 Multiple data model mapping ........................................... 39
4.3 Knowledge items and their context and type ......................... 43
4.4 Individuals are instance of superclasses .............................. 45
4.5 Direct knowledge linking ............................................... 45
4.6 Ontology class linking for virtual knowledge transition ........... 46
4.7 Providing "Instance-classes" to enable virtual knowledge mapping ........................................... 47
5.1 Web-tier controller ............................................. 53
5.2 Model2/MVC Architecture .................................... 54
5.3 Struts flow ....................................................... 54
5.4 Model data flow .................................................. 57
5.5 SPI Browser class containing the business logic .............. 58
5.6 Jena’s internal model structure in the SPI Browser .......... 58
5.7 SPI Browser levels for tree navigation ....................... 60
5.8 Protégé ontology editor ......................................... 61
5.9 Defining knowledge transition in Protégé .................... 63
5.10 Defining knowledge restricted classes in Protégé .......... 64
A.1 Process Model ontology for use of direct knowledge linking 68
A.2 Struts workflow for the SPI Browser ....................... 69
List of Tables

4.2 Concepts of ISO/IEC 15504-5 and their counterpart in CMMI . . . . . . . . . . 38

5.2 Required software packages for the runtime environment . . . . . . . . . . . . . 53
Listings

2.1 RDF statements transformed into XML ........................................ 23
2.2 Defining a superclass-subclass hierarchy ..................................... 24
2.3 Defining a property hierarchy ..................................................... 25
2.4 Defining the range and the domain of a property .......................... 25
4.1 Example of a defined class ....................................................... 47
4.2 Example of an union restriction class ......................................... 48
Acknowledgments

First I want to thank my attending tutor Timo Mäkinen at the Tampere University of Technology, Pori who has supported me with good insight and helpful comments on the topic. I also want to thank the chair of "Information Technology" that is controlled by Prof. Dr. Hannu Jaakkola for their financial project support and I would like to thank any person of the Tampere University of Technology, Pori, who has given me a great time during my stay in Pori.

My special acknowledgement goes to Prof. Dr. Bernhard Thalheim who made it possible to perform my internship at the Tampere University of Technology, Pori in which my research work passed into a student research project and finally into this thesis.

My personal thank are for my aunt Ilona and my brother Stefan who were so kind to cross-read my thesis and to point out mistakes in spelling and grammar. Last but not least I want to thank my parents for their emotional support during the last months.
Chapter 1

Introduction

The Quality of software products largely depends on the capability of their software engineering process. Improving the process has an impact on lower development and maintenance costs, increased predictability and controllability of software products and processes, and shorter time to market. \[RKd03\]

1.1 Motivation

Organizations are highly motivated to find improvement opportunities in their current practices. Process Assessment helps to evaluate the current state and capability of an organization’s Process Model in order to identify strength, weakness and risk in undertaking software development.

Process Assessment is performed against a Process Reference Model that defines practices and artifacts for a good software engineering process in an abstract, high-level way. After assessment has been performed, capability of processes is measured in capability profiles. These profiles are analyzed to figure out improvement opportunities. It depends on the organization’s business goals on what processes an improvement program is developed. Such an improvement program consists of activities, outcomes and criteria for their performing. Processes must implement these activities and outcomes, and must satisfy the abstract criteria to achieve a certain level of capability. However, these activities do not define "how" they get performed and there are also no rules for name, format and content of outcomes. Knowledge about their implementation, the "how", then is needed. This knowledge comes from practical models which include good practices for improvement implementation. For example, Extreme Programming is a software development process that is based on 13 good practices for rapid software development. Each practice covers a certain aspect of software engineering that will give you a benefit if you adapt these practices for your engineering processes. There is also another kind of knowledge feasible, e.g. templates for test plans, or theorems for resource planning.

Today, knowledge about improvement implementation comes from external companies that focus is on consulting the organizations in their improvement programs. Expensive consulting could be diminished, if this kind of knowledge could be transferred to quality or project
managers by means of a software tool. Such a software tool could provide support for improvement activities. It necessarily includes models used in Process Assessment, that define the high-level activities and outcomes, and knowledge about their implementation. In order to achieve knowledge transition their elements could be linked together.

An attempt to provide Software Process Improvement has been given by the Finnish Software Measurement Association (FiSMA). One of its goals is to transfer knowledge of improvement cases to people who are responsible to implement these improvements. As a part of supply a knowledge base, called Gnosis, has been realized. It is mainly intended for browsing through a selection of processes, practices and concepts of Capability Maturity Model Integration (CMMI) and Software Process Improvement dEtermination (SPICE) which provide, among others, good software engineering practices. However, Gnosis’s architecture is based upon a plain model where relationships are established by element mapping. This architecture makes it difficult to integrate further Software Process Improvement related models. Gnosis has also more shortcomings in usability and stability which make it not well suited to provide you serious benefit for the support of Software Process Improvement activities.

Hence, Lepasaar, Mäkinen and Varkoi suggest in [LMV02] to implement a meta-model generalizing the structure of SPICE and CMMI. Such a meta-model could then be implemented within a Software Process Improvement supporting software tool.

1.2 Requirements Analysis

This thesis deals with the question how Software Process Improvement could be effectively supported by a software tool. The term "effective" refers to integrating SPI related models into a meta-model without loosing model related features, the way in which knowledge transition is achieved and a proper Process Model browsing structure is built. Finally "effective" refers to a supposable architecture for a prototypical implementation of a supporting software tool. The topics this thesis mainly deals with we characterize in the following paragraphs.

Process Model Integration into a Meta-Model Models related to Software Process Improvement, we refer to Process Models, can be depicted in their data model structure or in addition with data in a database schema. If we model them into a single data model, we could make a meta-model upon the database structure. Constructing a single data model is driven by the question if there is a generic Process Model that preserves model specific features. If this question is denied, does a general structure exist that could be used as a basis to be described by a meta-model? The worked out meta-model approach must be capable to be described by a language that transforms the meta-model into a machine-readable format. Secondly the language’s grammar needs to be processable by an intelligent agent.

Transition of Knowledge Transition of knowledge about Software Process Improvement is the primary purpose for a SPI supporting software tool. Such transition is from elements of Process Models to elements of practical models. We will raise the question, how transitions could be accomplished.

Transition actually means mapping of elements from different domains. However, it has been
exposed in Gnosis knowledge base that mapping of knowledge elements to Process Model elements one-by-one is inappropriate and circumstantial. We raise the question, is there a better way of connecting these elements?

**Meta-Data System to support Process Model Browsing** Browsing of Process Models should be performed upon the database schema. This means, relations between elements should be used to construct a desired navigation structure. A supposable browsing structure could be a tree from which root node Process Model browsing starts. Such a root node or starting concept must be declared explicitly. We will ask the question, which meta-data system works well upon the meta-model? In addition, it should be able to be extended for other purposes concerning Process Model browsing.

### 1.3 Overview about the Remaining Chapters

**The first chapter** you are currently reading, is an introduction into the field of study. It motivates for the need of a supporting software tool concerning Software Process Improvement. The primary requirements for a supposable solution are explained here.

**The second chapter** provides the background for the thesis and its solution. It starts with giving you definitions to the terms this thesis frequently deals with. Subsequently, two widely used and popular Process Models are introduced in their history, purpose and structure. Among them is Software Process Improvement Capability dEtermination (SPICE). It is becoming International Standard for Process Assessment. The second is Capability Maturity Model Integration (CMMI). It provides a similar approach like SPICE does, but is also to certify organization’s maturity. Both models are considered in their structure. For this purpose we create two conceptional data models that describe the entities and their relations among them.

A software tool concerning Software Process Improvement consists of knowledge about process improvement implementation. We give a review about what knowledge about Software Process Improvement is. For this reason we reflect on its structure in terms of its attributes, context and type.

Ontologies play the key role in this thesis. Ontologies have become popular since the Semantic Web initiative has been started by Berners-Lee. We describe what Semantic Web means, and we introduce the key languages that have been developed by the World Wide Web Consortium (W3C) to build meaningful and machine-readable ontologies. For this purpose we start by describing the Resource Description Framework (RDF) that provides a simple data model. It is followed by the RDF Schema (RDFS) that is the base language for designing simple ontologies. Finally we give a short review about the Web Ontology Language (OWL) that provides the highest potentials for describing ontologies for the Semantic Web.

---

1international consent about a standard published by national organizations or design in-house within an industry
CHAPTER 1. INTRODUCTION

The third chapter introduces an interesting work done by FiSMA according to provide software tools which support Software Process Improvement. As we have already mentioned within the motivation part, this knowledge base is termed to Gnosis. We introduce Gnosis in its way of working and we finally evaluate it concerning conception, usability and stability.

The fourth chapter is about the proposed solution. The architecture’s solution is thereby driven by the way of transforming Process Models and Knowledge about Software Process Improvement into ontologies. For the first point we consider to build a generic Process Model of SPICE and CMMI. However, such a generic model misses some model specific features. For this reason we develop an approach that is based upon integrating several Process Models into a single database schema. We specify mapping rules to transform this schema into an ontology.

The way of representing a domain of interest within an ontology is sometimes different. From the scratch we follow an iterative methodology to design an ontology for knowledge about Software Process Improvement.

Transition of knowledge is a topic that is discussed in the following. The main idea for knowledge transition comes thereby from the Semantic Web. For the purpose of reusing domain specific knowledge, Ontology Web Language provides a feature to import other ontologies. We show how this feature is basically to make cross-references between model items and knowledge items. For knowledge transition can happen we introduce two different ways: by "direct knowledge linking" and by "virtual knowledge linking".

In the last section we describe another Web Ontology Language feature that provides comments on ontology elements. We consider examples to show the impact on the construction of a proper Process Model browsing structure.

The fifth chapter introduces the thesis outcome in the form of a prototypical Web-Application, termed to SPI Browser. Its implementation is driven by the Model-View-Controller (MVC) paradigm. Web-Framework Apache Struts implements this MVC approach. The Struts configuration is described in their components for the SPI Browser.

Another topic this chapter deals with is the matter of ontology maintenance. We introduce a way based on using Protégé ontology editor to manage all the facets that are included within ontologies.

The sixth chapter The last chapter is on reviewing the suggested solution. It summarizes the results and it makes an appraisal about the primary achievements of this thesis. If there are critics on solution, they are addressed here. We give an outlook about future work and further research on this topic.
Chapter 2

Background

This chapter provides a background for the actual solution within the solution part of this thesis. It gives you an introduction to the topics this thesis and its proposed solution deals with.

Section 2.1 starts by giving you definitions for necessary terms in the domain of process Software Process Improvement.

In the following two of the most popular models that focus on Software Process Improvement are considered: SPICE alias ISO/IEC 15504 and Capability Maturity Model Integration (CMMI). We pay attention to their data model structure in section 2.4 pointing out the concepts and their relations between them.

Because these models define high-level and abstract criteria for improvement, knowledge for their implementation is needed. In section 2.5 following questions are answered: What is knowledge about Software Process Improvement? From which domains does it come from? And how is knowledge structured?

In section 2.6 we talk about the vision of the Semantic Web. Its key technologies and languages for describing ontologies have impact on the solution of this thesis.

2.1 Definitions

Most of the terms we use in this thesis are pretty overloaded in literature. There are many different definition in the Web and in literature. Sometimes terms are synonymously denoted to other, sometimes the same terms are used in different contexts.

**Process**  A *Process* is ”a set of interrelated activities which transform inputs into outputs” [ISO04a].

**Software Process**  A *Software Process* is ”the process or set of processes used by an organization or project to plan, manage, execute, monitor, control, and improve its software related activities” [EDM98].
Discipline  A Discipline “provides grouping facility of a set of activities that are oriented towards the discipline’s purpose” [RKd03].

Software Process Improvement  Software Process Improvement is a discipline or a subset of Process Improvement that’s primary purpose is on continually improving the software organization’s effectiveness and efficiency. Its purpose is on elevating the performance of a software process.

Model  A Model is a simplified representation of a system or a process.

Process Model  In [WIK05e] a Process Model defines the ”demands on the desired processes (the ”what”) and it describes how they might/should/could be performed (the ”how”).

Process Reference Model  In Normative Part 1 of ISO/IEC 15504 a Process Reference Model is defined as ”model comprising definitions of processes in a life cycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes” [ISO04a].

A Process Reference Model is a general Process Model that compromises activities and artifacts in a general way for no specific type of organization. For the discipline of software engineering there exists several Process Reference Models, for example, ISO/IEC 12207 or CMMI-SW.

Process Assessment  Process Assessment is a discipline or a subset of Process Improvement that provides examining organizations processes to determine their capability in order to identify improvement priorities. Process Assessment is either performed during a Software Process Improvement initiative or as part of process capability determination. Process Assessment has two principal contexts depicted in figure 2.1. Within the Capability Determination Context assessing processes identify risks in undertaking current practices. Analysis of strengths and weaknesses may motivate for prioritizing improvements steps that means by the Improvement Context.

