

Roadmap for Tool Support for Collaborative Ontology Engineering

by

Yiling Lu

B.Sc., XiAn Transportation University, 1994

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

Dr. Margaret-Anne Storey, Supervisor (Department of Computer Science)

Dr. Daniela Damian, Departmental Member (Department of Computer Science)

Dr. Daniel M. Germán, Departmental Member (Department of Computer Science)

Dr. Eleni Stroulia, External Examiner (Department of Computing Science, University of Alberta)

© Yiling Lu, 2003
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopy or other means, without permission of the author.

Supervisor: Dr. Margaret-Anne Storey

Abstract

An ontology is an explicit formal specification of terms and relations between terms in a domain, and enables sharing and reuse of knowledge. More and more ontologies are being developed, including examples such as WebOnto for the semantic web and ontologies used to categorize products and services on large web sites (as ebay.com). Enabling and facilitating collaborative ontology development – in order to take advantage of distributed computing power and intellectual resources – is becoming a concern for researchers in the ontology development field. Enhancing the efficiency of collaborative work and reducing the time for ontology development will bring economic benefits.

This thesis reviews five existing ontology development tools and compares their strengths and weaknesses in supporting collaborative work. Our research investigates issues that arise when people work collaboratively on ontologies, review several groupware technologies, and discusses their potential to be used in supporting collaborative ontology engineering. In addition, we investigate a lightweight mechanism to support collaboration in a knowledge engineering tool known as Jambalaya.

Collectively this work constitutes a roadmap for tool designers creating or integrating collaboration support into an ontology engineering environment. From this road map, we conclude by providing a set of recommendations, for Protégé 2000, an established ontology editing tool, for adding and improving its collaboration support features in the future.

Examiners:

Dr. Margaret-Anne Storey, Supervisor (Department of Computer Science)

Dr. Daniela Damian, Departmental Member (Department of Computer Science)

Dr. Daniel M. Germán, Departmental Member (Department of Computer Science)

Dr. Eleni Stroulia, External Examiner (Department of Computing Science, University of Alberta)

Table of Contents

ABSTRACT.....	II
TABLE OF CONTENTS	IV
LIST OF TABLES	VII
LIST OF FIGURES	VIII
ACKNOWLEDGEMENTS	X
CHAPTER 1 INTRODUCTION.....	1
1.1 Outline of Thesis	2
CHAPTER 2 ONTOLOGIES AND ONTOLOGY ENGINEERING	5
2.1 What is an Ontology?	5
2.1.1 Use of Ontologies.....	7
2.1.2 Examples of Ontologies	9
2.2 Ontology Engineering	11
2.2.3 Different Ontology Engineering Approaches	12
2.2.4 Life Cycle Model of an Ontology	16
2.3 Summary	20
CHAPTER 3 COMPUTER SUPPORTED COLLABORATIVE WORK	21
3.1 Overview of Groupware.....	22
CHAPTER 4 SURVEY OF CSCW SUPPORT FOR ONTOLOGY	
DEVELOPMENT TOOLS	26
4.1 Ontolingua Server	26
4.2 OntoEdit	29
4.3 APECKS	31
4.4 CO ₄ System and CO ₄ Protocol.....	33

4.4.1	The CO ₄ Knowledge Base Network.....	35
4.5	Protégé-2000	38
4.6	A Comparison of Ontology Editing Tools	40
4.7	Summary	43
CHAPTER 5 DIMENSIONS OF THE PROBLEMS IN COLLABORATIVE ONTOLOGY DEVELOPMENT.....		45
5.1	Distance and Communication	46
5.2	Documentation and Knowledge Management	51
5.3	Version Control and Change Tracking	52
5.4	Summary	55
CHAPTER 6 USING GROUPWARE FOR ONTOLOGY ENGINEERING		56
6.1	Instant Messaging	56
6.2	Web Portals	58
6.3	Peer-to-Peer Networks	62
6.3.1	Taxonomy of Computer Systems.....	62
6.3.2	What are Peer-to-Peer Networks?.....	63
6.3.3	Historical View	65
6.3.4	Different Peer-to-Peer Systems	66
6.3.5	Challenges to Peer-to-Peer Technology.....	68
6.3.6	JXTA	70
6.4	Groove Workspace.....	72
CHAPTER 7 COLLABORATIVE SUPPORT FOR PROTÉGÉ-2000		75
7.1	Protégé-2000 and JXTA.....	77
7.2	Capturing Group Knowledge	80
7.3	PromptViz	86
7.4	Live Bookmarks	89
7.4.1	Advanced Visualization and Navigation Engine for Protégé-2000	90
7.4.2	Live Bookmarks in Detail	91
7.4.3	Live Bookmark Specification	95

7.5	Summary	96
CHAPTER 8	CONCLUSION.....	98
8.1	Summary of Contributions.....	99
8.2	Future Work	100
8.3	Concluding Remarks.....	101
BIBLIOGRAPHY		102
APPENDIX A		108

List of Tables

Table 1: Ontology representation with different formalisms.....	7
Table 2: Some well-known ontologies	11
Table 3: Approaches to ontology design [70]	12
Table 4: Classification of CSCW Systems	24
Table 5: Fine grained categorization of groupware	24
Table 6: Structure of the concept history table in the NCI Thesaurus TM	54
Table 7: Overview comparison of Bug/Issue tracking system requirements	61
Table 8: Table of ontologies developed using different methodologies. (Some methodologies are identified by the name of the designers.).....	108

List of Figures

Figure 1: Example of libraries of ontologies (modified from [55])	11
Figure 2: A collaborative approach to ontology design.....	15
Figure 3: Spiral model of software development [35]	19
Figure 4: Architecture of the Ontolingua Server ([53])	27
Figure 5: Client/Server architecture of OntoEdit ([102])	31
Figure 6: Hierarchical knowledge base network and message flow (dark arrows). Bases are organized in a tree whose leaves are individual bases and nodes [50].	36
Figure 7: The software architecture of CO ₄ system. Boxes represent a software module, Circled units are data/knowledge repositories and arrows represent the call of program functionality [50].....	37
Figure 8: Editing classes, slots, and instances with Protégé-2000.....	40
Figure 9: Work flow diagram for developing NCI Thesaurus TM	53
Figure 10: A KM architecture model shows how various parts of the system obtain, store, classify, and distribute knowledge	60
Figure 11:A Taxonomy of Computer Systems [85]	63
Figure 12 A classification of existing P2P systems [85]	67
Figure 13 The project JXTA virtual network [103]	71
Figure 14 JXTA software architecture [56]	72
Figure 15: A tentative idea for Protégé-2000 collaboration plug-in.....	78
Figure 16: myJXTA application in group chat mode	79
Figure 17: A diagram of the Shrimpbib architecture of integrated tools [28]	81
Figure 18: Discussion threads in Drupal forum	84
Figure 19: Screen capture for collaborative book writing in Drupal.....	85
Figure 20: Table view in Prompt lists all the merged concepts from two versions of the same ontology. There are about one thousand changes listed in this list	87
Figure 21: Treemap view in PromptViz	89
Figure 22: Treemap view in PrompViz, with all nodes open to leaf level	89
Figure 23: Ontology visualization in Jambalaya	91

Figure 24: Using Jambalaya working with the wine ontology; user is about to bookmark this view in the scene	94
Figure 25: The bookmark from Figure 24 is opened in a web browser. User has zoomed into part of the bookmark to see more details using the zooming capacity of the SVG graph.....	94

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr. Margaret-Anne Storey, for having me as her student, for granting guidance and mentorship and for all the support and encouragement she has been giving me throughout my graduate studies.

Thanks also go to Dr. Daniela Damian, Dr. Daniel M. Germán, and Dr. Eleni Stroulia for being my committee members and reviewing my research.

I would like to acknowledge the assistance provided to me by my fellow graduate students in the CHISEL lab for their feedback and for making all the research activities fun and exciting. Special thanks go to Neil Ernst, Liz Hargreaves and Victor Chong for proof reading and providing comments and advice in the final stage of writing. All the works presented in this thesis are not only from myself, but also include the diligent contributions from Rob Lintern, Neil Ernst, Polly Allen, and David Perrin. This thesis is another synergized result from the CHISEL research group at University of Victoria.

This work was funded by National Cancer Institute at United States.

To Debbie

Chapter 1 Introduction

Ontologies have moved beyond the domains of philosophy and library science, and are now the concern of knowledge engineering, knowledge representation, domain modeling, language engineering, database design, information retrieval and extraction, and knowledge management and organization [82]. As ontologies become increasingly common in a greater number of applications and as these applications become larger and longer-lived, more and more ontologies are developed in distributed environments by authors with disparate backgrounds [81].

This thesis examines the major ontology engineering methodologies being developed and currently practiced, and surveys five widely used ontology authoring tools. The result not only verifies the fact that collaborative ontology authoring is the inherent nature of ontology engineering, but also indicates that collaborative ontology development is not well supported by any of the existing ontology authoring tools or environments. This presents a new challenge for tool designers in finding and designing tools that can better support collaborative ontology development.

In searching of a solution to this, our research investigates a range of tools and technologies in the Computer Supported Collaborative Work (CSCW) domain in order to determine how they can be used to support collaborative ontology development.

Ontology engineering, from its infancy, has been closely related to software engineering in terms of its methodologies and life cycle models. This thesis examines some experiences in the Global Software Development (GSD) field in order to understand the correlation between distance and collaboration.

The focus of this work is not to completely cover the research area of collaborative ontology engineering, but rather to study how to help ontology developers coordinate their efforts across multiple, geographically distributed sites. The purpose of this undertaking is to understand the state-of-the-art in collaborative tool support for ontology engineering, and to explore the potential of some of the groupware technologies that have been used in the global software development domain to solve collaboration problems. The long term vision is to combine these two fields and create a collaborative ontology engineering environment that provides collaboration support in multiple dimensions. Ultimately this thesis aims to develop a roadmap for ontology tool designers, a road map that may lead them to produce a comprehensive collaborative ontology engineering environment.

1.1 Outline of Thesis

Starting from the second chapter, this thesis introduces the background and definitions of key concepts in the field of ontology engineering, with a discussion of several ontology engineering approaches and life cycle models. Chapter 3 briefly introduces the field of Computer Supported Collaborative Work (CSCW) and sets the stage for Chapter 6 where we further discuss the potentials and benefits of using CSCW technology to support collaborative ontology engineering.

Chapter 4 provides a detailed survey of CSCW support provided by existing ontology development tools. By investigating several ontology engineering tools developed and used in different domains, we gain valuable insights into the collaborative tasks these tools support poorly or not at all.

In Chapter 5 we explore the challenges to collaborative ontology engineering according to three dimensions: distance and communication, documentation and knowledge management, and version control and change tracking. We stand on the experiences and lessons learned in the global software development field, and look at collaborative ontology engineering from the perspectives that are represented by those dimensions. Global software engineering is a well-developed field, and problems such as the impact of distance and communication to the cost of software development has been well studied. On the other hand, collaborative ontology engineering is still in an early stage of research with formal ontology languages (such as the Web Ontology Language (OWL) and Resource Description Frame Work (RDF)) still evolving, and as such many of the existing ontology development tools were designed without taking future challenges, such as collaborative development, into account. Nevertheless, collaborative work across distance in software engineering and ontology engineering share many similar characteristics, as our research show. Therefore, we believe that solutions from software engineering can be borrowed and modified for use in collaborative ontology engineering.

In Chapter 6, we dive deeper into the groupware technologies we have examined, focusing on their potential to support collaborative ontology editing. Instant messaging techniques, when implemented properly, have the potential to support spontaneous and informal communication while web portals can be used to support tasks in the area of group knowledge management. We also explore the Peer-to-Peer (P2P) network technologies, which have been around for some time but are relatively undeveloped in the area of reliability and security. We also report an experiment where we evaluated the

possibility of adding collaborative support to a knowledge engineering tool based on a P2P network.

Our recommendations are laid out in Chapter 7, where we discuss the details of how collaborative support can be provided by the designers of the multi-user version of Protégé-2000, a software tool for creating and editing ontologies and knowledge bases. In particular, we explore how live bookmarks, a lightweight collaborative mechanism, can be used with Protégé-2000.

This thesis concludes in Chapter 8 with a summary of contributions and areas for future work.

Chapter 2 Ontologies and Ontology Engineering

In this chapter we examine the concept of ontologies and describe specific examples of ontologies used in scientific and engineering domains. Following this, our discussion moves into the area of ontology engineering, which succinctly, is about the processes and methodologies that are used in creating ontologies. We also discuss the life cycle model of an ontology, which covers all the activities from conceptualization to maintenance and ultimately retirement. The discussion of a life cycle model will help us understand the ontology development tools we present later in Chapter 4. Although not all the tools we discuss in Chapter 4 are built to strictly support a particular ontology life cycle model we discuss here, each one of the tools does support a particular ontology engineering methodology and ontology life cycle model, which is often a hybrid model that can find its roots from the distilled life cycle models presented in this chapter.

2.1 What is an Ontology?

The short answer to this question is that an ontology is the formal and explicit conceptualization of a specific domain; it defines all the concepts and relations among those concepts. The term ‘Ontology’ originates from the field of philosophy, where it refers to the science or study of being.

Even so, the rich meaning of ontology can not really be described by one sentence, and there has been much discussion over the exact definition for this term. A careful examination of some of the representative definitions will paint us a more complete picture of what ‘ontology’ means.

One of the best-known definitions is from Tom Gruber [57]:

“... An ontology is an explicit specification of a conceptualization. For Artificial Intelligence systems what exists is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationship among them, are reflected in the representational vocabulary with which a knowledge-base program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrains the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory”

In comparison, Neches' [91] believes: “An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary”, while Swartout [30] defines ontology as: “An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base.”

On the symbolic level, an ontology is the representation of a conceptual system via logical theory; it is the vocabulary used by a logical theory and also the meta-level specification of a logical theory [63]. At the knowledge level, we can look at an ontology as a formal, explicit specification of a shared conceptualization. The formalism makes an ontology machine understandable as the explicit specification encompasses the complete set of concepts, properties of concepts, relations, constraints, and axioms in the target domain, while the shared specification of conceptualization exemplifies an ontology as the abstract model of the world or a specific domain; it is the consensual knowledge of a community.

An important concept is ontology commitment. For a common ontology, it is the agreement to use a predefined vocabulary, developed through consensus from all users and stakeholders in a specific domain in a coherent and consistent manner [62]. It is also

a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology [57]. In the design and implementation of an ontology, we want to produce an ontology that requires a minimum amount of ontological commitments from its users. Since the primary purpose of creating an ontology is to enable knowledge sharing and reusing, a base level of ontological commitment sufficient must be ensured to support this purpose. An ontology should make as few claims as possible about the world being modeled, this allows the participating parties freedom to specialize and instantiate the ontology as necessary [57].

An ontology can be represented by languages with different degrees of formalism [106], as summarized in Table 1:

Table 1: Ontology representation with different formalisms

Degree of Formalism	Representation Language
Highly informal	Natural language
Semi-informal	Restricted and structured form of natural language
Semi-formal	Artificial and formally defined language
Rigorously formal	Language with formal semantics, theorems and proofs of such properties as soundness and completeness

The design of an ontology should be independent of the representation language; even though, the representation language is the technology that enables the exchanging, sharing and reusing of an ontology.

2.1.1 Use of Ontologies

In this section, we will briefly discuss how ontologies have been used in the Semantic Web and in software engineering field.

The concept of an ontology first gained wide application in the Artificial Intelligence (AI) domain. In AI, knowledge in a computer system is thought of as something that is explicitly represented and operated on by inference processes [39]. However, the reality is that all information systems live on their knowledge of their application domains and the model of the world.

The vision of the Semantic Web [34] is to add machine understandable semantics (meta-information) to the World Wide Web by using an ontology to define and organize this meta-information space. The Semantic Web aims to realize the integration of all the information sources on the World Wide Web, allowing the reuse of data across applications and to make intelligent Internet searching possible. The Semantic Web is an extension of the World Wide Web in which information is given well-defined meanings that better enable computers and people to work in cooperation [34]. The requirement to encode machine-interpretable information on the Web led to the development of a number of languages for representing this information [87]. Among these knowledge representation languages are Resource Description Framework (RDF), Web Ontology Language (OWL), DARPA Agent Markup Language (DAML), and Ontology Interface Layer (OIL). At this stage, there is no consensus on which language or set of languages should become the standard for the Semantic Web and so researchers and developers continue to experiment with existing languages and to develop new ones [87].

Object-oriented design of a software system depends on the developers' understanding of and commitment to the model of the relevant domain. The result of object-oriented analysis is actually a draft of the domain ontology [44]. Interfaces, classes (their attributes and procedures) and the hierarchical system that organizes them is a close

model to the application domain, and can often be reused for a different application program in the same or related domain.

One of the problems software engineering is now facing is the lack of concrete and consistent formal bases for making modeling decisions [32]. Advocates of the Unified Modeling Language (UML) are using UML as an ontology modeling language and some experimental works have been reported [43]. UML as a graphical language has its own problems as it requires a significant amount of ontology commitment instead of the minimal ontology commitments mentioned previously, and it does not have the formal semantics that an ontology modeling language requires [97]. We are witnessing the efforts and trends to make the research of domain modeling in both software engineering and ontology engineering field converge over time. As information systems model large knowledge domains, domain ontologies may become as important in general software systems as in many areas of AI [39].

Many other examples of ontologies use can be found in the domain of e-business with the aim to integrate heterogeneous business processes, in information-retrieval systems, digital libraries, and natural language processing.

2.1.2 Examples of Ontologies

Most of the researchers in the area of ontology development agree that two important goals of ontology research are to make ontologies sharable by developing common formalisms and tool, and to develop the content of ontologies [89]. Depending on the domain or the world an ontology models, the ontology can be put into three categories:

1. Knowledge representation ontologies (knowledge representation systems that embody ontological frameworks), such as the frame ontology that captures the representation primitives used in frame-based languages¹, or the Knowledge Interchange Format (KIF) is a computer-oriented language designed for the interchange of knowledge among disparate programs. It has declarative semantics (i.e. the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions), is logically comprehensive (i.e. it provides for the expression of arbitrary sentences in the first-order predicate calculus), and it provides for the definition of objects, functions, and relations [13].
2. Ontologies about general world knowledge (upper level ontologies) such as ontologies about time and space.
3. Domain specific ontologies, such as gene ontologies and cancer ontologies.

Figure 1 illustrates how ontologies can be built upon one another in sharing and reusing each other's knowledge. Table 2 lists some well known ontologies.

¹ A frame-based system has two parts: an ontology defining both the hierarchy of type definitions and the relationships between the types, and a collection of instances, or instantiations of those types. Any concept that is modeled by these types can have properties called slots. The slots can have values that are strings or numbers, or even other types as defined in the ontology.

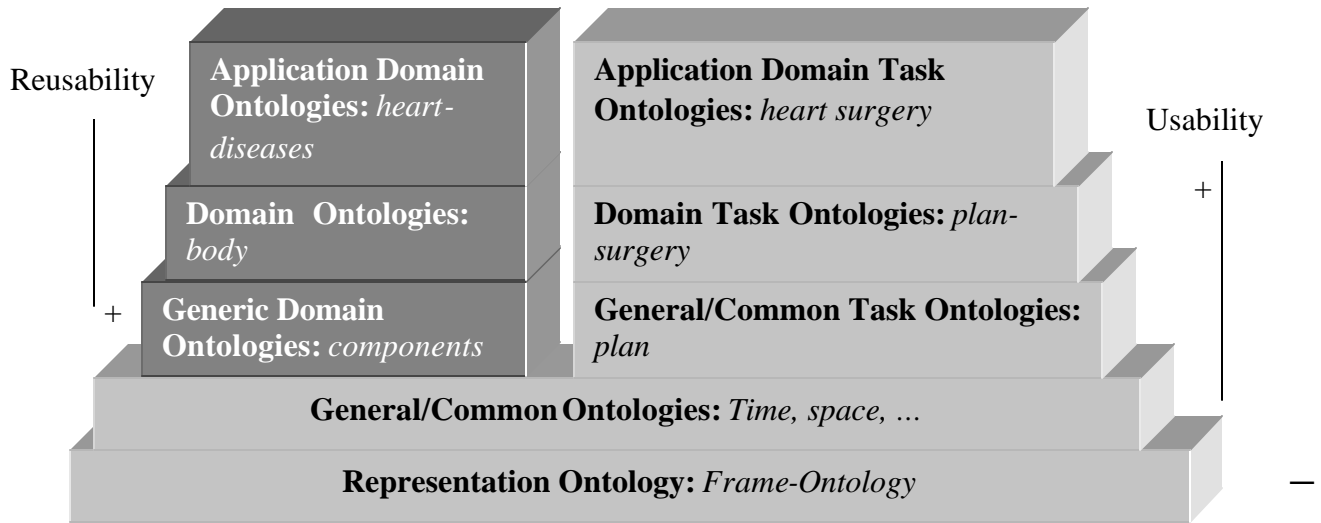


Figure 1: Example of libraries of ontologies (modified from [55])

Table 2: Some well-known ontologies

Name	Category	Brief Description	Developer
CYC	Upper level ontology	A general ontology for common sense knowledge to facilitate reasoning.	www.cyc.com
TOVE (Toronto Virtual Enterprise Project)	Domain ontology	For enterprise modeling, it includes ontologies about activities, state, causality, time, resources, inventory, requirements order, and parts.	University of Toronto Enterprise Integration Laboratory www.eil.utoronto.ca
UMLS (Unified Medical Language System)	Domain ontology	An ontology of medical concepts.	www.nlm.nih.gov/research/umls
WORDNET	Domain ontology	Most well-developed lexical ontology.	www.cogsci.princeton.edu/~wn

2.2 Ontology Engineering

Ontology engineering is concerned with the principled design, modification, application and evaluation of ontologies [70]. It encompasses a set of activities that are conducted during conceptualization, design, implementation and deployment of

ontologies. It covers a range of topics such as ontology evaluation, development methodology, knowledge sharing and reuse, knowledge management, business process modeling, and information retrieval from the Internet [44].

Ontology development has passed the stage where it lacked standardized methodologies, life cycle model and systematic approaches in its infancy stage. It is now moving from its early craftsmanship development stage towards an engineering activity, which has a set of well-defined criteria, techniques and tools [55]. This section takes a closer examination of the various development approaches and methodologies adopted by developers in practice.

2.2.3 Different Ontology Engineering Approaches

The following table outlines the five primary approaches to ontology design: inspiration, induction, deduction, synthesis and collaboration approaches.

Table 3: Approaches to ontology design [70]

	Approaches to ontology design
Approach	Basis for Design
Inspirational	Individual viewpoint about the domain.
Inductive	Specific case within the domain.
Deductive	General principles about the domain.
Synthetic	Set of existing ontologies, each of which provides a partial characterization of the domain.
Collaborative	Multiple individuals' viewpoints about the domain, possibly coupled with an initial ontology as an anchor.

