Introduction

From the early 1990s, there has been a fruitful series of over a dozen workshops, symposia and conferences on the emerging field concerned with the development and application of ontologies. Early workshops were focused in large part on identifying what ontologies were, and how they might be used. As the field developed and matured, we have obtained a reasonable understanding and consensus about the nature of ontologies. The core idea is to explicitly encode a shared understanding of some domain that can be agreed upon by different parties (be they people or computers). This shared understanding is the ontology - it is an explicit representation comprising a vocabulary of terms, each with a definition specifying its meaning. All parties commit to using these terms in accordance to their definitions.

Meanwhile, the benefits of using ontologies have been recognized in many areas such as knowledge and content management, electronic commerce and recently the emerging filed of the semantic web. These new applications can be seen as a great success of research in ontologies. On the other hand, moving into real application comes with new challenges that need to be addressed on a principled level rather than for specific applications. The main purpose of this workshop is two-fold:

1. identify and discuss these new challenges as well as ways towards solutions in terms of stable infrastructure, advanced methods and professional tools.

2. discuss unconventional new ideas to cope with the new challenges that arise in distributed and weakly structured environments

The workshop should be seen as an opportunity for comparing and discussing traditional and new methods for capturing, describing and using semantics descriptions of certain domains and to try to bridge gaps between the two where-ever necessary.

Fundamental Issues

One part of the workshop will be devoted to fundamental issues of ontological research. As such it will not be devoted to a specific application area, but will rather address more fundamental questions like:

- What is the language of choice when building an ontology for a specific purpose?

- What kind of reasoning services are needed to support the development and use of ontologies?

- How can we ensure the quality of ontologies using formal methods reuse of standard models and engineering methodologies?
Advanced Issues

The second part of the workshop is devoted to less well explored topics that have come into focus recently as a response to the new problems we face when trying to use ontologies in a heterogeneous distributed environments. These environments include the use of ontologies in peer-to-peer and multi-agent systems as well as the semantic web. A couple of additional questions arise with respect to these environments, such as:

- Can we automatically generate ontologies from information and data structures?
- How can we mediate between different viewpoints and contexts encoded in different ontologies?
- How can we build Systems of independent agents that are able to cooperate by negotiating different viewpoint and ontologies?

In general this second part of the workshop asks for new methods needed to handle ontologies under new circumstances and for changes in the way we use ontologies in general.
Contributions

We received 27 submissions from different areas connected with the overall topic of the workshop. Out of these, 17 papers were accepted to be presented at the workshop resulting in an acceptance rate of 63 per cent. Based on the submitted papers four major topic areas can be identified which will briefly be introduced in the following:

Modelling and Management

One of the biggest barriers for using ontologies not only in distributed environments, but in general is the so-called acquisition bottleneck: building and maintaining ontologies is a difficult and time-consuming process that requires a lot of effort prior to a return in terms of enhanced performance. In order to overcome this bottleneck sophisticated tools and methodologies are needed to help the developers of ontologies or ontology-based systems to build accurate models and to reduce the development time. Further, just as for any software artifacts, maintenance is a crucial issue that though often being neglected, deserves special attention.

Five out of the seventeen accepted papers address topics related to the development and maintenance of ontologies: Santos and Staab (page 37) introduce an operator for assembling independently developed ontologies and report their experiences using it to enhance a domain ontology with notions of time. Pease (page 45) presents the Sigma environment that integrates a set of tools supporting the development process as well as existing reference ontologies. Kokis and Vouros (page 29) describe a tool that focuses on the way people interact with ontologies and tries to support that interaction in order to improve the ontology-engineering life cycle. Corcho and others (page 13) present an extension of their engineering tool that goes beyond the development of ontologies towards the development of ontology-based web services. Klein and Noy (page 21) finally describe an extension of the PROMPT tool for the detection of atomic and higher level changes in different versions of an ontology in order to support the maintenance process.

Reasoning and Learning

One of the often mentioned benefits of using formal ontologies in distributed environments is the ability to reason about terminology and data. Reasoning has many applications in ontology creation (consistency checking), deployment (instance classification) and use (query answering). Reasoning engines exist for many ontology representation languages such as RDF schema and OWL. In a distributed and heterogeneous setting, however, new techniques have to be developed to be able to allow for complete and efficient reasoning.
Andreasen and others (page 53) describe techniques for processing ontology-based queries over full text resources by translating phrases into ontological expressions and determining the similarity of such expressions on the basis of weighted relations. Stuckenschmidt (page 83) summarizes and extends approaches for compiling implicit knowledge encoded in expressive ontology languages into explicit RDF statements that can be queried using already implemented query languages. Omelagenko (page 67) proposes a schema for providing generic reasoning support for semantic web data by translating RDF files into PROLOG and converting the results back into RDF.

Recently, machine learning techniques have gained more attention in the areas of ontologies and the Semantic Web. Researchers have realized, that in the presence of huge amounts of information, manual approaches do not scale. Techniques of machine learning can be used to generate metadata and to assign instances to ontologies. Recently, learning methods are also used to generate skeletal ontologies from data or text repositories.

Caragea and others (page 61) discuss the problem of learning from distributed sources by decomposing learning tasks into hypothesis generation and distributed extraction of learning data. Stanescu and others (page 75) discuss the combination of ontologies and machine learning for the development of personal information agents.

**Ontologies for Distributed Systems**

One of the central topics of this workshop is the use of ontologies in distributed and de-centralized systems. This class of systems has gained a lot of attention recently. Concepts like the semantic web, grid computing, peer-to-peer systems or ubiquitous computing or multi-agent systems are topics of ongoing research. Ontologies can have different functions in these systems. They might define shared terminologies, explicate hidden assumptions of system components or as a vehicle to specify system architectures and information formats.

Pinkston and others (page 101) address the problem of intrusion detection in networks and the use of ontologies and logical reasoning methods in order to more effectively solve this problem. Ranganathan and others (page 109) describe the enrichment of a CORBA-based pervasive computing environment with explicit ontologies in order to facilitate matchmaking and interoperability. Chen and Finin (page 93) address the same problem focusing on the recognition of the local context a user is currently in and the conclusions than can be drawn from this fact. Uschold and others (page 117) finally, discuss the use of ontologies to capture world knowledge necessary to build autonomous vehicles by supporting reasoning about potential obstacles and their characteristics.
Meaning Negotiation

One of the most pressing and challenging problems of decentralized semantic-based systems is the need to compare different representations of semantics. In the world of ontologies, this is known as mapping or integration. In the broader area of distributed systems, the term meaning negotiation has been used to indicate that the problem is not restricted to the use of formal ontologies. New and unconventional methods are needed to achieve at least partial results on this difficult problem. Existing approaches range from theorem proving methods to voting and bidding mechanisms each being applicable under certain assumptions. Up to now no general solution to this problem has been found.

Giunchiglia and Shvaiko (page 139) discuss the general problem of matching graph-based representations such as thesauri, XML documents of ontologies on a semantic basis. Bonifacio and others (page 125) report results of a large case study in applying their matching algorithm that combines linguistic and logical matching in to the problem of matching product classifications. Ge and others (page 131) address the special problem of finding a common terminology in peer-to-peer systems and proposes a methods for agreeing on a shared model based on local definitions.
# Table of Contents

## A. Modelling and Management

- **p. 13** An for Development of Semantic Web Services  
  O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez and M. Lama

- **p. 21** A Component-Based Framework for Ontology Evolution  
  M. Klein and N.F. Noy

- **p. 29** Human Centred Ontology Management with HCOME  
  K. Kotis and G.A. Vouros

- **p. 37** Engineering a complex ontology with time  
  J. Santos and S. Staab

- **p. 45** The Sigma Ontology Development Environment  
  A. Pease

## B. Reasoning and Learning

- **p. 53** On Ontology-based Querying  
  T. Andreasen, H. Bulskov and R. Knappe

- **p. 61** Statistics Gathering for Learning from Distributed, Heterogeneous and Autonomous Data Sources  
  D. Caragea, J. Reinoso, A. Silvescu and V. Honavar

- **p. 67** Engaging Prolog with RDF  
  B. Omelayenko

- **p. 75** Ontologies for Learning Agents: Problems, Solutions and Directions  
  B. Stanescu, C. Boicu, G. Balan, M. Barbulescu, M. Boicu and G. Tecuci

- **p. 83** Compiling Complex Terminologies for Query Processing  
  H. Stuckenschmidt

## C. Ontologies in Distributed Systems

- **p. 93** An Ontology for Context Aware Pervasive Computing Environments  
  H. Chen, T. Finin and A. Joshi

- **p. 101** A Target-Centric Ontology for Intrusion Detection  
  J. Pinkston, J. Undercoffer, A. Joshi and T. Finin

- **p. 109** Ontologies in a Pervasive Computing Environment  
  A. Ranganathan, R.E. McGrath, R.H. Campbell, M.D. Mickunas

- **p. 117** Ontologies for World Modeling in Autonomous Vehicles  
  M. Uschold, R. Provine, S. Smith, C. Schlenoff, and S. Balikirsky
D. Meaning Negotiation

p. 125 Context Matching for Electronic Market Places - A Case Study
M. Bonifacio, A. Dona, A. Molani and L. Serafini

p. 131 OntoVote: a scalable distributed votes collecting mechanism for ontology drift of P2P platforms
Y. Ge, Y. Yu, X. Zhu, S. Huang and M. Xu

p. 139 Semantic Matching
F. Giunchiglia and P. Shvaiko
Programm Committee

Finally, we would like to thank the members of the programm committee for helping us to have a high quality workshop. Members of the programm committee are:

- Troels Andreasen
- Stefano Borgo
- Paolo Bouquet
- Joost Breuker
- Stephen Cranefield
- Ian Dickinson
- Rose Dieng
- Jerome Euzenat
- Martin Frank
- Fausto Giunchiglia
- Asuncion Gomez-Perez
- Nicola Guarino
- Siegfried Handschuh
- Frank van Harmelen
- Vipul Kashyap
- Michel Klein
- Alexander Maedche
- Eduardo Mena
- Natasha Noy
- Leo Obrst
- Dan O’Leary
- Borys Omelayenko
- Adam Pease
- Alun Preece
- Guus Schreiber
• Luciano Serafini
• Steffen Staab
• Heiner Stuckenschmidt
• Gerd Stumme
• York Sure
• Valentina Tamma
• Ingo Timm
• Mike Uschold
• Ubbo Visser
• Steven Willmott
A. Modelling and Management
An Environment for Development of Semantic Web Services

Oscar Corcho\textsuperscript{(1)}, Mariano Fernández-López\textsuperscript{(1)}, Asunción Gómez-Pérez\textsuperscript{(1)} and Manuel Lama\textsuperscript{(2)}

\textsuperscript{(1)} Departamento de Inteligencia Artificial. Facultad de Informática. Campus de Montegancedo, s/n. Universidad Politécnica de Madrid. 28660 Boadilla del Monte, Madrid. Spain. email: \{ocorcho, mfernandez, asun\}@fi.upm.es

\textsuperscript{(2)} Departamento de Electrónica y Computación. Facultad de Física. Campus Sur, s/n. Universidad de Santiago de Compostela, 15782 Santiago de Compostela, A Coruña, Spain. email: lama@dec.usc.es

Abstract

Semantic Web Services (SWSs) are specified in a semantic markup language to enable other services (and agents) to reason about their capabilities, in order to decide whether a SWS should be invoked or not. In this paper an environment for development and description of SWSs is presented. This environment, called ODE SWSDesigner, consists of a graphical interface, which allows users to carry out the design and characterization of SWSs at a conceptual level, and a set of software modules, which verify the design correctness and perform the translations from the graphical descriptions to the languages used to specify SWSs. ODE SWSDesigner provides support for a layer-based framework that we have proposed with the aim of enabling a language-independent development of SWSs. This framework is based on the use of problem-solving methods that are considered as high-level specifications from which SWS descriptions can be generated and verified.