![Figure 2.1: Context of Process Assessment](image-url)
**Process Assessment Model**  A *Process Assessment Model* is “suitable for the purpose of assessing the process capability based on Process Reference Models” [ISO04a].

ISO/IEC 15504 Part 5 defines an exemplary Process Assessment Model. Another similar approach for assessing of a single process capability is Capability Maturity Model Integration (CMMI). For the purpose of process capability determination continuous CMMI formulates a framework that provides guidance for assessing process capability. After assessment has been undertaken resulting capability profiles represent process-improvement paths by illustrating improvement evolutions.

**Organizational Assessment Model**  Another assessing approach is followed by organizational assessment models. They focus on certificate the entire organization capability concerning quality aspects or engineering aspects. Likely the most famous model to certificate the quality assurance of organization is *ISO 9000/9001 series*. Similar to the ISO 9000 approach is staged CMMI which focuses also on the entire organization. However, evolution of the organization’s processes is indicated by means of maturity levels.

**Software Process Improvement Model**  A *Software Process Improvement Model* guides organizations in planning and implementing an effective Software Process Improvement program.

In [ISO04c] a Software Process Improvement Model as part of the SPICE initiative is introduced. The basic activities are illustrated in figure 2.2. Another Process Improvement Model is IDEAL [3].

![Figure 2.2: Activities of SPICE Process Improvement Process](http://www.sei.cmu.edu/ideal/)

---

1 continuous means the CMMI continuous representation
2 staged means the CMMI staged representation
3 IDEAL has been developed by SEI [http://www.sei.cmu.edu/ideal/](http://www.sei.cmu.edu/ideal/)
2.2 SPICE

SPICE is an abbreviation for *Software Process Improvement and Capability dEtermination*. Its name goes back to the roots when it was initially used for assessing software processes, however, it can also include Process Reference Models for different kinds of businesses.

It has become an International Standard for Process Assessment. Since 1992 its further development is committed to the Working Group 10 of Sub Committee 7 of the Joint Technical Committee of the International Organization for Standardization (ISO). From there it has been technically denoted to ISO/IEC 15504.

SPICE was first published in 1998 as Technical Report Type 2. Being in state of Technical Report Type 2 indicates this report is still under technical development or it is not sure if it can achieve an International Standard. In this thesis SPICE TR Type 2 is referred to as SPICE’98 or ISO/IEC TR 15504. In the meantime the SPICE project has been continually technical revised and it is being re-published to become a full International Standard during 2005.

The objectives of the ISO/IEC 15504 are to help software development organizations increase their development productivity and quality of their software products. Process assessments are the method to assess the current capability of organizations processes as well as to point out appropriate process improvements. The focus for ISO/IEC 15504 is put on self-assessments, not on certification.

2.2.1 SPICE Process Assessment Model

Process Assessment is performed against a SPICE Process Assessment Model. Its architecture is driven by two normative elements: Process Reference Model and Measurement Framework. Picture 2.3 shows the general context of a Process Assessment wherein these elements are highlighted.

**Measurement Framework**

The Measurement Framework provides a basis for rating the capability of processes. The process capability evaluation covers 6 capability levels starting from level 0 for incomplete processes to level 5 for optimizing processes. Progressing from bottom to top capability of processes increases. Within this framework the measure of capability is based upon a set of process attributes. A Process Attribute is "a measurable characteristic of process capability applicable to any process" [ISO04a].

As informational part capability levels and their corresponding process attributes are listed:

- **Level 0: Incomplete Process** - The process is not implemented, or fails to achieve its process purpose.
- **Level 1: Performed Process** - The implemented process achieves its process purpose.
  - PA 1.1: Process Performance Attribute
• **Level 2: Managed Process** - The previous performed process is monitored, planned and adjusted
  
  – PA 2.1 Performance Management Attribute
  – PA 2.2 Work Product Management Attribute

• **Level 3: Established Process** - The previous managed process is capable to achieve its process outcomes
  
  – PA 3.1 Process Definition Attribute
  – PA 3.2 Process Deployment Attribute

• **Level 4: Predictable Process** - The previous established process now operates within defined limits to achieve its process outcomes.
  
  – PA 4.1 Process Measurement Attribute
  – PA 4.2 Process Control Attribute

• **Level 5: Optimizing Process** - The previous predictable process is continuously improved to meet relevant current and projected business goals.
  
  – PA 5.1 Process Innovation Attribute
  – PA 5.2 Process Optimization Attribute
ISO/IEC 12207 Process Reference Model

The Process Reference Model is selected according to the application domain of interest. Processes included within the Process Reference Model must be described in terms of its process purpose and its outcomes.

ISO/IEC 12207 defines a framework for Software Life-Cycle Processes that constitutes a Process Reference Model for the exemplar Process Assessment Model in ISO/IEC 15504-5. It consists of a "comprehensive set of life cycle processes, activities and tasks for software that is part a larger system, stand alone software product and software services. The standard provides common software process architecture for the acquisition, supply, development, operation and maintenance of software. The standard also provides the necessary supporting processes, activities and tasks, and organizational processes, activities and tasks for managing and improving the processes" \[ISO01\].

In alignment to ISO/IEC 12207 AMD1 & AMD2 the life-cycles, the fundamental processes and its sub-processes are depicted in figure 2.4.

Process Assessment Model Architecture

The Process Assessment Model "forms the basis for the collection of evidence and rating of process capability" \[ISO04b\]. For this purpose it has a two-dimensional architecture. The first dimension is the Process Dimension, which is characterized by a set of processes entities which purpose statements describe in measurable terms what has to be achieved in order to fulfill the process purpose. The Capability Dimension describes the scale for measuring the efficacy of the process capability in terms of capability levels and process attributes.

Figure 2.5 depicts the architecture wherein process entities of the Process Reference Model have been defined in alignment with ISO/IEC 12207. The capability scale is mapped to the SPICE Measurement Framework organized into capability levels and process attributes. The level of achievement of a process attribute is thereby measured by an ordinal rating scale: N - not achieved, P - partially achieved, L - largely achieved, F - fully achieved.

2.3 CMMI Framework

Capability Maturity Model Integration (CMMI) is a framework for assessing and improving the capability of organizations development processes.

CMMI bases on the achievements of Capability Maturity Model (CMMI which was developed by SEI (Software Engineering Institute) as response to the needs of the US Department of Defense (DoD) for better techniques in the selection of contractors. \[EDM98\]

In 1991 CMM Version 1.0 was officially released for software engineering. Several Process Models has been developed during meantime to meet the needs of assessing specific disciplines, e.g. Software Capability Maturity Model (SW-CMM), the Systems Engineering Capability Maturity Model (SE-CMM), Integrated Product Development Capability Maturity Model (IPD-CMM) and Supplier Sourcing Capability Model (SS-CMM).

CMMI project was finally formed to sort out the problem of using several Process Models.
Its mission was to combine three Process Models into an improvement framework. The latest version of CMMI, released in year 2002 in version 1.1, combines four Process Models into a single modular framework. Several integrations are available which are build up from the Capability Maturity Model for Software Engineering: CMMI-SW, CMMI-SE/SW, CMMI-SE/SW/IPPD, CMMI-SE/SW/IPPD/SS. \[SEI04\]

### 2.3.1 CMMI Representation for Capability and Maturity

CMM Integration encompasses two representations: Continuous Representation and Staged Representation. It depends on the organization’s goal which representation is chosen. If organizations want to measure the maturity of their whole engineering process, they will select the staged representation. If they want to determine process capability in order to select improvements, they will choose the continuous representation.
CHAPTER 2. BACKGROUND

The staged representation provides a proved sequence of improvements beginning from basic practices and progressing through a proved path of successive levels, each level serving as basis for the next maturity level. Satisfying the criteria each level defines, gradual improvements of maturity for an organization is achieved.

The continuous representation provides a recommended order for approaching process improvements within each process area. This allows to compare the capability between process areas of different organizations. Another afford is that capability of process areas can be compared to capability of processes determined by the International Standard of process assessment defined in ISO/IEC 15504.

**Maturity Levels for Staged Representation**

Selecting a CMMI model that uses a staged representation makes distinction between 5 maturity levels that provides a way to predict the future performance of organizations:

- **Initial** - organization processes are ad hoc and chaotic
- **Managed** - requirements are managed and processes are planned, measured and controlled
- **Defined** - processes are established and improved over time
- **Qualitative Managed** - quality and process performance measurement and detailed analyzing of the given results
- **Optimizing** - processes are continually improved by using collected quantitative improvement objectives

![SPICE Process Assessment Model](Image)

**Figure 2.5: SPICE Process Assessment Model**

The staged representation provides a proved sequence of improvements beginning from basic practices and progressing through a proved path of successive levels, each level serving as basis for the next maturity level. Satisfying the criteria each level defines, gradual improvements of maturity for an organization is achieved.

The continuous representation provides a recommended order for approaching process improvements within each process area. This allows to compare the capability between process areas of different organizations. Another afford is that capability of process areas can be compared to capability of processes determined by the International Standard of process assessment defined in ISO/IEC 15504.
CHAPTER 2. BACKGROUND

Capability Levels for Continuous Representation

The continuous representation of a selected CMMI model distinguishes between 6 capability levels to describe discrete levels of process improvement:

- **Level 0: Incomplete** - Process is either not performed or partially performed
- **Level 1: Performed** - Process is performed and satisfied the specific goals
- **Level 2: Managed** - A managed process is a performed process that is also planned, monitored, controlled and reviewed
- **Level 3: Defined** - A defined process is a managed process that is tailored from the organization’s set of standard processes, and contributes work products, measures, and other process-improvement information to the organizational process assets
- **Level 4: Quantitatively Managed** - A quantitatively managed process is a defined process that is controlled using statistical and other quantitative techniques
- **Level 5: Optimizing** - An optimizing process is a quantitatively managed process that is changed and adapted to meet relevant current and projected business objectives

2.4 Data Model Structure of ISO/IEC 15504-5 and CMMI

Models related to Software Process Improvement could be the central part of a supporting software tool which guides organizations through process improvement. We need understanding of the structure of these SPI related models.

In [LMV02] Lepasaar, Mäkinen and Varkoi have analyzed the structure of ISO/IEC TR 15504-5 and continuous CMMI. Their work has great impact to an initial data model structure. However, today International Standard ISO/IEC 15504-5 is shortly before publishing. It has changed in terminology to the Technical Report and thus we will analyze the newer versions structure. Concerning CMMI, continuous CMMI has a similar approach to SPICE for selecting improvements. We will not focus on staged CMMI, because it for improving the entire organization.

The logical structure of Process Model concepts and their dependencies can be illustrated within an Entity-Relationship-Diagram by means of Entity-Relationship modeling. This modeling technique is based upon the Entity-Relationship Model that has been developed by Chen. [Che76] The original notation comes from Chen, but there are a number of notations used, among the more common is Bachman’s crow’s foot. We will use the Bachman-notation.

2.4.1 SPICE Exemplar Assessment Model Structure

ISO/IEC 15504-5 describes an exemplar Process Assessment Model that is compatible to the reference model defined in SPICE Part 2.

This example Process Assessment Model expands the reference model by adding assessment indicators. *Assessment Indicators* are "objective attributes or characteristics of a practice
or a work product that supports a judgment of the performance or the capability of an implemented process” [EDM98].

**Assessment Indicators**

ISO/IEC 15504-5 distinguish two types of assessment indicators: *Process Performance Indicator* (PPI) and *Process Attribute Indicator* (PAI). PPI apply solely to ”Capability Level 1” and are chosen explicitly to address the achievement of the defined process purpose and outcomes. Indicator for process performance are *Base Practice* and *Work Product*.

Process attribute indicator address practices that provides guidance on the implementation of the attribute achievements, resources that may be used when performing the process and basic types of work products that may be input to or output from all types of processes. Process attribute indicator are: *Generic Practice* (GP), *Generic Resource* (GR) and *Generic Work Product* (GWP).

**Process Dimension**

For the process dimension process entities have been adapted from the International Standard for Software Life-Cycle Processes defined in ISO/IEC 12207. This standard defines 48 processes covering the discipline of Software Engineering. Each *Process* is described in terms of a *Process Name*, a *Process Purpose* and in list of its *Process Outcomes*. Process outcomes are expected positive results of its process performance. Additional elements are *Process Notes* and *Process Identifier*.

Life-Cycles provide process grouping into three *Process Categories*: "Primary Life-Cycle", "Supporting Life-Cycle" and "Organizational Life-Cycle". Within each life-cycle a second grouping into 10 *Process Groups* is made concerning the type of activity processes address. For example "Supporting Life-Cycle” is classified into process groups "Configuration Control Process Group" and "Quality Assurance Process Group".