With an *inspirational approach*, a single developer takes the tasks of gathering requirements, performing the domain analysis, designing the ontology and verification of the ontology. The developer must be a domain expert and an ontology design expert in order to ensure the success of the design, and more importantly, to ensure the adoption of

the ontology in the user community. This process heavily depends on a single developer's creativity, inspiration and his or her personal perspective of the domain.

This approach is often applied in focused domains that can be well understood by a chief designer, where the designer is able to dominate the design process and ensure the adoption of the final product. However, when the knowledge in that domain starts to expand rapidly and the complexity of the ontology starts to grow, the method will soon become ineffective.

With an *inductive approach*, an ontology is developed by observing, examining, and analyzing one or more specific cases in the domain of interest. The resulting ontological characterization for a specific case is applied to other cases in the same domain [61]. The degree of ontology commitment is largely decided by the generality of the cases chosen during development.

With a *deductive approach*, some general principles are adopted first and then tailored and applied to the target domain. The resulting ontology can be viewed as an instance of these general notions.

In the *synthetic approach*, a set of related ontologies is identified. The developer then synthesizes the elements from these ontologies first together with the concepts in the new target domain, produces a new unified ontology.

The fifth approach is the *collaborative approach*. The hallmark of a modern ontology is its large size and high complexity; these ontologies encompass such a rich set of knowledge that its complete comprehension is beyond that of any single developer or even a small team of developers. The development of a large-scale ontology has to be a joint effort of many domain experts and software developers and the collaborative

approach for ontology development is the one best suited for the task as none of the first four approaches address the impact of complexity of the ontology to the development and the issue of coordinating team efforts.

From the literature reviewed in the field of ontology engineering, we inferred a range of strategies in identifying concepts. These are:

1. Bottom-up: starting from the most specific concepts and then grouping them in categories.
2. Top-down: identifying the most general concepts and creating categories at the general level first.
3. Middle-out: starting from middle layer of most relevant concepts and then growing the ontology in both directions.

Figure 2 illustrates one collaborative ontology engineering approach that uses the Delphi method in its iterative improvement stage.

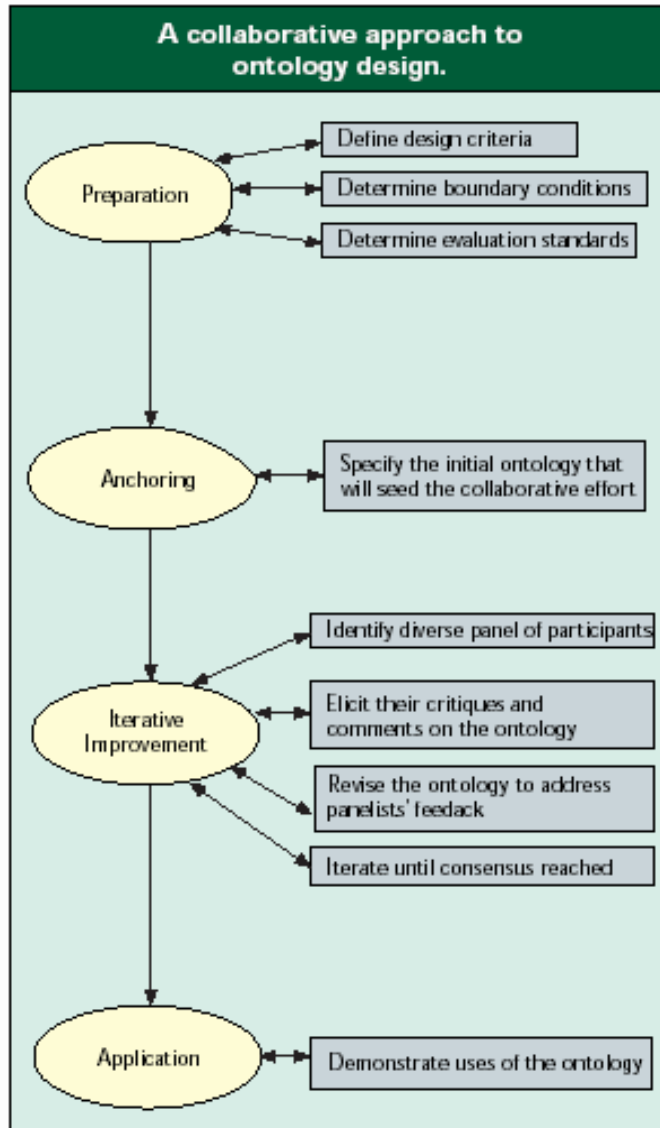


Figure 2: A collaborative approach to ontology design

A practitioner of the Delphi method described this method as [70]:

“[The Delphi method] is a formal technique for collecting and integrating the view of multiple persons about some topic. Each participant independently provides views in writing to a leader, who prepares a document reflecting the combined views as feedback for the next round. In the second round, participants furnish their independent written views in light of the feedback. In this way, the (development team) leader attempts to foster a convergence of views across successive rounds”.

For a more comprehensive discussion of a case study for this approach, please refer to [70].

Because this development process inevitably involves multiple developers and domain experts, collaboration among these people for the purpose of reaching a convergence of views becomes a decisive factor for the success of the resulting ontology. The collaborating nature of ontology engineering becomes even more prominent in the following discussion about the life cycle model.

2.2.4 Life Cycle Model of an Ontology

The ontology life cycle model defines a set of stages that occur along the time line for building an ontology, guidelines and principles to assist in the different stages, and relationships and transitions among the stages. From the literature review, we synthesize the development activities involved as the following:

1. Planning
2. Gathering requirements and specification
3. Eliciting knowledge using knowledge elicitation techniques, knowledge acquisition
4. Designing and building the conceptual model after acquiring enough domain knowledge
5. Formalizing the conceptual model using a formal language, such as a frame-oriented or description logic representation systems
6. Finding existing related ontologies and trying to reuse and integrate them into the formalized conceptual model
7. Implementing the ontology with a formal language
8. Evaluating the ontology against ontology evaluation frameworks (example presented in [92])

9. Documenting the design and the implementation

10. Maintenance of the ontology [79]

This list covers all the major activities in modern ontology development. There exist Some variations can be obtained by combining some categories or by further decomposing a step into finer steps. This list merely is an enumeration of the activities and does not impose any order on the execution of such activities. The life cycle model of an ontology also decides the grand order of the activities and when to move from one activity or state to another. Though we always do the planning first and the maintenance last, the activity of specification, knowledge acquisition, and conceptualization are often intertwined.

Inspired by the success of software development models, such as the waterfall model [99], incremental model, and spiral model [35], ontology development researchers have tried to incorporate these models into the ontology development process. Methodological approaches that incorporate those models in building ontologies have been reported by Uschold in the Enterprise Ontology project [105], Gruninger in the TOVE project, and also in the domain of enterprise modeling [21]. Fernandez [79] has a detailed report on his practice in mapping the above listed ontology building activities to the spiral model. In these methodological approaches, developers borrow the lessons learned from the software engineering practice and transfer them to ontology engineering. The following paragraphs are a closer examination of the waterfall and spiral models as used in ontology engineering.

The waterfall life cycle model, a popular software engineering methodology first defined by Royce [99], has been transferred to knowledge engineering. Development

organized according to this model is supposed to proceed linearly through the phases of requirements analysis, design, and implementation, testing (validation), integration and maintenance. The drawback of the waterfall model is the difficulty of accommodating changes after the process is underway.

When applied in the ontology engineering field, this model requires the ontology developers to identify all the terms at the beginning, and the implementation must be a mirror of the specification; that is, it has to satisfy the complete requirements specification [79]. When used by ontology engineers, this model faces the same problem as it has in the software engineering domain; it fails to address the inevitable problem of incomplete requirements specification at the early stage of the development process, and is not able to capture those requirements and specifications during the evolution of the ontology over its life time.

The spiral model [35], sometimes called the evolutionary life cycle, is borrowed from the software engineering field to solve some of these problems. The basic idea of the spiral model is to use the waterfall model in each step or development cycle as this helps manage risks of incomplete specification in each step in light of the nature of ontology development. The developers only define the highest priority features in every iteration. Feedback from ontology users is collected after those features are implemented and released. With this knowledge, developers then go back to define and implement more features in iterative steps (Figure 3).

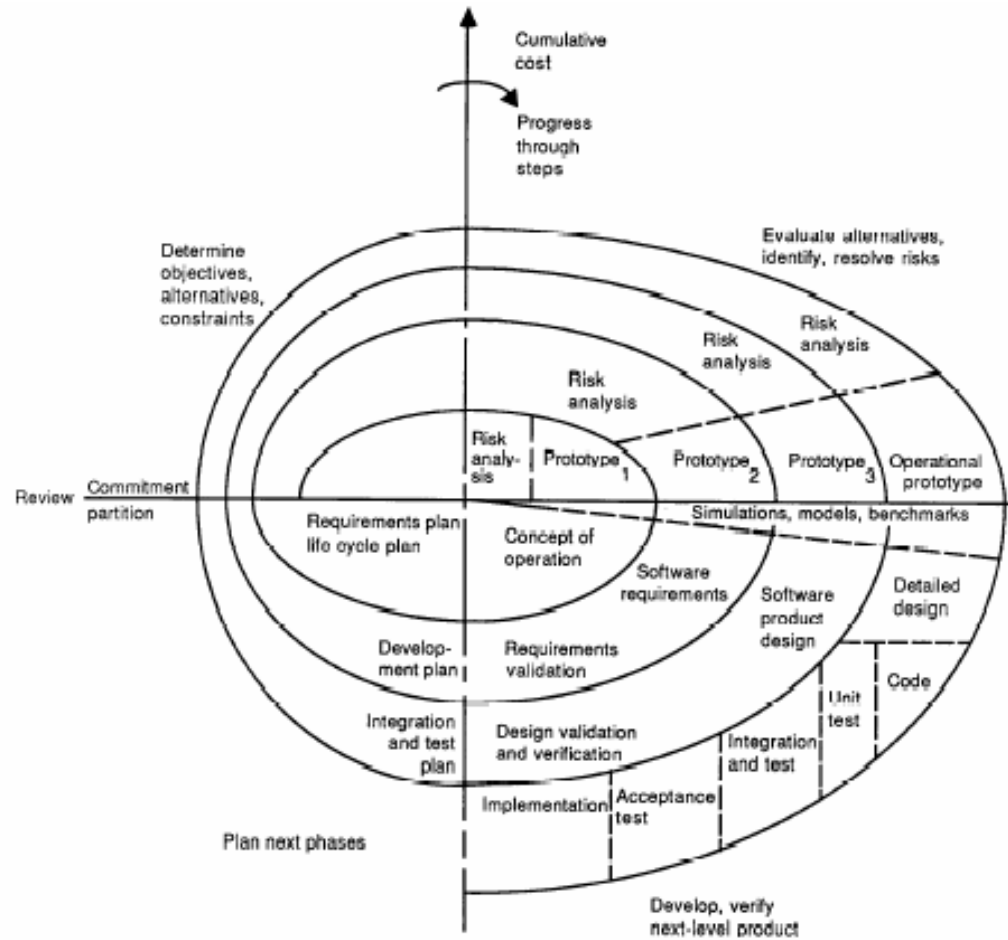


Figure 3: Spiral model of software development [35]

When applied in the ontology engineering field, each cycle of the spiral begins with the identification of:

1. the objectives of the portion of the ontology being elaborated (concepts, definitions, relationships, and the conceptual model)
2. the alternative means of implementing this portion of the ontology with a formal language
3. the constraints imposed on the application of the alternatives (cost, schedule) [35]

The next step after identification is to evaluate the alternatives relative to the objectives and constraints; this may involve prototyping and analytic modeling. Once one

of the alternatives is chosen, the process continues with the development of a more detailed prototype and the final implementation of this portion of the ontology.

The primary advantage of the spiral model is that it facilitates the process of including or excluding new definitions at any stage of the development.

Many ontologies have been built by learning and borrowing from software development process models. Appendix A contains a partial list of them. For detailed information on each one of the projects listed, please refer to [77].

2.3 Summary

By investigating the common practices/methodologies in creating ontologies and the life cycle model of an ontology, we can conclude that ontology development is increasingly collaborative.

In order to explore tool support for collaborative ontology engineering, we look at Computer Supported Cooperative Work (CSCW) in the next chapter, and further investigate how technologies from the CSCW field are used in ontology engineering practices.

Chapter 3 Computer Supported Collaborative Work

CSCW stands for Computer Supported Collaborative Work (CSCW); it is a multidisciplinary research field encompassing computer science, artificial intelligence, sociology, and psychology. CSCW involves studies about tools and techniques of groupware as well as their psychological, social and organizational effects. It is a generic term that combines the understanding of the way people work in groups with the enabling technologies of computer networking and the associated hardware, software, services and techniques [20].

The term groupware is often seen side by side with CSCW in computer science literature. Ellis [45] defines groupware as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." The notion of groupware is the computer-based support of work groups or project teams. Support may mean support by special software or hardware, by information and communication services as well as support of group work, in contrast to individual data processing.

While groupware refers to real computer-based systems, CSCW is the study of tools and techniques of groupware as well as their psychological, social and organizational effects [4]. In other words, the term 'groupware' usually refers to tool implementations whereas CSCW looks at everything that impacts groupware.

For fast and distributed development of ontologies, the development team often needs and uses collaborative technology support, which falls into the domain of CSCW. This

chapter briefly reviews the history of the CSCW domain and its state of the art technologies.

3.1 Overview of Groupware

In 1968, Doug Engelbart and other researchers from the Augmentation Research Center at Stanford Research Institute in Menlo Park, CA demonstrated the NLS/Augment system [46] that debuted the mouse and other innovations including shared-screen collaboration involving two persons at different sites communicating over a network with audio and video interface. It inaugurated the development of modern groupware such as real-time shared editing of documents, text messaging and audio and video-conferencing.

CSCW and groupware emerged in the 1980s as a research field from shared interests among product developers and researchers in different areas. It started as an effort by technologists to learn from economists, social psychologists, organizational theorists, educators, and anyone else who could shed light on group activity [58]. Many groupware applications have since been developed and used in the 1990s and have advanced further in recent years with the rapid growth of the Internet and the World Wide Web [33].

A conventional categorization of CSCW systems according to a time/location matrix is shown in Table 4 [74] and Table 5 [59].

From the literature reviews, we find that, in general, the cooperative support provided by CSCW system or groupware can be categorized as following:

- Data storage (e.g. file sharing system, CVS)
- Synchronous and asynchronous communication (e.g. email, video/audio conferencing, instant messaging, Web bulletin board, Web log, group chat)

- Organization of the work (workflow systems, collaborative editing/graphing applications, group decision support systems)
- Advanced groupware that integrates several of the above functions

Groupware functions in a heterogeneous world where different media, storage systems, and planned or impromptu collaboration are used. In the real world, a variety of tools and techniques are often used to get a job done. A single groupware product that adequately covers all aspects of cooperation does not exist and may never exist[36].

The typology in Table 4 and 5 help us identify groupware applications that pose common technical challenge. Developers do not necessarily map the tool to be developed rigorously to this time and space categorization, because tasks and activities do not always match this table below precisely [58]. For instance, Table 4 does not show that some of these activities overlap each other. Email can be used in same place / same time situation very effectively—for example when you need to talk to someone privately while you are in the same room but others are present, sending an email to this person may be a good solution.

Our daily work often involves some face to face meetings, some distributed synchronous and asynchronous communications, and some coordination. To provide adequate collaboration support for one task usually requires a set of tools.

Key functions of groupware are group awareness, multi-user interfaces, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing single-user applications [4].

Table 4: Classification of CSCW Systems

	Same time	Different times
Same place	Face-to-face (classrooms, Meeting rooms)	Asynchronous interaction (project scheduling, coordination tools)
Different places	Synchronous distributed (shared editors, video windows)	Asynchronous distributed (email, bulletin boards, conferences)

Table 5: Fine grained categorization of groupware

Space		Same	Different but Predictable	Different but Unpredictable
	Same	Meeting facilitation	Work Shifts	Team rooms
	Different but Predictable	Tele/Video Conferencing	Electronic mail	Collaborative Writing
	Different but Unpredictable	Interactive multicast seminars	Web bulletin boards	Workflow

Groupware applications have already infiltrated our office and home. Instant messaging systems such as MSN messenger, Yahoo messenger and AOL instant messaging have been widely used for informal communication both at work and home. With application sharing (e.g. co-authoring and shared writing tools, group calendar), a group of users simultaneously interacts with one or more programs and they all can see and share the results. Microsoft Windows' Remote Assistance application even allows sharing all the applications on your computer with another user over the network. The World Wide Web, initially designed as a passive medium, has also been extended with facilities to support cooperation[36]; Web-based bulletin boards, shared web calendars, web project planning and management systems have been used to support team awareness in business and academic settings.

As well as the technological advances made in groupware development, the non-technical aspects, such as social, psychological, and economical, are equally important.

David Coleman has this interesting account in his book “Collaborative Strategies for Corporate LANs and Intranets” [42]:

“... When addressing technical challenges, a technical solution must be found. However, even if the technology solves the problem, works well, and is rolled out efficiently, support from the corporate culture is essential to the implementation's success. Further, even if the culture supports the groupware success, but there is no economic justification for a groupware solution, the implementation will fail. Finally, even if technology, culture, and economics combine to support groupware, the success of a project can be destroyed by politics.”

Taking all the factors into consideration, Coleman expresses the success of any groupware application by the equation:

$$\text{Groupware Success} = \text{Technology} + \text{Culture} + \text{Economics} + \text{Politics}$$

The further to the right a factor is in this equation, the greater its potential impact on the success of the project [42]. Coleman again reminds us of the importance of human and social factors in any groupware implementation and deployment.

Though success and failure of a groupware application cannot be reliably predicted [60]. Overall, development of groupware requires a good understanding of the working environments and the social, political and motivational factors in the work place.

From this point of view, the willingness of the individual tool users, the project team, and even the entire organization, to participate and to use the groupware is critical to the acceptance of the tool. In the next chapter, we will begin by taking a closer look at the existing groupware support in some of the more widely used ontology development tools.

Chapter 4 Survey of CSCW Support for Ontology

Development Tools

Building ontologies in a collaborative environment has been an ongoing research topic. There are a number of tools for ontology development. In order to search for innovative ways to complement the collaborative support in the existing tools, it is important to study the existing approaches, and examine the weaknesses and strengths of each of them in providing collaborative support. The five tools presented here are all comprehensive ontology development environments and are widely used in the ontology development community. Ontolingua Server, OntoEdit and APECKS are tools built on a client-server architecture with different approaches in supporting collaboration. Protégé-2000 is a platform-independent tool widely used in the clinical and medical domain for creating and editing ontologies. It started with functions supporting solo development and is evolving towards supporting collaborative group works. For purpose of our research, we will mainly focus our discussion on the collaborative aspects of the tools.

4.1 Ontolingua Server

Developed by the Knowledge Systems Laboratory at Stanford University, the Ontolingua Server is a set of tools and services used to support the process of achieving consensus on common shared ontologies by geographically distributed groups. These tools make use of the World Wide Web to enable wide access and provide users with the ability to publish, browse, create, and edit ontologies stored on an ontology server [53].

These tools and services aim to promote the use of ontologies across the user community and knowledge-level agent interaction.

Figure 4 shows the architecture of the Ontolingua Server. The left hand box depicts the general purpose Ontolingua editor and server. In order to support ontology reuse, the server hosts a central library of ontologies, and provides developers controlled access to this library. The server supports the assembly, customization, and extension of ontologies from this library of ontologies.

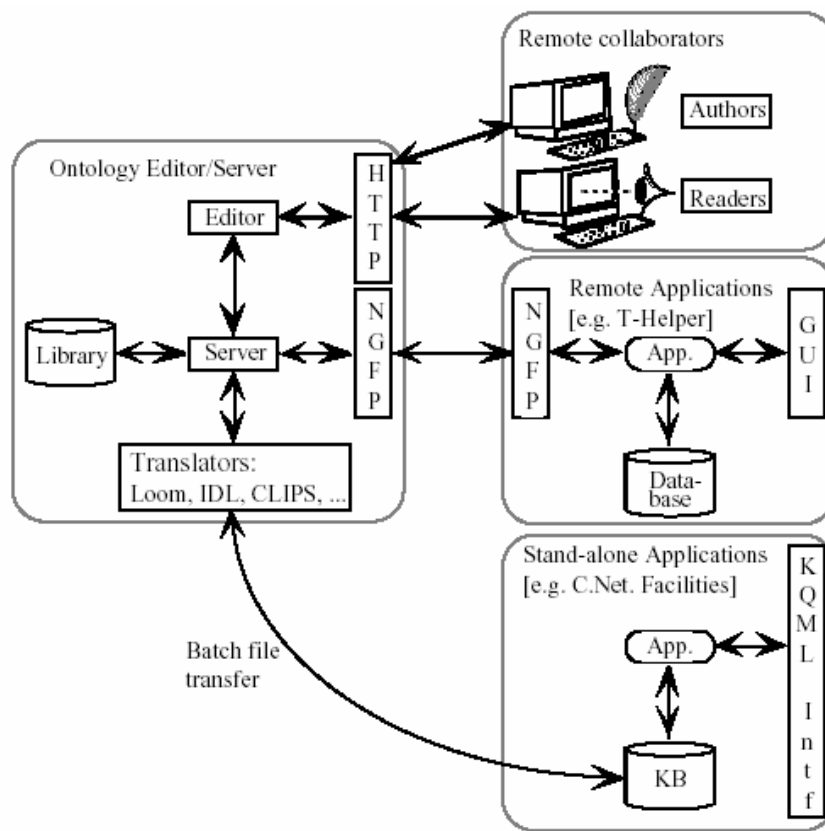


Figure 4: Architecture of the Ontolingua Server ([53])

This Ontolingua Server technology supports three modes of use:

1. Distributed groups may browse and retrieve ontologies from repositories using a Web browser. Users can also use the Ontolingua language to build and

maintain ontologies stored on the server. The Ontoligua language is based on the Knowledge Interchange Format [53].

2. Remote applications may query and modify ontologies stored on the server over the Internet using a network API that extends the Generic Frame Protocol [94].

3. Users can translate an ontology into a format used by a specific application.

For example, an Interface Definition Language (IDL) translation can produce an IDL header file that a CORBA compliant program can use to interact with an object request broker.

The three boxes on the right side of the Fig. 4 indicate these primary use modes.

Support for distributed and collaborative development of consensus ontologies is provided through the web interface by means of user and group access control and multi-user sessions. Locking mechanisms and analysis of alternative definitions from multiple authors facilitates the concurrent access to a shared ontology. The developers of the Ontoligua Server provide a detailed explanation [53] on this matter:

The Ontoligua Server uses a notion of users and groups that is typical in most multi user file systems. As with file systems, read and write access to ontologies is controlled by the ontology owner giving access to specific groups. This mechanism supports both access protection as well as collaboration across groups of people who are defined within the ontology development environment.