1 Introduction

Web Services (WSs) are software modules that describe a collection of operations that can be network-accessible through standardized XML messaging \cite{Kreger2001}. WSs are distributed all over Internet, and in order to enable this accessibility and interactions between WSs, it becomes necessary an \textit{infrastructure} offering mechanisms to support the WS discovery and direct invocation from other services or agents. Nowadays, there are a number of proposals (usually ecommerce-oriented) that claim to enable partial or totally this required infrastructure, such as ebXML \cite{Webber2000}, E-Speak \cite{Graupner2000}, or BPEL4WS \cite{Curbera2002}. However, the approach that has emerged as a de facto standard, due to its extended use and relative simplicity, is the Web Service Conceptual Architecture \cite{Kreger2001}. This framework is composed of a set of layers that, basically, enable: (1) WS \textit{publication}, where the UDDI specification \cite{Bellwood2002} is used to define the WS capabilities and characterize its provider; (2) WS \textit{description}, which use the WSDL language \cite{Christensen2001} to specify how the service can be invoked (input-output messages), and SOAP \cite{Biron2001} as the communication protocol for accessing to WS; and (3) WS \textit{composition}, which specifies how a complex service can be created from the combination of other services. The language used to describe this composition is WSFL \cite{Leymann2001}.

In this context, the Semantic Web \cite{Berners-Lee2001} has risen as a Web evolution where the information is \textit{semantically} expressed in a markup language (such as DAML+OIL \cite{Hendler2000}) and, thus, both agents and services could access directly to it. This approach considers that the Web Services in the Semantic Web, so-called Semantic Web Services (SWSs), should specify their capabilities and properties in a semantic markup language \cite{McIlraith2001, Hendler2000}. This markup would enable other services to reason about the SWS, and, as a result, decide whether it match their requirements. Taking this into account, two frameworks, SWSA \cite{Sollazzo2001} and WSFM \cite{Fensel2002}, have been proposed to describe a semantic Web infrastructure for enabling the automatic SWS discovery, invocation and composition.
Both frameworks use the DAML-S specification [Ankolenkar et al., 2001], which is a DAML+OIL ontology for SWS specification, and emphasize the SWS integration with de facto standard WS, in order to take advantage of its current infrastructure.

On the other hand, Problem-Solving Methods (PSMs) describe explicitly how a task can be performed [Benjamins and Fensel, 1998]. The aim of the PSMs is to be reusable components applicable to different, but similar, domains and tasks. A PSM description specifies the tasks in which the PSM is decomposed (methods-tasks tree); the input-output interactions between the tasks; the flow control that describes the task execution; the conditions in which a PSM can be applied to a domain or task; and, finally, the ontology used by the PSM (method ontology), that is specified in a general manner to become PSM reusable in different domains (characterized by a domain ontology). The UPML specification [Fensel et al., 2003] provides containers in which these PSM views can be described, and, also, it incorporate elements that enable the PSM reuse. UPML has been developed in the context of the IBROW project [Benjamins et al., 1999] with the aim of enabling the semi-automatic reuse of PSMs. This objective could be interpreted as a composition of PSMs.

In this work our aim is to provide a development environment of SWSs, which would allow the user to design SWSs on the basis of PSM modeling (at a conceptual level). This environment also should perform verification about the soundness and completeness of the design created by the user. Once the design is verified, the user will select the specific languages in which the SWS will be specified. Thus, the SWS development process supported by this environment does not depend on a specific SWS language. These two features (PSM-based and language-independent design) are the main differences between our environment and other tools [Narayanan and McIlraith, 2001; Sirin et al., 2003], which use DAML-S to specify SWS description and composition, and to verify the SWS consistency.

The structure of the paper is as follows. In section 2 a PSM-based framework to develop SWSs (and WSs) is presented. In section 3 we describe the software architecture of the environment that supports this framework, and in section 4 the current capabilities of its graphical interface are explained. Finally, in section 5 the main contributions of the work are summarized.

2 Framework for SWS development
The framework that we propose for SWS development is based on the assumption that, in essence, SWSs (and WSs) could be considered as PSM specializations. This specialization means that SWSs do not need to be expressed in a general manner, because they are not aimed to be reusable in different domains or tasks. Therefore, the PSM method ontology is the same ontology as the one used in the SWS specification.

Relation between PSMs and SWSs
Both SWSs and PSMs are paradigms in which an operation (or equivalently a method) is executed to perform a task in a domain, and, as a result, it may obtain new domain information or provoke an effect in the real world. Taking this common objective into account, it seems to be reasonable to analyze whether PSMs may be used to enable the publication, description and composition of both WSs and SWSs.

Publication. A PSM definition does not usually show detailed information about its provider or the industry segment in which the PSM could be included. Although the UPML specification provides some information, in order to publish and discover SWSs it becomes necessary to extend it with data typically used in ecommerce interactions, such as quality or geographical situation of the provider.

Description. PSM specification details the input-output interactions between the PSM components (task interactions and method ontology). Figure 1 shows how the elements which define the WSDL specification (de-
noted with white boxes) can be completely extracted from a description of both task interactions and method ontology (dashed arrows), and from other WSDL elements (solid arrows). This knowledge will be enough to describe the SWS in order to enable its invocation. However, PSMs do not specify the communication protocol that allows them to be invoked through a network.

Composition. PSMs specify in detail how a task must be executed (flow control) and of which elements the PSM is composed of (methods-tasks tree). These specifications include the conditions in which the PSM elements (or subtask) should be executed and how those elements are combined to obtain the required result. On the basis of this information, the SWS composition could be enabled.

Considering this analysis we can conclude that there is a direct relation between PSMs and SWSs: PSMs can be used to specify SWSs (and WSs) features that are related to their internal structure (description and composition). However, we need to extend the PSM specification with knowledge related to ecommerce features, to enable SWS discovery, and communication protocols, to provide network-accessibility.

Framework Requirements
The design of the framework has been guided by a set of criteria (or requirements) that establish the conditions for defining an open and extensible framework for SWS development. These criteria are the following:

1. **SWS conceptual modeling.** SWS development must be carried out at conceptual level: characterization and description of the SWS capabilities and internal structure (for composition and description) cannot depend on specific languages that could limit the expressiveness of the SWS model.

2. **Integration with Web Service standards.** SWS specifications should be integrated with Web service de facto standards (both frameworks and languages) to be able to use its benefits and the current infrastructure that supports these standards [Sollazzo et al., 2001]. This criterion complements the SWS conceptual modeling, because it fixes the specific languages the SWS model must be translated to.

3. **Modular design.** The framework must be composed of a set of independent, but related, modules, which contain knowledge about different views of the SWS development process. This criterion guarantees the extensibility of the framework, because we can include new modules without modifying the others.

2.1 Layer-Based Framework
In order to cover these criteria we propose a framework with a layered design, whose layers are identified following a generality criterion, from the data types (lower layer) to the specific languages in which SWSs will be expressed (higher layer). Each layer is described by an ontology that defines its elements on the basis of well-known standards. These ontologies (or layers) are the following (see figure 2):

- **Data Types (DT) Ontology.** It contains the data types associated with the concept attributes of the domain ontology. The data types included in the DT ontology are the same as the ones defined in the XML Schema Data Types specification [Biron and Malhotra, 2001].

- **Knowledge Representation (KR) Ontology.** It describes the representation primitives used to specify the domain ontology managed by SWSs in its operations. That is, the components of the domain ontology will be KR instances. KR ontology is needed because tools that use the framework higher ontologies (PSM and SWS ontologies) could need to reason about the domain ontology itself. For example, preconditions of a method could impose that the input-output data should be “attributes”. Usually, the KR ontology will be associated with the knowledge model of the tool used to develop the domain ontology.

- **PSM Description Ontology.** This ontology describes the elements that compose a PSM, which, as we have previously discussed, can be used to generate SWS descriptions. The PSM ontology is constructed following the UPML specification [Fensel et al., 2003], that has been extended with (1) a programming structures ontology, which describes the primitives used to specify the PSM flow control (such as conditional and parallel

![Figure 2. Framework for SWS development. It is composed of a set of design layers, each one described by an ontology that is based on well-known specifications (de facto standards).](image-url)
loops, conditional statements, etc.); (2) inferences, which are new PSM elements defined as in the CommonKADS knowledge model [Schreiber et al., 1999], that is, as building blocks for reasoning processes; and (3) relations among PSM elements to explicitly declare whether an element may be executed independently of the others or not and whether they can be invoked by an external agent (or service). On the other hand, the PSM ontology contains a number of axioms that constrains how PSM element instances are created. That guarantees the soundness of the PSM model. For example, it exists an axiom establishing that the inputs method must be covered by the inputs associated with the tasks that compose the method.

- **SWS Ontology.** This ontology is constructed on the basis of the PSM description ontology, which is extended with both knowledge related to ecommerce interactions, which enable the publication and advertisement of services, and communication protocols. These extensions are performed using the DAML-S specification as reference [Ankolentar et al., 2001], because it describes containers to include these types of knowledge.

- **Standard Language Ontologies for Web Services.** They describe the elements associated with the de facto Web standard languages for service publication (UDDI), description (WSDL/SOAP), and composition (WSFL). These ontologies complete the SWS specification, because they facilitate its integration in the current infrastructure of the Web.