The process dimension is extended by a set of *Base Practice* for each process providing definitions of tasks and activities to accomplish the process purpose and fulfill the process outcomes. *Work Product* can either appear as input or output to processes. To review input and outputs of process implementation work products can be characterized by their *Work Product Characteristics*.

**Capability Dimension**

Evolving process capability is expressed in terms of *Process Attributes* grouped into 6 *Capability Levels*. These capability levels and their corresponding process attributes have been adapted from the Measurement Framework defined in ISO/IEC 15504-2.

Each process attribute has a set of associated process attribute indicators to indicate the extend of achievement of the process attribute in the instantiated process: *Generic Resource* and *Generic Work Product* apply directly to process attributes. *Generic Practice* provide guidance on the implementation of *Process Attributes Achievements*. Generic work
products are characterized by a set of **Generic Work Product Characteristic** that would be expected to be evident for work products.

Certain processes relate to the capabilities addressed by a process attribute and evidence of their performance support achievement of the attribute.

**SPICE Data Model**


Concepts of **Process Purpose**, **Process Note**, **Process Name** and **Process Identifier** become attributes of entity type **Process**.

The concepts of **Assessment Indicator**, **Process Performance Indicator** and **Process Attribute Indicator** do not necessarily become entity types. They provide just grouping facility.

Relationships represent associations between these entities. Relationship names are not represented within the model, instead, role names are denoted. Sometimes it is not obvious in which relation an entity stands to another. Then we denote a role name to the entity in the relation.

Entity attributes which have not been stated in the previous paragraphs are retrieved from the Process Assessment Model documentation in [ISO04d](#).

According to the given information, an entire conceptional data model has been created. It is illustrated in figure [2.6](#).

### 2.4.2 CMMI Structure

Capability Maturity Model Integration is a framework that provides guidance to implement an effective process based on a selected Process Reference Model. The continuous representation follows a similar approach to SPICE in the area of capability determination. The staged representation focuses on capability of the entire organization measuring its maturity.

**Model Components**

CMMI components are grouped into three categories:

- **Required**: **Generic Goals** and **Specific Goals** define required achievements to satisfy the corresponding process area. These components must be achieved by an organization planned and implemented process.

- **Expected**: Expected achievements are defined by **Generic Practices** and **Specific Practices**. They describe what an organization will typically implement to achieve a required achievement.
Informative: Subpractices, Typical Work Products, Discipline Amplifications, Generic Practice Elaborations, Goal Titles and Practice Titles, Goal Notes and Practice Notes, and References are informative model components that help model users understand the goals and practices and how they can be achieved.

Continuous Representation

The continuous representation uses 6 Capability Levels to indicate improvement evolution for each of the Process Area organized by 4 Process Area Categories, for example "Project Management" or "Engineering".

Specific Goals apply to process areas to address a unique characteristic that describes what needs to be implemented to satisfy the process area. Specific Practices are grouped to specific goals that collectively describe the activities to achieve the associated goal. As it is pictured in figure 2.7 these specific practices are mapped to capability levels. Specific practices at capability level 1 address Base Practices and specific practices at higher levels...
are called Advanced Practices. Advanced practices may build on a base practice, but it is not necessary.

Each capability level has one Generic Goal that describes what an organization has to achieve at that capability level. Generic Practices are the required model components that describe the activities to achieve the generic goals. Generic practices apply to just one generic goal.

A Typical Work Product provides example output from specific or generic practices. Sub-practices provide guidance for performing specific and generic practices.

Discipline Amplifications are informative components that contain information to specific practices relating to a particular discipline.

Generic Practice Elaborations provides guidance on how generic practices should be applied to one certain process area.

![CMMI continuous representation](image)

**Figure 2.7: CMMI continuous representation**

**CMMI Data Model**

The entity-relationship model illustrated in [LMV02] is modified concerning the fact that we want to track the practices which have impact on the process performance. They are called base practices. They are directly applied to specific goals. Advanced practices are also associated to specific goals. Both concepts are generalized into specific practice. From there we can figure out to which capability level a specific practice belongs.

Specific practice and generic practice are generalized by practice, because subpractice can apply to generic and specific practice.

The entire data model is depicted within figure 2.8

**2.5 Knowledge about Software Process Improvement**

After process assessment has been performed improvement actions are prioritized. These improvement actions "consist of good software engineering practices, however, defined as abstract, high-level concepts without constraining the ways in which they may be implemented. The criteria are the set of attributes that might be found in an instantiation of a process" [EDM98].
Guidance how to implement improvements is more important after assessment. So far external consultants have been supported improvement activities by their knowledge and experience about Software Process Improvement implementation. This knowledge comes from more practical models that give guidance for implementation. For example Extreme Programming is software development methodology that turns project faster into success. Basically it bases on 13 practices covering good practices that have great impact on software quality and project time. They are described in their practice and their benefit. Implementing these practices could contribute to process improvement. This is what we mean by knowledge.

Knowledge in general meaning is defined in [WIK05c] as "the awareness and understanding of facts, truths or information gained in the form of experience or learning, or through introspection". But knowledge is "neither data nor information. Knowledge is distinct from simple information. Both knowledge and information consist of true statements, but knowledge is information that has a purpose or use. It is information with intention" [WdHvdS97].

We limit discussion to experience needed for Software Process Improvement implementation. This is why we are talking about SPI Knowledge.
Knowledge Item

Knowledge Items are the smallest pieces of experience that can be created, stored and shared. When we talk about knowledge items we usually mean individuals within the class of knowledge.

Based on the experience of Gnosis knowledge base, that has been evaluated in section 3.1, a possible structure for knowledge items can be adapted and reused. Any knowledge item included in Gnosis owns following structure:

- **Title** - name of this knowledge item
- **Problem** - what problem is addressed by this item
- **Purpose** - what is the purpose
- **Description** - a detailed description
- **Examples** - useful examples
- **Solution** - what is the proposed solution
- **Risks** - what are the risks
- **References and Links** - references to other documents
- **Resource Requirements** - which resources are additional needed
- **Author** - the author of this item
- **Applicability** - to which organizations this knowledge is applicable
- **Introduction** - brief description
- **Related Knowledge** - items that are subsequently related

Knowledge Context

A Knowledge Context is a domain to which knowledge items provide benefit. A knowledge context is a kind of higher-level domain to which a couple of knowledge items provide implementation help. A knowledge item can have at least one context, but it does not necessarily have one. In this case, it is related to another knowledge item that provides benefit to a certain context.

For example, we could add a knowledge item that includes experience about good defect classification. It might be called "Classification of defects and errors". It could be in context to "System Testing" or "Software Component Testing" or "Testing" in general.

Another example comes from Extreme Programming. XP is an software engineering method that is based on 13 practices. Any practice is related to a knowledge item that states about an "Agile Software Development". Any of these base practice is related to this item. However, not any practice has a context. There is a practice called "Metaphor" for which no proper context could be found. But in contrast practice "40h Week for Software Engineers" could be context to "People Workspace" and "Project Planning".
Knowledge Type

Another attribute for knowledge that supports SPI is its type. A Knowledge Type indicates the naturally form of an knowledge item. For example, if we had a knowledge item that supplies making a project plan by a template, it might be therefore of the type "template". Following list of types might possible:

- **Good Practice** - practice that helps to achieve a specific goal (sometimes known as best practice)
- **Theorem** - statement that has been proved by someone
- **Template** - a sheet of a special pattern
- **Information** - facts about an area of interest
- **Guidance** - gives help or advise to perform a task
- **Paradigm** - example that shows how something works or is produced

2.6 Vision of the Semantic Web

*Semantic Web* is for the time a buzzword that is more and more mentioned when experts discuss about the Internet of the future.

The word *semantic* implies meaning or, as the DUDEN[4] defines it, "study of the meaning of speech symbols". For the Semantic Web, *semantic* indicates that the meaning of data on the Web can be discovered - not just by people, but also by computers.

The vision behind *Semantic Web* is "the World Wide Web becomes a giant database, linking both human readable documents and machine readable data in a way useful for mankind and machine" [BL03].

Currently the World Wide Web consists of textural files that are evaluated just for displaying purposes. Information and knowledge are encapsulated within Hyper Text Markup Language (HTML) or Hyper Text Markup Language (XML) tags used for information displaying or structuring.

Until now just humans can make accessible the sense of documents, but the idea of Semantic Web is to give data more meaning through the use of meta-data, the data about data, that describes how, when and by whom a particular set of data was collected. By adding meta-data to existing Web, the Semantic Web will allow both humans and machines to find and make in ways that previously haven’t been possible. Actually the benefit of the Semantic Web is for machines. They become able to discover and to extract desired information from Web resources. [BL03]

But how machines can do, how humans have ever could? Berners-Lee refers in [BL04] that the Semantic Web uses two different approaches: A common readable data format and a well

---

[BL03] German standard book for grammar
defined ontology. The former one addresses a standardized machine readable format that base on standards XML, Resource Description Framework (RDF) and on the top of them the Web Ontology Language (OWL).

"The father" of the Semantic Web initiative, Berners-Lee, has considered in [BL00] the architecture that is sometimes called "Semantic Web layer cake" depicted in figure 2.9 because of its stack view on languages.

![Figure 2.9: Layers of Semantic Web pyramid](image)

The layers are explained as follows:

- **Unicode** - Character encoding format
- **URI** - Unified Resource Identifier. URI provides a common syntax for naming resources
- **XML Schema** - Language used to describe the structure of specific XML languages
- **RDF** - Resource Description Framework. Its data-model describes meta-data of resources
- **RDF Schema** - Language for describing schemes in terms of classes and properties. Allows to design simply ontologies.
- **OWL** - Web Ontology Language. More complex and expressible language for ontologies.
- **Logic** - Logical reasoning ensures the consistency and correctness of data sets and infers conclusions that aren’t explicitly stated
- **Proof** - Trace the steps of logical reasoning
- **Trust** - Authentication and trustworthiness of data, services and agents

The sections following deal with underlaying techniques that allows to state informations into ontologies. In particular we lay discussion on RDF that provides the fundamental data-model, RDF Schema describing an ontology vocabulary and OWL is for more expressive ontologies.
2.6.1 Resource Description Framework

The World Wide Web Consortium (W3C) has been developed the Resource Description Framework (RDF) as a key component of its vision for a Semantic Web. RDF is a language for representing information about resources in the World Wide Web. It was especially designed to provide meta-data about Web resources. [MM04]

RDF Data-Model

RDF uses a simple data-model to structure information about resources in the form of Statements. An RDF statement is equivalent to English Grammar that is also being built of these different parts:

- **Subject** - Resource about is made this statement
- **Predicate** - Property of the subject
- **Object** - Property value. Can be either a resource, a literal or a blank node

Because of the three parts structure a statement is called a *Triple*. A triple is the smallest piece of knowledge that is human readable as well as machine readable.

![Figure 2.10: Graphical representation of a RDF statement](image)

The parts of a statement - subject, predicate and object - are described by resources names URI references. An object is not a resource at all. Objects can be either a resource addressed by URI references, a Literal or a Blank Node. Literals are a characters representation of the object value. Blank Nodes are anonymous resources which are not denoted by an URI. There are some applications that make up their own identifier for blank nodes, but in natural manner they are left. Blank nodes are particular used for aggregating things to be described as resource and then making statements about the resource. They act thereby as group of facts.

An URI references is an identifier for resources. It contains a prefix, which corresponds to a namespace, following by a colon and a local name. The last part is also denoted as fragment identifier. A fragment identifier is optional and may be only used to identify resources.

The structure of a URL reference is:

\[
\text{URL Reference} = \text{URI} \ [\text{fragment identifier}]
\]

\[
\text{fragment identifier} = '#' \text{local-name}
\]

The prefix refers to the term of Unified Resource Identifier (URI). The Network Working Group describes a URI in [BL+98] as "compact string of characters for identifying an abstract or a physical resource".
CHAPTER 2. BACKGROUND

A URI can refer to a location as well to a name. URI are related to URL (Uniform Resource Locater) in that URL is a subset of URI of an URL scheme based on a known protocol (e.g. HTTP or FTP protocol). URI references from a specific domain form thereby a collection that is called a vocabulary.

Examples of URI, URI references and URL:

http://www.debian.org/intro/about.html - URL or URI
http://www.w3.org/1999/02/22-rdf-syntax-ns#Class - URI reference
this.is.just.a.name - URI

XML Serialization of Statements

XML is a widely used data exchange format that can be processed by XML Parsers. It allows to encode RDF statements into XML files that is known as rdf-xml serialization. In the matter of representing the RDF data model into XML, the RDF Syntax Recommendation [Bec04] provides a XML vocabulary to transform RDF statements into XML terms.

```xml
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description rdf:about="http://www.hornet-home.de">
      <dc:Title>Honda Hornet Homepage</dc:Title>
      <dc:Creator>Andre Frenzel</dc:Creator>
      <dc:Description>Largest community for Honda Hornet Drivers in Germany.</dc:Description>
   </rdf:Description>
</rdf:RDF>
```

Listing 2.1: RDF statements transformed into XML

2.6.2 Ontologies

Ontologies play a key role in the terminology of the Semantic Web. There are many definitions in literature about what an ontology is. John Davies defines in [DFvH03] ontologies as a "shared and common understanding of a domain of interest that can be communicated between people and application systems".