The server provides support for simultaneous work through group sessions. When a user opens a session, she may assign a group ownership to it. This enables any other members of that group to join the session and work simultaneously on the same set of ontologies. A notification mechanism informs each user of the changes that other users have made. Notifications are hyperlinked to the changed definitions and describe changes in terms of basic operations such as add, delete, and modify (e.g., “Farquhar added the slot motor-of to the class vehicle”). The synchronous nature of the web protocols makes this sort of notification somewhat clumsy — the Server cannot notify users that a change has occurred until they visit a new page.

Researchers also found that merging ontology is critical in coordinating collaborative development for both co-located and geographically distributed teams [83]. Chimaera [83] is an optional web-based ontology merging and diagnostic tool used by Ontolingua Server. Chimaera allows users to choose the level of rigor before merging and suggests a list of potential merging candidates based on this rigorous level.

4.2 OntoEdit

OntoEdit [15, 102] is a collaborative ontology editor for the Semantic Web. It is available in freeware and professional versions. The professional version typically includes an additional set of plug-ins, e.g. plug-ins that provide the collaborative environment and the inferencing capabilities. OntoEdit is a methodology-based ontology construction tool that supports an iterative development process with three phases: a requirement specification phase, a refinement phase, and evaluation phase. The professional version has two plug-ins to support collaboration: OntoKick and Mind2Onto.

OntoKick supports the collaborative generation of requirements specifications for ontologies. The collaborative aspect of Ontokick is not so much support for distributed work of team members, but rather support for personal interaction of ontology engineers and domain experts [16]. OntoKick allows ontology engineers to specify important meta-aspects of the ontology, and it supports the creation of a semi-formal ontology description in the requirements specification phase.

Mind2Onto integrates the commercial tool MindManagerTM into the development process. MindManager supports collaborative engineering of mind map through peer-to-peer communication, and it presents the hierarchical mind maps in graphical format. OntoEdit uses MindManager to facilitate brainstorming and discussion sessions about

building ontology structures, while OntoEdit imports the mind map in XML format and converts it to a semi-formal description of an ontology. It also supports collaborative editing of mind map through peer-to-peer communication [102]. The peer-to-peer communication of the mind map tool provided the necessary workgroup functionalities. Individual developers and workgroups on the peer-to-peer network can easily participate in a joint session in creating a mind map.

The goal of OntoEdit in the refinement phase is to turn the semi-formal description of the ontology according to the captured requirements into a mature ontology. OntoEdit uses a client-server architecture as shown in Fig. 5. The server will send out notifications to all the connected clients when there is any modification made to the ontology. Ontology engineers store and share their comments; for example, the design rationales in the documentation fields for each concept and relation.

The OntoEdit ontology server employs a very fine-grained locking mechanism and transaction management system to coordinate the concurrent accesses and modifications to the shared ontology to ensure a safe and consistent ontology development environment. Developers have options ranging from locking concepts, instances, and relationships to locking a complete subtree of the concept hierarchy. To guarantee consistency and to avoid deadlock situations, OntoEdit forces clients to obtain locks to all the needed resources pertaining to the object to be locked. For example, locking a concept X implies read-lock for all super-concepts of X and all their super-concepts. A read-lock marks a resource as being read-only; that is, modifications to it are disallowed. If a read-lock for at least one super-concept cannot be achieved, the concept X cannot be locked and the transaction fails [102].

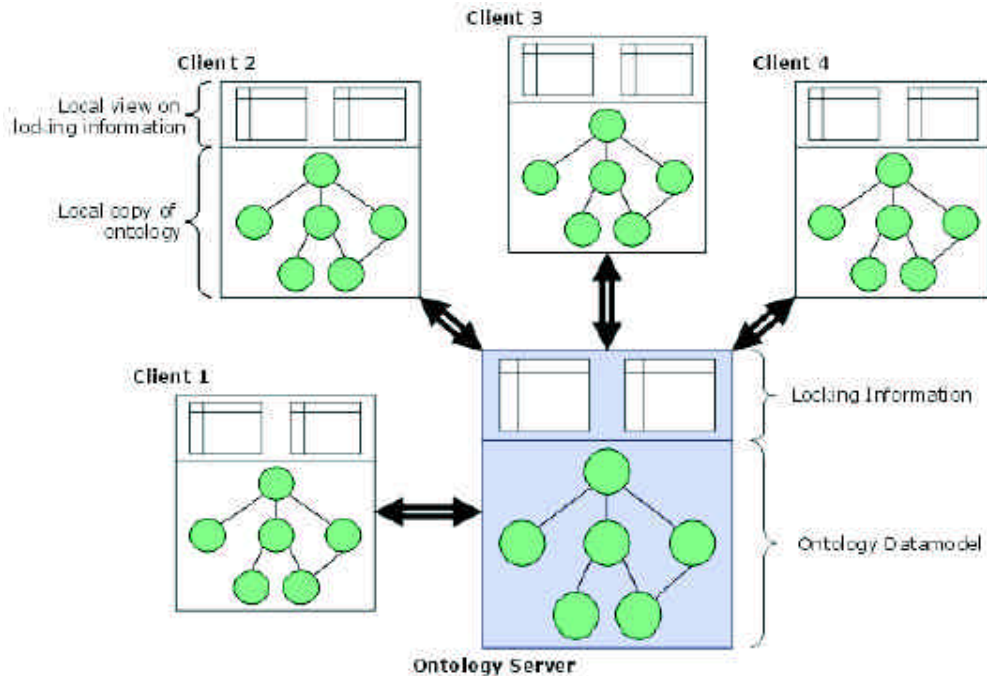


Figure 5: Client/Server architecture of OntoEdit ([102])

4.3 APECKS

The Adaptive Presentation Environment for Collaborative Knowledge Structuring (APECKS) system was developed by the Department of Psychology at the University of Nottingham. It is an ontology server supporting collaboration by allowing domain experts to create ontologies based on their own perspective. APECKS allows users to compare their perspective to another perspectives (prototype, design rational, etc) to prompt discussion about the sources of their differences and similarities. The developer of APECKS believes that ontologies should develop in the form of “living ontologies”, and as such the tool for the development should support collaboration among their creators through structured and unstructured communication. APECKS tries to enable experts to address the sources of their disagreements and to argue with a productive end. The

emphasis within APECKS is not on the outcome it produces, but rather the process: the disagreements and discussion that are involved in creating a consensual ontology [73].

The APECKS ontology servers are designed to let a number of users create an ontology together and communication among these developers is supported by the following mechanisms [73]:

- *Subscription*: users subscribe to certain areas of interest within a knowledge representation. They are then notified of any changes that occur to those areas.
- *Annotation*: users' annotations can be recorded and attached to any concepts or instances for cross-references.
- *Group sessions*: users subscribed and working in the same group session can receive synchronized notification when changes are made by other users within the session.
- *Synchronous communication*: collaborators can send short messages to each other, including images and ontology files.

APECKS supports unstructured synchronous communication and unstructured asynchronous communication by allowing free annotation of objects in the system, and by preserving annotations and establishing referential links between annotations and objects. It supports structured communication by:

1. Using the questions, options and criteria (QOC) methodology. For a design question, it uses a questionnaire to poll each user's answers in order to make the decision. It also recognizes that structure communication often limits the expressiveness of discussion and can be subverted by users to enable them to make their point, which detracts from its utility [73].

2. Allowing users to compare their ontology from time to time in terms of the consensus, conflict, correspondence, and contrast classification. Users of APECKS are prompted to take actions or communicate with each other on the basis of the comparisons. The process results in further discovery of domain related details, and many criteria used in the construction of ontologies are made explicit. One such scenario is documented in [73]:

If one ontology classifies rocks in terms of their 'quartz content' while another does so in exactly the same way, but uses the term 'silica content', APECKS would recognize this as a state of correspondence (different terminology being used for the same concept). The users who constructed these ontologies would be prompted to either change the term to bring them into line with the other or to start a discussion about why different terms were being used.

APECKS is geared towards two overlapping but distinct groups of people: knowledge engineers who have the expertise of constructing ontologies and domain experts who have the domain knowledge. The knowledge engineer's task is to construct consensual ontologies from the knowledge of a number of experts. This leads to the type of assertion [98] that ontologies, by definition, represent consensual knowledge, the single accepted definitions of technical terms used within the domain.

4.4 CO₄ System and CO₄ Protocol

The Collaborative Construction of Consensual Knowledge (CO₄) system [49, 52] is designed for the incremental and concurrent building of a knowledge base. This system is first developed and used in the domain of molecular genetics. The CO₄ protocol and the implementation of the CO₄ system supports collaborative construction of a formal knowledge base, and it allows collaborators to freely annotate, express and manipulate their knowledge with hypertext, images, and experimental data [50]. It uses a Web

browser as its presentation medium. It specifically addresses the problem of consensus of the knowledge base with the help of the CO₄ protocol for integrating knowledge through several levels of consensual knowledge bases. Capturing corporate memory through cooperative creation of knowledge bases and hyper-documents is the motivation behind the CO₄ system.

The principles underlying the CO₄ protocol are derived from the peer-reviewing protocol [95]: before being committed into a consensual knowledge base, the knowledge must be submitted, reviewed and accepted by the community [50]. It is intended to ensure that at the end of the development process, knowledge stored in the knowledge base is safe enough so that anybody can accept it and use it confidently and easily. Informal knowledge is also subject to submissions, reviewing and so on. The CO₄ system uses a peer-reviewing protocol in knowledge base development.

The designer of the CO₄ system discovered that simply using a “computer as [a] medium” for stating, editing and storing knowledge bases is not enough [50], it does not promote communication among its individual participants or developers.

The motivations behind the design of this system lie in three aspects:

1. Knowledge must be stated as formally as possible. Formalizing knowledge has the advantage of capturing the semantics, creating a standard ontology and allows the software agents to manipulate the knowledge, do intelligent reasoning and searching.
2. Not everything can be formalized, and even if everything is formalized, the formal system can still suffer from problems such as complexity and incompleteness [50]. Thus, the formal knowledge must be supported with rich documentation,

annotations, images, animations and even video/audio material. Formal knowledge, knowledge that has not yet reached a formal state, and knowledge that cannot be formalized are well supported by the informal forms of knowledge.

3. People must be supported in discussing the knowledge introduced in the knowledge base. In this perspective, creating, re-using, modifying and maintaining knowledge should be a participatory activity of all the involved people (providers and users) [49, 50]. Supporting discussion and consensus building ensures the consistency and coherence of the produced knowledge base, and its acceptance and usefulness in the user community.

4.4.1 The CO₄ Knowledge Base Network

Figure 6 illustrates the hierarchical structure of the CO₄ knowledge base network.

Each collaborator plays a base role in the system. Bases are organized into a tree structure whose leaves are user bases and whose intermediate nodes are group bases; each group base represents the knowledge consensual among its children, the subscriber bases [50]. Group bases can be further grouped together to some higher level groups until it reaches the top of the tree. This network of collaboration can be imposed on the structure of a particular organization or that of a group/department in the organization [49].

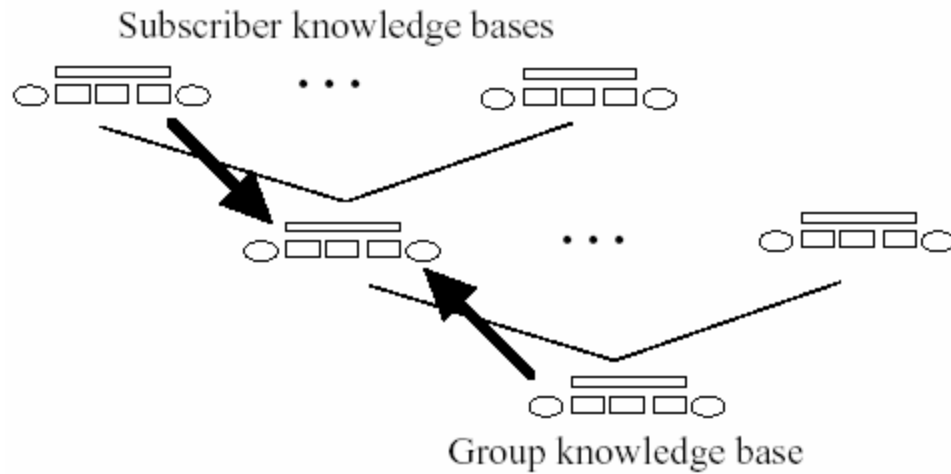


Figure 6: Hierarchical knowledge base network and message flow (dark arrows). Bases are organized in a tree whose leaves are individual bases and nodes [50].

Each group base acts as a rendezvous point for its subscriber bases to submit its knowledges, critique and comment on each other's submission, explore alternatives, revise the changes and eventually reach consensus. The dark arrow in Figure 6 represents the flow of message from one base to another. A group base broadcasts messages for a change accepted by everyone, or calls for comments in order to establish whether a change should be committed or not. Similarly, a base (group base or individual) sends to its group base changes or new knowledge it wants the group base to integrate. A human user can subscribe to different group bases, and knowledge can be transferred from one base to another. Several human users can share the same base. Group bases have the same structure as the subscriber bases as they are the same pieces of software; however, the group base is automated to respond to stimuli from its subscribers or higher-level bases.

Each group knowledge base is implemented on top of a web server. Object references in the knowledge base are transformed into URLs. The mixing of hypermedia with the

knowledge base makes the knowledge base accessible through a web browser, and the hypermedia provides navigation assistance for the users of the knowledge base.

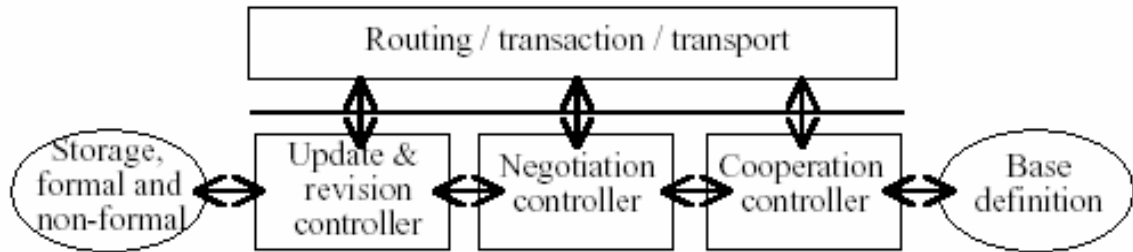


Figure 7: The software architecture of CO₄ system. Boxes represent a software module, Circled units are data/knowledge repositories and arrows represent the call of program functionality [50]

Figure 7 illustrates the software architecture of the CO₄ system. The communication layer handles routing and transport of messages among subscribed groups. Other the components in the system include the:

- knowledge base storage component
- update and revision controller that manages and analyzes the stored knowledge base, detects inconsistencies and suggest possible repairs
- negotiation controller that interacts with the peer collaborators by broadcasting messages such as call for comments, and possible repairs and alternatives to the knowledge base. It also handles queries submitted by other collaborators
- cooperation controller is a library of functions that implements the CO₄ protocol
- base definition defines the connection among peer bases

The primary goal of CO₄ is to construct a consensual knowledge base. It achieves this goal by combining the hypermedia navigational assistance in a large information space with the collaboration protocol. Its implementation details can be found in [51].

4.5 Protégé-2000

Developed by Stanford Medical Informatics at the Stanford University School of Medicine, Protégé-2000 is an extensible, platform-independent software tool for creating and editing ontologies and knowledge bases. It allows domain experts and ontology developers to create and modify reusable domain ontologies, customize knowledge-acquisition forms, and enter domain knowledge [88]. Its plug-in architecture allows developers to add customized components to provide new functionality, such as customized graphical widgets for tables, and diagrams and animation components to access other knowledge based systems. It also has a set of libraries that can be used by other applications to access and display knowledge bases [18]. Protégé-2000 is currently being used in clinical medicine and the biomedical sciences, but it can also be used in any field where the concepts can be modeled by a frame-based system. A knowledge base in a frame-based system is built around the notion of frames or classes which represent collections of instances, and a hierarchy of type definitions and the relationships between the types. Each frame has an associated collection of slots or attributes which can be filled by values or other frames. In particular, frames can have a kind-of slot which allows the assertion of the frame taxonomy. This hierarchy can then be used for inheritance of slots. As well as frames representing concepts, a frame-based representation may also contain instance frames, which represent particular instances.

The beta version of the multi-user version of Protégé-2000 is available for testing at the time of this writing. In this version, the ontology under development is stored in a shared database where multiple users can read the same database and make incremental changes or changes that do not conflict with one another. Protégé-2000 uses the Open

Knowledge-Base Connectivity protocol (OKBC) as its common query and construction interface for its frame-based knowledge representation system in order to achieve interoperability with other knowledge representation systems. Consequently, Protégé-2000 users can import ontologies from and export to other OKBC-compatible tools. It also complies with the new OWL (Web Ontology Language) knowledge model, which is developed by the World-Wide Web consortium and is emerging as the standard for defining metadata for encoding machine-readable semantics on the Web. Protégé-2000 can translate an RDF knowledge base created in Protégé-2000 into standard RDF syntax; effectively making Protégé-2000 an editor for RDF documents [90], adding on another degree of interoperability to this tool.

Figure 8 shows the ontology-editing environment in Protégé-2000. The left-hand panel contains the class hierarchy. The left-most pane in Figure 8 shows the class hierarchy with multiple inheritances; the hierarchy can be re-arranged by dragging and dropping. The right side of the pane shows the detailed information for the selected class. The details include the slots for the class, their values, the template slots attached to the class, and the value restrictions. The small window shows the form for editing an instance of the selected class.

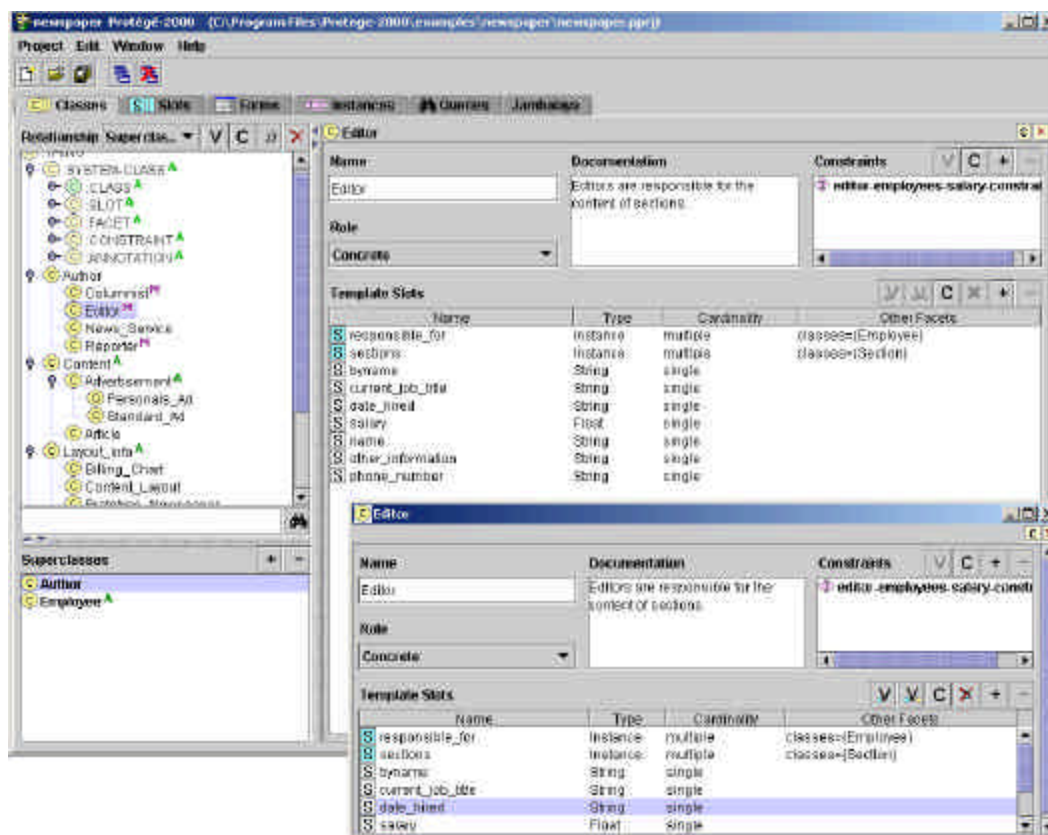


Figure 8: Editing classes, slots, and instances with Protégé-2000

Protégé-2000 also has an advanced visualization and navigation engine which we will discuss in detail in Chapter 7. This plug-in provides not only the ability to visualize many ontology concepts in ways that are easier to understand, but it also gives the users a rich set of controls with which users can interact with the data with greater freedom.

4.6 A Comparison of Ontology Editing Tools

Both Ontolingua Server and OntoEdit adopt a server-client infrastructure to support concurrent and distributed collaboration.

OntoEdit uses a locking and transaction protocol and implements a distributed event model on the basis of Java remote method invocation to provide consistency and concurrency.

Ontolingua Server uses the HTTP protocol to propagate change notifications to the clients; but because of the asynchronous nature of HTTP, this notification mechanism does not satisfy the synchronous collaboration criteria well. Using the ubiquitous Web browser as the interactive medium for browsing and editing ontologies on the Ontolingua Server attracts a large user community because it improves the rate of user acceptance, and avoids many of the pitfalls of software distribution [98]. However, the Web-based interface in Ontolingua also imposes a set of constraints on the user interface, such as the responsiveness of the application.

The fine-tuned locking mechanism, transaction management and inference support in OntoEdit make it stand out among many collaborative tools. However, using locking to coordinate collaboration also creates some problems when there is a large development team. The locking method uses a mutual exclusive algorithm, which has long been used in operating systems, to coordinate multiple processes running simultaneously. In the case of the operating system, the daemon scheduler running in the background does all the coordinated scheduling to achieve maximum resource utilization, load balance, fairness of time sharing, etc, among processes. In the case of OntoEdit, each task a developer is working on is treated as a running process, which are suspended and resumed according to the availability of resources (concepts, instances, ontology hierarchical subtree). The scheduling process among the ontology engineers and domain experts is realized through communication such as email, phone calls, face-to-face collaboration, instant messaging, and even corridor talks. It is not a process that can be completely automated to achieve maximum resource usage. Better collaborative support

that facilitates both formal and the ad-hoc nature of communication among developers is needed in this case to support coordination.

The APECKS Ontology Server developers identified communication among ontology developers as an important task and tried to provide explicit support for it. The APECKS designers found that more communication between the builders of ontologies will allow deeper discussion of a domain, leading to both richer ontologies and design rationales that can be reviewed and used to inform the construction of future ontologies [73].

For Ontolingua Server and APECKS, the communication and collaboration support are both built as extensions of HTTP servers, and they present to the users a frame-based view of the ontology in the web browser. In APECKS, any user can define multiple ontologies within a single domain, representing different aspects of the domain or different tasks that might be carried out within it [73]. Individual ontologies are shared in a different manner from Ontolingua shared sessions; these are shared in a similar manner to the knowledge bases in CO₄ discussed in the following section, which are based on group sharing and peer-reviewing.

APECKS uses the frame-based knowledge representation schema. The translation between the APECKS representation system and that in other frame representation systems can be facilitated by Ontolingua Server and the Generic Frame Protocol [73]. This makes APECKS able to access network-accessible Ontolingua ontologies and can incorporate them into the APECKS ontology representation.