This framework satisfies the design requirements. In effect, conceptual modeling of SWSs is performed in the PSM layer, which is not constructed following a specific language, but is modeled at knowledge level [Newell, 1982]; integration with Web service standards is explicitly enabled in the framework highest layer, which, if required, could be easily extended to include new standards; and, finally, modular design is associated with the layered approach itself.

## 3 Environment for SWS development

In order to provide support for the framework, we have developed an environment with the aim of allowing users to design the conceptual model of a SWS by means of a graphical interface. Once this model is created, it could be exported to a DAML+OIL specification (such as DAML-S), which will be complemented with Web service standard languages. This environment, called ODE SWSDesigner, has been designed following the framework requirements: to develop an open and easily extensible environment that, if required, could be adapted to support new SWS (and WS) specification languages or frameworks.

In addition, ODE SWSDesigner is integrated into WebODE [Arpírez et al., 2001], which is a workbench for ontology development that provides additional services for exporting ontologies to different languages (such as DAML+OIL, RDF, etc.), merging and evaluating ontologies, and reasoning with ontologies using their axioms. Integration in WebODE will allow SWSDesigner to invoke those services whose capabilities needs in its operation, such as services for exporting an ontology (SWS and PSM) to a specific language (DAML+OIL and Java respectively) or checking constrains in ontologies.

### 3.1 Software Architecture

In accordance with the proposed framework, the design and development of SWSs could be viewed as the process of instantiating an ontology set that contains the knowledge needed to generate the SWS specifications. Software architecture of ODE SWSDesigner is based on this consideration and, as we can see in figure 3, it is composed of two different types of modules: a graphical interface, which allows the users to develop SWSs at a conceptual level, and a set of instance processors, which are software modules that process the SWS graphical descriptions (created by the users) to generate the instances associated with the ontologies of the framework, and, if required, to check the correctness of the generated instances. The instance processors, which have been included in WebODE as services, are the following:

- **KR service.** This processor gets as input the ontology used in SWS operation (usually the domain ontology) and establishes the instances associated with the KR and Data Types ontologies. The domain ontology can be available in WebODE or could be translated from an ontology language into the WebODE specification. In both cases, this processor will invoke the ODE service to access to the domain ontology elements, which are saved in a database (figure 3).

- **PSM service.** It uses the graphical descriptions of the SWS model created by the user to generate an instance set that describes completely the PSM internal structure and flow control (PSM model). Once the instance set is created, this processor must invoke the inference WebODE service [Corcho et al., 2002] to verify the soundness and completeness of the PSM model. In this verification the axioms that constrain how the PSM elements can be combined with each other are used. For example, if we defined a general service that is decomposed in two sub-services, it is necessary to verify that the inputs of these sub-services have the same (or subsumed) type as the general service inputs. In order to perform this verification, the PSM processor must operate with an explicit description of the representation primitives in which the domain ontology will be instanced.
SWS service. Instances created by this processor will enhance the knowledge included in PSM ontology instances by adding the information used in ecommerce interactions. This information will be directly obtained from the graphical interface.

These three instance processors represent the SWSDesigner core, because they support the generation of the SWS model and their operation does not depend on the languages in which the SWS will be expressed. Thus, these processors will be modified only if their associated layers are changed.

WSLang service. It gets as inputs the SWS ontology instances and generates an instance set from which the SWS model is specified in UDDI, WSDL/SOAP and WSFL languages.

DAML-S service. It obtains the DAML-S specification of the SWS getting as inputs the instances of the SWS ontology. This operation, nevertheless, is not straightforward because in the DAML-S ontology a service is modeled as a process, while in our framework a service is considered to be a specialization of a PSM (or method). Once this operation is performed, this processor must invoke the WebODE service that exports an ontology to the DAML+OIL language.

Java service. Using the PSM ontology instances, this processor generates the skeleton of the programming code (Java beans) needed to execute the SWS and perform its operation. Once this code has been created, the user must fill in the methods responsible of carrying out the operation modeled in the PSM.

These three processors represent SWSDesigner additional processors, because they have been specifically included into the framework to obtain SWS (or WS) specifications in various languages. This means that these processors would be changed (or substituted) if it was required to use other languages or if the core processors were also modified.

On the other hand, instance processors are directly invoked from the graphical interface when the users, after creating the SWS conceptual model, require to export that model to well-known WS languages or when the graphical interface itself needs to verify whether an operation carried out by the user has generated an inconsistent model of the SWS.

4 Graphical Interface

ODE SWSDesigner graphical interface is based on the assumption that the design and development of a service should be performed from different, but complementary, points of view (such as in PSM modeling). These different views help the user to understand the internal structure of a service and the interactions between its components (sub-services). Taking this into account, the graphical interface contains the following views, which reflect how PSMs are designed (see example of figure 4):

Definition view. In this view the user defines a service by specifying its name (mandatory) and, optionally, by introducing the information needed for enabling service discovery and advertisement, such as a description of the provider that offers the service, the types of business for which the service is oriented (industry classifications), etc.

Decomposition view. This view allows the user to specify (and also create) the services (sub-services) that could be executed when a service (composite) is activated. That is, a service hierarchy is specified. This hierarchy could be used for service composition and for checking inconsistencies in the other graphical interface views.

Interaction view. In this view the input-output interactions between the sub-services of a composite service are specified. This operation requires that the domain ontology be previously loaded from WebODE database to the graphical interface. Figure 4 shows the main window of the ODE SWSDesigner, where we
can see the specification of the interactions between the sub-services of *buyMovieTicket* composite service. All these services have been created in the decomposition view, which generates the service tree shown in the right side of figure 4.

*Flow control view.* In this view the user specifies the flow control of a service, where its sub-services are combined with programming structures to obtain a description of the service execution. This view, which is not implemented yet, will be used to model the service composition by means of several diagrams that the user will create to describe the different compositions of services. On the other hand, this view and the decomposition view could be used to export to languages (as WSFL) that specify the service composition.

The graphical interface guarantees the soundness and completeness of the models that have been created in each one of its views. For example, if the user specifies that a service is composed of three sub-services (decomposition view), the graphical interface will invoke the PSM processor to assure that the interaction view contains exactly those three services (as shown in figure 4).

### 5 Conclusions

ODE SWSDesigner enables the users to develop SWSs following a PSM-oriented design, which is based on a language-independent framework for SWS development. Furthermore, ODE SWSDesigner will assure the soundness and completeness of the SWS designs created by the users. Once the SWS design correctness is verified, the user can select the languages in which the SWS will be described. Thus, in ODE SWSDesigner the user does not need to know specific details about the languages used to specify the SWSs.

Nowadays, we have implemented the definition, composition and interaction views of the graphical interface, and DAML-S is the current SWS language into which the designed SWS are translated. In addition, we are extending the PSM and SWS ontology with new axioms. These extensions will allow us to cover additional conditions to check SWS consistency and completeness.
On the other hand, the ODE SWSDesigner integration in WebODE has simplified its software architecture and implementation, because (1) it uses directly the WebODE services, which offer support for ODE SWSDesigner operations; and (2) it uses the infrastructure itself that WebODE provides for including software modules as services, which could be easily accessed from the graphical interface. Thus, the integration in WebODE favors the ODE SWSDesigner modularity, which is a key requirement to adapt the environment to new standard languages or frameworks.

References


A Component-Based Framework For Ontology Evolution

Michel Klein
Vrije University Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands

Natalya F. Noy
Stanford Medical Informatics,
Stanford University,
Stanford, CA 94305

Abstract
Support for ontology evolution becomes extremely important in distributed development and use of ontologies. Information about change can be represented in many different ways. We describe these different representations and propose a framework that integrates them. We show how different representations in the framework are related by describing some techniques and heuristics that supplement information in one representation with information from other representations. We present an ontology of change operations, which is the kernel of our framework.

1 Support for Ontology Evolution
Ontologies are increasing in popularity, and researchers and developers use them in more and more application areas. Ontologies are used as shared vocabularies, to improve information retrieval, or to help data integration. Neither the ontology development itself nor its product—the ontology—is a single-person enterprise. Large standardized ontologies are often developed by several researchers in parallel (e.g. SUO1 [9]); a number of ontologies grow in the context of peer-to-peer applications (e.g. Edutella [5]); other ontologies are constructed dynamically [2]. Successful applications of ontologies in such uncontrolled, de-centralized and distributed environments require substantial support for change management in ontologies and ontology evolution [7].

Given an ontology $O$ and its two versions, $V_{old}$ and $V_{new}$, a complete support for change management in an ontology environment includes support for the following tasks.2

Data Transformation: When an ontology version $V_{old}$ is changed to $V_{new}$, data described by $V_{old}$ might need to be translated to bring it in line with $V_{new}$. For example, if we merge two concepts $A$ and $B$ from $V_{old}$ into $C$ in $V_{new}$, we must combine instances of $A$ and $B$ as well.

1 http://suo.ieee.org/
2 Note that $V_{new}$ is not necessarily a unique replacement for $V_{old}$. There might be several new versions based on the old version, and all of them could exist in parallel. The labels are just used to refer to two versions of an ontology where $V_{new}$ has evolved from $V_{old}$.

Ontology Update: When we adapt a remote ontology to specific local needs, and the remote ontology changes, we must propagate the changes in the remote ontology to the adapted local ontology [8].

Consistent Reasoning: Ontologies, being formal descriptions, are often used as logical theories. When ontology changes occur, we must analyze the changes to determine whether specific axioms that were valid in $V_{old}$ are still valid in $V_{new}$. For example, it might be useful to know that a change does not affect the subsumption relationship between two concepts: if $A \subseteq B$ is valid in $V_{old}$ it is also valid in $V_{new}$. While a change in the logical theory will always affects reasoning in general, answers to specific queries may remain unchanged.

Verification and Approval: Sometimes developers need to verify and approve ontology changes. This situation often happens when several people are developing a centralized ontology, or when developers want to apply changes selectively. There must be a user interface that simplifies such verification and allows developers to accept or reject specific changes, enabling execution of some changes and rolling back of others.

This list of tasks is not exhaustive. The tools that exist today support these tasks in isolation. For example, the KAON framework [10] supports evolution strategies, allowing developers to specify strategies for updating data when changes in an ontology occur. The SHOE versioning system specifies which versions of the ontology the current version is backward compatible with [3]. Many ontology-editing environments (e.g., Protégé [11]) provide logs of changes between versions. While these tools support some of the ontology-evolution tasks, there is no interaction or sharing of information among the tools. However, many of these tasks require the same elements in the representation of change. I mple-
mentation of support for one task can, and should, use the change information acquired for another, rather than try to determine it from scratch. Having a general framework for ontology evolution that allows tools supporting different evolution tasks to share change information and leverage change information obtained by other tools, will make ontology evolution much more efficient.