The term ontology comes thereby from the philosophy and means the study of "being" or existence trying to find out what entities and what types exist. In computer science, the meaning for ontology has derived from the philosophy and it is defined in [WIK05d] as "formal system of concepts, describing all the relevant entities, their relationships and the rules within the domain of interest".

Ontologies can be expressed by ontology representation languages. The next section introduce two languages for describing ontologies.
2.6.3 RDF Schema

RDF vocabulary description language (RDF Schema or RDFS) is the base language for describing ontologies. It bases on the top of RDF and it is intended to use for limited and inexpressive ontologies. [Zie03]

RDFS is an extension to RDF to specify domain specific vocabulary and object structures. It provides mechanisms for describing groups of related resources and the relationships between these resources. [BG04]

In other words, RDFS provides meta-model functionality to describe areas of interest as it XML Schema does, however, not on the syntax level, rather than on the semantic level.

Within the next sections the basic features of RDFS are introduced.

Classes

A significant feature that RDFS provides is to group resources into Classes. Classes are "concrete representation of concepts" [Hor04]. They are declared using property rdfs:Class.

Classes are typically arranged into class hierarchy which is also known as Taxonomy. RDFS provides the property rdfs:subClassOf to make statements about classes that are sub-classes of another.

Consider the example of printers in order to introduce the concept of classes and sub-classes. The statements below create a class hierarchy of several kind of printers. The class of Printer is the most general concept. LaserPrinter, MatrixPrinter and InkJetPrinter are more specific classes because of their different printing technique. They apply as subclasses to class Printer. The concept of InkJetPrinter is further divided into BubbleJetPrinter and PiezoJetPrinter. The resulting hierarchy is pictured in figure 2.11.

```xml
<rdf:Class rdf:ID="BubbleJetPrinter">
    <rdfs:subClassOf rdf:ID="InkJetPrinter"/>
</rdf:Class>
<rdf:Class rdf:ID="InkJetPrinter">
    <rdfs:subClassOf rdf:ID="Printer"/>
</rdf:Class>
<rdf:Class rdf:ID="LaserPrinter">
    <rdfs:subClassOf rdf:resource="Printer"/>
</rdf:Class>
<rdf:Class rdf:ID="MatrixPrinter">
    <rdfs:subClassOf rdf:ID="Printer"/>
</rdf:Class>
<rdf:Class rdf:ID="PiezoJetPrinter">
    <rdfs:subClassOf rdf:ID="InkJetPrinter"/>
</rdf:Class>
<rdf:Class rdf:ID="Printer">
    <rdfs:subClassOf rdf:resource="Resource"/>
</rdf:Class>
```

Listing 2.2: Defining a superclass-subclass hierarchy
Instances

An *Instance* represents an object if it applies to a class. We also say an instance is a member of a class. Resources must be typed by RDF property *rdf:type* to that class to which it is instance.

Properties

RDFS defines the concept of *Properties*. Properties, that are denoted by *rdfs:Property*, are relations on subject resources and object resources.

Properties can be arranged into property hierarchies in the same way as it is done for classes. The related property is called *rdfs:SubPropertyOf*.

Consider the following example: *b rdfs:SubPropertyOf a*. The meaning of the semantic level is that all pairs of resources that relate to property *a* also relate to sub-property *b*.

```xml
< rdfs:Property rdf:ID ="b">
  < rdfs:SubPropertyOf rdf:ID ="a">
  </ rdfs:Property >
</ rdfs:Property >
```

Listing 2.3: Defining a property hierarchy

Properties can state about its domain (*rdfs:domain*) and its range (*rdfs:range*). In statements where its predicate is a property, a subject resource must satisfy the domain and an object value the range. This means a subject resource must be a class or a sub-class to which the domain is set. Object values must be instance of the class, or its sub-class, the range is set to.

```xml
< rdfs:Property rdf:ID="a">
  < rdfs:domain rdf:ID="A">
  < rdfs:range rdf:ID="C">
  </ rdfs:Property >
</ rdfs:Property >
```

Listing 2.4: Defining the range and the domain of a property
2.6.4 Web Ontology Language

The fundamental language for representing semantic facts is called Web Ontology Language (OWL) that is currently a recommendation of the World Wide Web Consortium (W3C).

OWL facilitates greater machine interpretable of Web content than that supported by XML, RDF, and RDF Schema. OWL builds on the top of RDF Schema. Thus, it has all the capabilities RDFS has, but it provides a couple of new facilities. Some of new features are introduced in the following.

Species of OWL languages

OWL comes with three sub-languages:

- **OWL Lite** - provides simple class hierarchies and simple constraints
- **OWL DL** - includes all OWL Lite languages constructs and add additionally constructs for restrictions, but the language remains decidable and computable. OWL DL corresponds to the field of Description Logics which deals with decidable fragments of the First Order Logic.
- **OWL Full** - supports the full specification of OWL constructs. However, there is no guaranty that computation stops in finite time

Changes to RDFS

However, there are slightly different syntax representing classes and individuals. In OWL classes are denoted by `owl:Class`. Each user-defined class is implicit a sub-class of `owl:Thing`.

In RDFS the concepts of properties have been launched. Properties let us assert facts about member of classes. OWL split properties into two different types:

- `owl:DatatypeProperty` - relations between instances of classes and RDF literals or XML datatypes
- `owl:ObjectProperty` - relations between instances of two classes.

Property Restrictions

Within the context of `owl:Restriction`, restrictions can be applied on range and domain of properties using OWL construct `owl:onProperty`. Restrictions fall into three categories: Quantifier Restriction, Cardinality Restriction and `hasValue-Restriction`.

Quantifier Restrictions limit which values can be used. They exist in two alternatives:

- `owl:allValuesFrom` - a particular class may have a restriction on a property in such every instance has only values from a certain type. This is also known as universal quantifier.
• **owl:someValuesFrom** - a particular class may have a restriction on a property such every instance member of that class has *at least one* value of a certain type. Also named *existential quantifier*

*Cardinally Restrictions* limit the number of values on this property on instances of that class. They exist also in two alternatives: *owl:minCardinality* and *owl:maxCardinality* that refer to the upper and lower limit.

Last but not least, *HasValue-Restriction* require to a property to have a certain individual as its value.

### Equality and Inequality

We can state about two classes being equivalent *owl:equivalentClass*. Equivalent classes have the same individuals and are therefore called *Synonymous Classes*.

Look at figure 2.12 Both classes *ColourPrinter* and *ScannerPrinterCombination* are sub-classes of *Printer*; but they are red colored. That means they are equivalent classes to mostly not named classes. These unnamed classes are restricted on property *isColourPrinter* and on property *hasScannerFacility* respectively to a given value. Any individual that satisfy this condition is member of this class. By means of *Reasoning* these individuals can be deducted.

![Synonymous classes created with a hasValue-Restriction](image)

**Figure 2.12: Synonymous classes created with a hasValue-Restriction**

### Boolean Expressions

Boolean combinations help to define arbitrary classes and restrictions: *unionOf*, *complementOf*, *intersectionOf*. Consider the example before, we could have constituted a class which intersects classes *LaserPrinter* and *ColourPrinter* to a new class of *ColourLaserPrinter*.

---

5 within RDF Graph these classes become blank nodes
Chapter 3

Related Work

There has been interesting work done concerning development Software Process Improvement related software tools. One attempt has been given by FiSMA\(^1\). We take a closer look at one of the achievements by FiSMA.

3.1 GNOSIS - a Knowledge Base for Software Process Improvement

Finland has been established as one of the leader in Software Process Improvement. Hightechfinland.com describes the role of continuous Software Process Improvement. To provide advice and share experience in the field of Software Process Improvement FiSMA has been founded in 1998.

FiSMA is a network about 40 member organizations. Their activities cover guiding development of the estimation tools and experience database, guiding support and development of the method based on ISO 15504 standard, meetings and seminars for the members about software measurement topics, participation in ISO standardization in the software process and measurement areas.

One of the out-coming tools released by FiSMA was a knowledge base known as Gnosis for primarily support of internal assessments and SPI planning activities. However, Gnosis knowledge base is still in prototypical state and not for being thought for production use.

Gnosis implements a common set of best practices and company specific knowledge for context specific SPI needs denoted as knowledge items within the system. This knowledge has been linked to those processes, concepts and practices of Process Reference Models CMMI-SW and ISO/IEC 15504 that needs to be given supplementary information e.g. for guidance.

Within the next sections the basic possibilities of Gnosis is introduced and finally discussed and rated.

\(^{1}\text{Finnish Software Measurement Association}\)
3.1.1 User Interface

The Gnosis Knowledge Base is a Web application and therefore can be easily accessed by any Internet Browser.

First of all user and password are requested for getting authorized by the system. After authentication has been succeeded, the user is forwarded to the welcome page that is actually the SEARCH Web page. Furthermore there are three other Web pages: BROWSE, MAINTENANCE and GRAPH.

SEARCH Web page

At this page a keyword based search facility is offered. Gnosis serves a list of items as result. This list is ordered by the calculated strength coefficient. The strength coefficient describes the relationship’s relative importance.

Items are not only picked in the result because of the name includes the keyword, also related items are put in the result. This is why the strength coefficient is important.

Items which are found are denoted by different kind of circles or by their type respectively. Particular knowledge items are emphasized by a green colored sign that makes user easier to see interesting items at the first glance.

BROWSE Web page

The browsing functionality enables the user to access all the Process Model elements and knowledge items within a Java applet depicted in figure 3.2. Reading the data from the Java applet takes up to 45 seconds also by using a fast Internet connection.
The applet enables the user to group data by a certain style: type grouping, none grouping, level grouping, level-type grouping, model grouping.

Within the applet a search operation is offered which directly operates on the tree. After a keyword is entered not matching items are immediately skipped.

By selecting a certain item from the tree a detailed description is displayed.

**MAINTENANCE Web page**

At this Web page the user can maintain all the items defined in Gnosis, in particular goals, impacts, concepts, knowledge items. These items can be edited by a WYSIWYG HTML editor that comes as 3rd party applet pictured in figure 3.4.

**GRAPH Web page**

Within the Java applet relations of connected items are graphically represented. Different colors are used to indicate the different categories. The initial graph is computed by a depth
CHAPTER 3. RELATED WORK

3.1.2 Data-Model Structure

Gnosis distinguishes data between seven several categories:

- **CMMI Specific Practices** - base practices from CMMI Process Model
- **SPICE Base Practices** - base practices from SPICE Process Model
- **Concepts** - further CMMI and SPICE Process Model elements
- **Goals** - serve as reference points to elements which attain the goal
- **Impacts** - serve as reference points to elements which have impact
- **Knowledge Items** - good practices or specific knowledge
- **Processes** - processes which either belong to CMMI or SPICE Process Model

The structure of these items is as follows:
Figure 3.4: Gnosis’ maintenance page

\[
\text{structure}(\text{dataItem}) = (\text{title}(\text{dataItem}), \text{properties}(\text{dataItem}), \text{relatedItems}(\text{dataItem}))
\]
\[
\text{properties}(\text{dataItem}) = (\text{attribute}, \text{value})^*
\]
\[
\text{relatedItems}(\text{dataItem}) = (\text{link}(\text{relatedItem}), \text{type}, \text{strength}^*)
\]

\text{type} = \text{has-a} | \text{supports} | \text{overlaps} | \text{helps} | \text{see also} | \text{association}

\text{strength} = \text{strong} | \text{medium} | \text{weak}

The basic type of a relation is indicated by property \text{type}. Relationship of type "has-a" is most frequently used.

The \text{strength}-property denotes the impact of related items to the relationship. A strong relation means there is a high impact on the item, while a weak relation means there is just a low impact on the related item.

3.1.3 Evaluation

Although Gnosis is in prototypical state its shortcomings in concept, usability and stability do not make it well suited for professional use. Its user interface responds slow because of Java Applets. Session time-out and internal exception messages forwarded to the user are user unfriendly. Gnosis has not been designed to adapt further Process Models. Its architecture is restricted to concepts from SPICE and CMMI.

We shortly discuss and classify any issue concerning conception, usability and stability.
CHAPTER 3. RELATED WORK

Figure 3.5: Gnosis’ graphical view on relationships

Conceptual Issues

**item to item linking** Relations are established by one to one mapping. Knowledge items which are added to the knowledge base must be manually connected to related items. Managing the knowledge base means that a maintainer needs also skills in field of Process Models to apply the knowledge to the right concepts.

**no distinction of concepts** Gnosis unionizes several concepts like Process Category, Work Product, Process Attribute into a single category Concept. However, these concepts are unlike each other. They distinguish in their meaning and in their of structure.