APECKS foresees the use of distributed networked-resources on an individual developer's workspace. It is envisioned as acting as a client as well as a server, accessing network-accessible knowledge acquisition applications and other ontology systems [73].

The CO₄ system emphasizes collaboration and consensus building among developers; Protégé-2000 emphasizes ontology reuse and interoperability among different tools. The CO₄ system particularly pays attention to the acquisition of informal knowledge, and uses it effectively as annotation materials to the formal knowledge, and as communication assistant materials among users.

The ability to present and let the user interact with the ontology effectively is of key importance to a tool's usability. Ontology models usually support multiple inheritance in the concept hierarchies and relation hierarchies; the standard approach is the use of multiple or nested tree views with the ability to expand and contract hierarchical levels [26]. Using web browsers and hypermedia for presentation, navigation and user interaction certainly makes CO₄ easy to learn and use, but it remains to be a challenge for the web browser to present complex hierarchical information. Protégé's visualization plug-in provides various levels of abstraction on the complex ontology structure through a fully zoomable user interface making the user interaction more intuitive, enhancing understanding and allow easier exploration [78].

4.7 Summary

Five ontology development tools widely used by the ontology and semantic web community have been reviewed. Collaborative ontology development, consensus building in the development process, interoperability, and reusability are the primary themes common throughout tools. Each of these five ontology and knowledge base development tools has its strength in one or two of these aspects, and weakness in others. Part of the reason for this diversity is that each tool stemmed from a different domain and

each of them is tailored for a particular ontology design methodology, or a specific development process model.

The insufficiencies in these tools with regard to collaboration support are:

- Coordinated group work, such as collaborative editing, discussion or annotation is not well supported, mainly because the systems lack the functions for keeping developers informed of each other's tasks and activities.
- Contextual communication support is not considered in these tools, forcing users to rely on general purpose tools, like email applications, to communicate with each other. Email is good for formal or semi-formal communications, but considering that in many cases users have to fumble through emails and establish links between the email correspondents and the development environment, email is by far not the best choice. Documentation reflecting a coherent discussion is inevitably scattered in many people's mailboxes. In theory we could join all these documents/views to reconstruct the complete transcript and to get an entire picture of the topic. In practice, however, nobody has the time, skill, or motivation to stitch the whole quilt together [104].

Chapter 5 Dimensions of the Problems in Collaborative Ontology Development

With each successive deployment of networked application, our world is becoming ever more connected. Since Ray Tomlinson developed the first email application for the ARPANET in 1971, various forms of communications have arisen, such as voice mail, instant messaging, cellular phones. This has even led to the claim that distance is becoming irrelevant [37] in our work and lives.

It is true that Internet and communication technologies connect us and enable us to communicate in many incredible ways; nevertheless, we still have problems determining how to effectively set up a globally distributed team and organize a-round-the-clock software development project. With problems like these, we are still left with good reason to be skeptical about the aforementioned claim.

Distances were first introduced into software engineering because of the globalization of the software market, the growing practice of software outsourcing, and the need for lower cost and to gain access to skilled resources [66]. In order to take advantage of these distributed resources, the software industry began to decompose the software development activities and resources, and arrange them into geographically distributed locations. Many software development projects have become globally distributed coordinated tasks. Global Software Development (GSD) studies the effect of this software development practice, the complexities, and new variables introduced into software development, such as distance, communication, cultural issues, management

issues, etc., while aiming to help improve the efficiency of distributed development and the quality of the software products. Researchers in GSD and the telecommunications engineering domain have found that communication, especially informal and unplanned communication, is extremely important in supporting collaborative work [96] [67].

With the increasing recognition of the importance of ontologies and the amount of development and usage of ontologies in business, distributed ontology development is inevitably going to face challenges similar to GSD.

Developing an ontology in multiple sites is still a relatively unexplored territory. In some cases, the insurmountable hurdle created by distance has forced the development work to be done in one central site, at the expense of losing the ability of leveraging the knowledge and expertise of other peer experts in remote sites.

Drawing from the lessons and experiences from the GSD field, this chapter explores how the difficulty of collaborative ontology development manifests itself in three dimensions: distance and communication, documentation and knowledge management, change tracking and version control.

5.1 Distance and Communication

Collaboration over distance must face the loss of the rich, subtle interactions that co-located teams use to coordinate their work. Research in distributed software engineering suggests that working across sites introduces substantial delays to the development cycle because of reduced communication, difficulty in finding the right person and establishing contact, as well as not having an effective collaborative session [69].

For teams with all the members working in a collocated work space, the informal *ad hoc* communications can happen anytime during the work day (in the corridor or beside

the coffee machine, for example) and team members take advantage of this all the time. Because it is seamlessly blended into the work processes, the effects of informal communication are often overlooked, and we tend to neglect its significance. However, in a geographically distributed team, the chances for team members to have informal communications are reduced sharply, and the team loses the benefits of it. A study at Carnegie Mellon University showed that the rate at which scientists collaborated spontaneously with one another was a function of distance between offices [75]. Similar results were also found among engineers. When engineers' offices were about 30 meters or more apart, the frequency of communication would drop to nearly the same low level as people with offices separated by many miles [29].

Meetings are one kind of formalized communication; however, communication need not always occur within a formal, hierarchical communication. When a distributed team collaborates, informal channels of coordination are critical since they help developers fill in the details in the work, handle exceptions and correct mistakes [68]. Empirical studies suggest that software developers also rely heavily on informal and unplanned communication [67]. Informal communication has a direct impact on the development process. For example, the news of a requirement change can propagate and reach each team member much more quickly than going through the formal mechanisms of communication, such as specification documents. This phenomenon potentially makes workers react more quickly to changes and consequently shortens the development period.

During a quantitative study at Bell Labs [65], scientists studied two software development teams, one in collocated and the other in distributed environment. They measured and compared the length of delay for both teams to respond to and implement a

modification request, which is a request to incorporate a specific functionality into the software. The results show that the average distributed team took about 2.5 times longer to complete the task than the collocated team. A similar study was done to measure the length of delay for developers to get needed information from co-workers; the result is that there is no significant difference in the number of delays reported, but the delays in the distributed team took almost a day and half longer than a collocated team.

Coordination is the act of integrating each task with each organizational unit, so each unit contributes to the overall objective; orchestrating this integration often requires intense and ongoing communication [38]. Control refers to the process of adhering to project goals, specifications, and standards. The inadequate communication caused by distance also poses a serious challenge towards team coordination and control. For software engineers and knowledge workers, coordination and control have in many ways blended together where communication becomes the mediating factor between coordination and control [38], and the exchange of unambiguous information is one stepping stone towards achieving consensus.

Preliminary findings from a qualitative study [67] [69] in GSD have identified three general types of coordination problems experienced by software developers, depending on the sub-task interdependencies that are involved. These coordination problems provide the basis for constructing the coordination success variables: technical (i.e. software parts do not work well together), temporal (i.e. software parts not ready on schedule), and process (i.e. lack of adherence to the established software process) [48].

Distributed ontology development suffers from similar problems, for example, the lack of consistency in the concepts and relationships being developed. The following

example was documented by my colleague, Polly Allen, during her visit to the Digital Anatomist Foundational Model of Anatomy group [27] (also known as the Foundational Model of Anatomy or FMA) at the University of Washington in Seattle. As part of their research, the FMA team develops ontologies for the structure of the human body. During their study, it was found that surgeons in North America and Europe both divide the human heart into five regions for study, and for each part, surgeons from both sides share many similar terminologies. When they wanted to merge the ontologies they have each developed for the heart, they found that the way they have divided the heart into five parts are slightly different with each other, which brings complications for the merging task.

While coordination and control are necessary to manage interdependencies within the tasks, they are also important for the development and maintenance of shared mental models [100]. Shared mental models are based on organized shared knowledge, which helps collaborators form accurate explanations and expectations about the task and each other, thus helping them coordinate explicitly. The effectiveness of shared mental models in supporting team work has been extensively studied and proven not only in global software development, but also in other work environments, such as air traffic control, aircraft cockpits, fire fighting, and emergency medicine.

The functions of communication in supporting shared mental models are two-fold [100]:

1. Communication during task execution refines team members' shared mental models with contextual cues, which may result in more accurate explanations and predictions of the team task demands.

2. For maintenance purposes, communication is needed to keep the shared mental models up-to-date with regard to the changes that occur during task execution. Especially in dynamic or novel situations, communication is needed to maintain an up-to-date shared mental model of the situation and to adjust strategies or develop new ones to deal with the situation.

Weak shared mental models in asynchronous tasks like software development can lead to uncoordinated activities and productivity losses due to re-work and missed deadlines because important task interdependencies may not be adequately managed [48]. This is further aggravated in geographically distributed environments where team members have fewer opportunities to interact with each other, and often have to do so through less rich media (e.g., e-mail, shared databases, etc.), thus making it more difficult to develop shared mental models.

These findings are also becoming troublesome in the rapid evolving ontology engineering field where geographically distributed knowledge engineers work on tasks that have dependencies among each other.

In summary, effective communication plays a critical role in the successful orchestration of a global software development project in order to facilitate coordination and build team shared mental models [38]. Effective communication support for distributed ontology development, which has similar development processes and life cycle models as software engineering, will be of equal importance.

The importance of informal communication has led to a variety of tools designed to stimulate casual conversation among workers at different sites. Those tools mainly fall into two categories [69]:

1. Tool support to compensate for the loss of frequent informal communication.
2. Tools that bring group awareness to the work and keep team members informed of each other's work status.

Designing tools to support and enhance informal communication is the major approach to bridge the missing link in distributed development work. Researchers in the GSD field have also suggested designing the organization and assigning work in order to reduce the amount of informal communication required; although this will not eliminate the need for informal communication, the goal is to reduce it to a more manageable level [96].

5.2 Documentation and Knowledge Management

The software development community has realized that a large number of problems can be attributed to un-captured and unshared knowledge [84], specifically the need of knowing 'who knows what', the need for distance collaboration, and the need for recording the lessons learned and best practices. Information and knowledge obtained during meetings, email correspondences, and instant messaging need to be captured easily, stored and shared effectively. The distribution of resources and developers in space, and time combined with the the dynamic evolution of knowledge make the use of tools for knowledge management a necessity in software development. The content management and bug/issue reporting systems used in software engineering may well be useful in the ontology engineering field.

Poor documentation can also lead to ineffective collaboration. In addition to having the design and the system well documented, keeping the documentation up-to-date is equally important. To prevent incorrect assumptions and ambiguity and to support

maintainability, documentation must be current and reflect what various teams or members are working on [66].

A close examination of the collection of tools used for documentation and knowledge management in software engineering will provide us with insights on how effectively they can be used effectively for ontology development.

5.3 Version Control and Change Tracking

In software engineering, there are a variety of tools for version control, source code management, and change history tracking, such as, CVS, RCS, and SourceSafe. Even so, advanced technologies for supporting version control and change tracking in ontology development has yet to emerge. Version control systems in software engineering are not directly applicable to ontology development. For instance, the minimum unit of version control in CVS is file; the system does not have the notion of concepts, instances and relationships, which are the basic building block of an ontology, and are the subject of versioning and change tracking.

The following diagram illustrates how the developers at the NCI (National Cancer Institute) handle version control and change tracking while developing an ontology (NCI ThesaurusTM) for cancer research. Due to the rapid rate of evolution of cancer science and the demand for developing an ontology for it, researchers at the NCI find it vital to do multi-editor development [64]. In order to overcome the difficulties described above, they invented this work flow system to manage multi-editors' work:

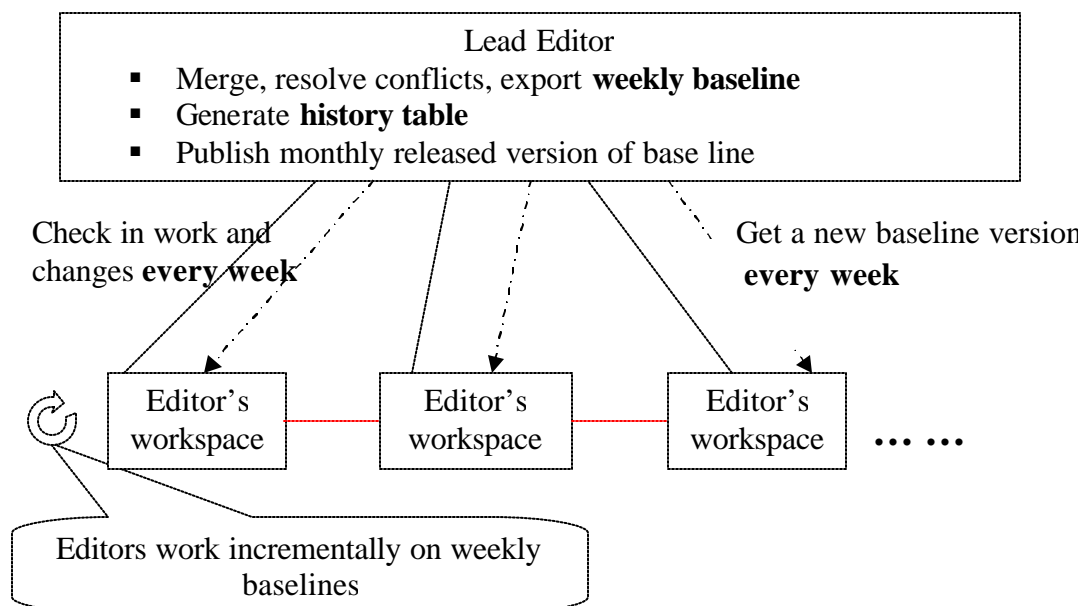


Figure 9: Work flow diagram for developing NCI Thesaurus™

We created the above work flow diagram based on the description of the system given by Hartel in [64]. As illustrated in this diagram, each week, the common baseline of the ontology is loaded onto each editor's work space. Each editor also gets a work assignment from the lead editor. Each editor then sends a set of changed concepts for a particular assignment to the lead editor weekly. The lead editor takes the bulk of the work of merging the changes submitted by multiple editors, resolving the conflicts, using a software application to produce and update a history table for each concept, and in the end, produces a consistent baseline version. The process repeats after each baseline is released.

The history table is published as an API and shared among developers after it is exported. Each record in the table reflects the trace of evolution of a particular concept, and is the source of information for developers to understand the history and state of the system. The structure of a history table is presented in Table 6.

Table 6: Structure of the concept history table in the NCI Thesaurus™

Column Name	Description
History_ID	Record number
Concept_Code	Concept code
Concept_Name	Preferred name of concept
Action	Edit action
Reference_Code	Referenced concept code
Edit_Date	Timestamp
Edit_Name	Name of edited NCI Thesaurus™ schema
Host	IP address of editor's workstation
Published	Publication state of history entry

The work flow constraints in the system are as follows:

- Individual editors cannot commit their changes at any time
- Merging, which is a large undertaking, has to be handled by one person, the lead editor, who does not necessarily understand all aspects of each developers work and the rationale behind every merged of concept.

The red lines in Figure 9 suggest collaborations among individual editors which are not supported by the system right now. The idea is that by increasing collaboration among ontology editors, conflicts about concepts can be discovered earlier, and possibly resolved before submitting to the lead editor. Fewer conflicts will be helpful in reducing the development time and reducing the work load of the lead editor.

The development environment used by the Thesaurus project at NCI is the commercial software system called the Apelon Terminology Development Environment, the Apelon Workflow Manager©, and the Apelon Distributed Terminology Server [1].

In projects where such an advanced system to support distributed development is not available, the development resources have to be placed in a collocated fashion, and

knowledge engineers have to carefully coordinate the work among team members to make sure that two developers do not work on the same part of the ontology in order to avoid the complication of resolving conflicts.

5.4 Summary

This chapter presents three significant challenges that collaborative ontology engineering is facing: communication problems intensified by distance; documentation and knowledge management; version control and change tracking. Although these challenges are inferred from the findings in the GSD domain, they also manifest themselves in ontology engineering field. Our team has also had experiences with these problems while developing a bibliography ontology and we believe these problems warrant a set of solutions that would be beneficial to the existing ontology development tools and practices.

The next chapter looks at tools and technologies that have been used to support collaborative software engineering, and further discusses their potential for being used by or integrated into ontology development tools.

Chapter 6 Using Groupware for Ontology Engineering

We have observed many of the problems and insufficiencies in collaborative ontology development; but the growing demand for ontologies for the semantic web, artificial intelligence and many other scientific domains is calling for the rapid development of higher quality ontologies. The ontology development community needs tools to support their collaborative development work.

Inspired by the success of groupware and the promise of newly emerging peer-to-peer technology, this chapter presents functions and characteristics of some groupware technologies that have been used in areas such as GSD (global software development).

6.1 Instant Messaging

As instant messaging gains its popularity, sending text messages and, increasingly, audio, video, files of any sort, interactively, it is already being put to use in business environments [40]. Workers are using it to share documents remotely, ask a quick question or exchange notes during a meeting, all despite being hundreds of kilometers apart. Instant messaging has also established itself in cell phones, PDAs and many other electronic devices as part of the short messaging service (SMS). The instant messaging “buddy list” provides group awareness by continuous updating the presence information of friends and coworkers. It brings synchronous, real time communication to work places.

In summary, instant messaging applications are primarily used [80] to:

- Ask quick questions or request clarification;
- Coordinate and schedule work tasks;
- Coordinate impromptu social meetings;

- Keep in touch with work colleagues.

Some studies found that most instant messaging communications tend to be brief, with media switching and multitasking prevalent among instant messaging users [72].

While instant messaging is becoming as ubiquitous as email, they are limited in terms of expressiveness, since they convey nothing other than the words we put on the screen; the tone, expressions, and other nuances we have in face-to-face communication that are essential for creative and innovative work are not communicated in instant messenger dialogues [54]. Currently, we use emoticons in email and instant messaging to supplement the lack of emotion; there is still a long way to go before we can bring all the intimate elements of face-to-face collaboration into remote collaboration supported by software.

In the interface design for instant messaging applications, there seems to be no single answer to the question of how to best handle the interruptive nature of instant messaging. The task of dismissing pop-up windows, recognizing and processing audible alerts have the potential to distract the user; to what level those signals constitute interruptions really depends on user preferences or tolerance, the nature of the work at hand, and even varies throughout the day (for example, interruptions become more acceptable in the afternoon for some). The design of the interface should look at ways of making user alerting mechanisms customizable to accommodate these different needs.

Awareness of what one's distant colleagues are doing and their availability for interaction is often found to be useful in initiating further interactions. Instant messaging applications provide peer presence information and support for informal communication that may be helpful in supporting collaborative work.

There are many proprietary (i.e., MSN, AOL, Yahoo, and ICQ) and open source instant messaging services available. Each of these instant messaging systems has its own communication protocols and are therefore not interoperable. In order to enable the interoperability among all the proprietary and open source systems, the Internet Engineering Task Force (IETF) is trying to establish a standard for instant messaging: the Instant Messaging and Presence Protocol (IMPP) is at the stage of request for comments (RFC 2779), it is expected to become a standard when it stabilizes in the user and research community.

The problem with simply using general purpose instant messaging applications in ontology development environments is that IM is not integrated with the ontology development environment and it is hard to associate the message with contextual information of the host system. The tight integration of an instant messaging system with an ontology development system at both the data and presentation levels would be an important step towards fully harnessing the power of instant messaging. Implementing an instant messaging system from scratch is obviously not necessary; it is worthwhile to investigate various IM systems or platforms and choose candidates for integration.

6.2 Web Portals

A web portal is a web site that provides information content on a common topic, for example, a specific city or domain of interest. A web portal allows individuals that are interested in the topic to receive news, find and talk to one another, build a community, and find links to other web resources of common interest [25]. Portals create a customized single gateway to a wide and heterogeneous collection of data, information, and knowledge. They also provide different kinds of personalization so that content is

presented in a manner that suits the individual's role within the organization and reflects personal preferences [84]. Portals support knowledge distribution as well as organization of the display information. Web portals use web browsers as the infrastructure to provide a ubiquitous accessing medium for distributed content and knowledge.

Figure 10 shows a knowledge management system architecture model in an enterprise setting [76]. The shaded area illustrates the position of the knowledge portal in a knowledge management system. Through portals, knowledge can be distributed to different users and applications in the business application layer.

Examples of web portals support team collaboration include Bugzilla [3], Tack+ [22], and group web log/project log systems (e.g., Drupal [7]).

A project log is a collaborative web log focusing on a specific topic. Members write about issues relating to the development of the project by posting messages to a website where the messages are stored for chronological display and commentary.

A project log is good for logging change information and managing group knowledge in general. In some ways a project log can be compared to a mailing-list, the difference being that the user does not have to send or receive e-mail to participate in a discussion. The project log provides web forms to post messages or comments. Because they can use many of the features web technology provides, project logs can be equipped with a lot more helpful features (e.g. customization and personalization).

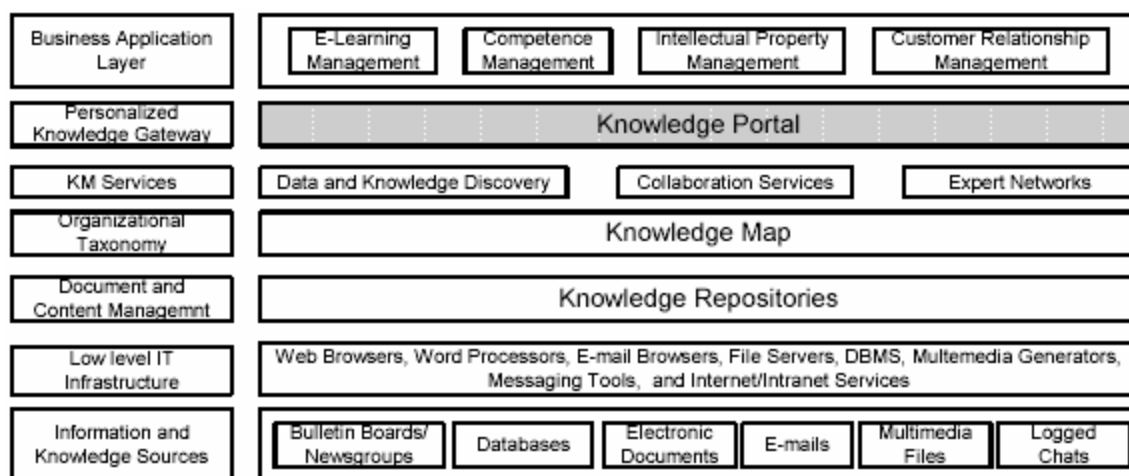


Figure 10: A KM architecture model shows how various parts of the system obtain, store, classify, and distribute knowledge

Project logs can become the center stage for communication to the whole project group. They are a convenient source for finding updates about a project's status.

Issue tracking systems are similar to web portals in the sense that they both provide a single gateway and personalized access portal to a central knowledge repository. Issue tracking systems focus on bug/issue reporting, tracking and managing, while project logs can be used in ontology engineering teams for managing knowledge bases, sharing project information and group discussion. Issue tracking systems are specifically designed for issue and bug reporting, inter-related task tracking, work status and progress reporting.