In this paper, we propose a framework that integrates several sources of information about ontology change. We explain how these different change representations are related to one another and show how we can supplement information in one representation with the information from other representations. More specifically, this paper makes the following contributions:

- We study formalisms for representing ontology change (Section 2).
- We present a component-based framework for defining ontology change (Section 3).
- We present an ontology of basic change operations that provides the basis for inter-operations between various components in the framework (Section 4).
- We define complex ontology changes, which provide the basis for more efficient data transformation and advanced user-interaction capabilities (Section 4).
- We present rules and heuristics for identifying complex changes between ontology versions (Section 5).

Throughout this paper, we use the following example to illustrate our concepts and ideas. Suppose that we are developing an ontology of wines. In the first version (Figure 1a), there is a class Wine with two subclasses, Red wine and White wine. The hierarchy also includes some specific types of red and white wines. Figure 1b shows a later version of the same ontology fragment. Note the changes: we introduced a new subclass of Wine, Rosé wine; the classes that were previously subclasses of White wine—Cabernet blanc, White Zinfandel, and Vin gris—are now subclasses of the new class Rosé wine; we renamed the Riesling class to Weisser Riesling.

2 Formalisms for Representing Change

There is a number of ways in which we can represent change information. On the one end of the spectrum of representation forms, we may have very few details about changes from $V_{old}$ to $V_{new}$. For instance, the two versions of the ontology may be all the information that we have. On the other end of the spectrum, we may have a complete and detailed representation of changes from $V_{old}$ to $V_{new}$; both versions, a detailed log of changes, conceptual description of changes, metadata about them, and so on.

The following are some of the ways to represent change information for an ontology version $V_{new}$ (in a particular environment we can have one or more of these elements in place):

- the old version of the ontology $V_{old}$ (providing the basis for finding change information but no explicit change information)
- a log of changes applied to $V_{old}$ that result in $V_{new}$ (providing a record of the ontology-transition process)
- a structural diff between versions that describes differences between them (providing a declarative view of the ontology transition)
- a set of conceptual changes between versions (providing an explicit specification of conceptual relations between concepts in $V_{old}$ and corresponding concepts in $V_{new}$)
- a transformation set that describes a sufficient set of change operations for the transition from $V_{old}$ to $V_{new}$ (providing an operational view of the changes)

One of the easiest change representations to create, with the appropriate tool support, is a change log between versions. A change log records an exact sequence of changes that occurred when an ontology developer updated $V_{old}$ to arrive at $V_{new}$. Many ontology-editing tools, such as Protégé [1], OntoEdit [11] and others, record changes that developers make. There are several detailed proposals for the information that logs should contain (e.g., versioning in KAON [10], Concordia [8]). For example, the evolution framework of KAON provides a number of “add”, “set” and “delete” operations. The log contains a list of specific operations, such as “AddPropertyDomain” or “RemoveSubConcept” with references to the concepts or properties that they operate on.

Most logs of ontology changes are quite similar to the KAON format. They contain simple ontology changes, where the level of granularity at which changes are specified is close to a single user-interface operation. A log is that it provides a complete and unambiguous change specification at a very fine level of detail. Figure 2 shows a possible log of changes between versions from Figure 1.

Figure 1: Two versions of a wine ontology (a and b).
Change logs may not always be available however. In a dynamic and de-centralized environment such as the Semantic Web, we may have access only to the old and the new version of an ontology, but not to the record of the change. Furthermore, change logs are less useful in an environment where several editors update an ontology at the same time: interleaving the logs to find out the final effect of changes is a difficult task in itself. Therefore, there are a number of ways to represent change that relate $V_{\text{old}}$ and $V_{\text{new}}$ directly, without taking into account the specific sequence of changes that has actually taken place.

A structural diff [6] provides a map of correspondences between frames in $V_{\text{old}}$ and $V_{\text{new}}$. For each from in $V_{\text{old}}$, it identifies whether or not there is a corresponding frame in $V_{\text{new}}$ (its image) or whether a frame was deleted, or a new frame was added. Figure 3 shows a structural diff between ontology versions in Figure 1. The structural diff shows that the class Rosé wine was added, the class Riesling was renamed into Weisser Riesling, the class Cabernet blanc changed its superclass, and so on. PROMPTDIFF [6] is an example of a tool that uses heuristics to create a structural diff automatically. It uses persistent identifiers of the frames in different versions, or, if such identifiers are not present, structural relations between ontology elements.

A structural diff provides a declarative view of changes: it represents the mapping between versions but not the operations to get from one version to another.

A set of conceptual changes specifies the conceptual relation between frames across versions, that is, the relation between a frame in $V_{\text{old}}$ and the image of that frame in $V_{\text{new}}$. In our example in Figure 1, after creating the class Rosé wine, we moved a number of classes that were previously subclasses of White wine to the Rosé wine subtree. In this case, a conceptual change could specify that the class White wine in $V_{\text{new}}$ is a subclass of the class White wine in $V_{\text{old}}$. Similarly, it could specify that Riesling in $V_{\text{old}}$ is equivalent to Weisser Riesling in $V_{\text{new}}$. Sometimes, when a consistent interpretation of already annotated datasets is essential, updates are intentionally specified as sets of conceptual changes. For example, the EMTREE thesaurus, which is used by Elsevier to index scientific publications, specifies updates by defining that specific terms become subsumed by other terms, or that they became synonyms of other terms.

In the OntoView system [4], developers can augment a change description with conceptual relations between frames across versions.

A transformation set provides a set of change operations that specify how $V_{\text{old}}$ can be transformed into $V_{\text{new}}$. Figure 4 presents one possible transformation set for versions in Figure 1. The transformation set in the figure contains only basic changes: each change is a single knowledge-base operation. The set can also include complex changes: for example, we can combine two operations that add a superclass and remove a superclass for the same class into a single move operation. We define transformation sets formally in Section 3 and we introduce basic and complex operations in Section 4.

A transformation set is different from a log in several aspects. First, while a log contains a record of all the operations that actually took place (including all intermediate steps) during the ontology-editing process, a transformation set specifies only the necessary operations to achieve the resulting change. Second, while a log is an ordered sequence of actions, there is only very limited partial ordering in a transformation set (mainly, that all “create” operations happen before all other operations). Third, while a log is a unique representation of the actual change process, there can be several (and often there are many) valid transformation sets for any two versions $V_{\text{old}}$ and $V_{\text{new}}$.

Note that if a log of changes between two ontology versions consists of operations that do not undo other operations, this log is by definition a transformation set between these two versions.

The list of change representations in this section is not exhaustive. For example, some systems store concept-history information, associating with each concept a list of concepts that it was derived from, whether a concept was “retired” and which concept replaced it [8]. The systems with the primary purpose of data transformation may store a set of operations that is a specific “recipe” for transforming data instances. Other ways to represent change may develop as ontology evolution becomes more and more common.

---

3See http://www.elsevier.com/locate/emtree.
A transformation set between two versions, specified with operations from the change ontology, and possible interactions with other change representations.

3 A Framework for Ontology Evolution

We now bring together the different formalisms that we described in Section 2 in a single ontology-evolution framework. In a distributed evolution environment, such as the Semantic Web, given an evolving ontology \( O \), we can have some information about the change between two versions of \( O \). For instance, we may have only the log of changes, or only a structural diff. However, once we have some of the change information, we can use additional tools to derive other information. For instance, we can use a log to derive a transformation set or we can use a structural diff to derive the definition of conceptual changes.

The purpose of our framework for ontology evolution is twofold:

1. relate the change information that is available in different formalisms (Section 2), and

2. provide mechanisms to derive new pieces of information from existing information.

Figure 5 shows the components of the ontology-evolution framework and some of the possible interactions between them.

The kernel of the framework is a minimal transformation set, which provides a set of operations that are necessary and sufficient to transform \( V_{\text{old}} \) into \( V_{\text{new}} \). We already introduced the idea of a transformation set, here we define it formally.

**Definition 1 (Transformation set)** Given two versions of an ontology \( O \), \( V_{\text{old}} \) and \( V_{\text{new}} \), a transformation set \( T(V_{\text{old}}, V_{\text{new}}) \) is a set of ontology-change operations that applied to \( V_{\text{old}} \) results in \( V_{\text{new}} \). The operations in \( T(V_{\text{old}}, V_{\text{new}}) \) can be performed in any order, with one exception: all operations that create new classes, properties, and instances are performed first.

A transformation set \( T(V_{\text{old}}, V_{\text{new}}) \) is minimal if removing any operation from the set results in a set that is no longer a transformation set from \( V_{\text{old}} \) to \( V_{\text{new}} \).

A transformation set is not necessarily unique. The requirement that create operations are performed first has a very practical reason: some of the operations in the transformation set may refer to the newly created concepts. Thus they need to exist in order to be used in other operations.

We also define an ontology of change operations that can constitute a transformation set. This ontology is also a central element in our framework, because different tools using the framework must agree on the part of the ontology describing basic change operations. As tools use different formalisms for change representation for different tasks or augment information represented in one formalism with information in another, this set of basic operations is the “common language” that they share. This requirement to agree on a common set of basic change operations is similar to the requirement that agents on the Semantic Web share a common ontology language, such as OWL. Defining such a standard set is not unrealistic: once there is a common ontology language (e.g., once OWL becomes a standard), developing and agreeing on an ontology of basic changes is doable. Essentially, an ontology of basic changes is directly related to the ontology language itself and constitutes a set of simple operations to build an ontology in this language. We present the ontology of change operations in Section 4.

3.1 Interaction of Framework Components

As we mentioned earlier, we often have only an incomplete description of the change, with only some of the components in place. Different tools in the framework can use the available representations to derive new ones. As a result, having some information about change enables us to complete the picture by deriving additional elements of the change description. We use existing pieces of the puzzle to fill (some of) the missing pieces. Even in the case that we cannot fill in all the pieces, we might still be able to support tasks that we could not support before.

We now describe some of the transformations from one change description to another. Tools already exist for most of these transformations.

**Change log \( \rightarrow \) minimal transformation set** Many ontology-editing tools provide logs of changes (e.g., Protégé, OntoEdit). These changes are often at the level of simple knowledge-base operations: adding a superclass to a class or removing one. We can transform logs into transformation sets by translating the operations into our vocabulary of basic changes (Section 4) and removing redundant changes.

**Basic changes \( \rightarrow \) complex changes** If we have a transformation set consisting of basic operations, we can use heuristics to combine these simple operations to create complex change operations. For instance, if we have a set of siblings in a class hierarchy and each of these siblings had the same class added as a superclass and the original superclass removed, we can infer that the whole set of siblings was moved from one part of the hierarchy to another. In addition to the set of basic changes, we may need direct access to \( V_{\text{old}} \) to find complex changes. We describe such heuristics in Section 5.