**fixed relationship types drive into loss of information** Relations among items are denoted by a fixed identifier. Model items are connected by relationship type "has-a". However, sometimes their meaning is more than "has something”. For example, relationship between elements from category Process Category and elements from category Process could be more meaningful denoted by type has-Process.

Usability Issues

**use of client-side Java applets** Use of Java applets is almost reserved for interactive applications. They have negative impact on the performance of Web-applications. Before applets can be used in Web pages, the Java Virtual Machine needs initialized before. This takes time and a lot of memory capacity.

**slowly data-model reading** Although the user has a fast Internet connection reading the data-model takes a couple of time caused by the transfer of data to the applet. Even by DSL connection Gnosis need up to 60 seconds.
Stability Issues

out of session  After a short period of time (about 15 min) while inactivity session is getting lost. The session period is too short to seriously work with Gnosis. However, this can be fixed setting a higher value to the session time-out.

no exception handling  There is no exception handling implemented when the application crashes. Therefore, the user is confronted with an incomprehensible exception message. One of the design goals of good software is to prepare meaningful error messages to the user.
Chapter 4

Proposed Solution

This chapter suggest an architecture that refers to provide effective support for Software Process Improvement within a software tool.

One of the system’s design goal is, the database is not fixed on the structure for a number of Process Models. This means, Process Model’s structures might differ in their concepts and their relations among the concepts. As we have pointed out in section 2.4 the data model of SPICE and CMMI is not similar. But, if we identify a generic Process Model that provides a generalization about the concepts and relations, we could implement its data structure into a (relational or object-oriented) database. However, a database is “a collection of information [...] that can be managed and queried by a database management system” [WIK05a], but there is no system that provides meta-data about the database concepts.

Ontologies are like databases. The structure (schema) and the content (data) form a database. If we think of the schema and the data, then the ontology becomes a meta-model. In section 4.1 and in the following we discuss this meta-model approach concerning a database schema mapping into an ontology.

There are two alternatives how we could Process Models bring into a single data schema. Either by developing a generic Process Model or by integrating the structure of several Process Models. The first option is discussed in section 4.1.2. However, it has been shown transformation into a generic model comes with loss of model specific features. Therefore, we advise in section 4.1.3 to develop by a single data schema that is based upon an integration approach for several sub data models that are mapped into an ontology meta-model.

As well as ontologies can be used for describing a schemata, they are "formal defined systems of terms and/or concepts and relations among them to represent knowledge in the domain of discourse" [WIK05a]. The domain of discourse is here knowledge about Software Process Improvement. By an iterative approach we show in section 4.2 how SPI Knowledge could be successive transformed into a simply ontology, called taxonomy, that provides classification for the knowledge objects. Afterward we discuss classes as "sets of individuals that are collected together for the purpose of classification" [Pas04] according to provide a finer granular ontology of SPI Knowledge.

From the Semantic Web comes the possibility "to allowing ontologies to be related, including explicitly importing information from other ontologies" [S+04]. We show in section 4.3 how we
can use the import feature from Web Ontology Language to accomplish knowledge transition. As we have figured out in section 3.1 knowledge items could be one to one mapped to model items requires knowledge about Process Models. We introduce and propose another way of knowledge transition can happen by means of "virtual knowledge linking".

Ontologies provide the schema and the data about Process Models and SPI Knowledge. However, they do not contain information about a proper navigation through each Process Model. By attaching annotations to existing ontologies, without changing them, we can supply information for a software tool. This OWL meta-data construct is originally used to provide additionally comments on classes, properties and individuals. How we are going to reuse it for presentation purpose is described in section 4.3.

4.1 Ontology-based Meta-Model

In [LMV02] it has been suggested by Lepasaar, Mäkinen and Varkoi in their conclusion to "integrate the structure of SPICE and CMMI and later on, to develop a meta-model that generalizes the Process Model structure of SPICE and CMMI". Additional information have not been given about what is meant by structural integration and generalizing the structure of CMMI and SPICE into a meta-model.

Johannes Ernst defines in [Ern03] a meta-model as "an explicit model of the constructs and rules needed to build specific models within a domain of interest". With other words, a meta-model describes the methods of a model that represents a cutout from the real world. If we had such a meta-model, we could make statements about the methods the underlaying model describes the real world.

4.1.1 Mapping Database Schemes into Ontologies

As we have seen in section 2.6.4 ontologies can be expressed by Web Ontology Language. OWL has a grammar that can describe things in ontologies in its terms of classes, properties and individuals.

OWL capabilities work also for expressing database schemes. A database schema can be split into table definitions (the corresponds to the schema) and a data dictionary, that contains the data. The relations between data in database schemes can be represented by data models.

If we describe a database schema, by its schema and data, by means of Web Ontology Language into an ontology, then the resulting ontology constitutes a meta-model for the schema.

Database schemes are graphical representations of concepts and their relationships. For example, the Entity-Relationship Model is "a data model for high-level descriptions of conceptionsal data schemes" [WIK05]. Consider the fact, we can transfer database schemes into an ontology, therefore we can transfer Entity-Relationship Models into an ontology, too. We will provide a mapping from the terms used in the Entity-Relationship-Model to the languages constructs defined in OWL. This mapping is depicted in figure 4.1.

- **Entity Types** - Each entity type in the data model becomes a class within the ontology
CHAPTER 4. PROPOSED SOLUTION

37

Figure 4.1: Single data model mapping

defined by property \texttt{owl:Class}. Entity types that have a specialized meaning to generalized entity type - their relationship is denoted by "is a", become a sub-class to the more general class. Class to sub-class relation is denoted by property \texttt{rdfs:subClassOf}.

- \textit{Relationships} - Relationships exist between entities. They can be seen as uni-directional if there is no relationship name denoted. Any direction is a relation that becomes an object property \texttt{owl:ObjectProperty}. An Object property has a domain \texttt{rdfs:domain} and a range \texttt{rdfs:range}. Property domain is the class of the mapped entity type from which the relationship goes out. The range is defined by the class to which the relationship goes in. If there are roles defined, what means an entity has a role name within the relation, then this relation becomes labeled \texttt{rdfs:label} with the role name.

- \textit{Attributes} - Entity types in conceptional data models are described by attributes. A attribute becomes a datatype property defined by \texttt{owl:DatatypeProperty}. The domain of the datatype property is the class name to which the entity has been mapped. The datatype property range is a XML datatype corresponding to the domain of the attribute.

- \textit{Entities} - Entities are the data. They are mapped to instances of its classes.

4.1.2 Considerations of a Generic Process Model

In section 2.4 we have pointed out the structure of SPICE and CMMI that can be represented within separate data models. Using the mapping rules we have given in the section above, we are able to transform each of these schemes into a separate ontology.

However, if we integrate both models into a generic Process Model, and therefore into a single data model, we could develop upon the generalized structure a meta-model.

At first we discuss the question, if there could be a generic Process Model that represents the concepts and relations of ISO/IEC 15504-5 and continuous CMMI?

A generic Process Model provides the terms and relationships at a higher abstraction level. Specific model concepts which have the same meaning, are merged into a generic concept. The same takes place to attributes and relationships. Finally we obtain a generic Process Model that includes concepts, attributes and relationships, which provide an intersection of the model specific components. Designing a generic model comes often along with loss
of information of specific model features. Because it provides an intersection of common concepts, model specific features were cut off from the generic model. This makes just sense, if the specific models have similar concepts and provide relations that is shared among any specific model.

Valuable preparation work concerning the structure has been already done in 2.4. We start by comparing the concepts of SPICE and CMMI. Table 4.2 faces the concepts and their model counterparts. Within the process dimension CMMI has a hierarchical structure among Process, Specific Goal, Specific Practice and Typical Work Product. In SPICE concepts of Base Practice, Work Product and Process Outcomes are parallel elements. There are no relations that document a Base Practice accomplish a Process Outcome and a Work Product is output of a Base Practice.

In SPICE Capability Levels are characterized by Process Attributes. There is no counterpart grouping mechanism in CMMI. Generic Goals apply directly on Capability Levels and Specific Practices are mapped to Capability Levels for a given Process Area.

Within the process and capability dimension both models have similar concepts, that sometimes differ just in names. A couple of concepts are additional to SPICE and others to CMMI. However, relations among concepts are slightly different. CMMI forms a hierarchical process dimension, SPICE only to a certain degree. There could be a generic model, but it is limited, because CMMI or SPICE define some relationships that are not denoted in their model counterparts. Actually generic models make sense, if their specific models differ only in minor matters.

<table>
<thead>
<tr>
<th>ISO/IEC 15504-5 concepts</th>
<th>CMMI concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life-Cycle Process Category</td>
<td>-</td>
</tr>
<tr>
<td>Process Group</td>
<td>Process Category</td>
</tr>
<tr>
<td>Process</td>
<td>Process Area</td>
</tr>
<tr>
<td>Process Outcome</td>
<td>Specific Goal</td>
</tr>
<tr>
<td>Base Practice</td>
<td>Base Practice (subset of Specific Practice)</td>
</tr>
<tr>
<td>Work Product</td>
<td>Typical Work Product</td>
</tr>
<tr>
<td>Work Product Characteristic</td>
<td>-</td>
</tr>
<tr>
<td>Capability Level</td>
<td>Capability Level</td>
</tr>
<tr>
<td>Process Attribute</td>
<td>-</td>
</tr>
<tr>
<td>Process Attribute Achievement</td>
<td>Generic Goal</td>
</tr>
<tr>
<td>Generic Practice</td>
<td>Generic Practice</td>
</tr>
<tr>
<td>Generic Work Product</td>
<td>-</td>
</tr>
<tr>
<td>Generic Work Product Characteristic</td>
<td>-</td>
</tr>
<tr>
<td>Generic Resource</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>Generic Practice Elaboration</td>
</tr>
</tbody>
</table>

Table 4.2: Concepts of ISO/IEC 15504-5 and their counterpart in CMMI
CHAPTER 4. PROPOSED SOLUTION

4.1.3 Integration Approach for a General Process Model Structure

The previous section has focused on a generic Process Model that is based upon a single database schema. Unfortunately, there does not exist a generic database schema that could be mapped into an ontology.

Another way to integrate Process Models is to find a generalization that preserves the concepts and relationships within a single data model. We call this approach Process Model Integration into a general Process Model.

Figure 4.2 depicts the solution. Each data model is integrated as kind of sub data model within a single data model. We include abstract entity types into the general data model that are termed to the domain name of its sub data model, and it is connected to each sub entity type. We include another abstract entity type that generalizes the previous added entity types. This entity type is denoted by a name concerning to the domain of the entire data model.

![Figure 4.2: Multiple data model mapping](image)

4.1.4 Mapping the General Process Model into an Ontology

Applying the method that we have described in the previous paragraph, we become able to transform any Process Model’s database schema into a single schema and finally into an ontology. This transformation is done without losing model related features.

We have concerned with a data model for ISO/IEC 15504-5 and continuous CMMI in section 2.3. We provide by example a mapping of these data models. The resulting class hierarchy for both models is depicted in the appendix in figure A.1.

Step 1: Renaming Entity Types and Relationships If entity types, relationships and attributes have equal names, they have to be renamed to get a unique identifier. For example SPICE and CMMI include entity type Capability Level or Base Practice. In order to preserve the names, we add prefix "SPICE-" to entity and relationship names of SPICE and to entity and relationship names of CMMI we add "CMMI-".

**Step 2: Entity Type to Class Mapping** Any entity-type within the data model becomes an ontology class. We additionally create classes for each Process Model that has been included: SPICE and CMMI. The scope of the resulting ontology is also represented as a unique class called Process-Model.

**Step 3: Arranging the classes within a class hierarchy** Classes of SPICE and CMMI become sub-classes of Process-Model. Process model specific classes specialize its corresponding general class and become therefore sub-class of them. For example, any class name that begins with "SPICE-" becomes sub-class to SPICE and so on. If there are more class sub-class relations, they become arranged within the class hierarchy. This applies for instance to CMMI-BasePractice or CMMI-AdvancedPractice which become sub-class to CMMI-SpecificPractice.

**Step 4: Relationship to ObjectProperty Mapping** Within the data model relationships are not denoted by a relationship name. Instead, role names are denoted at relationship ends, but not anywhere. Relationships are bi-directional, therefore any direction becomes an object property termed as follows: "Has-" + Name of the Class. For example, the relation from SPICE-ProcessGroup to SPICE-Process is termed to Has-SPICE-Process. The opposite direction is termed to Has-SPICE-ProcessGroup. Domain of the object property is the class from which the relation is read. The range is specified to the class to which the relation goes. Some relations have role names. For example, Class SPICE-Process is in relation to SPICE-ProcessAttribute by the role "Support Achievement Of". Then the object property is labeled to this role name.