Many older issue and bug tracking systems are client server based desktop applications that not only require setting up a server but also require the installation of clients on each individual user's computer. The newer systems developed in recent years are mostly web based and use a web browser as the universal client. The primary benefit of web based systems is the reduced cost of making the client portable over multiple hardware and software platforms together with the decreased cost of distributing clients,

release patches, and upgrades. The primary concern left is to ensure the client is supported by web browsers from different manufactures. Leveraging standard web browsers makes the system more accessible to a larger user community, thus improving the rate of user adoption.

Regarding server side technology, Microsoft Windows-based bug tracking systems usually require Windows 2003 or Windows XP server as the operating system with an application server and a Microsoft SQL Server for database support. Most free bug/issue tracking systems are UNIX-based and usually require a MySQL or Postgres database. Table 7 shows an overview of system configuration for the most common bug/issue tracking systems; integration with version control systems is not available on all systems.

Table 7: Overview comparison of Bug/Issue tracking system requirements

	Windows (2003/xp server)	Linux/UNIX
Database	MS SQL, MySQL	MySQL, PostgreSQL
Web Server	IIS server	Apache, iPlanet
Application Server	WebSphere, JBoss	WebSphere, JBoss, J2EE,
Programming Language	ASP, Visual Basic, C#	Perl, PHP, JSP, Java, C/C++
Version Control Integration	SourceSafe	CVS
Initial Cost	Commercial software	Commercial software and open source software (GPL license, BSD license)

Although free software and open source software have a low initial cost of procurement, for calculating the total cost of ownership, the following factors need to be taken into account when selecting the right bug/issue tracking system.

1. IT infrastructure (i.e., whether CVS or some database system is already available in the environment)
2. Support (i.e. whether there are IT professionals in your organization to provide daily operational and maintenance support, or whether you have to rely on

support from the software vendors or commercial support for open source software)

3. the skill of individual users (the cost of training for using the system may vary largely depending on the skill level of the software system users)

When the system and users requires technical support and it is not freely available, the expense for hiring professional support for an open source system can be high enough to drive the total cost of owner ship (TCO) higher than a commercial system.

6.3 Peer-to-Peer Networks

Peer-to-Peer (P2P) generally refers to technology that enables two or more users to collaborate spontaneously in a network of equals (peers) by using specialized information and communication systems without the necessity for central coordination [5]. One might still think P2P systems are useful for illegal music swapping, but further research would reveal this to be a hasty conclusion [31]. This section takes a closer look at peer-to-peer technology, and many of the benefits it promises to deliver.

6.3.1 Taxonomy of Computer Systems

All computer systems can be categorized as centralized or distributed. Distributed systems communicate and coordinate their actions within a network and may be further divided into those implementing the client-server model or the P2P model.

When all clients only communicate with a single server or a cluster of servers, it is called a flat client-server model. When servers are arranged at various levels in order to improve the scalability of the entire system, it is called a hierarchical client-server model.

The P2P model can either be pure or hybrid. In a pure model, there is no centralized server. In a hybrid model, peers initially contact a central server to get meta-information, or to verify security credentials [85], and from then on, peer-to-peer communication is carried out without the involvement of servers. Examples of the hybrid model include Groove [9], Napster and Magi [11].

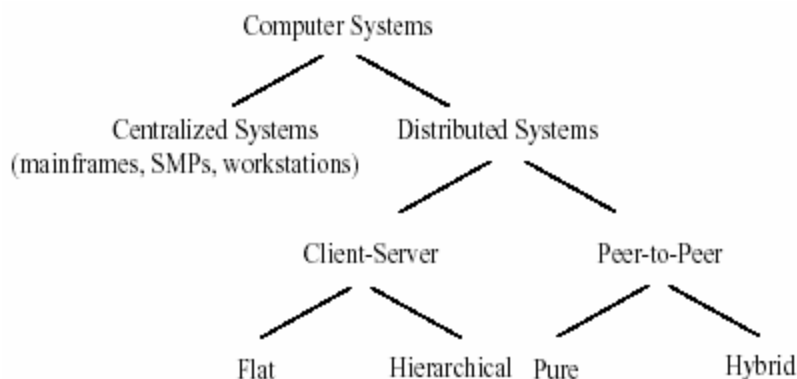


Figure 11:A Taxonomy of Computer Systems [85]

6.3.2 What are Peer-to-Peer Networks?

A peer-to-peer system, (or P2P), leads many of us to think about things like decentralization, self-organization, anonymity, etc. However, it is hard to give a simple definition of what P2P is and what it is not; no agreement has yet emerged either in research or industry. Since P2P is many things to many people, P2P can be a mind set, an implementation choice, a property of a system or an environment [85].

The best example for peer computing, in the eyes of the public, continues to be Napster whose number of users reportedly reached 50 million [47] at the peak of its service. Before it was forcibly shut down, it demonstrated that it was easy to be adopted, and that it had outstanding scalability. Its business model was soon replicated by hundreds of companies.

Fortunately, the domain of peer computing is much broader than Napster. Some of the benefits of a P2P approach include: improving scalability by avoiding dependency on centralized points, eliminating the need for costly centralized infrastructure by enabling direct communication among clients; and enabling resource aggregation [85].

The Peer-to-Peer Working Group (P2PWG), created by a group of leading IT companies including Intel, Cisco and HP, is looking to draft industry standards for P2P technologies in order to lead the advancement and interoperability of peer-to-peer computing. It defines P2P as the sharing of computer resources and services by direct exchange between systems [16]. These resources and services include the exchange of information, processing cycles, and files. Milojevic [85] considers P2P as a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. Biz2Peer technologies, one of the leading P2P technology companies [2], defines it as the coordinated use of geographically distributed resources in the absence of central control, based on direct exchanges of information. Finally, Clay Shirky at O'Reilly [19] defines P2P as:

A class of applications that takes advantage of resources storage, cycles, content, human presence available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.

If a peer is defined as one that is of equal standing with another [33], then a P2P system is one in which autonomous peers depend on other autonomous peers for information and computing resources. Peers are autonomous when they are not wholly

controlled by each other or by the same authority; the peers connected to their peer network make up the system as a whole [85].

6.3.3 Historical View

The concept of P2P is not totally new. In the 1960s, the Internet was originally conceived as a peer-to-peer system, and the goal of ARPANET was to share computing resources. The first few hosts on the ARPANET were from a handful of universities including University of California at Los Angeles, University of California at San Barbara, and the University of Utah. Those hosts were already independent computing sites with equal status. The ARPANET connected them together not in a master/slave or client/server relationship, but rather as equal computing peers [11]. It later evolved into a client-server system and is now dominated by large organizations and corporations. Today's Internet expands the horizons of the ARPANET infrastructure along several dimensions, such as scale, performance, and higher level functionality.

In UNIX, the talk utility has always been peer-to-peer, and while File Transfer Protocol (FTP) is based on a client-server architecture, its usage pattern is symmetric, meaning that any individual can run an FTP server on their home computer. Furthermore, the Domain Name System (DNS) has the characteristics of a hierarchical peer-to-peer system [19]; it scales very well from a few hundreds nodes to millions of nodes on the Internet at this stage. Lessons learned from the DNS system are directly applicable to our contemporary peer-to-peer data sharing applications.

The increase of computing power and storage capacity on personal computers has opened a market for a new form of distributed system, a P2P system. Many believe that there are three fundamental enablers/drivers for peer computing that recently reached the

point where the technology began to mature, and thus make P2P system an important system model today, as opposed to years ago. They are [47] the:

- ubiquity of computing hardware usable as a peer platform;
- ubiquity of network connectivity, and;
- need for local control over networked application;

Both researchers and industry are looking for the killer application for P2P. At this stage, many people believe that an application that finds the right niche to provide users with proper local control over collaboration, communication and customization is likely to be this “killer application”. Imagine doing some financial analysis on your bank’s web site – financial analysis that you want, but had not been or may not ever be programmed into your bank’s server [47].

6.3.4 Different Peer-to-Peer Systems

Figure 13 shows the distribution of the existing tools in the following dimensions:

- File sharing. File sharing in P2P is a way of aggregating many peoples’ file systems into one logical, integrated “file system” to which access is optimized in spite of bandwidth and disk space constraints [47].
- Collaboration and communication. The inherently ad-hoc nature of P2P technology makes it a good fit for user-level collaborative applications.

Collaborative P2P applications allow users to collaborate, in real time, without relying on a central server to collect and relay information and without being constrained by the collaborative features provided by the server. Instant messaging and coauthoring word processing applications are examples of this

class of application. There still exist some technical challenges in implementing this type of system, like fault tolerance and real-time constraints.

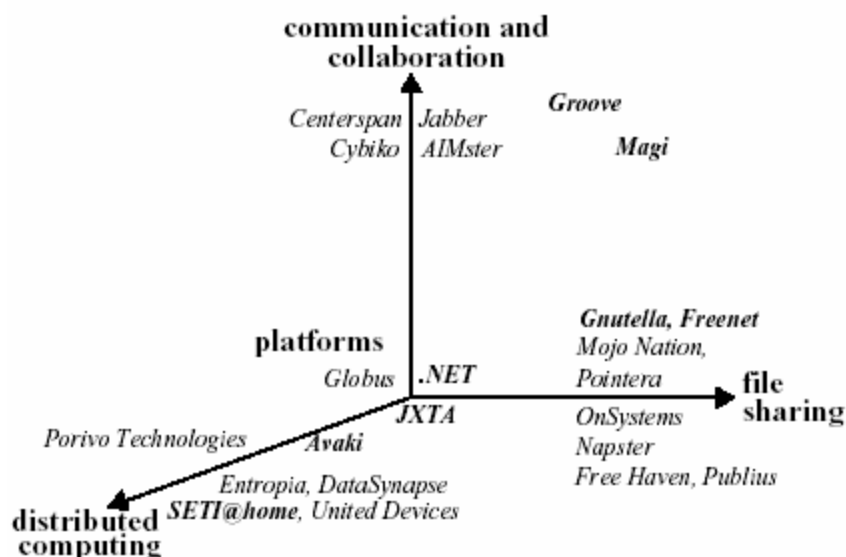


Figure 12 A classification of existing P2P systems [85]

- **Platforms.** Platform for peer-to-peer infrastructure aims to provide solutions for the challenges mentioned above, in addition to support for primary P2P components, such as naming, discovery, communication, security, and resource aggregation. The JXTA platform [12] is Sun Microsystem's approach in providing strong interoperability in P2P systems.

P2P systems often bring many benefits such as cost sharing and reduction, improved scalability and reliability, resource aggregation and interoperability, increased autonomy, anonymity and privacy and enabling ad-hoc communication and collaboration.

Decentralization is the most prominent characteristic of a P2P system. Centralized systems are ideal for some applications and tasks, such as transaction management, where access rights and security are more easily managed. However, a centralized system is costly to set up and maintain. On the flip side, in a fully decentralized system, users have ownership and control over data and resources. This can be problematic in terms of

security and discovering the first peer to which one should connect. For this reason, we see many P2P systems built as hybrid approaches.

The most significant benefit of decentralization is improved scalability. Sometimes, in order to spare other desirable features, such as determinism and performance guarantees, hybrid systems often centralize some of their operations, such as centralized file storage [85]. SETI@home [35], a project started at University of California at Berkeley to search for extraterrestrial intelligence, has the implementation characteristics of a hybrid P2P system. Its developers were hoping for approximately 100,000 participants initially, but they surpassed this number within a week. Currently, the system supports more than 3,000,000 contributors.

It is very hard to predict the scale and load of a P2P system at any given time, which in turn requires self-maintenance and self-repair of the system. There is still a serious challenge in P2P system design in terms of handling intermittent peer availability, variances in network latency and bandwidth.

Other potential benefits of a P2P system include reduced cost of ownership, ability to handle ad-hoc connectivity, and more efficient usage of processing power and storage capacity.

6.3.5 Challenges to Peer-to-Peer Technology

Despite many of the benefits that come with P2P technology, there are challenges and open questions to be answered.

1. All P2P system must deal with the lookup problem, which is how to find any given data item in a large P2P system in a scalable manner, without any centralized servers or hierarchy [31]. There are two traditional solutions to this

problem on the Internet. One is to maintain a central database that maps a file name or object identity to the locations of the servers that stores the data. The other is to use a hierarchy (a well-known example is the Domain Name System) and start searches at top of the hierarchy, and reach the desired node by traversing references from node to node. Both of these two approaches are vulnerable due to the failure of central node or nodes that are sufficiently high in the hierarchy.

2. Interoperability is a serious issue because P2P systems are being developed by different companies using proprietary protocols and interfaces. Users are locked into a product once they start using it, which eventually leads to isolated P2P islands on the network. There are very few efforts in major P2P developers to improve interoperability. The P2P Working Group [16] led by Intel Corp. and the JXTA[12] project led by Sun Microsystems are the only two claiming to address this problem in their systems.
3. The implementation of P2P creates additional security challenges on an already security-challenged network [85]. Security remains the most complex and most important requirement for the P2P infrastructure [5]. Sharing local files or granting access to third parties can sometimes have critical side effects sometimes. For example, unencrypted instant messaging that circumvent firewalls remains a great concern for businesses.

It is hard to compare P2P with alternatives in terms of scalability, performance, security, self-organization, and fault-tolerance. This is mainly due to the fact that P2P is a relatively new technology and is not yet fully understood by the research community; it has to be widely used or receive comprehensive evaluation in industrial environments.

P2P systems will remain an important solution to certain inherent problems in distributed systems. However, it is not the only solution and it may not be appropriate for all problems, but it will be a strong alternative in supporting data sharing, collaboration, and distributed computing [85].

In the next section, we examine one of the Peer-to-Peer network designs and implementations from Sun Microsystems.

6.3.6 JXTA

JXTA [12] is an open source project founded by Sun Microsystems with the participation of a small but growing number of experts from academic institutions and industry. It was officially unveiled in April 2001. According to its mission statement, it explores a vision of distributed network computing using peer-to-peer topologies, and develops basic building blocks and services that would enable innovative applications for peer groups.

It defines a set of protocols for ad hoc, pervasive and peer-to-peer computing, which enables applications that are collaborative and communication-focused. As shown in Fig. 14, the JXTA protocols establish a virtual network overlaid on top of the Internet and non-IP networks, allowing peers to directly interact and organize independently of their network location [103]. This virtual network provides transparency to the complexity of the underlying network topology.

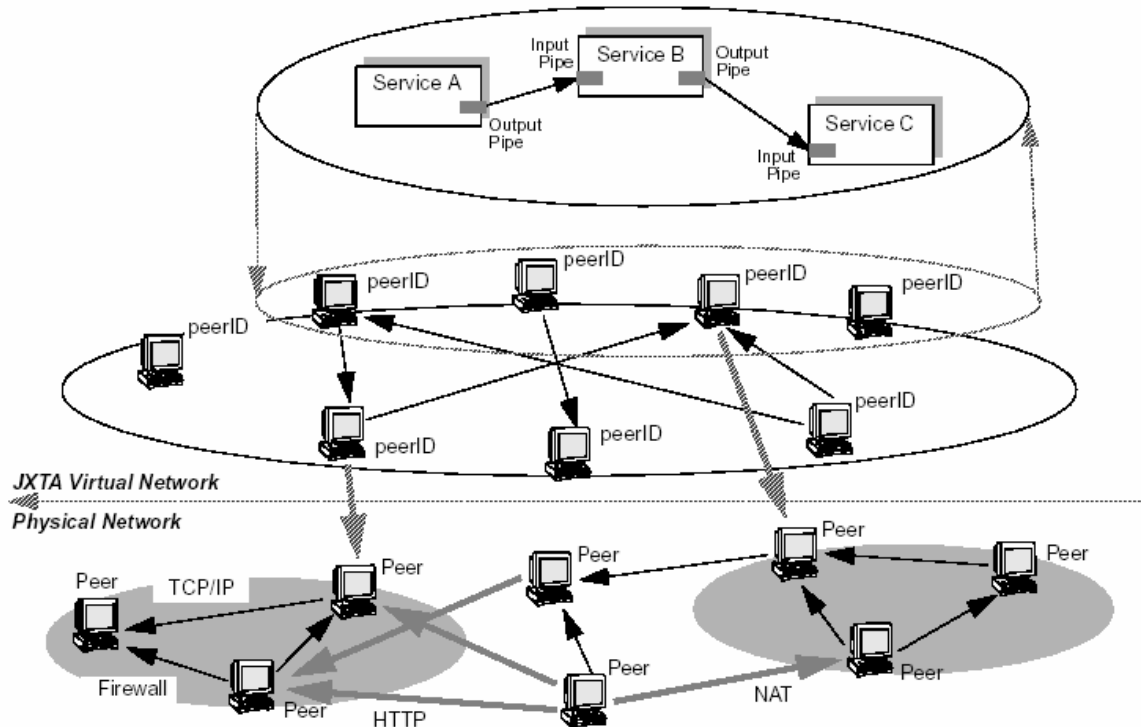


Figure 13 The project JXTA virtual network [103]

JXTA sees the most precious Internet resources such as information, bandwidth, and computing powers as being vastly under utilized, partly due to the traditional client-server computing model [56]. JXTA sets its objectives as:

- **Interoperability.** Enabling interconnected peers to easily locate each other, communicate with each other, participate in community-based collaborations, and offer services to each other seamlessly across different P2P systems and different communities.
- **Platform independence.** The JXTA technology is designed to be independent of programming languages (e.g. C or the Java programming language), system platforms (e.g. Microsoft Windows and UNIX operating systems), and networking protocols (e.g. TCP/IP or Bluetooth).

- Ubiquity. JXTA is designed to be implementable on every device with a digital heartbeat, including desktop computers, network routers, data-center servers, storage systems, sensors, consumer electronics, PDAs, and appliances,

An abstract view of the 3-layered JXTA architecture is shown in Fig. 14.

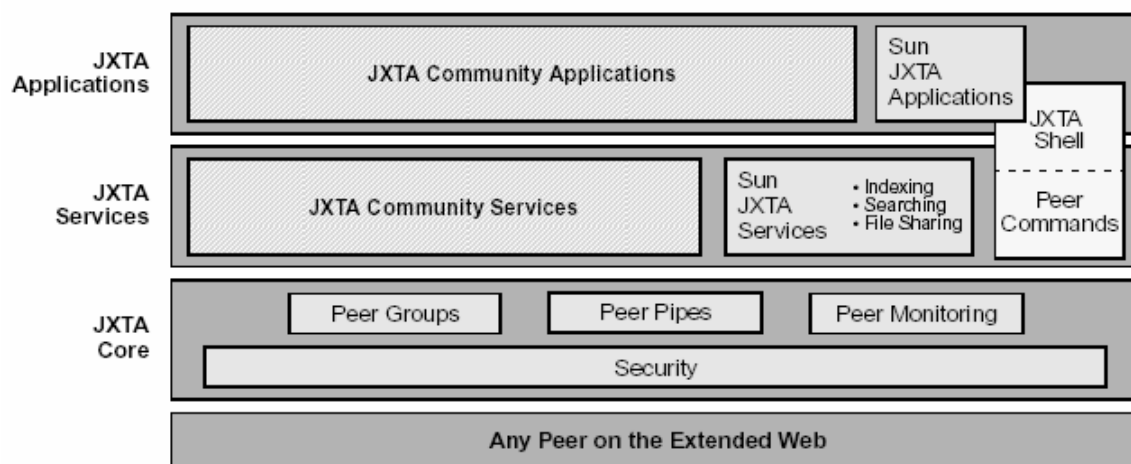


Figure 14 JXTA software architecture [56]

6.4 Groove Workspace

Developed by Groove Networks, Groove Workspace is an application for person-to-person communication and small group interaction and collaboration. Besides the instant messaging functions, Groove has a rich set of collaboration features. With Groove Workspace, users can create virtual shared spaces where they make immediate and direct connections to perform a wide variety of activities, from working on a project, brainstorming, planning an event, discussing issues, sharing drafts and proposals, to coordinating schedules. Users can take advantage of team file sharing and co-editing, project and meeting management, co-browsing, document review and approval, image markup, group discussions, chat and instant messaging functions. Teams of users across

the boundary of organization and firewalls can gather all the information in a shared context so that they can make better decisions and get work done more efficiently.

Groove Workspace is designed to operate on a hybrid peer-to-peer network; its prominent differences as opposed to the client-server approach are [6]:

1. The Groove Workspace sits directly on users' PCs, not on a web site or centralized server. All application logic and data is stored locally on the desktop of each member of a shared space. If there are four members in the space, there are four local copies of the space. The Groove platform uses an XML object store as its foundation.
2. All content, activity and gestures made by any member of the space are immediately duplicated on the desktops of the other members. Peer-to-peer communication is handled via XML message passing, which makes for lightweight traffic and efficient use of the network.
3. There is no 'master copy' of the data in a shared space. Each member's copy of the data is a peer in the network. The Groove platform ensures that the content and state of the shared space is synchronized across all members' machines.
4. The Groove architecture is a hybrid peer-to-peer architecture that includes a role for a server or switch, called the Relay Server, to broker connections between offline members (who may never be online at the same time) and members separated by network firewalls.

Group discussion sessions in Groove are less disruptive than instant messaging. The system tray icon is always on when user is logged on and there is no alerting sound or flashing icon when a message arrives or other users log in. Since many messages are

directed toward a group or other individuals, users do not feel compelled to read new messages immediately. However, for users who are accustomed to instant messaging style and rely on alerting signals, they have a high chance of missing the message on the discussion board or are not able to keep up with the discussion while multi-tasking.

In a graduate directed study course in the Department of Computer Science at the University of Victoria, Groove was used by a team in designing a software requirement engineering tool. The group used the Groove collaborative software to provide both formal collaboration capability (document-sharing) as well as informal communication (such as instant messaging, threaded discussion groups, and physical awareness). Team members found Groove's absence of an alerting cue a problem in recognizing who was participating in the discussion. It was reported several times that users were unaware that another user had logged in and had started monitoring the discussion and work until they engaged in instant messaging with each other. It was because that member's presence information was silenced when he/she logged in.

Since Groove Workspace depends on peer-to-peer technology, team members had to depend on "running into" a team member with the latest version of each document online. Due to this problem with sharing documents, users grew to mistrust the tool and resorted to e-mail to share documents [28].

In general, users found that Groove was best for highly collaborative and interactive tasks. It can potentially be used to support distributed ontology authoring among multiple developers, real time collaboration and communication, and support both ad hoc and structured communication.

Chapter 7 Collaborative support for Protégé-2000

We introduced the ontology editing tool Protégé in Section 4.5. The recent Protégé 2.0 beta release has added multi-user support capability by storing the ontology being developed in a shared database. This allows multiple users to read the same database and make incremental changes or changes that do not conflict with one another. There is, however, no support for multiple users trying to modify the same elements of a knowledge base or notification of changes made by other users. Concurrent changes to the same section will cause severe problems. This multi-user capability is still experimental at this stage and has not undergone any “real life” testing [17].