**\( V_{\text{old}} \) and \( V_{\text{new}} \) \( \rightarrow \) structural diff** If we do not have any specific information about the change, but we have both \( V_{\text{old}} \) and \( V_{\text{new}} \), we can compare the two versions to create automatically a structural diff between them. For example, if concepts in an ontology have immutable concept ids, a simple tool can
create a diff between versions identifying for each frame $F$ in $V_{old}$ its image in $V_{new}$. If we do not have immutable concept ids, tools such as PROMPTDIFF[6] use a set of heuristics based on concept names, class-tree structure, and concept definitions to create a structural diff.

**Structural diff → transformation set** If we have a structural diff, we can use it to create more useful change descriptions. Consider for example the structural diff in Figure 3. Knowing that Riesling became Weisser Riesling, we can add a changeName operation to the transformation set.

**Transformation set → conceptual relations** If we have a transformation set with both simple and complex operations defined between versions, we can use a set of heuristics to suggest conceptual relations between frames in versions to the user. For example, if we add a property to a class, we might suggest to the user that the new version of the class has become a subclass of the old version.

**Structural diff → conceptual relations** Similarly, we can use a structural diff to derive conceptual relations. For instance, the mappings of a structural diff directly suggest equivalence relations between concepts. In fact, we have integrated PROMPTDIFF, a tool that finds a structural diff [6], and OntoView, a tool for specifying conceptual relations [4], using the information produced by PROMPTDIFF to suggest initial conceptual relations in OntoView.

We outlined some of the ways to fill in missing pieces of the puzzle based on the pieces that we have. However, we cannot possibly envision what future tools will exist and what other ways to fill in new pieces researchers will come up with. Moreover, there could be more potential pieces of information that we may want to have, that our picture is currently missing. For example, several years ago, we might not have had either structural diffs or conceptual relations between versions in this picture. The main idea is to have this framework extensible. The guiding principle is “we’ll take whatever input we have and do the best to find out more information.”

### 3.2 Framework Components and Evolution Tasks

Each way to represent change in our framework is useful for some ontology evolution tasks that we listed in Section 1.

**Minimal transformation set** The minimal transformation set provides an operational view of the change. Thus we can use it for translation of other ontologies, for reversing changes, and as a starting point for data translation.

**Complex changes** Together with the minimal transformation set, we can use the complex operations to create data-transformation scripts. Also, complex changes allow us to determine in more detail the effect of changes on data accessibility and specific logical queries. Moreover, visualizing complex rather than basic changes, makes validation and approval tasks much easier for users.

**Structural diff** A structural diff is also useful for visualizing differences between ontologies. In addition, it is an essential information source for most of the supplementation techniques that are described in the next section.

**Conceptual relations** Finally, conceptual relations between concepts across versions facilitates data access by improving the interpretation and query of data sources that were described with different versions of ontologies.

### 4 Ontology of Change Operations

We have developed an ontology of change operations for the OWL knowledge model as an example of a common language for the interaction of tools and components in our framework. The operations in this ontology are the elements for the specification of a transformation set (Definition 1). The ontology consists of two parts. The basis is an ontology of basic change operations and there is an extension that defines complex change operations. We chose the set of basic operations in such a way that the required commitment is minimal, while the set is still rich enough to capture enough knowledge about the change to derive new information.

#### 4.1 Basic Change Operations

Each of the basic change operations modifies only one specific feature of the OWL knowledge model. Examples of such operations are cardinality change, changing the cardinality of a property restriction, or property transitivity addition, declaring a specific property as transitive. Altogether, we distinguish more than 80 different basic change operations.

We model change operations as a hierarchy of classes, where each class represents a specific type of change operation. The ontology specifies characteristics of each change type via property restrictions. All change classes use the from and to properties to refer to the source and target of the change. In addition, most change classes have properties that specify an argument for a change operations. For example, one of the arguments for the cardinality_change operation is the integer specifying the new cardinality restriction.

By organizing the change operations in a class hierarchy, we exploit the inheritance mechanism to specify common properties of change operations in an efficient way. For example, all changes in property restrictions require two arguments to identify the source, namely the class identifier and the property identifier. Since all changes in property restrictions are subclasses of property_restriction_change, we can easily specify this fact for all operations at once.

The ontology of basic changes contains “add” and “delete” operations for each feature of the OWL knowledge model. This set of operations ensures the completeness of the ontology of basic changes since it is sufficient for defining a transformation set from any ontology $V_{old}$ to any other ontology $V_{new}$. While not being the most useful or efficient, such transformation set can contain the operations that delete all elements in $V_{old}$ and then add all elements in $V_{new}$.

The ontology of basic changes also contains ‘modify’ operations, which specify that an old value is replaced by a new value. For example, a range_change operation specifies that

---

4 A snapshot of a recent version of this ontology can be found at [http://ontoview.org/changes/1/3/](http://ontoview.org/changes/1/3/).

5 See [http://www.w3.org/TR/owl-features/](http://www.w3.org/TR/owl-features/).

6 Additions and removals of classes or properties are the only exceptions, since they only have one of these two properties.
the filler of the range of a property has changed. We can form these operations by combing a ‘delete’ and an ‘add’ operation. However, we have included them in the ontology of basic changes because this information will often be available. For instance, logs of changes provided by tools will often contain information on modifications. ‘Modify’ operation classes have properties for both old and new values, which give a reversible specification of the modification. Figure 6 shows a screenshot with some basic changes.

4.2 Complex Change Operations

In addition to the basic change operations, the ontology of change operations also contains complex change operations. Complex change operations are operations that are composed of multiple basic operations or that incorporate some additional knowledge about the change.

Complex operations thus provide a mechanism for grouping a number of basic operations that together constitute a logical entity. For example, a complex operation siblings_move consists of several changes of superclass relations.

Complex changes could also incorporate information about the implication of the operation on the logical model of the ontology. For example, a complex change might specify that the range of a property is enlarged, that is, that the filler of the range changed to a superclass of the original filler. To identify such changes, we need to query the logical theory of the ontology. In contrast, basic changes can be detected by analyzing the structure only.

We define the ontology of complex changes as an extension of the ontology of basic changes. We model specific variants of a basic change as subclasses of the basic change class. For example, the basic change superclass_change has two subclasses: superclass_changed_to superclass and superclass_changed_to subclass. Figure 7 shows a number of complex operations in our ontology.

Knowing complex operations rather than only the basic ones has a number of benefits.

• First, we can use complex operations to improve the user interface for the task of verifying and approving changes. Quite often, an ontology editor performs a number of changes that are all part of one “conceptual” operation. Some complex operations, like sibling_move, capture this knowledge. Visualizing these operations helps the user to verify modifications.

• Second, knowing complex operations, we can transform instance data with less data loss. For example, consider the move of a class: if we just had the “remove class” and “add class” operations, we will lose all instances of that class; knowing that the class was moved allows to move the instance data, too.

• Finally, knowing complex operations enables us to determine the effect of operations more precisely. If we only know that the range of a property has changed, we cannot tell anything about the effect on data. However, if we know that the range of the property is enlarged, we know that all old instance data is still valid.

If we know a set of basic change operations, we can use a set of rules and heuristics to enrich the basic changes with complex ones. We describe some of these enrichment procedures in Section 5.

The set of complex change operations is never finished or complete. It is always possible to define new complex changes that are useful in some setting. At the same time, a specific application does not have to use (or commit to) all complex change operations. The set of basic operations is already sufficient to specify all possible transformations.

Representing actual modifications

By using an ontology to structure the operations, we do not need to define a syntax for the representation of actual changes. Instead, we can use the representation format that comes with the ontology language that we use. Our ontology of basic change operations uses RDF Schema as the “ontology language.”

Therefore actual changes can be represented as RDF data. Because both our ontology of changes and OWL itself can be represented

7 We used RDF Schema instead of OWL because the expressivity of RDF Schema was sufficient to capture the main aspects of the ontology (the hierarchy of operations and their properties). In fact, because RDF Schema is a subset of OWL, we can also say that our ontology is represented in OWL.
in RDF it is very simple to represent complex arguments. The RDF representation of the complex argument can just be inserted as value of a property in our ontology.

5 Finding Complex Operations

Existing tools that currently do provide change information, usually do it at the level of basic changes (e.g., logs of operations [10]). We are currently working on tools for identifying and presenting complex operations based on a set of basic operations or a structural diff between versions. In some cases, we can use a set of rules to generate a complex change from a set of basic changes (Section 5.1). For other changes, we may not have a definitive set of rules for finding the complex changes and will need to use heuristics to determine if a complex change occurred (Section 5.2). In the rest of this section we sketch some of the approaches for supplementing change description with complex operations that we are working on. However, as with everything else in the framework, these rules and heuristics can be extended with additional ones that other tools provide.

5.1 Using combination rules to find complex operations

Consider again the example in Figure 1. And consider the change to the classes that were subclasses of White wine in $V_{old}$ and became subclasses of Rosé wine in $V_{new}$. We assume that we have an instantiated ontology of basic changes between the two versions. We can view this change as a set of several basic operations:

1. add a superclass relation between Rosé wine and Vin gris
2. remove a superclass relation between Vin gris and White wine
3. repeat the same for classes Cabernet blanc and White Zinfandel

If we look at this set of operations conceptually, we can see that a complex operation was performed: a set of siblings was moved to a different location in the class hierarchy. Note that there are two levels of “enrichment” that we can identify in this example. First, we can recognize the “add superclass” -- “remove superclass” sequence for each of the classes Vin gris, Cabernet blanc, and White Zinfandel as a move in the tree. Second, we can recognize that Vin gris, Cabernet blanc, and White Zinfandel were and remain siblings in the class hierarchy and thus we have a “move siblings” operation.

The set of rules to recognize this change is rather simple: as long as we know that the class $A$ in $V_{new}$ is the same as the class $A$ in $V_{old}$ (this information is readily available in a structural diff) and their superclasses are different, we can identify a “move” operation. To recognize that a set of siblings was moved together, we compare arguments to the “move” operations. If the to and from arguments for a set of “move” operations are the same, we have a “move siblings” operation.