**Step 5: Attribute to DatatypeProperty Mapping** Attributes from entities become datatype properties. Any datatype property address a string and therefore property range is always restricted to xsd:string. Equally named attributes are mapped to a unique datatype property.

**Step 6: Entity to Individual Mapping** Entities are mapped to individuals of its classes. Their resource identifier is named to the concept it belongs plus an individual identifier that is sometimes predefined in the Process Model documentation. SPICE uses identifier for its processes. For example, process "PIM.3 Process Improvement" is then termed to SPICE-Process-PIM.3. Other elements that have no identifier such as capability levels, work product, etc. get an unique identifier.

### 4.2 Ontology-Driven Knowledge Representation

It is not unusual and often suggested in literature to organize knowledge into ontologies or structuring them into knowledge trees. Ontologies provide the vocabulary of terms that we can talk about in a system or context.
Ontology-Development of SPI Knowledge

Defining ontologies for a certain domain of interest is mostly a pretty challenging task that needs for a good-quality ontology the ability to conceptualize. There is no correct way of developing ontologies. Noy and McGuinnes have worked out in [NM02] a tutorial that describes a possible methodology for developing ontologies by an iterative approach. The terminology they use for describing ontologies is according to frame-based systems, where their central modeling primitives are classes (or frames) with certain properties (or slots). These terms roughly corresponds to the terms used in the terminology of Web Ontology Language, we are going to use.

Noy and McGuiness suggest to proceed by the following iterative steps:

1. Determine the domain and scope
2. Consider reusing existing ontologies
3. Define important terms
4. Define classes and the class hierarchy
5. Define properties of classes
6. Create instances

**Step 1: Determine the domain and scope**

Defining the domain and the scope is started by asking the question: What is the domain the ontology will cover? For what we are going to use the ontology? And what are the questions an ontology provides answers? [NM02]

Our to developing ontology is supposed to cover the domain of knowledge for Software Process Improvement implementation. Knowledge should be classified according to their context they supply implementation help for high-level activities, general work products and other criteria defined in Process Models.

**Step 2: Consider reusing existing ontologies**

Many ontologies have been made available in electronic form. If we could find one, we could reuse the concepts and relationships. [NM02]

There exists no previous defined ontology for knowledge in context of Software Process Improvement.
Step 3: Define important terms

We enumerate important terms we would like to make statements about. What are the terms we want to talk about and what properties those terms have? [NM02]

We talk about Software Process Improvement knowledge. We shorten this term to SPI Knowledge. There are terms which address properties of knowledge items such as Purpose, Description, Author, Applicability, Problem, Examples, Solution, Risks, References and Links, Resource Requirements, Introduction and Related Knowledge.

SPI Knowledge is characterized by its SPI Knowledge Type or its SPI Knowledge Context. There are different kind of types, for instance Template, Theorem, Good Practices, Guidance, Information, Paradigm.

Terms addressing the SPI Knowledge Context are high-level terms which must be chosen concerning the domain knowledge provides benefit. For example Project Management, Risk Management, Software Testing, Innovation Management, Process Improvement and so on. We define the context exemplary according to the process definitions in ISO/IEC 12207.

The next two steps - developing the class hierarchy and defining classes properties - are closely intertwined. Typically, we create a few definitions of the concepts in the hierarchy and then continue by describing properties of these concepts and so on. [NM02]

Step 4 + 5: Define classes, properties and the class hierarchy

There are several approaches in developing class hierarchies. For example, by bottom-up and by top-down. Classification from bottom-up starts with the definition of the most specialized concept and subsequently to the most generalized concept in the domain. The top-down approach starts from the most general concept in the domain and builds up the hierarchy by specializing the concepts. It depends on which approach relate better to the personal preferences. There is also a combination of both approaches possible. [NM02]

Within the previous paragraph we have identified three different domains which become classes in the ontology: SPI-Knowledge, SPI-Knowledge-Context and SPI-Knowledge-Type.

Properties only apply to class SPI-Knowledge. A property that addresses a character string becomes a datatype property. A property that addresses values from another class becomes an object property. Object properties are these which describe the context, the type and related knowledge. They are denoted by Has-Knowledge-Context, Has-Knowledge-Type and Has-Related-Knowledge. Their range is determined concerning the class to which allowed values come from. For example, property Has-Knowledge-Context has class SPI-Knowledge-Context as range and property Has-Knowledge-Type has class SPI-Knowledge-Type as range. This fact is depicted in figure 4.3 between some individuals of these classes.
We start forming a class hierarchy for class SPI-Knowledge by top-down. We could provide knowledge classifying concerning the process categories to which they provide help. In the International Standard ISO/IEC 12207 processes are grouped into 8 process categories: Engineering, Improvement, Acquisition, Management, Operational, Supply, Reuse and Resource and Infrastructure. These process categories can be further classified by their sub-processes. For example, class Improvement are classified into sup-classes Process-Improvement, Process-Establishment and Process-Assessment.

Step 6: Create instances

The last step is creating individual instances of the classes in the hierarchy. [(NM02)]

We define possible values for the classes SPI-Knowledge-Context and SPI-Knowledge-Type which can be referenced by instances of class SPI-Knowledge. Individual instances for class SPI-Knowledge-Type are GoodPractice-TYPE, Theorem-TYPE, Guidance-TYPE, Information-TYPE, Paradigm-TYPE, Template-TYPE. As we have mentioned before, we define instances for class SPI-Knowledge-Context according to the process definitions in ISO/IEC 12207:

Individual instances, called knowledge items, for class SPI-Knowledge and its sub-classes are appended in the course of time. They are mapped to the single class to which they can be grouped. These instances cannot be member of two or more classes.

4.3 Knowledge Transition

Process Models consist of "model elements" in process dimension and capability dimension in the form of processes, process groups, capability levels, practices, resources, and model related elements such as process attributes, or generic goals and so on. Some of these model items define criteria, activities, resources or artifacts which are essential to perform a process or to achieve a certain level of process capability. In particular, we could provide these items by "practical elements" in the form of knowledge about how they are implemented. Both, the "model elements" and the "practical elements" we have stored within two separate ontologies. For the purpose of knowledge transition we need now a mechanism to connect both kinds of elements.

Our designed ontologies are described by Ontology Web Language. OWL defines the language construct owl:imports. This statement "references another OWL ontology containing definitions, whose meaning is considered to be part of the meaning of the importing ontology" \[B^{-}04\]. By importing the facts from SPI Knowledge ontology we could make them available for the Process Model ontology to make references among their individuals.

In the next section two different approaches are discussed. The first makes knowledge transition by means of a simple mapping, while the latter advises a new approach by means of virtual mapping.

4.3.1 Direct Knowledge Linking

In section 4.1.4 we have constituted a hierarchy of Process Models. Its concepts have been arranged within a superclass-subclass hierarchy. This means, instances of a sub-class are also instance of their superclass. See figure 4.4 for an illustration.

Within the illustration, any instance of class SPICE and CMMI is also instance of class Process-Model. If we now define a property that applies to the root class, then it becomes inherit by the sub-classes. In order to establish knowledge transition, a new object property called Has-SPI-Knowledge is applied to the root class Process-Model. The property class range is SPI-Knowledge of the imported ontology. Thus, any instance of the Process Model ontology can be connected with instances, or knowledge items, from the SPI Knowledge ontology. This approach is called "direct knowledge linking" because related items must be asserted and stated respectively. In figure 4.5 this fact is depicted.
4.3.2 Virtual Knowledge Linking

Consider the fact, the number of knowledge items continually increase during the time, we are constrained to connect knowledge items to appropriate model items. There could be many connecting operations at one time because a couple of ”model elements” could benefit from the experience a knowledge item provides. Instead of directly element mapping, we could supply additional information about appropriate knowledge to the Process Model ontology.

In section 4.2 we have created an ontology wherein knowledge has been properly classified into a taxonomy. Within each created class, similar knowledge items exist concerning to the context they provide implementation help. We could these class definitions use to specify appropriate knowledge to each ”model element”. This definition contains no information about related individuals, but its classes. Knowing the class name we can deduce the class members. This is why we call it ”virtual knowledge linking”. It is based upon restricting the range of property Has-SPI-Knowledge. This restriction provides a kind of meta-data for an intelligent agent to finally deduce appropriate knowledge items.

Before we can restrict the property range, the Process Model ontology and the SPI Knowledge ontology need to modified. For the first one we extend the ontology by classes for model items. For the latter one, we create additional classes wherein their members fulfill certain conditions.

Figure 4.6 illustrates the ”virtually knowledge linking” approach. Model and knowledge items are not directly - they are virtually connected.
Extending the Process Model Ontology

Restricting the property range on a class definition always affects every instance that has instances of the specified property. There is no possibility to restrict the range of values on every individual.

The obvious solution is, we add classes which represent the individuals in the ontology. These classes consist of a single individual. Restricting the property range of this class applies then to a single individual.

Before these classes can be appended, every instance is renamed to avoid naming conflicts. To its former resource name the string "-1" is appended. After every instance has been renamed and its "individual-class" has been created, they become members of them. In picture 4.7 there is a fragment of the extended ontology, wherein CMMI capability levels and generic goals have been defined as classes.

Extending the SPI Knowledge Ontology by Class Definitions

In section 4.2 we have created a simple ontology, called taxonomy. We have grouped knowledge items into classes by listing all the members. But individuals can be classified by a second way: by their properties. This means, things are members of classes by definition. This method specifies a criterion for inclusion. [Pas04]

A criterion for inclusion could be the knowledge context. Because knowledge items can have more than one context, we could specify classes which define its member by definition of a certain context. We say, we create a Defined Class. Technically speaking, such a defined class has a has Value-restriction on a property. Hence, an individual becomes a member of such a
Figure 4.7: Providing "Instance-classes" to enable virtual knowledge mapping

class whenever at least one of its property values is equal to the hasValue resource.

Concerning SPI Knowledge we create classes which restrict on property Has-Knowledge-Context by a certain context value. They are then arranged as sub-classes of class SPI-Knowledge. Individuals which are member of class SPI-Knowledge and which have a context of the specified value, are member of the defined class.

Suppose we wanted to describe a sub-class of class SPI-Knowledge called Software-Design with a hasValue-Restriction on property Has-Knowledge-Context to value Software-Design-CONTEXT. The condition we have specified is read as, "every knowledge item that has the specified value on this property, is member of the defined class and also member of its super-classes". In listing 4.7 a source-code fragment is showed for an owl:hasValue-Condition.

```xml
<owl:Class rdf:ID="Acquisition-Preparation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Acquisition"/>
  </rdfs:subClassOf>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="Has-Knowledge-Context"/>
      </owl:onProperty>
      <owl:hasValue>
        <SPI-Knowledge-Context rdf:ID="Acquisition-Preparation-CONTEXT"/>
      </owl:hasValue>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```
Relevant, not explicit given facts, could be only by means of *Logical Reasoning* inferred. Logical reasoning goes back to the research field of description logics (DL).

### Knowledge Range Restriction

Web Ontology Language allows a lot of options to create simple or more complex conditions for property restrictions. For example, we could restrict the range of a property to a single class or to a union or an intersection of classes to make highly customizable expressions. However, we limit the extent of expressions to relatively simple expressions which effect a single class and a union or an intersection of separate classes:

∀ Has-SPI-Knowledge (KNOWLEDGE-RESTRICTION)
∀ Has-SPI-Knowledge (KNOWLEDGE-RESTRICTION ⊔ KNOWLEDGE-RESTRICTION)
∀ Has-SPI-Knowledge (KNOWLEDGE-RESTRICTION ⊓ KNOWLEDGE-RESTRICTION)

An example is given in listing 4.2 wherein related knowledge items of SPICE Management Process come from a union of two separate knowledge classes.

```
<owl:Class rdf:ID="SPICE-Process-MAN.1">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#SPICE-Process"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="http://www.andre-frenzel.de/ont/spi-knowledge#Engineering-RESTRICTION="/>
            <rdf:Description rdf:about="http://www.andre-frenzel.de/ont/spi-knowledge#Improvement-RESTRICTION="/>
          </owl:Class>
        </owl:unionOf>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 4.2: Example of an union restriction class

This example is an XML source-code snippet wherein OWL, RDFS and RDF statements are serialized. In line 1 resource *SPICE-Process-MAN.1* is denoted as ontology class. It is arranged as sub-class to *SPICE-Process* in line 2-4 and to an anonymous class that is defined
in line 5-19. A new class is defined by the anonymous class wherein any individual is member of this class, if it satisfy the condition defined as restriction (line 6-18) on property Has-SPI-Knowledge (line 7-9). The condition is defined by an universal quantifier (line 10-17). In line 12 to 15 applies a union operator for classes Management-RESTRICTION and Improvement-RESTRICTION that are defined in namespace http://www.andre-frenzel.de/ont/spi-knowledge#.