The word “multi-user” is used to describe the fact that the beta version has functions to support limited collaborative work among more than one developer. By looking at the dimensions of collaborative ontology engineering discussed in Chapter 5, we realize that there is still a gap between simple multi-user support and full-scale collaborative support. The discussions we had in Chapter 6 about using groupware to enable collaborative ontology development can be applied as a guide towards the long term design of collaborative features in Protégé.

We see our exploration of groupware technologies as a road map for designing and transforming Protégé into an ontology development environment with collaboration support in more than one dimension, however we believe the following work will need to be done in order to weave these technologies together and find a path to achieve this goal:

- First, an ethnographic field study on an ontology development team that uses Protégé should be done. We recommend this study be conducted in order to better

understand the type of the work of Protégé users perform, the context under which Protégé is used, and how their collaborative work is currently organized in the development team. The result could be used to determine which groupware technologies, without being invasive to the users or disruptive to the team work, and in what form, should be integrated or used to support collaborative work in Protégé. The aim is to ensure that groupware being designed and integrated with Protégé will be adopted by Protégé users and teams, and will effectively facilitate the collaborative work.

- Based on the results of the ethnographic field study, a preliminary set of groupware technologies could be selected as candidates for integration with Protégé. From here, it would be feasible to design and build a prototype system that incorporates Protégé and multiple groupware technologies, which in turn could be used for further user studies in order to verify and refine the design.

We conjecture that the specific needs for collaborative support will vary depending on the domain the ontology is being developed for and the setting of the development team. This is also part of the reason why we suggest the above studies. Several groupware technologies integrated under a highly user customizable system will likely be one of the ways to satisfy these diverse needs.

To further investigate the application of groupware technologies, we have explored several of the technical alternatives we discussed based on the needs of users in specific domains. We took Protégé as the core ontology authoring module for the development environment, and did an experiment to incorporate peripheral collaboration support around it, such as using the Peer-to-Peer technology. The experimental results are

presented in the next section. These results are useful for the Protégé developers when designing future multi-user versions of Protégé. Moreover, we also recommend the usage of web portals such as Drupal (<http://chiselog.chisel.cs.uvic.ca>), which, as an example, was customized and administered by Neil Ernst and used by the CHISEL Research Group for organizing and sharing group knowledge. Finally, we suggest the adoption of a change tracking/visualizing systems, such as PromptViz, which is a work in progress by my colleague David Perrin, also with the CHISEL Research Group. Finally, we present our design and implementation of a live bookmark – a system artifact that aims to diversify the means of information sharing in collaborative work.

7.1 Protégé-2000 and JXTA

This section describes the experiences and lessons learned during an attempt to implement a collaborative environment for Protégé on top of a peer-to-peer network.

We explored the idea (see Fig. 15) of making Protégé operate on top of a P2P network; where each Protégé instance acts a node/peer on the JXTA network, utilizing the network's basic messaging and security service. The goal was to enable informal and spontaneous communication among Protégé users through instant messaging, and provide group awareness as well. The network was to become the group knowledge repository (with gather and retrieval functions) for a Protégé project and the collective storage spaces on developers' workstations would be the physical medium.

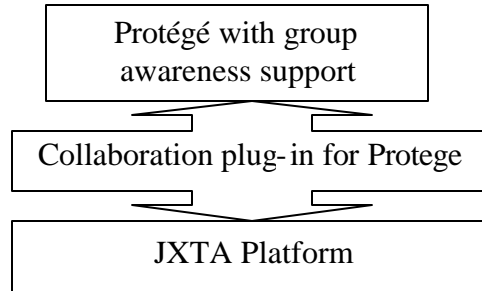


Figure 15: A tentative idea for Protégé-2000 collaboration plug-in

The JXTA platform has the notion of a user group, which allows developers to form working groups and join or leave a group as their interests or the focus of their work shifts. Compared with the Groove application that puts everyone in one default working group, this function helps collaborators filter irrelevant information during collaboration.

The challenge with implementing applications directly on the JXTA platform is that JXTA lacks a high level programming interface, and since it is still in the initial development stage, the constantly changing API makes it even more difficult to produce a prototype. This is also one of the reasons that many of the projects listed on JXTA's site remain in a crude phase [8].

In our experiment, the project myJXTA [14] was chosen to be used as the base to develop the collaboration plug-in for Protégé. It is an application designed with instant messaging, file sharing and resource searching functions.

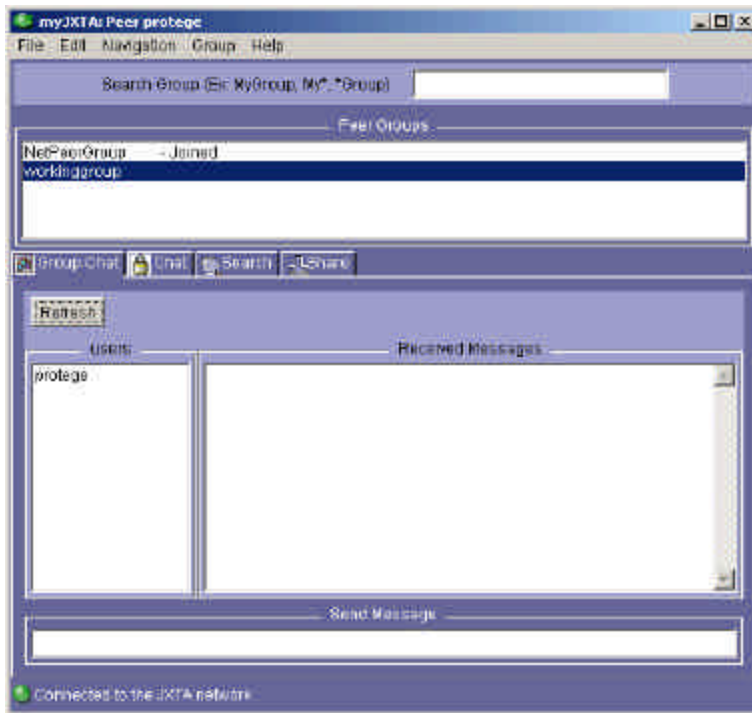


Figure 16: myJXTA application in group chat mode

Unfortunately, our attempts to integrate Protégé with JXTA were not successful. Although it is one of few working prototypes that demonstrates the ability of JXTA network, it has not yet reached the state that can be easily extended and integrated into other applications. A great deal of time and effort was spent on studying its sparsely documented code, and very little documentation could be found on more detailed design programming interface. Ultimately, it was determined that this single factor effectively prohibited any further progress to be made. A more informative programming guide for some of its modules was recently published on the JXTA project web site by the time of writing, but was too late to help our prototype for this thesis.

JXTA is not the first implementation of a P2P protocol and P2P network infrastructure, nor will it be the last. Though the project does make a good exploratory attempt at determining what such a framework might look like. Its success lies in moving from a research tool to useable framework and its adoption by developers and extension by the

community [63]. As a strong P2P network research initiative from SUN Microsystem and with a growing developer community, JXTA continues to be a promising candidate to grow into a mature P2P platform.

Though the infrastructure is still evolving towards its maturity and does not satisfy our requirements at this moment, we believe the exploration is on track towards achieving group awareness support. Moreover, our results suggest that this technology needs to further mature before it can support integration with Protégé.

7.2 Capturing Group Knowledge

Groups that manage and maintain their organizational knowledge have been found to have higher capacities for absorbing and making use of new knowledge [41]. Polly Allen in the CHISEL Research Group designed a system called Shrimpbib [27] to capture and share bibliography references accumulated during each group member's research.

Through this system, the knowledge about the literatures in a particular research area is no longer closely tied to a specific student and will be preserved and passed on even after the student leaves the group. New members of the group will not have to re-acquire this knowledge through experience, and can hence build their work on top of another. As this system starts being adopted and used in our daily research activities, it saves us precious research resources, such as time, funding, and human resources.

Figure 17 illustrates the system architecture of the Shrimpbib system, which is an integration of several tools. Group members usually use the web front-end for entering/capturing new bibliographical information, while expert Protégé users may directly use the Protégé interface for this task. In the current development phase,

members of the group need to use Protégé to share this knowledge, the goal for the next stage is to make this shared information also available through the World Wide Web.

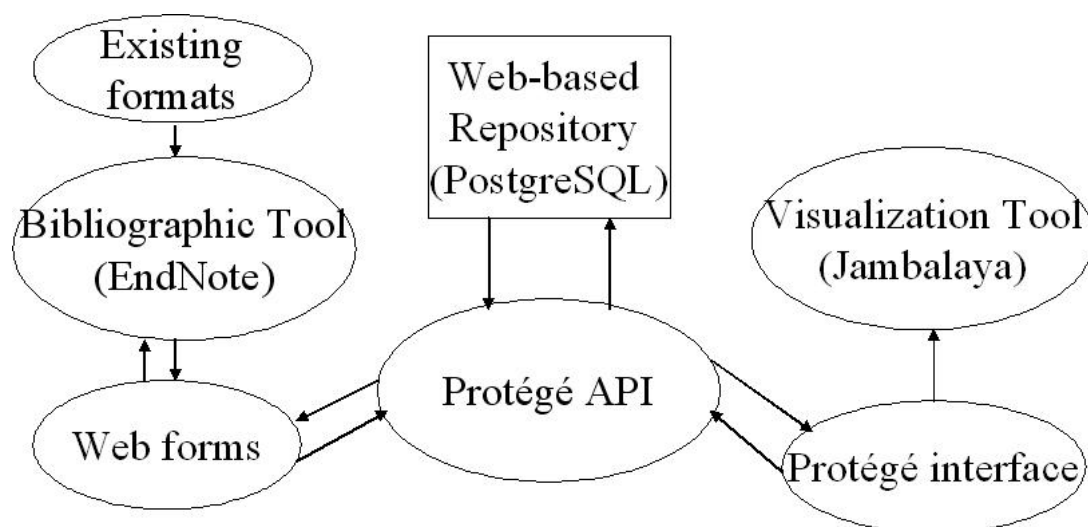


Figure 17: A diagram of the Shrimpbib architecture of integrated tools [28]

The Shrimpbib system enables us to easily capture and share bibliographical information; but we also experimented with a web content management system in order to accomplish a wider range of knowledge capturing and sharing tasks. We used Drupal (<http://drupal.org>), which is an open-source platform and content management system for building dynamic web sites. It offers a broad range of features and services including user administration, publishing workflow, discussion capabilities, news aggregation, metadata functionalities using controlled vocabularies and XML publishing for content sharing purposes. Equipped with a powerful blend of features and configurability, Drupal can support a diverse range of web projects ranging from personal web logs to large community-driven sites [10].

The CHISEL research group set up and configured Drupal (<http://chiselog.chisel.cs.uvic.ca>) as a formal mechanism for maintaining or managing the organizational knowledge created during each member's research. It is being used for web logging of research ideas and progress, as a group forum, for organizing and sharing bibliographical data among members of the group. It has been heavily used by some members of the group and is gaining more users as many of the advantages it brings start to surface.

Drupal meets many types of collaboration needs, and has found its way into supporting lots of other organizations, such as the community web portal for the Debian operating system (<http://www.debianplanet.org>) and the Linux user community (<http://kerneltrap.org>).

We believe that Drupal can be customized to support the collaboration of an ontology development team that uses Protégé because it can help to address the knowledge management and documentation type problem we discussed in Chapter 5.2. The tasks it supports and its usage scenarios include:

FORUM FUNCTION

The forum function captures and disperses project information among the Protégé users. Many of us may have had the experience where at some point email broke down for highly collaborative and interactive teamwork. As soon as we participated in a seemingly endless thread of “reply-to-all” emails with multiple attachments, we would readily recognize that email is poorly suited to project work that requires more than messaging back and forth [6]. The forum is well suited for group discussion including one or more topics. An ontology design or an implementation topic that merits group

discussion is a good example. Each discussion thread represents a coherent topic, and contains all the postings related to this topic. All these postings, contributed by team members representing their opinions and ideas, are organized under the “respond to” relationship (see Fig. 18) and presented in a hierarchical tree structure. A discussion thread naturally reinforces and facilitates a consensus building process, which is much needed and critical to a collaborative ontology engineering methodology.

Once recorded in this central repository, the group knowledge about a certain design idea/rationale or the creation of a concept in the ontology is not scattered in many members’ inboxes anymore. After visiting and reviewing the discussion threads in the forum, a new team member will be able to get a pretty good picture about the ins and outs about the team’s work. This will greatly help the member get up to speed. The indexing and searching functions in the forum improve the knowledge retrieval speed and reuse rate. The group knowledge preserved in the forum becomes invaluable information capital within the team and the organization.

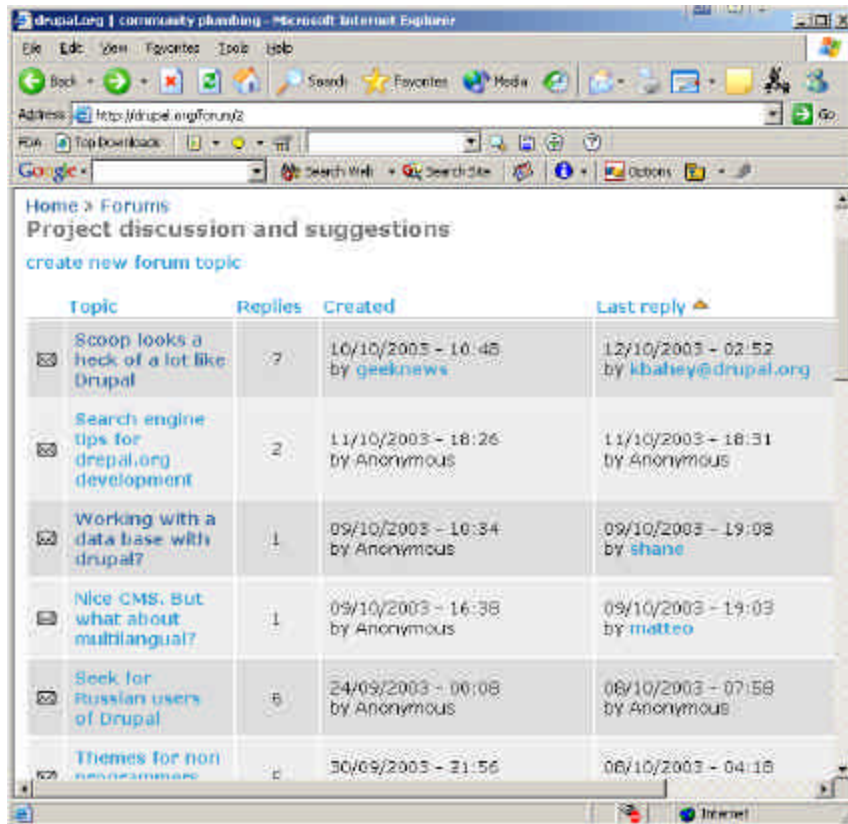


Figure 18: Discussion threads in Drupal forum

COLLABORATIVE BOOK FEATURE FOR DOCUMENTATION WRITING

The book organizes content contributed by users into a nested hierarchical structure that conforms to a book. It is particularly good for enabling multiple team members to cooperatively maintain an online user manual, or for Frequently Asked Questions (FAQs). It allows users to create and edit individual chapters, sections, etc. The advantage it brings is that updates to the documentation can be collected from anybody and at anytime as necessary. We can also easily use the collaborative book writing function to write a Frequently Asked Questions (FAQ) section on the project web site. The main benefit is that not all the questions and answers have to be written by the developers, as the user community will do a large part of it.

Two burdens are lifted by this function: one is the requirement for dedicated personnel to update the user manual or system documentation; the other is the need to rely on the web master to post the updated content.

Figure 19 is a snapshot of the collaborative book writing function.

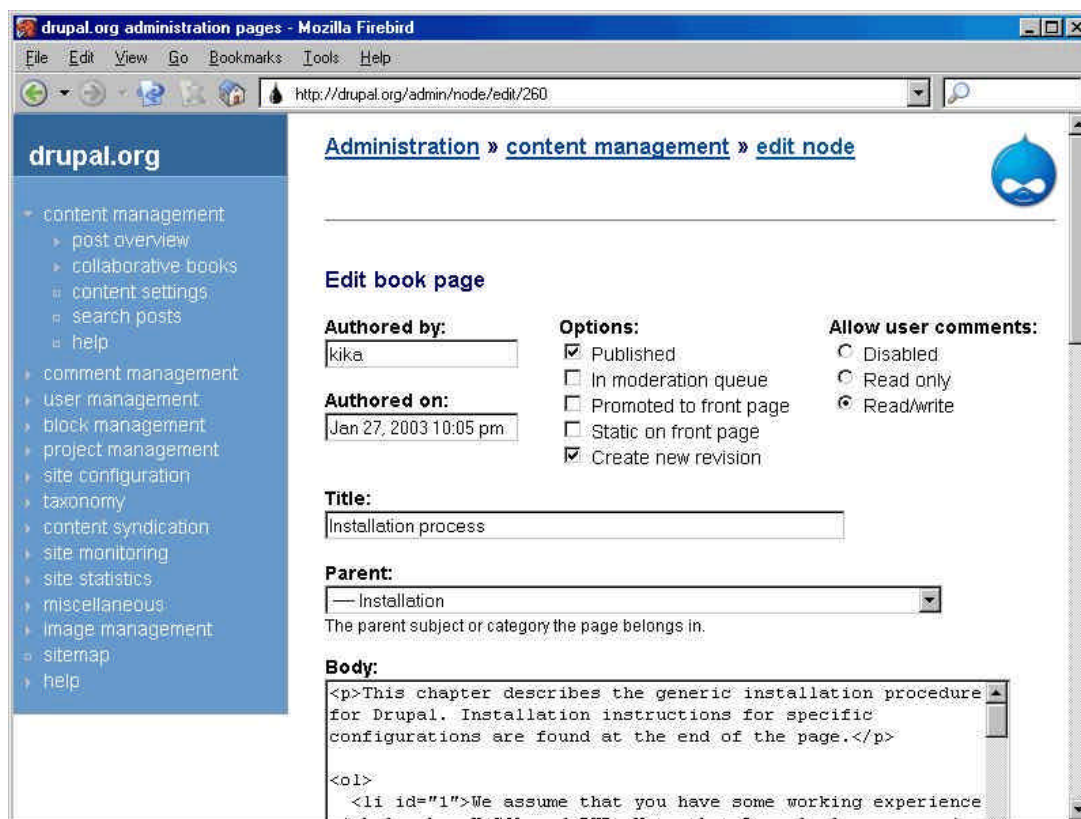


Figure 19: Screen capture for collaborative book writing in Drupal

WEB LOGS FOR INDIVIDUAL MEMBERS

Once registered as a user of the Drupal powered project web site, each member can start his own web logging. He or she can define access permissions for each log entry created under his or her account. Public log entries are available to the world, whereas private ones are hidden from all other users.

Members can be required to log the daily/weekly work progress/status report in their public log entry. For the benefit of each individual member, it is a good way of keeping

track of the work process and for storing notes and thoughts in a clear and structured format. From a managerial point of view, the project lead may, at any time, get a good handle on the working progress of the entire team.

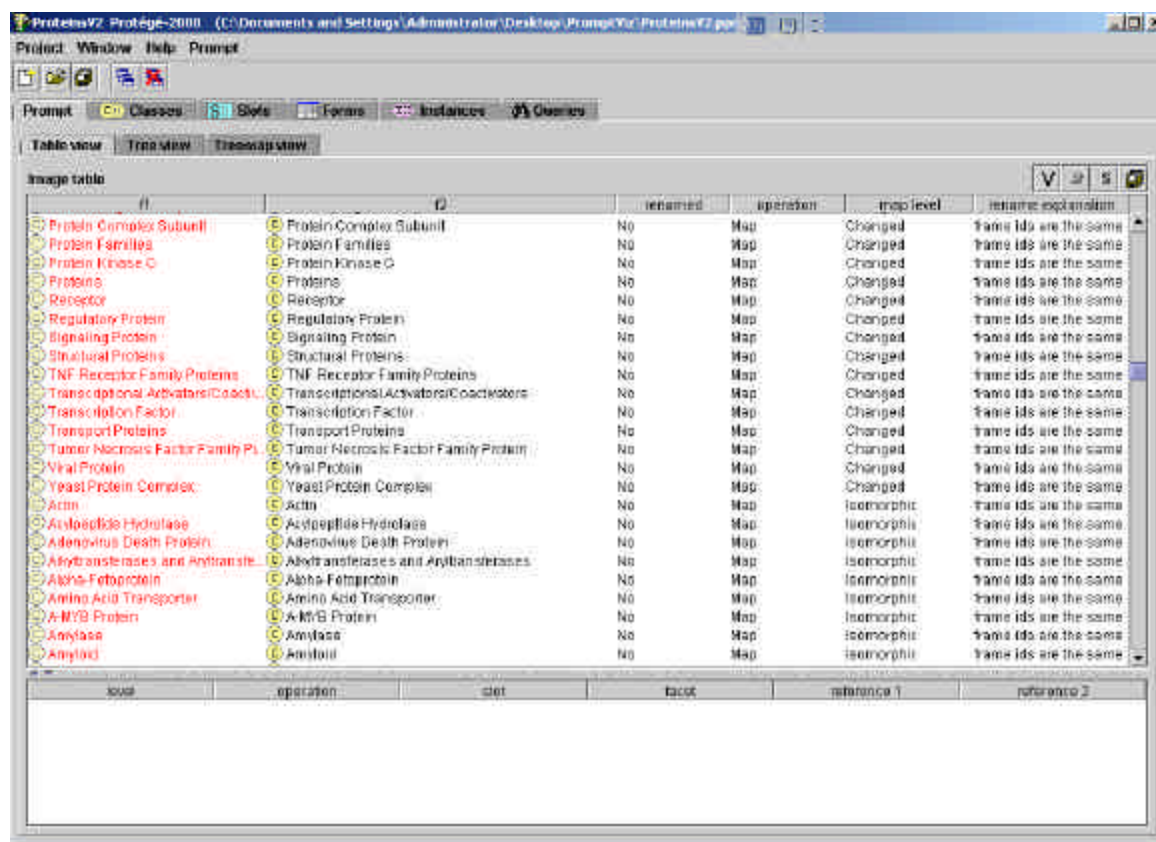
Browsing each other's log entries are another effective collaboration form in sharing new ideas and opinions. Even for a collocated team, merely relying on informal face-to-face discussions is not sufficient for group collaboration [27] mainly because knowledge and the spark of ideas will easily be lost. A web log entry is a formal way of maintaining and managing this knowledge. It becomes an inexpensive alternative to large-scale knowledge management solutions, such as Lotus Notes [71]. Effectively preserving and managing the knowledge accumulated from one project can be invaluable for the team and successive projects.

7.3 PromptViz

PROMPT is a plug-in for Protégé-2000 that has been developed to help knowledge engineers merge overlapping ontologies and find the differences between versions of the same ontology. One of its primary functions is to compare two versions of the same ontology and compute the differences between them. This is of great interest to us because it will help solve the change tracking problem we discussed in Chapter 5.3.