For some complex operations, just having a set of basic operations and access to $V_{new}$ is not sufficient to determine that the complex operation has occurred. We may also need to have direct access to $V_{old}$. Suppose we know that a range of a property $P$ was changed from $C_1$ to $C_2$. If we have access to $V_{old}$, we can check whether $C_2$ is a subclass of $C_1$ in $V_{old}$. If it is, then the range of the property $P$ was restricted. As a result, for example, some instances that use this property may become invalid. On the other hand, if we know that $C_2$ is a superclass of $C_1$ in $V_{old}$, we also know that the range of the property $P$ has become less restrictive (another complex operation). If our task is data transformation, we can conclude that no instances were affected. Therefore, if all we have, for example, are $V_{new}$ and a transformation set with basic operations, these are the complex operations we will not be able to identify. Restricting a range of a property is an example of such operation.

5.2 Adding Uncertainty to Finding Complex Operations

While we can precisely identify when a group of sibling classes was moved to a new place in the class hierarchy, some other complex operations in practice may not have such precise definitions and may require a set of rules involving uncertainty to determine if the complex operation has occurred. In other words, while in principle we can specify a precise set of rules to determine when a complex operation occurs, in practice additional changes in the ontology involving the same concepts, may make a decision that a complex change has occurred less clear-cut.

Consider for example the following operation: group a set of siblings to create a new superclass (create a new abstraction). We would have such an example if we grouped the rosé wines in Figure 1a together to create the Rosé wine class, but have left this new class as a subclass of White wine.

In the ideal case, we have the following conditions that describe the case when such an operation has occurred:

1. A class $C \in V_{old}$ has $n$ direct subclasses: $subC_1, subC_2, \ldots, subC_n$
2. There is a class $newC \in V_{new}$ such that:
   - $newC$ is a direct subclass of $C$
   - $\forall subC \in V_{new}$ such that $subC_i$ is a direct subclass of $newC$, $subC_1$ was a direct subclass of $C$ in $V_{old}$

However, the user may have also, for example, added other subclasses to $newC$ (e.g., adding new types of rosé wines). He may have also added another level of classes between $C$ and $newC$. Thus, we can rephrase the conditions for $newC$ above defining the heuristic to the following form:

- $newC$ is a subclass of $C$ (not necessarily direct)
- Among the direct subclasses of $newC$, most come from $subC_1, subC_2, \ldots, subC_n$

We need to determine empirically what is a practical value for $most$. Our heuristic rule could say that more than 50% of subclasses of $newC$ must be former subclasses of $C$; or that there is at most one level of classes between $C$ and $newC$.

Determining whether a class was split or two classes were merged into a single class has to rely on such heuristics even more unless an ontology-editing tool provides the merge and split operations directly and records them in the log. By just
looking at two versions of an ontology and even at the transformation set including simple operations, it is often hard, if not impossible, to determine whether either of these operations has occurred. However, if we have some additional information, we may be able to identify such complex operation in some cases. In particular, if concepts have sets of instances associated with them or sets of properties that differ significantly, we can have the following set of heuristic rules (for merge in this example):

1. A class \( C \in V_{\text{old}} \) has subclasses \( subC_1 \) and \( subC_2 \); \( I_1 \) and \( I_2 \) are sets of instances of \( subC_1 \) and \( subC_2 \); \( S_1 \) and \( S_2 \) are sets of properties for \( subC_1 \) and \( subC_2 \).

2. The class \( C \in V_{\text{new}} \) has a subclass \( subC \); \( I \) is a set of instances of \( subC \); \( S \) is a set of properties for \( subC \) and the following is true:
   - \( I \) is similar to \( I_1 \cup I_2 \)
   - \( S \) is similar to \( S_1 \cup S_2 \)

Here again we need to determine empirically the meaningful interpretations of similar to. Should the sets overlap by 90%? 50%? What if the classes were merged and then some of the instances of the merged class were deleted? Or new instances were added to the merged class?

We define the same set of rules and need to answer the same set of questions for the operations of splitting classes.

## 6 Summary And Outlook For Future Work

When we use ontologies in a distributed and dynamic environment, we need support for several ontology-evolution tasks, ranging from data translation to change visualization. Providing this support is difficult, as the distributed environment does not allow us to enforce specific development procedures, and we cannot count on having complete information about changes.

As a first step towards a solution of this problem, we have analyzed different formalisms for representing changes between ontologies and developed a framework that integrates and relates them. The framework specifies how we can supplement information in one representation with information from other representations.

An ontology of change operations is a key element in this framework. The ontology of change operations makes a distinction between basic operations and complex operations. Complex operations are often more useful for specification of the consequences of a change. We have described a number of rules and heuristics to distill complex operations from a set of basic operations.

The framework and the procedures that we described make it possible to supplement information about change, and thus to support ontology-evolution tasks that the original change information did not allow. However, the procedure does not guarantee, nor depend on, a complete description of a change. The more information we have, the more we can do with it.

Although many elements of the framework are already in place, a lot more work remains to be done. We need to experiment with the heuristics that we defined to test their effectiveness and to determine the optimal values for the parameters.

We will need to build translators from existing change representations, such as change logs provided by specific tools to the vocabulary of basic changes. A crucial feature of the framework is its extendability: it is always possible to define new interesting complex changes, new heuristics, or new and more complete ways of filling in missing change representations from available ones. Finally, we need change representations to build tools to support various evolution tasks. The more complete representation of change enables us to build better, more robust and more efficient tools.

## References


Human Centered Ontology Management with HCONe

Konstantinos Kotis, George A. Vouros
Department of Information and Communications Systems Engineering
University of the Aegean
Samos, Karlovassi, 83200 Greece
{kot, george}@aegean.gr

Abstract
The development of ontology management environments that empower communities of knowledge workers to shape their information space by being actively involved in the ontology engineering life-cycle is emerging. This is motivated by the need to incorporate ontology management tasks in knowledge-empowered organizations and the need to populate the semantic web with agents exploiting semantically annotated information items. Towards this target we aim at ontology management environments that focus to the way people interact with their conceptualisations and to the way conceptualisations are formed as part of the day-to-day activities of knowledge workers. The paper points on important issues for the development of human-centered environments for the management of ontologies and presents a prototype system, HCONe (Human Centered Ontology Management Environment), which has been designed with the aim to address these issues.

1. Introduction
Ontologies are formal conceptualizations of domains, describing the meaning of (abstract and concrete) domain aspects by means of concepts and their interrelations [Chandrasekaran et al, 1999]. Ontologies have been realized as the key technology to shaping and exploiting information for the effective management of knowledge and for the evolution of the Semantic Web. Ontologies establish a common vocabulary for community members to interlink, combine, and communicate knowledge shaped through practice and interaction among community members, binding the knowledge processes of creating, importing, capturing, retrieving, and using knowledge [Staab et al, 2001; Benjamins et al, 1998].

The incorporation of ontology management tasks in knowledge-empowered organizations can prove to be a hindrance if not done in a way that is seamless to the day-to-day activities of the community members [Vouros 2003]. Traditionally, knowledge engineers develop the ontologies that are required by knowledge workers\(^1\), and provide these ontologies for exploitation. However, due to constant updating, changing, and evolution of ontologies, there must be a close collaboration between knowledge engineers and workers, requiring the active and decisive involvement of the latter in many stages of the ontology management processes [Stojanovic, 2002].

In conjunction to the above, the proliferation of the semantic web aims at explicating the meaning of the worldwide available information. Ontologies constitute the backbone of this effort, providing machine-exploitable semantic information for the Knowledge Web [Fensel et al, 2000]. But how could we possibly populate the semantic web, enabling information providers to attach semantic information to every published information item, shaping in conjunction their consensual conceptualisations? Current efforts support the semi-automatic and manual structuring of web pages using ontologies, in conjunction to ontology learning and enrichment (e.g. [Erdman et al, 2001; Maedche et al., 2000]). In conjunction to these technologies, we need tools that would empower people to develop and manage ontologies in a seamless and transparent way, without bothering with the formalities and symbol-level details of specifications.

In this paper we conjecture that in parallel to research efforts concerning methods and techniques for engineering, learning and enriching ontologies, the need for tools that would empower people to minimize the up-front knowledge engineering effort and accelerate the knowledge processes is great. Towards this aim we adopt a human-centered approach to designing and developing ontology management environments. Such environments must support “knowledge workers” to shape their information space by being actively involved in ontology management tasks throughout the ontology engineering life cycle.

\(^1\) A knowledge worker is any member of an information production-exploitation community. Such communities may involve workers within an organization, or World Wide Web users with common interests.
To further support our conjecture for the need for human-centered tools, let us consider the following ontology management scenarios in a living organization setting:

**Scenario No 1:** Involved in a knowledge retrieval process, a worker is searching for a specific piece of information about best practices concerning the design of a product type. The retrieval tool exploits the ontology concerning product designs, but the worker can neither find the terms that she thinks to be appropriate for querying the system, nor can she get the needed information by any combination of existing terms. She soon finds out that the definitions of some terms must be changed to reflect the information related to the new case at hand. The information is there, but cannot be reached, since the ontology does not reflect the up-to-date practice of the organization. Imagine now the same case happening for five workers per day in a fast changing domain. We suggest that workers must be empowered to shape their information space, possibly working in collaboration with knowledge engineers.

**Scenario No 2:** In a knowledge use process, a worker browses, recalls existing knowledge items, and process them for further use. During this process the worker may produce derivations that should be captured as new knowledge, indexed by new terms, or combinations of existing terms. Capturing derived knowledge is very important. Empowering this worker with the proper tools for describing her conceptions formally, incorporating them in organization’s information repository, submitting and sharing this information with co-workers readily, accelerates much the knowledge processes.

**Scenario No 3:** In day-to-day information creation and import tasks, workers are devising business documents, proposals, product reports, best practices, problem/fault reports, etc. Indexing such information using formal ontological commitments should be done in a seamless way by knowledge workers themselves, during authoring, allowing them to devise, expand and update their shared conceptualisations at the same time.

Summarizing the above, this paper emphasizes on devising environments that focus to the way people interact and shape their conceptualisations and to the way conceptualisations are formed as part of knowledge workers communities’ day-to-day activities [Tennison et al, 2002]. The paper presents important issues for such environments and presents the overall architecture and functionalities of a human-centred environment for ontology engineering (HCOME), which has been designed and implemented, based on these requirements.

2. Management of ontologies

As it is widely argued and shown in the above scenarios, ontologies explicate conceptualizations that are shaped and exploited by humans during practice. Being part of knowledge that people possess, ontologies evolve in communities as part of knowing [Cook and Brown, 1999].

Ontology management in the context of communities of knowledge workers involves the development, evaluation and exploitation of conceptualizations that emerge as part of knowing. In particular it involves:

- **The development of individual ontologies.** People develop their own conceptualizations that may either explicate (e.g. by formalizing concepts, by taking notes about their meaning or just by naming them) or not (by storing them in the background of their minds). In their day-to-day activity people develop their conceptualizations either by improvising, by specializing/generalizing/aggregating existing concepts based on their experiences and on interaction with other community members, or by synthesising existing conceptualizations.