4.4 Meta-Data for the Purpose of Process Model Browsing

Process Models consist of a multiplicity of concepts and plenty of model elements. A proper Process Model browsing structure is therefore the most important matter for a supporting software tool. To achieve a good browsing structure we need rules which define a feasible way to browse through Process Models.

Web Ontology Language provides a feature that helps us to make statement about classes, e.g. to exclude classes or relationships from a tree navigation. The feature is called annotation to provide explanatory notes, like comments, which are allowed to classes, properties and individuals. They need to be explicitly declared by property owl:AnnotationProperty. Such annotation property could be processed as a kind of metadata statement by an intelligent agent that is, for example, in charge of creating the navigation structure.

In the following use of annotation is described for some cases.

Including/Excluding Resources for a Tree Browsing Structure Fortunately, the ontology of Process Model exists of a hierarchical structure upon a tree navigation structure could be built. However, any concept and relation should not be appear within the tree structure. Using annotation property AllowAsNode we become enable to state about classes that they should appear within a tree navigation structure. Allowing or disallowing implies a boolean datatype.

We could reuse this annotation property to state about object properties. Object properties represent relations from and to entities that could be also included/excluded from a tree navigation. It is up to the ontology maintainer if he decides to include a relationship or not. For example, object property Has-SPICE-ProcessGroup should not be appear in a tree, however, the opposite site could.

Labeling Tree Model Items Classes and relationships are denoted by its resource name. However this resource name is just a unique identifier within the ontology vocabulary. For presentation purpose we need extra information, for instance, a well-defined name or a label that appears in navigation tree instead of the resource identifier. We could use the annotation property rdfs:label for this purpose that resists in RDF Schema namespace.

Avoiding Infinite Loops The Process Model structure is sometimes not hierarchical. There might be cross-references between classes. This could be cause infinite loops during the time the tree data model is being built.
For example, class SPICE-ProcessAttribute and SPICE-Process exist in different dimensions, the first is in the capability dimension, the latter is in the process dimension. Both classes have relationships to each other. There could be cross-references in deeper classes or even in the same class that cause trouble when a tree model is being constructed. There could be two possible solutions:

1. Both relations are excluded from the tree by using property `AllowAsNode`
2. The resource at the end of the relation becomes a leaf in the tree. If it is selected, it is switched to the "original" item existing within the other dimension. Such relations could be denoted by a annotation property called `SwitchToItem`. 
Chapter 5

Implementation

This chapter introduces the thesis outcome in the form of a prototypical software tool that implements the elaborated ontology of a generic Process Model and it enables knowledge transition by using "virtually knowledge linking".

In the following the programming and deployment environment will be shortly described, before a detailed description of the software architecture is given, which is driven by a Model-View-Controller paradigm.

Frameworks that notably contribute to the implementation will be briefly explained and their impact on the implementation will be shown. Wherever applicable open-source software becomes first draw to commercial products.

This thesis finally has produced an outcome of software that has been termed to SPI Browser. This name denote the primary intension for being a software tool that supports Software Process Improvement.

5.1 Programming and Runtime Environment

Considerable the best programming language available for implementation purpose is sometimes up to the individual preferences. The market is full of professional programming languages. But for the purpose of ontology processing it is up to the 3rd party software which Application Programming Interfaces (API) is offered in which language.

Two favorite open-source ontology processing frameworks are presently on market. One of the most popular is JENA Ontology Framework. It has been taken under detailed consideration in section 5.3. Its enormously functional range is offered through a Java API.

For the second criterion, the programming language should be able to offer good practices for Web-Development. Web-Development means, making application available for the Web. SUN Microsystems is offering a key technology for enabling "rapid development of web-based applications” naming to JavaServer Pages (JSP)\textsuperscript{1}

\textsuperscript{1}JavaServer Pages Technology is available at http://java.sun.com/products/jsp/
JavaServer Pages

JavaServer Pages are part of Java 2 Enterprise Edition (J2EE) that is based on the Java 2 Standard Edition (J2SE) which is primary for local desktop applications. J2EE is additionally enhanced by server-side components for business applications.

JavaServer Pages is basically an extension to Java Servlets. Servlets are mainly designed to create dynamic Web pages. A JSP page is a text-based document that contains two types of text: static template data, which can be expressed in any text-based format, such as HTML and XML; and JSP elements, which construct dynamic content.

Apache Tomcat

Java Servlets life-cycle is managed by a Servlet Container. Servlet container that additionally support JSP technology are called Web-Container. Such an application is responsible for the entire servlet life-cycle: creating, loading, initialization, destroying and handling client requests for servlets.

Apache Tomcat\(^2\) is the Web-Container that is used in the official reference implementation for the Java Servlet and JavaServer Pages technologies. It has been developing within the Jakarta sub-project of the Apache Software Foundation. Tomcat has been published under open-source ever since and it has become the most popular Web container in the open-source scene. The current stable development tree is 5.0.x available for any operating system where a Java Virtual Machine (JVM) greater version 1.4 is available.

The prototype of the SPI Browser is deployed under following address: \texttt{http://www.andre-frenzel.de:8080/SPIBrowser}.

Required Software Packages

For tracking the progressive of the prototypical implementation a non-production server has been set-up. The operating system the server is running on is Linux under Debian 3.1 Sarge distribution. The Software packages are almost on the latest version. A list of packages is quoted in table 5.2.

5.2 Model-View-Controller Web-Architecture

One of the ever mentioned design goals of particular web application is to separate business logic, presentation logic and controller logic. This paradigm has become famous as Model-View-Controller-Architecture, shortly MVC-Architecture. The purpose of MVC is to provide flexibility of the software architecture and re-use of software components.

MVC is a design pattern for interactive applications that is organized into three separate modules: the application model for data representation and business logic, the second for

\(^2\)Apache Jakarta Tomcat is available at \texttt{http://jakarta.apache.org/tomcat/}
views responsible for user input and presentation layer and the third for controller to dispatch user requests. [SUN02]

Figure 5.1: Web-tier controller

A Web application architecture that uses MVC to organize the Web application design is named to Web-tier architecture. The architecture’s center is formed by the Web-tier controller depicted in figure 5.1. It receives incoming HTTP request and invokes the requested business logic method. Based on the results of the operation and state of the application model, the controller then selects the next view to display. Finally the controller generates the selected view and transmits it to the client for presentation.

5.2.1 Struts Web-Framework

One of the most popular and powerful open-source Web-Framework at the time is Struts. It is part of the Jakarta project hosted by Apache.

Struts Framework provides the application architecture based on Model 2 approach, a variation of MVC design paradigm, where the request processing is performed in Java Servlets rather than in JSP pages (JSP Model 1). [Ses99]

The JSP Model 2 architecture, shown in figure 5.2, combines the use and advantages of Java Servlets and JSP. Servlets perform as the controller for process-intensive task while JSP is for generating the presentation layer. [Ses99]

---

3 Apache Struts is available at [http://struts.apache.org/](http://struts.apache.org/)

---

<table>
<thead>
<tr>
<th>Software Package</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Tomcat</td>
<td>5.0.28</td>
<td>Servlet and JSP Container</td>
</tr>
<tr>
<td>Java Virtual Machine</td>
<td>1.4.2-06</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>Struts Framework</td>
<td>1.2.4</td>
<td>Web-Framework based on MVC-Architecture</td>
</tr>
<tr>
<td>Jena Framework</td>
<td>2.2 beta</td>
<td>Framework for RDF, RDFS, OWL processing</td>
</tr>
<tr>
<td>CommonControls</td>
<td>1.2.075</td>
<td>Control elements e.g. for trees</td>
</tr>
</tbody>
</table>

Table 5.2: Required software packages for the runtime environment
Struts Model, View and Controller Components

The Struts information flow illustrated in picture 5.3 is very similar to plain Model 2 applications. Any incoming request (1) is dispatched (2) through a single Servlet Controller. Depending on the user request the controller creates responsible Action classes. Action classes are built by the developer to perform the work of the application. Within action classes the application Model is changed either by updating (3) the internal state or by invoking JavaBeans to perform the actual business logic. Actions focuses rather on error handling and control flow, but not on performing the business logic.

As result of the Business Logic operation the attained Java Bean is stored in the application request or session context. The control is then forwarded (4) by controller to the view. The corresponding JSP is now able to fetch the JavaBean from the application context, to create the output page and to transmit (5) it to the client.

Central part of the Struts framework is the Struts deployment descriptor `struts-config.xml`. All configuration work for Struts-based applications is done here. The Struts configuration consists of several parts that set-up Model components (FormBeans), Controller behavior (Actions), plugins as well as data sources and message resources.
5.2.2 Struts Configuration for the SPI Browser

Relevant Struts components can be best described on the SPI Browsers architecture itself. Additionally, it is given some further explanation about significant components.

Form Beans

FormBeans are derived from class *ActionForm* which can be considered both as Model and as Controller. Input of your application such as properties of input forms or request parameters in hypertext links are mapped to the properties of the corresponding FormBean. They are finally forwarded by the controller to the scope of a responsible Action class.

- **itemRequestForm** - formBean that consists of attributes for current user resource request
- **treeForm** - consists of the request parameters when invoking a tree operation
- **searchForm** - stores the parameters when user invokes the search operation

Global Exceptions

Within *Global Exception* configuration section exception handler are defined that treat occurring exceptions thrown by Action classes.

- **java.lang.Exception** - defines a global exception handler for the most general exception that is not catched within an Action class

Global Forwards

*Global Forwards* are instances of *Action Forwards* which map logical names to specific resources. Target could be either JSPs or another logical name that is processed by an Action. *Global* means that the flow from any Action could be forwarded a Global Forward. For example to the home site or to a login site when the session has been timed out.

- **Page.Start** - Global Forward to the application start site. It is sometimes used when session is timed out and the application flow needs to be forwarded to start.

Action Mappings

Within this section the URI is mapped to an appropriate Action class. It is a good design rule to never access JSPs directly from a URI but from Action classes. The reason is either to let application logic and business logic processed by Action (MVC paradigm) and to globally access resources by the unique URI without changing any reference if the JSP will have been renamed.
CHAPTER 5. IMPLEMENTATION

- /Page.Start - forwards the flow to the application initialization action (/Action.Start.init)

- /Page.Search - forwards the request for the Search page to an Action (Action.Search.init) that initializes the user search

- /Page.Browse - forwards the flow for the Browse page request to an Action (/Action.Browse.init)

- /Page.History - forwards to the action (/Action.History.init) that initializes the History page

- /Action.Start.init - Action that initialize the application scope such as treeControl bean and RDF data model bean. If succeeds, the request is forwarded to the Browse page (/Page.Browse).

- /Action.Search.init - initialize the search options that possible to the user. Forwards finally to action /Page.Search.display

- /Action.Browse.init - checks if treeControl is in session and forwards to the action that points to the JSP.

- /Action.History.init - forwards to a action (/Page.Search.display) pointing at the History page

- /Action.Item.dispatch - dispatches the user request for a resource that could be either a Model Item or Meta-Model Item and forwards finally to the proper action for processing this item.

- /Action.ModelItem.handle - fetches a Javabean from the Model that contains the informations to the requested Model Item.

- /Action.MetaModelItem.handle - currently not implemented

- /Action.KnowledgeItem.handle - retrieves an Knowledge Item from the Model and put it as JavaBean into the session scope

- /Action.Tree.browse - dispatches the request for treeControl. The state of the treeControl is managed by package common-controls.

- /Page.Browse.display - forwards to Browse.jsp

- /Page.History.display - forwards to History.jsp

- /Page.Error.display - forwards to Error.jsp

- /Page.Search.display - forwards to Search.jsp

SPI Browser Workflow

The Struts workflow for the SPI Browser can be visualized within a diagram. The entire workflow is illustrated within appendix in figure A.2.
5.3 Role of Jena Ontology Framework

Jena\(^4\) is a framework for building Semantic Web applications which business model is described in RDF or in ontologies by RDFS or OWL language. The creator of Jena Semantic Web Framework has been Hewlett Packard’s Semantic Web Team. It has released Jena as open-source for the community. Jena’s API architecture focuses on the RDF documents. RDF documents can be considered on three different levels of abstraction [DFvH03]:

- at the syntactic level ⇒ XML document
- at the structure level ⇒ RDF data model
- at the semantic level ⇒ graphs with predefined semantics (Ontologies)

Jena Framework enables to query about each of the abstraction levels. In figure 5.4 a possible data flow within the Jena Ontology Framework is depicted.

![Figure 5.4: Model data flow](image)

Business Logic

The entire business logic is encapsulated within class \textit{RDFModelLoader}. Its structure is illustrated in figure 5.5. This class is responsible among other things for generating the Process Model tree navigation, querying the ontology model against a search request, achieving knowledge transition, and so on.