Since PROMPT is relatively new and is still in a late prototype stage, it has not been widely adopted by the Protégé community. When PROMPT does a comparison of ontologies, it generates a merged version of the two ontologies and a table of differences. PROMPT provides two different views of this comparison: one is a table view (see Figure. 20) that lists all of the frames from both ontologies and a description of the

change. The other is a tree view that displays the merged version of the ontologies in a tree using the “*is-a*” hierarchy [93].



i1	i2	renamed	operation	map level	rename explanation
Protein Complex Subunit	Protein Complex Subunit	No	Map	Changed	frame ids are the same
Protein Families	Protein Families	No	Map	Changed	frame ids are the same
Protein Kinase C	Protein Kinase C	No	Map	Changed	frame ids are the same
Proteins	Proteins	No	Map	Changed	frame ids are the same
Receptor	Receptor	No	Map	Changed	frame ids are the same
Regulatory Protein	Regulatory Protein	No	Map	Changed	frame ids are the same
Signaling Protein	Signaling Protein	No	Map	Changed	frame ids are the same
Structural Proteins	Structural Proteins	No	Map	Changed	frame ids are the same
TNF Receptor Family Proteins	TNF Receptor Family Proteins	No	Map	Changed	frame ids are the same
Transcriptional Activators/Coactivators	Transcriptional Activators/Coactivators	No	Map	Changed	frame ids are the same
Transcription Factor	Transcription Factor	No	Map	Changed	frame ids are the same
Transport Proteins	Transport Proteins	No	Map	Changed	frame ids are the same
Tumor Necrosis Factor Family Protein	Tumor Necrosis Factor Family Protein	No	Map	Changed	frame ids are the same
Viral Protein	Viral Protein	No	Map	Changed	frame ids are the same
Yeast Protein Complex	Yeast Protein Complex	No	Map	Changed	frame ids are the same
Actin	Actin	No	Map	Isomorphic	frame ids are the same
Adenopside Hydrolase	Adenopside Hydrolase	No	Map	Isomorphic	frame ids are the same
Adenovirus Death Protein	Adenovirus Death Protein	No	Map	Isomorphic	frame ids are the same
Alkytransferases and Aryltransferases	Alkytransferases and Aryltransferases	No	Map	Isomorphic	frame ids are the same
Alpha-Fetoprotein	Alpha-Fetoprotein	No	Map	Isomorphic	frame ids are the same
Amino Acid Transporter	Amino Acid Transporter	No	Map	Isomorphic	frame ids are the same
A-MYB Protein	A-MYB Protein	No	Map	Isomorphic	frame ids are the same
Amylase	Amylase	No	Map	Isomorphic	frame ids are the same
Amyloid	Amyloid	No	Map	Isomorphic	frame ids are the same

Figure 20: Table view in Prompt lists all the merged concepts from two versions of the same ontology. There are about one thousand changes listed in this list

Both views can sometimes become incomprehensibly long, and hard for the users to establish contextual links between the items in the list and the two versions of the ontology. This puts additional cognitive load on the users when trying to find out the relationships between concepts and the impact of a particular change to the overall ontology.

PromptViz tries to alleviate these concerns by using information visualization technologies. PromptViz creates a treemap view based on the data extracted by PROMPT. Figure 21 presents a snapshot of the treemap view in PromptViz. The treemap

is built using a zoomable user interface, allowing users take control, zoom in and out, and pan on the graph. Green nodes represent classes newly added, yellow nodes represent a class relocated to somewhere else, blue nodes represents the new location a class has been moved to. An arc between nodes represents the changes to the selected node's slot including addition and deletion of allowed values. Figure 22 displays a view where the ontology hierarchy is fully expanded to the leaf descendent level; the colored nodes we can see represent all the changes that happened between two versions of the same ontology. We consider this view to be a powerful way for developers to get an overview of all the changes that have happened. If desired, they can zoom to a particular area of interest to get detailed information.

Upon the release of PromptViz in the near future, we recommend adding it to the Protégé collaboration environment. Users can benefit from using it to understand the

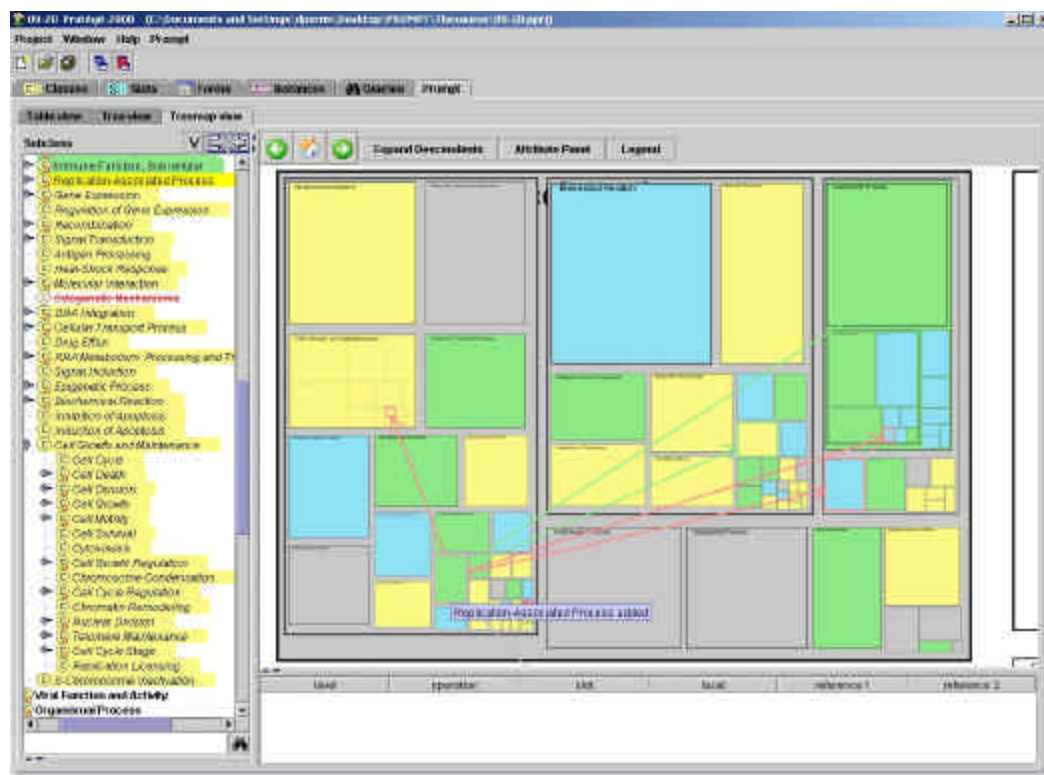


Figure 21: Treemap view in PromptViz

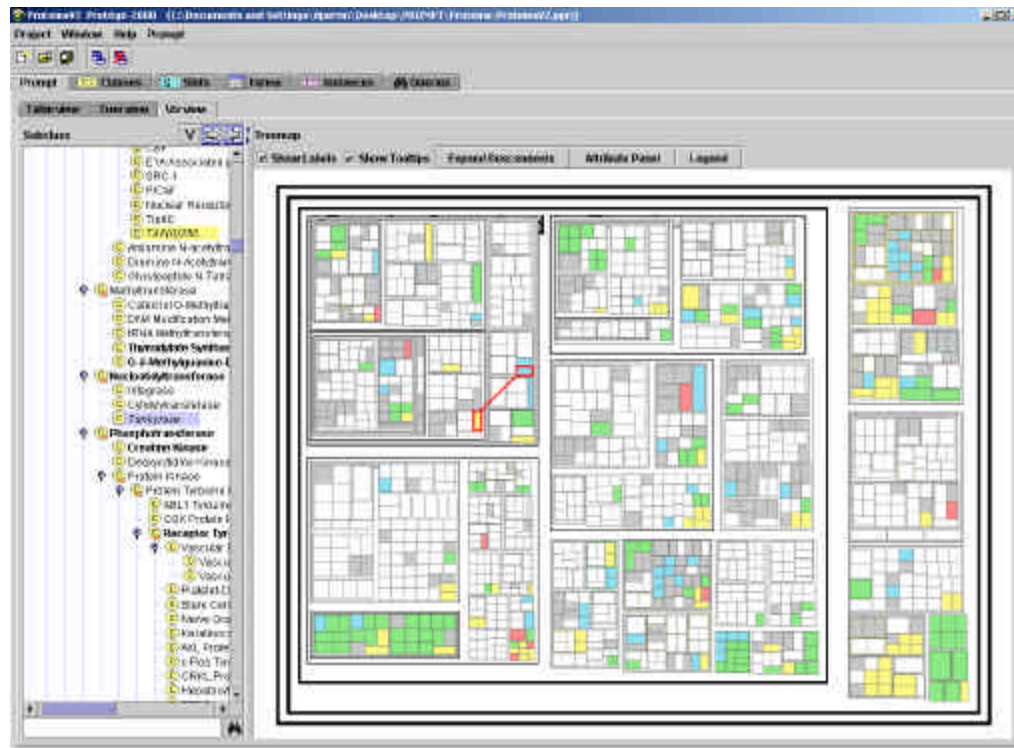


Figure 22: Treemap view in PromptViz, with all nodes open to leaf level

differences among versions – much like the way CVS system is used in software engineering to understand the evolution of code base and differences between code bases. The above recommendations are made with regard to some of the problems we identified and discussed in Chapter 5. They are among the many technological possibilities upon which we can build a full scale, fully integrated collaborative ontology development environment. Some of the recommendations are not confined to be only applicable to the Protégé environment and we believe they will also be valuable and inspirational for collaborative ontology authoring tool developers from other fields.

7.4 Live Bookmarks

While working with Protégé users, we have also discovered that visualization techniques can also be helpful in facilitating collaborative work among multiple developers. The live bookmark designed by the CHISEL Research Group demonstrates how this can be realized in a collaborative setting. Live bookmarks can be used to supplement the Protégé-2000 multi-user support and are discussed in the following section.

7.4.1 Advanced Visualization and Navigation Engine for Protégé-2000

Before we get into the details of live bookmarks, let us first look at the host application of live bookmarks, Jambalaya.

Jambalaya is an advanced information visualization and navigation system for Protégé-2000. The visualization techniques used in Jambalaya includes a zoomable user interface, a rich set of layout mechanisms, many navigational aids, information filtering and abstraction. The Jambalaya tool is designed to enhance how people browse, explore, model and interact with complex information spaces [101]. Users dealing with very large information spaces, like ontologies and knowledge bases, often face problems of disorientation. Jambalaya's solution to this problem is to present the hierarchically structure of information in a nested graph view. It introduces the concept of nested interchangeable views to allow a user to explore multiple perspectives of information at different levels of abstraction [101]. Jambalaya allows users to directly pan and zoom on the nested graph to achieve a continuous navigation experience and at the same time maintain contextual information; this technique also takes advantage of hyper link navigation wherever it is available in the document.

The left snapshot in Figure 23 shows the grid layout of concepts in the Jambalaya tab. Classes and instances are represented by blue and pink boxes respectively. The boxes are nested based on the “*is-a*” relationship with directed arcs representing slots that relate classes. When zooming in on a particular node, the contextual information is preserved. The right snapshot shows the treemap layout. Arcs are filtered in this view and the size of each node corresponds to the number of children it has. The treemap algorithm allows users to size the layout based on many different properties such as number of relationships or instances.

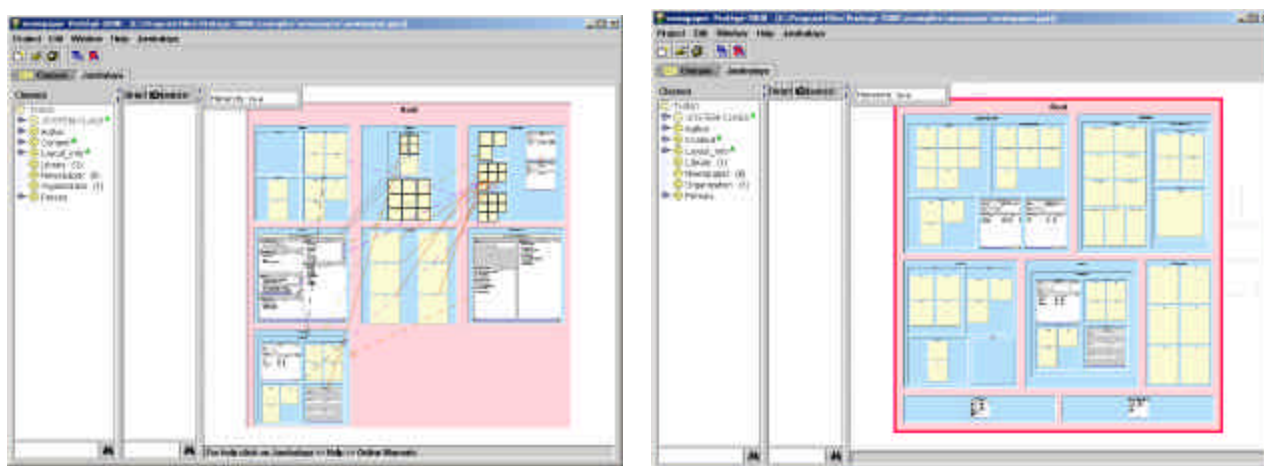


Figure 23: Ontology visualization in Jambalaya

7.4.2 Live Bookmarks in Detail

With its advanced interactive graphic techniques, Jambalaya helps users discover and reveal tacit knowledge, such as hidden patterns, tendencies, different perspectives based on levels of abstraction of the information space, and other knowledge that is often not visible to the users when the information is presented in its original format. For example, the cancer Bioinformatics Infrastructure Objects (caBIO) [23] defines entities and the relationships among the entities found in biomedical research. These entities/domain

objects are related to each other, and examining these relationships can bring to the surface biomedical knowledge that was previously buried in the various primary data sources [23], such as genomic and clinical information stored in the database. Jambalaya applies information visualization technologies to help biomedical researchers visualize and examine millions of entities and relationships.

Even with the tool assistance from Jambalaya, discovering underlying knowledge and producing useful or insightful views is a time consuming task that require a certain degree of knowledge and expertise with the knowledge base. It can be frustrating to users when they find some useful views and later have to go back to repeat many steps (such as a sequence of combined filtering, layout, and zooming in/out) in order to reproduce that view.

Another problem reported by Protégé users is the difficulty in effectively exchanging information about the knowledge base among developers. One of the difficulties for the knowledge engineers using Jambalaya is that when they have discovered a useful view and would like to discuss it with another developer at a remote site, they do not have an easy way of accomplish this, other than explaining to others all the steps required to reproduce the same view. This communication process can be slow and ineffective when the operation is complex. In this case, the developers working collaboratively need an effective way of communicating to each other about their different perspectives on the knowledge base.

One of the approaches to solve the problem described above is in the form of the live bookmark. A live bookmark is a lightweight system artifact created to:

1. capture the tacit knowledge created during the exploration of an information space, and facilitate the tacit-to-explicit knowledge conversion process.
2. support collaboration and data exchange among multiple knowledge engineers with live documents. The Jambalaya plug-in can use the live bookmark to restore the system back to the state represented by that view.
3. be used as part of the system documentation as bookmarks are easy to create and they contains a rich set of information. As they are lightweight and suitable for web publishing compared to taking an image of a particular view, live bookmarks can also be easily sent to others as an email attachment, or exchanged using instant messaging.

One of the differences between the live bookmark in Jambalaya and a bookmark in a web browser is that the live bookmark can exist as a standalone artifact to provide information about a particular view of the system and it allows for a certain degree of user interaction without requiring special tools.

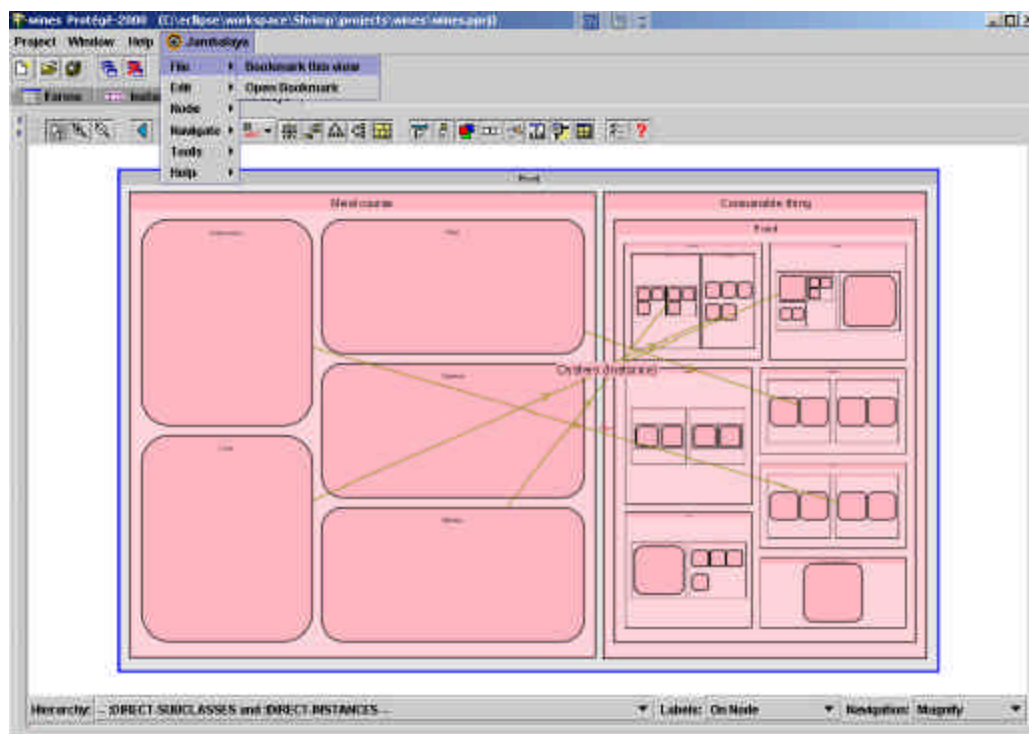


Figure 24: Using Jambalaya working with the wine ontology; user is about to bookmark this view in the scene

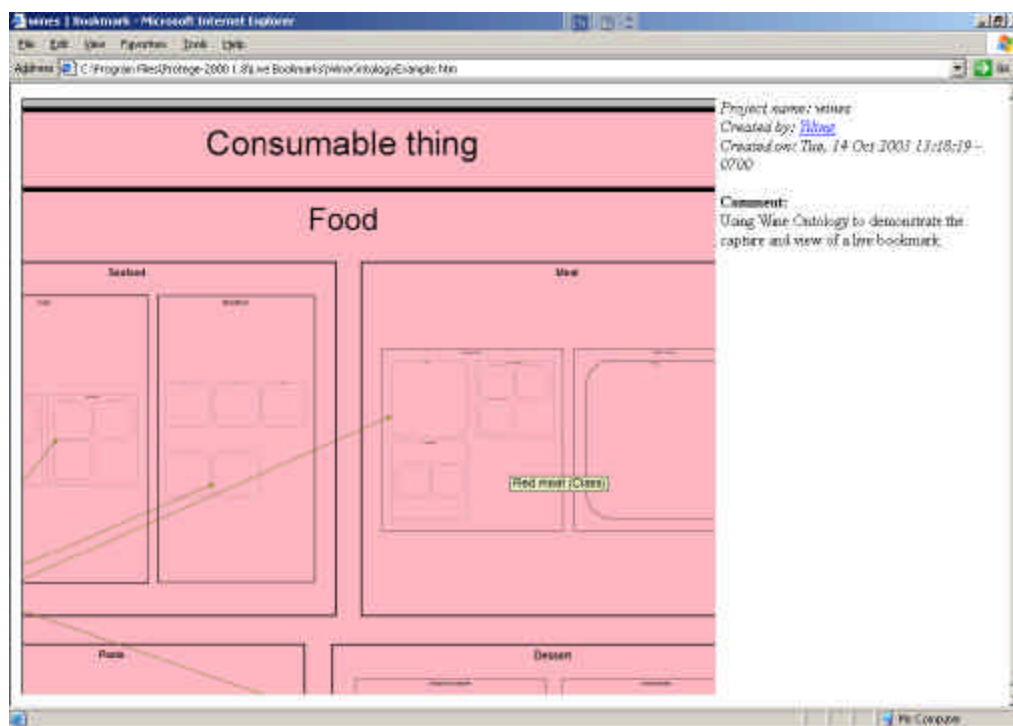


Figure 25: The bookmark from Figure 24 is opened in a web browser. User has zoomed into part of the bookmark to see more details using the zooming capacity of the SVG graph.

7.4.3 Live Bookmark Specification

The live bookmark is stored as an SVG (Scalable Vector Graphics) file format, which is a language for describing two-dimensional graphics in XML [24]. The animation in a live bookmark is implemented in JavaScript and the bookmark can be directly opened in any web browser with the SVG viewer plug-in that is available from Adobe Systems Inc.'s website (<http://www.adobe.com/svg/viewer/install/main.html>).

The bookmark takes advantage of SVG's ability to efficiently render high quality, large, complex images in an interactive and dynamic fashion, and the universal support of the SVG viewer in all mainstream web browsers. One of the SVG animation features is its zooming capacity, which makes the information visualization metaphor in live bookmarks correspond to the one used in Jambalaya. This saves the users the inconvenience of having to learn another set of user control mechanisms. Animations in SVG can be defined and triggered either declaratively (i.e. by embedding SVG animation elements in SVG content) or via scripting [24].

The bookmark XML file has three parts: the displayable graphics portion, the non-displayable portion, and the script portion. The displayable portion takes a 2D graph drawing on a Java Swing canvas and encodes this graph with XML elements. The non-displayable portion is essentially an SVG element with rendering turned off; it contains the meta-data associated with the displayable graph, such as the name of the project associated with this bookmark, layout type of this graph, time of creation, creator's name, creator's comments about this view, and the creator's email address. Other viewers of this bookmark will be able to send feedback to the creator by a single click on this email address. The scripting portion contains the JavaScript that enables the animation of the

displayable part. At this stage, it enables the tool tip that shows the name of nodes and arcs when the user moves mouse over them. The script can also be extended to enable more sophisticated animations, such as issuing layouts. There is a trade-off between keeping the bookmark lightweight and repeating all functions available in Jambalaya. Any decision will be based on the users' feedback and requirements as it gains wider acceptance and use.

Email, instant messaging, web portal, and peer-to-peer network applications can be used as the sharing and distributing channel of the knowledge captured by live bookmarks.

7.5 Summary

Looking at this array of recommendations we provide, we are gradually building an information ecology around Protégé. Bonnie Nardi and Vicki O'Day [86] define an information ecology as a system of people, practices, values, and technologies at work in a local environment. Local environments, not necessarily in a geographical sense, are defined by the participation, engagement, and commitment to a set of shared motivations and values. This ecology metaphor provides a distinctive, powerful set of organizing properties around those values and goals.

The collaborative ontology development system we are envisioning here has some of the characteristics of information ecology, and those characteristics are the same as – not surprisingly – some of the ones that a biological ecology has.