- **The development of commonly agreed group ontologies.** Developing commonly agreed and understandable ontologies is a very difficult and resource-demanding task that requires members of the communities to work synergistically towards shaping the information they exploit. Working synergistically, workers map others’ conceptualizations to their own and put them in the context of their own experiences. This leads to a conversation whose back-and-forth, as it is pointed in [Cook and Brown, 1999], not only results in exchanging knowledge but also in generating new knowledge.

- **The evaluation and exploitation of ontologies.** Exploitation and evaluation of ontologies as part of the day-to-day practice of communities can be considered only as part of knowing. Conceptualizations are put in practice or in the criticism of community members who, as already pointed, have to compare them with their own conceptualizations and put them in the context of their own experiences. Evaluation can result in new meanings since concepts are seen under the light on new experiences and evolving contexts.

Impediments for knowledge workers to participate actively in these tasks include their unfamiliarity with formal languages and knowledge engineering principles/methods, as well as with methods and techniques for constructing and synthesizing ontologies. Most of the existing ontology management environments have been designed and implemented for the knowledge engineer, concentrating mostly on the ontology development process at the symbol level. This implies that the deployment of these environments in organizations with limited experience in ontologies is essentially prohibited, leading communities to develop semantically-poor thesauruses for their domains, or even abandoning the trial for semantically annotating their resources, since in most of the cases they are not willing to pay the costs implied by employing knowledge engineering resources.

Leading tools for ontology engineering [ONTOWEB, 2002] employ powerful methods for manipulating ontologies but mostly at the symbol level. To empower knowledge workers to participate actively in the ontology engineering process in collaboration with knowledge
engineers, tools must enable people to improvise new ontologies, synthesize and map existing ontologies, and collaboratively develop ontologies with their co-workers, in ways that are natural for them to interact with their conceptualizations. This must happen in the background of the day-to-day knowledge intensive activities of workers, in a seamless way to their working practices, and so that the semantic validity of specifications is assured. Tools that are close to our aims at supporting knowledge workers to develop/manage ontologies, possibly in the absence of knowledge engines, include APECKS [Tennison et al., 1999] and WebODE [Arpírez et al., 2001]. APECKS supports “living” ontologies, allowing discussion and evaluation of the evolving domain conceptualizations. WebODE offers an extensible workbench to ontology management, based on a well-defined methodology for ontology development. However, both tools lack important facilities for managing the evolution of ontologies, such as version management and ontologies libraries.

In a greater extend than existing tools, we emphasize at providing greater “opportunities” for systems and tools to be used by members of knowledge management organizational units in their day-to-day activities, proposing the design of integrated ontology management environments using human-centered design principles [Hoffman et al., 2002].

The paper presents a human-centered ontology management environment (HCONE) that provides advanced functionalities for editing, viewing, managing the evolution, reasoning, integrating ontologies, and mapping ontologies to upper-level (top) ontologies and lexicons, putting emphasis on the way humans interact with their conceptualisations and on the way groups reach to an agreement on domain conceptualisations. Viewing ontology management from this point, we reveal “opportunities for using the technology that we never considered” [Clancey, 1999], providing also insights for technological advances in managing and engineering ontologies.

3. Human-centered Ontology Management

Human-centered computing (HCC) is a viewpoint intent on helping us achieve the full potential of computing so that might to maximize the value of computing to society in a new age of human and machine symbiosis [Hoffman et al., 2002]. Joint research efforts on cognition, ergonomics, psychology, social sciences and information technologies contribute to the emergent of human-centered systems, leading to a new world of applications and services. In this new era, systems’ designers have to really understand the use of systems in conjunction with the cognitive abilities, contextual constraints and activities of users.

It has been argued that the human-centered development of systems must take the triple of "people-machine-context" as the unit of analysis [Hoffman et al., 2002]. This involves studying people capacities and goals, computational mechanisms and interface capabilities, all within their appropriate contexts. Context involves organizations, activities, practices, norms and constraints that are inherent in these activities, policies, procedures, as well as devices and media that people use for communicating and doing their work.

Viewing people activities for managing conceptualizations of domains within the context of emergent communities we shall not focus on object-centered representation formalisms or on other technological issues, but put special emphasis on the physical way of interacting with these conceptualizations and on the way conceptualizations are formed by means of people interacting among themselves and practicing.

This section points on important issues for the development of human-centered environments for the management of ontologies. The list is by no means complete. However it points on issues that we consider to be the most important:

- **Provide the tools needed for the management of ontologies in an ecletic way.** Members of the communities must be allowed to follow any approach or combination of approaches for the development of ontologies, which better fits their practice, their working norms and constraints: They may improve by integrating concepts in a conceptual system, provide concepts with informal definitions attaching information items to them, compare, merge and refine/generalize existing ontologies. In this respect, version management and libraries of ontologies must provide people with the ability to keep track of changes in their conceptualizations.

- **Support people to communicate their conceptualizations without dealing with low-level implementation details.** The major issue for human-centered ontology management is that low level implementation details must be hidden from people who do not understand knowledge representation formalisms’ terminology. People must be given the power to express subtle ontological distinctions in a way that is natural to them but satisfies the formal constraints of the specifications too.

- **Provide powerful reasoning services for checking specifications’ consistency and coherency.** This is important for people to develop well-formed ontologies (i.e. ontologies without inconsistencies among the defined concepts, with coherency, and well organized) and to compare/map existing ontologies. Reasoning services must perform in the background, providing sufficient information to people about the well formed-ness of their ontologies. It is very important to notice that these services must not prohibit people to follow any approach to ontology development. On the contrary, feedback from the reasoning services must be provided in the form of help and advice about the validity and consistency of specifications, with respect to people cognitive capacities.

- **Provide facilities for exchanging and creating ontologies conversationally.** The aim is to support conversations between individuals in order to enable criticism and
encourage feedback among community members to ontological specifications. These facilities must provide the means for detecting new opportunities for collaboration, as well as for getting out of deadlocks within problematic situations that may arise during collaboration.

- Provide facilities for bridging different perspectives via concepts’ meaning disambiguation. This is important for the development of commonly agreed conceptualizations especially in communities where people from different disciplines may use the same term with different meanings or use different terms for the same concept.

- Provide facilities for the management of multilingual ontologies. Multilingualism in this context aims to support ontological specifications in different languages. Doing so, members of organizations may specify an ontology using their native language and may evaluate and exploit an ontology originated in another language, in their native language too.

The development of human-centered ontology management environments is challenging. Not only due to the expectations it creates, but also in that it requires the synergistic deployment of many technologies: intelligent and collaborative interfaces for editing and deploying knowledge bases, object-centered knowledge representation systems with powerful reasoning services, language-engineering technologies for terms’ disambiguation, to mention a few. HCON (Human Centered ONtology Environment) is a prototype environment that has been designed and developed taking into account the above-pointed issues.

4. HCON

HCON (Human Centered ONtology Environment) follows a decentralized model to ontology engineering that is shown in Figure 1. According to this model people can create their own ontologies stored in a personal space. Ontologies can be later publicized and shared among groups of workers that jointly contribute to ontologies development, with the aim to reach an agreement in conceptualizing their domain. During this process, workers may evolve ontologies by improvising in their personal space, map and synthesize their conceptualizations with the conceptualizations of their co-workers and discuss their arguments, objections and positions within the group. During collaboration, workers follow a structured argumentation process in which they may raise issues, propose solutions via positions, provide arguments for or against a position etc. Agreed ontologies are stored in a virtual space and can be further shared, evolved in workers’ personal space and so on.

HCON is a modular and integrated environment, providing access to any integrated tool in any HCON point. Doing so, workers are free to combine their own method for using the environment, following an eclectic way to ontology engineering. For instance, a worker may construct an ontology in her personal space while receiving comments on a previous version of the same ontology that has shared with co-workers. In the meantime, she is trying to comply with generic ontological commitments that the group has agreed to comply with, while in another slice of her work she is trying to merge her ontology with an
ontology issued by a co-worker. Towards these targets, HCOME provides facilities for (a) users to improvise their conceptualizations, (b) consult generic ontologies that provide important semantic distinctions, (c) manage different versions of their ontologies, tracking the differences between the versions, (e) track the generalization/specialization of an ontology during ontology development, (d) get proper consultation from machine exploitable readable lexicons by mapping concepts’ meaning to word senses, (e) merge ontologies and further manipulate merged conceptualizations, and (f) share their ontologies with groups of co-workers, following a structured conversation towards agreeing in domain conceptualization.

HCOME supports people to interact with their conceptualisations hiding low-level implementation details, enabling them to express subtle ontological distinctions, complying at the same time with formal constraints of specifications. Formal specification of ontologies in HCOME is done in the NeoClassic Description Logic [Patel-Schneider et al, 1996].

In particular, workers communicate their conceptualizations using the full expressive power of NeoClassic, without dealing with low-level implementation details. This is achieved by following canned natural language dialogues based on a slight variation of the What-You-See-Is-What-You-Meant (WYSIWYM) [Power et al, 1997] knowledge specification paradigm. As figure 2 depicts, while users specify the definition of a concept, they get a feedback text that reflects the definition of the corresponding concept. HCOME provides seamless access to reasoning services provided by Description Logics. These services include automatic concepts’ classification, concepts’ definitions consistency checks (e.g. between a concept and its subsumers) and detection of concepts’ definitions differences. Feedback from these reasoning services to users is constantly provided during ontology development/management and is of high significance. For instance, while tracking the differences between two versions of the same ontology or during merging, reasoning services identify semantically equivalent definitions allowing proper handing of versions, and disallowing semantic errors.

Critical to the ontology specification process is the lexicons consultation process. Through lexicon consultation users are guided to the consensual definition of terms, guided to follow well-established norms and practices in the community they are exercising their practice (e.g. by consulting a terminological lexicon) or in the wider context (e.g. by selecting the appropriate lexicalizations of their conceptions). In HCOME, lexicon consultation can be supported in any of the following three ways: (a) by mapping concepts definitions to word senses in a machine readable/exploitable lexicon through the disambiguation process, (b) by formally complying with generic ontological commitments of top level ontologies or (c) by simply consulting lexicons and other ontologies.