The logic makes use of Jena’s ontology processing capabilities to traverse the ontology files at the structural level. First of all, each ontology file is set up as instance of class \textit{OntModel}, but

\footnote{Jena’s Web-site is at \url{http://jena.sourceforge.net}}
Figure 5.5: SPI Browser class containing the business logic with different OntModelSpec. While the ontology of SPI Knowledge is enabled for entailment reasoning, the Process Model ontology is not. This is caused by the fact, knowledge items are not asserted in sub-classes. In figure 5.6 the internal model structure of Jena is depicted.

![Jena's internal model structure in the SPI Browser](image)

Figure 5.6: Jena’s internal model structure in the SPI Browser

Only by means of reasoning not stated knowledge items can be inferred. There exists three rea-
soner options: OWL Micro reasoner that comes with Jena, Pellet OWL Reasoner\(^5\) or RACER\(^6\). OWL Micro is the default reasoner.

5.4 Role of Common-Controls Framework

Common-Controls\(^7\) is a commercial presentation framework optimized for Struts that consists of several control elements like lists, trees, tabfolders, menus and forms.

One of these controls, namely the treeControl-component, provides a tree navigation structure. It generates a tree which nodes can be exploded and closed. The programmer merely provides a tree data model through a simple interface. In the JSP-page treeControl configuration takes place within a corresponding Java Taglib. The state of tree is managed by the treeControl-component itself.

Building the Tree Model

The tree data model is defined by implementing the common-controls tree interfaces: TreeNodeDataModel and TreeGroupDataModel. However, it is sufficient to implement interface TreeGroupDataModel.

The data to build up the tree is retrieved from the ontology in terms of classes, properties and individuals. The tree data model is constructed starting from class Process-Model. subclasses are mapped to folders in the tree to the point they include instances (⇒ Class Level). Those classes which haven not been annotated by property AllowsAsNode are left out from the mapping and are not further processed. Classes which have an individual instance, their corresponding tree node becomes a tree item (⇒ Item Level). This item is further considered concerning its properties which are explicit allowed to be displayed in the tree (⇒ Property Level). In figure 5.7 a sample tree is depicted wherein the colored nodes indicate the different processing levels.

5.5 Ontology Managing using Protégé Ontology Editor

There is no user interface in the prototype yet that is for maintaining the data. Editing data on the structural level (RDF level) is also not pretty. Fortunately there exist applications that provide an integrated view on the data and a smooth integration of ontology files. Such an application is Protégé. Protégé\(^8\) is an Java-based ontology and a knowledge base editor that makes it possible to modify ontologies. Is has been released as open-source and is therefore free of use.

Protégé has a quite open architecture. The functionality can be easily extended by appropriate


\(^6\) RACER is a Semantic Web inference engine for query answering over RDF documents and OWL ontologies. It is available at [http://www.cs.concordia.ca/~haarslev/racer/](http://www.cs.concordia.ca/~haarslev/racer/)

\(^7\) Common Controls is available at [http://www.common-controls.com](http://www.common-controls.com)

\(^8\) Protégé Ontology Editor is available at [http://protege.stanford.edu/](http://protege.stanford.edu/)
plugins. Such a plugin is Protégé-OWL which comes innately to the Protégé environment. This plugin extends Protégé in order to support the Web Ontology Language (OWL).

Within the scope of the proposed solution, Protégé and OWL Plugin act the role of creating and managing ontologies. Protégé-OWL offers separate views for the most common components (classes, properties, individuals) in which they can be easily managed. In figure 5.8 is a screenshot of Protégé showing the class view.

5.5.1 Ontology Infrastructure

Several options are available for providing a proper ontology infrastructure. Protégé is enabled to read ontologies on three ways: locally file reading, remote file reading and database reading. This options are shortly below.

Local Access This is standard option and also current practice within the prototype environment. Ontology files are locally read, modified and stored. Files need then propagated to a server by a remote file protocol using FTP or SSH. This approach allows ontology maintainer to put these files under version control and to locally test the integrity of data.
Remote Access Another option is to update the ontology on a remote server. Protégé offers an option to open a remote project and to update ontology files over network. This option has not been proved if it is working properly.

Anyway this option has a couple of drawbacks:

- If more than one people are updating the ontology at the same time, racing conditions could appear
- It is difficult to make an update of the remote ontology. It is even rather difficult to provide version control for these files

Database Access Last but not least ontologies can be serialized into databases and accessed by Protégé and by Jena Framework using API connectors to the most popular open-source databases such MySQL or PostgreSQL.

Databases have a lot of advantages:

- Restrict or allow user to have access on data
- Allow concurrent access
- Backup of database is supported
- Optimized data storing and querying

5.5.2 Editing Scenarios

Ontology maintainer should be supported by changing, adding or dropping of information that are stored within an ontology. These tasks are further referred to as ontology editing.

In the following sections we give a briefly explanation on Protégé-OWL, how ontology editing could be managed concerning the following exposed use cases:

- Define virtual knowledge transition
- Define knowledge restricted classes

Hereby we refer to the terminology used in Protégé-OWL that is unlike to the terminology of OWL specification. More on Protégé is given in [Hor04] that provides a good tutorial.

Define Virtual Knowledge Transition

As suggested in the solution part of this thesis, transition of knowledge is achieved by property restrictions. In Protégé-OWL we add restrictions either as NECESSARY CONDITION or NECESSARY & SUFFICIENT CONDITION a class. The first one can be read as, "if someone is a member of this class, then it is necessary to fulfill these conditions". Necessary & sufficient condition means, "the conditions are necessary for membership, they are also sufficient to determine that any individual that satisfies them must be a member of these class. [Hor04]

In order to define knowledge transition we make a necessary condition on property Has-SPI-Knowledge. The condition is of type hasAllValuesFrom. This means, every instance of this class has on property all values from the specified condition. This condition specifies either a class, then it is a simple expression, or an operation on more classes, then it is a complex expression. In picture 5.9 there is complex condition about a union of two knowledge restricted classes.

Define Knowledge Restricted Classes

Knowledge restricted classes are defined concerning their context. Such classes define a certain set of knowledge items that satisfy the given condition. Hereby inclusion depends on a necessary & sufficient condition. The restriction we create is a hasValue condition on property Has-Knowledge-Context. Only these knowledge items are member of a restricted class which have at least a value from the specified knowledge context. For an example see picture 5.10.

5.6 Technical Issues

Distinct Identifier for TreeControl Items The feature of switching branches when items are selected that exist within another dimension is considered to be necessary for the SPI
When switching between branches it is necessary to know the key of the selected items. Unfortunately a tree item identifier is calculated during the tree is being built. A key is not a plain identifier, it is structured in parts that are calculated during the tree is processed from the top across the tree structure. Any element within the tree becomes an identifier part that is separated by colon. This ensures making unique tree identifiers.

Following question arises: Why resource identifier cannot be used as a tree item identifier? If they can, selecting a certain tree item would lead to open more then one branch. This is because some items exist in several branches. However, tree item identifier cannot be recalculated from the ontology yet. By providing meta-data they could be determined.

**Performance vs. Reasoning** Performance of Semantic Web applications heavily depends on entailment reasoning. Reasoning demand a lot of processor and memory resources. For complex ontologies the process of computing not asserted facts could take unreasonable time. The process of reasoning is thus enabled for the sub-model that is tied to SPI Knowledge. Allowing reasoning for the entire model could greatly enhance the speed for reasoning knowledge items. However, some performance tests showed when reasoning is enabled for the entire model the speed was low. The time to calculate the union graph took sometime longer than a minute.
5.7 Future Development

The SPI Browser is in a prototypical phase. This means it is still under development even though the core functionality has been integrated into the browser. Following listed features could be implemented into a future version.

Switching the Perspective  The structure of Process Model is driven by two perspectives: capability dimension and process dimension. They represent different views within the browser. While some concepts appear just in a single dimension, for instance Base Practice, others appear in both dimensions. For example concept of Process has an interrelationship to concept of Process Attribute, that appears in the opposite dimension. These concepts could be properly denoted by property to change switch perspective if the user chooses such an item.

Intersection of Knowledge for Knowledge Transition  Currently only the union-operator is supported for knowledge transition. It could be sometimes useful, but mostly not needed, to provide intersection of SPI Knowledge classes. This means, knowledge items that exists in any intersected class become virtually connected to the model item.

Providing Context-Sensitive Search Options  This could be a short cut for the browsing feature. Different kind of restrictions for model and knowledge items can be chosen by user. For example, checkboxes could provide different types of knowledge. Model concepts could become select criterion for searching. Corresponding items could be context based displayed within comboboxes.
Comments on Knowledge Items  Comments could user encourage to provide their experience to actual improvement cases. User might give response on knowledge items such as critics, general comments, feasibility, realization, and so on. Comments could be considered as a separate concept within the ontology of SPI Knowledge.

Separate Namespace for Browsing Annotation Properties  The concept of annotation within OWL is primary used for presentation purposes e.g. labeling, building the tree structure, etc. (see 4.4).

Currently these annotation properties exist within the namespace for Process Models. However, their purpose is for browsing. In a future version these properties could be arranged within a separate namespace.
Chapter 6

Conclusion

This thesis has introduced an architecture that effectively supports Software Process Improvement after Process Assessment. This architecture is based upon ontologies in which several Process Models are integrated and knowledge for Software Process Improvement is represented. By languages intended for the Semantic Web these ontologies are implemented within a software tool related to Software Process Improvement. We will now summarize the results and the contributions and we will give an outlook on the future research.

Summary

In this thesis we have pointed out that a generic Process Model is difficult to achieve and it comes along with loss of information. Instead, several Process Models can be represented within a single data model, then all the model related features are preserved. The data model forms a hierarchical structure that can be transferred by mapping rules into an ontology. The attained ontology works hereby as a meta-model that allows to make statements about the specific concepts of Process Models. Upon this ontology a hierarchical browsing structure can be constructed. Browsing related features can be specified by applying annotation upon ontology classes, properties and individuals.

Knowledge for Software Process Improvement can be represented and classified within an ontology, too. Since Web Ontology Language provides an import statement to include other ontologies, we use this feature to attain knowledge transition between elements from different ontologies. The way how knowledge transition takes place is up to two options. By means of "direct knowledge linking", knowledge items are connected to Process Model items one to one. But knowledge is continually appended. Therefore, we advise to consider the option "virtual knowledge linking". This means, knowledge and model items are not physically mapped. It is based upon restricting the range of related knowledge. Such restrictions are meta-data that could be extracted by a software agent to obtain related knowledge items.

We have presented a methodology how ontology facets can be maintained by an open-source ontology editor, called Protégé. It produces ontologies serialized into XML-RDF files which can be read and processed by Semantic Web frameworks.

The achievement of this thesis is a browser for Process Models, termed to SPI Browser.
It is implemented as J2EE Web-Application that is driven by Apache Struts Model-View-Controller Architecture. Its business logic reads the serialized ontology files, and builds upon meta-data a properly browsing structure. The intended use of the application is to browse through Process Models, in order to obtain the requirements and criteria for processes and to obtain knowledge for implementation.

**Critique**

Process Model browsing is based upon the structure of its data model. This structure is the determining factor for a proper navigation through the models. If these models have a hierarchical structure, browsing could be well supported. However, if in Process Models a semantic exists it could not be represented within the data model. For example, CMMI capability levels work as a kind of a selection mechanism for generic and specific practices, those are grouped into goals, for a certain process area. A possible navigation would be from capability level to process area to goal to practices. But practices are functional dependent on capability levels and process area and not on goals, such a hierarchical structure is difficult, if at all, to attain.

Virtual knowledge linking is based upon a well designed knowledge ontology. Their classes are then referenced to Process Model classes. Thus, changing the ontology class structure should not be done afterwards. This means, any facets of knowledge must be considered during ontology designing in respect to the level it should provide implementation help. However, knowledge is multifarious and therefore good classification is hardly to achieve.

**Outlook**

The integration approach that several data model merge in a single data model can be used as a general basis for integrating arbitrary models. There could be simple models possible which represent general goals in Software Process Improvement programs. For example, goals like "Better Productivity" or "Better Team Collaboration". These elements could be supported by knowledge as well as by practices from Process Models.

If "virtual knowledge linking" methodology is chosen, it is essential to construct an appropriate ontology for knowledge that is based on a "good" classification structure. Such classification structure must be granular enough to supply knowledge classes that can be mapped, for example, to base practices and work products, if there is a need for it. In another thesis there could be dealt with this question.

Further research is needed if continuous and staged CMMI should be integrated within a software tool. Because they address two different views on the same content, they could be implemented within a single data model.
Appendix A

Diagrams

Figure A.1: Process Model ontology for use of direct knowledge linking
Figure A.2: Struts workflow for the SPI Browser
Bibliography


Ontology (computer science), Feb 2005. URL: http://en.wikipedia.org/wiki/Ontology_(computer_science)

Process model, Apr 2005. URL: http://lexikon.freenet.de/Prozessmodell

Statutory Declaration

I declare that I have developed and written the enclosed Diplomarbeit completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Diplomarbeit was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Location, Date  
André Frenzel