- Diversity: each tool takes a different niche in the environment and works together in a complementary way. While the Drupal powered project web site helps on managing group knowledge, PromptViz can help each individual developer sort

out the complex relationships between versions, while the live bookmark makes visual representation of the ontology data and the intertwined relationships among them available for sharing on the Web.

- Co-evolution: ecology implies continual evolution, and information ecologies evolve as new ideas, tools, activities, and forms of expertise arise in them [86]. Along with our quest for more comprehensive collaboration support for Protégé and for ontology engineering in general, we are expecting this research field to become a brewing pot for innovation and creativity. Tools will change, migrate and influence each other to fill the needs that come from the user community. This reflects the dynamic balance achieved in ecologies, a balance found in motion, not stillness.

As we look to the collaborative ontology development as a type of information ecology, we must remember that it will take time to grow and evolve. Evolution in an information ecology begins with our own efforts to shape and direct the technologies we use and be our choice of the scenarios in which we use them [86].

Chapter 8 Conclusion

Ontology development with developers located in geographically distributed sites is becoming a common practice. Emerging applications of ontologies in areas such as web services, content management and genetic research require large, complex ontologies that must be built and maintained by distributed teams.

This work provides a detailed survey of ontology engineering tools currently used in the field and the results show that there is a lot of room for adding and improving on the collaboration support in these tools. The thesis further discusses the specific problems in supporting collaborative team work with these tools.

This thesis also examines a range of groupware technologies, explored on their strengths and potential for supporting collaborative ontology engineering. These technologies include the peer-to-peer network infrastructure and its applications, instant messaging, and web portals. The exploration of these technologies establishes a roadmap that can help ontology development teams decide which collaboration tool to choose to better support their work, or ontology developers can use the survey and recommendations from this work as a reference framework to help identify their important requirements. This roadmap can also be beneficial for tool designers in understanding the state-of-the-art with respect to ontology development tools and help them decide which collaborative functions need to be improved or added in future tools. This thesis also records the technical difficulties and lessons learned during our attempt of implementing an application for collaboration on the JXTA platform. These recorded

experiences may be of help to future tool developers, allowing them to make informed technology choices and hopefully avoid implementation pitfalls.

Live bookmarks are an innovative approach to help facilitate collaboration among Protégé and Jambalaya users. Live bookmarks not only captures the tacit and newly discovered knowledge, but also codifies and expresses this knowledge with the XML format and thus helps complete the tacit-to-explicit knowledge conversion process.

The results of this thesis are important given the growing number of ontologies developed for business and scientific applications, and the increasing number of requirements from ontology developers for more and better tool support for collaborative work. The creation of an advanced collaborative ontology environment will help us produce ontologies faster and of better quality.

8.1 Summary of Contributions

The contributions of thesis include:

1. a detailed survey of existing ontology development tools and environments;
2. the investigation of three groupware technologies: instant messaging, web portals, and peer-to-peer networks, on their feasibility and potential for being used to provide collaborative support in ontology editing environments. The technical survey on peer-to-peer networks is useful for future researchers who will continue the design and implementation of multi-user Protégé-2000 on a P2P platform.
3. an identification of three issues related to collaboration in ontology engineering, and an analysis in correlation to the problems and solutions in the global software development field. These three issues are: distance and communication;

documentation and knowledge management, and version control and change tracking;

4. a record of our Protégé specific experiment on peer-to-peer network as a reference for Protégé designers interested in planning and implementing long term collaborative features on peer-to-peer network;
5. a set of recommendations for the future designers of the Protégé multi-user version, based on the problems we identified;
6. the live bookmark created as a collaboration-enabling mechanism for Protégé-2000. CHISEL Research Group continues to refine and improve it.

8.2 Future Work

Groupware technologies, such as instant messaging and peer-to-peer network applications, have never been integrated into the ontology development environments to support collaborative team work. In order to fully understand the nature of this ontology development team work (by analyzing existing work flow) and to produce an early prototype of an ontology development tool with sufficient collaboration support functions, it will be necessary to carry out an ethnographic study on an ontology or knowledge base development project.

Based on the recommendations we have produced in this thesis, we would like to organize an ontology development or knowledge engineering team to experimentally use different groupware technologies to support collaboration, and make careful observations on the usage patterns of the different groupware technologies. The usage patterns of instant messaging, web portals and other groupware among ontology developers will shed light on the question as to what extent general purpose groupware can satisfy the

requirements of collaborative ontology editing and what types of other groupware need to be developed to support specific collaborative ontology engineering tasks. This experiment will also help identify an initial set of user requirements for the design of collaborative ontology development tools.

8.3 Concluding Remarks

We believe that the need for more comprehensive collaborative support in the ontology engineering field is growing and will emerge as an important future research topic. This thesis takes an important step further towards understanding how groupware technology can be leveraged to support collaborative ontology development. Combining research fields is both exciting and challenging. We hope this work invigorates the discussion on designing and building a comprehensive collaborative ontology development environment using advanced groupware technologies.

We believe that this thesis offers a road map for future ontology editing tool designers that will help them understand the past difficulties and challenges of providing tool support for collaborative ontology engineering, and a glimpse at some of the technologies that may lead to future success.

Bibliography

- [1] Apelon <http://www.apelon.com/about/about.htm>, 2003, June 19
- [2] Biz2Peer Technologies, <http://www.biz2peer.com/p2p.htm>, 2003, February 2,
- [3] Bugzilla, Mozilla's bug tracking system, <http://bugzilla.mozilla.org/>, 2003, July 14
- [4] Computer Supported Collaborative Work <http://www.telekooperation.de/cscw/>, 2003, June 3
- [5] Desktop collaboration: project Backgrounder. <http://groove.net>, March 10, 2003, March 10, 2003
- [6] Desktop collaboration: Project Backgrounder. <http://groove.net>, 2003, March 10
- [7] Drupal - Content Management System, <http://www.drupal.org/>, 2003, July 14
- [8] Fact Fetch Project web site <http://factfetch.jxta.org/>, 2003, June 11
- [9] Groove Networks <http://www.groove.net>, 2003, January 28
- [10] <http://www.drupal.org/>, 2003, June 23
- [11] Imesh <http://www.imesh.com/>, 2003, February 3
- [12] JXTA <http://www.jxta.org>, 2003, February 3
- [13] Knowledge Interchange Format Manual, <http://www-ksl.stanford.edu/knowledge-sharing/kif/>, 2003, Sept. 17
- [14] myJXTA Project web site <http://myjxta.jxta.org/servlets/ProjectHome>, 2003, June 11
- [15] Ontoprise http://www.ontoprise.de/home_en.htm, <http://www.m-w.com/cgi-bin/dictionary>, 2003, April 25
- [16] Peer-to-Peer Working Group <http://www.peer-to-peerwg.org/whatis/index.html>, 2003, January 27
- [17] Protégé 2.0 Multi-User Tutorial, http://protege.stanford.edu/doc/protege2_alpha/index.html, 2003, Sept 16
- [18] Protégé project, Stanford University <http://protege.stanford.edu/index.html>, 2003, April 21
- [19] Shirky, C. 2001.What is P2P... and what Isn't. An article published on O'Reilly network. <http://www.openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html>, 2003, February 2
- [20] Terzis, S. CSCW & Groupware. <http://www.cs.tcd.ie/Sotirios.Terzis/CSCW.html#COOCOL>, 2003,
- [21] TOVE project publications <http://www.eil.utoronto.ca/enterprise-modelling/papers/index.html>, <http://www.m-w.com/cgi-bin/dictionary>, 2003, April 25
- [22] Track+ Project Manager, <http://sourceforge.net/projects/trackplus/>, 2003, July 14
- [23] US National Cancer Institute, Center for Bioinformatics, <http://ncicb.nci.nih.gov/core/caBIO>, 2003, July 28
- [24] W3C SVG Specification, <http://www.w3.org/TR/SVG/intro.html>, July 24
- [25] Web Ontology Language, <http://www.w3.org/TR/2003/WD-webont-req-20030331/#onto-def>, 2003, July 28

- [26] XML.com: Ontology Building: A Survey of Editing Tools XML.com
<http://www.xml.com/pub/a/2002/11/06/ontologies.html>, 2003,
- [27] M. M. Allen, "Empirical Evaluation of a Visualization Tool for Knowledge Engineering," in *Department of Computer Science*. Victoria: University of Victoria, 2003, pp. 109.
- [28] P. Allen, "Requirements Engineering in a Global Software Development: A Case Study," University of Victoria, Victoria May 2003.
- [29] T. Allen, *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information Within the R&D Organization*: MIT Press, Cambridge Massachusetts, 1984.
- [30] R. P. B. Swartout, K. Knight, T. Russ, "Toward Distributed Use of Large-Scale Ontologies," *Ontological Engineering* 138-148, 1997.
- [31] H. Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, "Looking Up Data in P2P Systems," *Communication of ACM*, vol. 46 (2) pp., 2003.
- [32] C. W. Barry Smith, "Ontology: Towards a New Synthesis," *ACM Press* 3-9, 2001.
- [33] M. Beaudouin-Lafon, *Computer Supported Co-operative Work*: John Wiley & Sons, 1999.
- [34] T. Berners-Lee, James Hendler, and Ora Lassila, "The Semantic Web," *Scientific American* (May) pp., 2001.
- [35] B. W. Boehm, "Spiral Model of Software Development and Enhancement," *Software Engineering Project Management IEEE* (May 1998) pp. 61-72, 1998.
- [36] U. Busbach, David Kerr, and Klaas Sikkell, "Forward of Conference Proceeding," presented at CSCW and the Web: An international Workshop organized by ERCIM/W4G, Sankt Augustin, Germany, 1996.
- [37] F. Cairncross, *The Death of Distance 2.0: How the communications Revolution Will Change Our Lives*: London, Texere Publishing, 2003.
- [38] E. Carmel, and Ritu Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Software*, vol. 18 (2) pp. 22-29, 2001.
- [39] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins, "What Are Ontologies, and Why Do We Need Them?," *IEEE Intelligent Systems, Special Issue on Ontologies*, vol. 14 (January/February) pp. 20-26, 1999.
- [40] S. M. Cherry, "IM Means Business," in *IEEE Spectrum* 39 (11), vol. 39, Nov 2002, pp. 28-32.
- [41] W. M. Cohen, Daniel Levinthal, "Absorptive Capacity: A New Perspective on Learning and Innovation," *Administrative Science Quarterly*, vol. 35 (1) pp. 128-152, 1990.
- [42] D. Coleman, *Groupware: Collaborative Strategies for Corporate LANs and Intranets*. San Francisco, CA: Prentice Hall PTR, 1997.
- [43] S. Cranefield, and Martin Purvis, "UML as an Ontology Modelling Language," presented at IJCAI'99 Workshop on Intelligent Information Integration, Sweden, 1999.
- [44] V. Devedzic, "Understanding ontological engineering," *Communications of the ACM*, vol. 45 (4(April 2002)) pp. 136-144, 2002.
- [45] C. A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: some issues and experiences," *Communications of the ACM*, vol. 34 (1, Jan 1991) pp. 39-58,, 1991.

- [46] D. C. Engelbart, William K. English, "A research center for augmenting human intellect," presented at AFIPS Fall Joint Computer Conference, San Francisco, CA, Dec. 1968.
- [47] J. Ernst, "Is Peer Computing Real?," presented at the 5th World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, Florida, July 2001.
- [48] J. A. Espinosa, Robert E. Kraut, Javier F. Lerch, Sandra A. Slaughter, James D. Herbsleb, and Audris Mockus, "Shared mental models and coordination in large-scale, distributed software development," presented at 22nd Annual International Conference on Information Systems, New Orleans, Louisiana, USA, 2001.
- [49] J. Euzenat, "Building consensual knowledge bases: context and architecture," presented at 2nd international conference on building and sharing very large-scale knowledge bases (KBKS), Enschede (NL), 1995.
- [50] J. Euzenat, "Corporate Memory through Cooperative Creation of Knowledge Bases and Hyperdocuments," presented at 10th Knowledge Acquisition, Modeling and Management for Knowledge-based Systems Workshop (KAW'96), Banff, Canada, 1996.
- [51] J. Euzenat, "HyTropes: A WWW front-end to an object knowledge management system," presented at Actes 10th knowledge acquisition workshop demonstration track, Banff, Canada, 1996.
- [52] J. Euzenat, "A protocol for building consensual and consistent repositories, Research Report RR-3260, INRIA," September 1997.
- [53] A. Farquhar, Richard Fikes, and James Rice, "The Ontolingua Server: a Tool for Collaborative Ontology Construction," presented at 10th Knowledge Acquisition Workshop, Banff, Canada, 1996.
- [54] H. Goldstein, "Collaboration Nation," in *IEEE Spectrum* 49 (6), vol. 49, June 2003, pp. 49-51.
- [55] A. Gómez-Pérez, "Ontological Engineering," presented at International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 1999.
- [56] L. Gong, "Project JXTA: A Technology Overview," Sun Microsystems, Inc, Palo Alto, CA, USA 2001.
- [57] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," presented at International Workshop on Formal Ontology, Padova, Italy, 1993.
- [58] J. Grudin, "Computer-Supported Cooperative Work," *IEEE Computer* 1994, vol. 27 (5) pp. 19-26, 1994.
- [59] J. Grudin, "Computer-Supported Cooperative Work: History and Focus," *IEEE Computer*, vol. 27 (5) pp. 19-26, 1994.
- [60] J. Grudin, "Groupware and social dynamics: eight challenges for developers," in *Communications of the ACM*, vol. 37, 1994, pp. 92-105.
- [61] M. Gruninger, Jintae Lee, "Ontology Applications and Design," *Communications of the ACM*, vol. 45 (2) pp. 39-41, 2002.
- [62] N. C. Guarino, M.; Giaretta, P., "Formalizing Ontological Commitments," presented at 12th National Conference on Artificial Intelligence, Seattle, WA, USA, 1994.

- [63] N. G. Guarino, P., "Ontologies and Knowledge Bases: Towards a Terminological Clarification," in *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*: IOS Press, 1995, pp. 25-32.
- [64] F. W. Hartel, Gilberto Fragoso, Kim L. Ong, Robert Dionne, and M.S., "Enhancing Quality of Retrieval Through Concept Edit History," 2003.
- [65] J. D. Herbsleb, and Audris Mockus, "An Empirical Study of Speed and Communication in Globally-Distributed Software Development," *IEEE Transactions on Software Engineering*, vol. 29 (6) pp., 2003.
- [66] J. D. Herbsleb, and Deependra Moitra, "Global Software Development," *IEEE Software*, vol. 18 (2) pp. 16-20, 2001.
- [67] J. D. Herbsleb, and Rebecca E. Grinter, "Architectures, Coordination, and Distance: Conway's Law and Beyond," *IEEE Software* (September/October) pp. 63-70, 1999.
- [68] J. D. Herbsleb, and Rebecca E. Grinter, "Splitting the organization and integrating the Code: Conway's Law Revisited," presented at 21st International Conference of Software Engineering (ICSE'99), Los Angeles, CA, USA, 1999.
- [69] J. D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter, "Distance, Dependencies, and Delay in a Global Collaboration," presented at 2000 ACM conference on Computer supported cooperative work, Philadelphia, Pennsylvania, United States, 2000.
- [70] C. W. Holsapple, and K.D. Joshi, "A collaborative approach to ontology design," *Communications of the ACM*, vol. 45 (2) pp. 42-47, 2002.
- [71] IBM, Lotus Notes official web site, www.lotus.com, 2003, Oct. 11
- [72] E. Isaacs, Alan Walendowski, Steve Whittaker, Diane J. Schiano and Candace Kamm, "The Character, Functions, and Styles of Instant Messaging in the Workplace," presented at Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA, Nov. 16-20 2002.
- [73] N. R. S. Jenifer Tennison, "APECKS: A tool to support living ontologies," presented at 11th Knowledge Acquisition Workshop (KAW'98), Banff, Canada, 1998.
- [74] Y. R. Jenny Preece, Helen Sharp, David Benyon, Simon Holland, Tom Carey, *Human-Computer Interaction*: Addison-Wesley Publishing Company, 1994.
- [75] R. Kraut, Egidio, C., and Galegher, J., "Patterns of contact and communication in scientific research collaboration," presented at Conference on Computer-Supported Cooperative Work (CSCW '88), Seattle, Washington, USA, 1998.
- [76] G. Lawton, "Knowledge Management: Ready for Prime Time?," *IEEE Computer*, vol. 34 (2) pp. 12-14, 2001.
- [77] F. López M., "Overview Of Methodologies For Building Ontologies," presented at IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, 1999.
- [78] M. M. Margaret-Anne Storey, John Silva, and N. E. Casey Best, Ray Ferguson, Natasha Noy, "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé," presented at Workshop on Interactive Tools for Knowledge Capture, K-CAP-2001, Victoria, BC, Canada, 2001.

- [79] A. G.-P. Mariano Fernández, Natalia Juristo, "Methontology: From ontological art toward ontological engineering," presented at Spring Symposium Series, Stanford University, Stanford, CA, 1997.
- [80] J. D. H. Mark Handel, "What is Chat Doing in the Workplace?," presented at Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA, Nov. 16-20 2002.
- [81] D. L. McGuinness, "Conceptual Modeling for Distributed Ontology Environments," presented at Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues, Darmstadt, Germany, 2000.
- [82] D. L. McGuinness, "Ontologies Come of Age," in *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, J. i. H. Dieter Fensel, Henry Lieberman, and Wolfgang Wahlster, Ed. Massachusetts: MIT Press, 2002.
- [83] D. L. McGuinness, Richard Fikes, James Rice, Steve Wilder, "The Chimaera Ontology Environment," presented at the Seventeenth National Conference on Artificial Intelligence (AAAI 2000). Austin, Texas, US, 2000.
- [84] I. R. Mikael Lindvall, Sachin Suman Sinha, "Technology Support for Knowledge Management," presented at Fourth Workshop on Learning Software organizations (LSO), Chicago, Illinois, USA, 2002.
- [85] D. S. Milojicic, Vana Kalogeraki, Rajan Lukose., J. P. Kiran Nagaraja, Bruno Richard., and a. Z. X. Sami Rollins "Peer-to-Peer Computing," Hewlett-Packard Company, HP Laboratories Palo Alto March 8th , 2002 March 8th , 2002.
- [86] B. Nardi, Vicki O'Day, *Information Ecologies: Using Technology with Heart*. Massachusetts, USA: MIT Press, 1999.
- [87] M. S. Natalya F. Noy, Stefan Decker, Monica Crubézy, Ray W. Ferguson, and and M. A. Musen, "Creating Semantic Web Contents with Protégé-2000," *IEEE Intelligent Systems* (March/April) pp. 60-71, 2001.
- [88] M. S. Natalya F. Noy, Stefan Decker, Monica Crubézy, Ray W. Ferguson, Mark A. Musen, "Creating Semantic Web Contents with Protégé-2000," *IEEE Intelligent Systems*, vol. 16 (2) pp. 60-71, 2001.
- [89] C. D. H. Natalya Fridman Noy, "The State of the Art in Ontology Design," *AI Magazine*, vol. 18 (3) pp. 53-74, 1997.
- [90] R. W. F. Natalya Fridman Noy, Mark A. Musen, "The knowledge model of Protege-2000: Combining interoperability and flexibility," presented at 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France, 2000.
- [91] R. F. Neches, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; Swartout, W.R., "Enabling Technology for Knowledge Sharing.," *AI Magazine* 36-56, 1991.
- [92] G. Pérez, "Towards a Framework to Verify Knowledge Sharing Technology," in *Expert System With Applications*, vol. 11, 1996, pp. 519-529.
- [93] D. Perrin, "Prompt-Viz: A Visualization Tool for Exploring the Differences Between Ontology Versions.," in *Department of Computer Science*. Victoria: University of Victoria, 2003.
- [94] T. G. Peter D. Karp, "The Generic Frame Protocol," presented at 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995.

- [95] J. Peters, "The Hundred Years War Started Today: An exploration of electronic peer review," *Journal of Electronic Publishing, University of Michigan Press*, 1996.
- [96] J. D. H. Rebecca E. Grinter, Dewayne E. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work," presented at 23rd international conference on Software Engineering, Toronto, Ontario, Canada, 2001.
- [97] G. G. Ricardo A. Falbo, Katia C. Duarte, Ana Candida C. Natali, "Developing Software for and with Reuse: An Ontological Approach," presented at ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, Foz do Iguacu, Brazil, 2002.
- [98] J. Rice, Adam Farquhr, Philippe Piernot, and Thomas Gruber, "Using the Web Instead of a Window System," presented at Computer Human Interaction Conference, Vancouver, Canada, 1996.
- [99] W. W. Royce, "Managing the development of large software systems: concepts and techniques," presented at International Conference on Software Engineering, Monterey, California, United States, 1987.
- [100] J. M. Schraagen, and Peter C. Rasker, "Communication in Command and Control teams," presented at The 6th International Command and Control Research and Technology Symposium, the United States Naval Academy (USNA) in Annapolis, Maryland, 2001.
- [101] M.-A. Storey, K. Wong, F.D. Fracchia, and H.A. Muller, "On Integrating Visualization Techniques for Effective Software Exploration.," presented at InfoVis'1997, Phoenix, USA, 1997.
- [102] Y. Sure, Michael Erdmann, Juergen Angele, Steffen Staab, Rudi Studer, Dirk Wenke, "OntoEdit: Collaborative ontology development for the semantic web," presented at International Semantic Web Conference 2002 (ISWC 2002), Sardinia, Italy, 2002.
- [103] B. Traversat, Mohamed Abdelaziz, and J.-C. H. Mike Duigou, Eric Pouyoul and Bill Yeager, "Project JXTA Virtual Network," Sun Microsystems, Inc, Palo Alto, CA, USA 2002.
- [104] J. Udell, *Practical Internet Groupware*: O'Reilly, 1999.
- [105] M. Uschold, "Building Ontologies: Towards Unified Methodology," presented at the 16th Annual conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK, 1996.
- [106] M. Uschold, and Michael Gruninger, "Ontologies: Principles, Methods and Applications," *Knowledge Engineering Review*, vol. 11 (2) pp., 1996.

Appendix A

Table 8: Table of ontologies developed using different methodologies. (Some methodologies are identified by the name of the designers.)

Methodology	Time	Domain	Ontology developed	Applications that use the developed ontology
Uschold and King	1995-96	Enterprise Modeling	Enterprise Ontology	Enterprise Toolset
Gruninger and Fox	1995-96	Business processes and activities modeling	TOVE ontology	<ul style="list-style-type: none"> Enterprise Design Workbench, Integrated Supply Chain Management Project Agents
Bernaras	1996	Electrical network	Several ontologies for electrical system	KACTUS toolkit, an interactive environment for browsing, editing and managing (libraries of) ontologies.
Methontology	1998	Chemical Scientific research	CHEMICALS Environmental pollutants ontologies	<ul style="list-style-type: none"> Chemical OntoAgent An ontology-based web broker
SESUS	1997	Natural language processing	SENSUS	Knowledge-based application for air campaign planning