HCOME uses the Latent Semantic Indexing (LSI) technique [Deerwester et al, 1990] for mapping terms’ meaning to word senses. LSI is a vector space technique for

![Figure 2. HCOME concept specification: 1) natural language, 2) formal, 3) graphical representation](image)
information retrieval and indexing. It takes a large matrix of term-document association data and constructs a semantic space wherein terms and documents that are closely associated are placed near one another. Currently HCOME exploits the WordNet [Miller et al, 1990] lexical database to match descriptions of terms provided by the user with word senses in WordNet. LSI in this case constructs a semantic space of terms and word senses [Agirre and Rigau, 1996]. Word senses that closely match the given terms are provided to the user in an ordered way.

On the other hand, knowledge workers may follow a deductive approach to terms specifications by elaborating a generic top ontology. In this case, concepts definitions can be checked for their semantic validity against generic conceptualisations by means of the consistency checking mechanisms provided by the representation and reasoning system. In doing so, the construction of domain specific ontologies is speed-up and guided by important semantic distinctions specified in generic ontologies. HCOME, in its current status, provides the option of importing any generic ontology, given that it has been implemented in NeoClassic. We have kontakted experiments with a generic ontology for terminological resources, which incorporates EuroWordNet and SIMPLE semantic commitments [Vouros and Eumeridou, 2002].

Having mapped terms to word senses, concepts in different ontologies can be associated among themselves, resulting in effective merging of ontologies. Merging functionality in HCOME is performed in “batch mode”. The result of this automated process is a merged ontology whose creation is driven by heuristics, a mapping algorithm and the reasoning processes of NeoClassic. As figure 4 shows, HCOME presents the steps of the merging process to the end user as a feedback report, describing the actions the system has followed to the merged ontology. The resulted merged ontology can be further manipulated according to users preferences.

Furthermore, HCOME exploits the association of concepts to WordNet word senses for computing equivalent terms in other European languages.

Finally, a very important built-in HCOME functionality concerns sharing ontologies to group members and supporting group members’ participation in structured conversations about conceptualizations. This is a built-in, rather than a patched-on facility, since it has been designed in order to support people to discuss ontological aspects and incorporate their suggestions / positions to specifications, rather than a generic argumentation or discussion facility. This facility enables criticism, identifying possible opportunities for members’ collaboration, as well as overcoming deadlocks within problematic situations that arise in ontologies specification.

5. Concluding Remarks
HCOME provides a modular environment supporting an eclectic approach to ontologies management, aiming to provide and multiply the opportunities of exploiting ontology management tools in working places. We conjecture that this target can be achieved via designing human-centred ontology management environments.
The fast emergent areas of the semantic web and knowledge management push researchers to new development efforts, but to our view no specific principles have been raised for evaluating the impact of tools on making the foreseen targets achievable. In the latest EON-Workshop\textsuperscript{2}, assessments of the most well known ontology engineering environment have resulted to some very important conclusions:

a) ontology engineering environments’ development criteria cannot be considered in isolation from situations in which the ontology-based tools will be used. Criteria need to be connected to scenarios of use. Such scenarios need to be explained, further analyzed and be connected to activity models. Furthermore, there must be a balance between usage and technology, and between formality and informality [Giboin et al, 2002]. HCOME, following the human centered design approach, emphasizes on integrating ontology engineering environments with knowledge providers/consumers’ practices considerably, enabling knowledge workers to interact with their conceptualizations at a high-level.

b) There are two types of ontology engineering environments: (1) Environments for developing ontologies and (2) environments for mapping, aligning, or merging ontology [Noy and Musen, 2000]. Mapping environments may identify potential correspondences between concepts, provide the user with guidelines for defining these correspondences, or both. HCOME follows an eclectic approach to ontology management, integrating techniques for ontology management in a toolset.

\textbf{c)} The development of ontologies must be seen as a dynamic process that in most of the cases starts with an initial rough ontology, which is later revised, refined and filled in with the details. Ontology evolution has to be supported through the entire lifecycle, resulting to a living ontology [Stojanovic and Motik, 2002]. HCOME incorporates functionalities (including development, versioning, argumentation, merging) for supporting the management of continuously evolving ontologies.

\textbf{References}


\footnote{2 http://km.aifb.uni-karlsruhe.de/eon2002}


Engineering a complex ontology with time

Jorge Santos
GECAD - Knowledge Engineering and Decision Support Research Group
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática
4200-072 Porto - Portugal

Steffen Staab
Universität Karlsruhe (TH)
Institut AIFB
D-76128 Karlsruhe - Germany

Abstract

Because it is difficult to engineer a complex ontology with time, we here consider a method that allows for factorizing the complexity of the engineering process, FONTe (Factorizing ONTology Engineering complexity). FONTe divides the engineering task into building a time-less domain ontology and a temporal theory independently from each other. FONTe provides an operator $\otimes$ that assembles the two independently developed ontologies into the targeted ontology. We investigate the quality of the proposed operator $\otimes$ by applying it to a practical case study, viz. the engineering of an ontology about researchers including temporal interactions.

1 Introduction

In recent years, we have seen a surge of ontologies and ontology technology with many ontologies now being available on the Web. At the same time one could observe that most ontologies (e.g., consider the DAML ontology library at http://www.daml.org/ontologies/) engineered exhibit only rather simple structures, viz. taxonomies and frame-like links between concepts.

This observation might indicate that such — comparatively — simple structures are sufficient for the large majority of ontology-based systems. According to our own experiences about ontologies for knowledge portals [?] and power systems [Santos et al., 2001], however, one frequently needs intricate concept descriptions and interactions — in particular ones about time and space.

Because of intense (and fruitfully ongoing) research, many practical theories about time are now well understood. While the same can be said about the engineering of concept hierarchies and concept frames, the issue of how to engineer complex ontologies with intricate interactions based on time has not been researched very deeply, yet, rendering the engineering of a new complex domain ontology with time a labor intensive, one-off experience with little methodology.

In this paper, we present our ontology engineering methodology, FONTe (Factorizing ONTology Engineering complexity), that pursues a ‘divide-and-conquer’ strategy for engineering complex ontologies with time. FONTe divides a targeted ontology that is complex and that includes time into two building blocks, viz. a temporal theory and a time-less domain ontology. Each one of the two subontologies can be built independently allowing for a factorization of complexity. The targeted ontology is then build by assembling the time-less domain ontology and the temporal theory by the operator $\otimes$.

Thereby, the assembling operator $\otimes$ is very different from existing operators for merging or aligning ontologies [Noy and Musen, 2000; Rahm and Bernstein, 2001]. Merging ontologies is a process that intends to join different ontologies about overlapping domains into a new one and most of its problems and techniques are related to the identification of similar concepts through structure analysis (e.g. graph analysis, path length, common nodes or/and edges and lexical analysis). For instance, car from ontology 1, $O1.car$, and auto from ontology 2 $O2.auto$ may be defined to be identical in the merging process because of results of the structure analysis. To formalize the merging and aligning process, Wiederhold proposed a general algebra for composing large applications through merging ontologies of related domains [Wiederhold, 1994] and actually, the operations proposed (Intersection, Union and Difference) are about the similarities and differences of two ontologies.

In contrast, the result of $\otimes$ needs rather to be seen in analogy to the Cartesian product of two entities. For instance, car from ontology 1, $O1.car$, with its frame $O1.licensedInState$ is assembled by $\otimes$ with ontology 2 and its $O2.timeInterval$ in a way such that every car in the result ontology has a lifetime as well as multiple $O1.licensedInState$-frames with different, mutually exclusive life spans.

$\otimes$ is operationalized by an iterative, interactive process. It starts off with a human assembly — in the sense just explained — between an ontology $O1$, the time-less domain ontology, and an ontology $O2$, the temporal theory. It is then propelled by a set of rules and a set of constraints. The set of rules drives a semi-automatic process proposing combinations. The set of constraints narrows down the set of plausible proposals to valid ones.

We have applied the methodology FONTe with its opera-
tor ⊗ to a case study, where O1 is a time-less ontology about a semantic web research community and O2 is a temporal ontology. We have investigated how many assembling steps were proposed and evaluated their adequacy. The study results suggest that indeed FONTE provided a way to factorize the complexity of building large applications leading to more reliable and cheaper final products.

The rest of the paper is organized as follows. In Sections 2 and 3, we sketch the temporal ontology and the time-less domain ontology we have used for our case study, respectively. In Section 4 we describe the semi-automatic process of assembling two ontologies by ⊗, with some emphasis on the tool support developed to drive the process. Then we give an evaluation of our sample case in Section 5.

2 Temporal Ontology

The temporal ontology used for our case study (see Figure 1 for the depiction of an excerpt) embodies many concepts like Instant or Period routinely found in ‘standard’ ontologies like Time-DAML [Hobbs, 2002] or SUMO [Niles and Pease, 2001]. As argued by [Vila and Schwalb, 1996] a temporal representation requires the characterization of time itself and temporal incidence, which are represented in our temporal ontology by TemporalEntity and Eventuality, respectively.

We defined a further notion TimedThing that bridges between temporal concepts and the domain concepts that will be used during the assemble process. In particular, we have included the notion of Role as a core concept. While there are concepts that give identity to their instances (i.e. they are semantically rigid), e.g. while the identity of a particular person depends on being an instance of Person, the identity of the same person does not change when it ends being a student and starts being a professor. Thus, the notion of Role is important when connecting a temporal theory, e.g. Allen’s interval calculus, with a concrete domain, e.g. an ontology about researchers (cf., e.g., [Lehmann, 1996; Sowa, 1996; Guarino and Welty, 2000; Steinnann, 2000]).

Some of the choices for the temporal ontology that we made were driven by modelling objectives particular to the knowledge portal application that we had in mind when performing the case study. For instance, our objective was to have a 3-dimensional model of the world with time as an extra variable. For future applications an ontology engineer may prefer a 4-dimensional view, leading to a somewhat different temporal ontology and, thus, to an overall different target ontology in the end (cf. [Hayes et al., 2002]). The scope of this paper is to show a factorization of complexity into two sub-ontologies and a re-assembly into a target ontology in a way that is independent of such underlying concerns.

Temporal Entities In the temporal ontology we used for the case study there are two subclasses of TemporalEntity: Instant and Period. The relations before, after and equality can hold between Instants, respectively represented by the symbols: <, >, =, allowing to define an algebra based on points [Vilain and Kautz, 1986]. It is assumed that the before

1 The reader may note that we could not directly use a ‘standard’ temporal ontology like Time-DAML or SUMO as they do not include the notion of role.
3 Case Study Example: The SWRC Ontology

The assemble process may be used either for development of ontologies with time from the scratch as well as for re-engineering existing ones in order to include time. For our case study we have used the SWRC (Semantic Web Research Community) ontology (http://ontobroker.semanticweb.org/ontos/swrc.html) that served as a seed ontology for the knowledge portal of OntoWeb.

SWRC comprises 55 concepts, 151 relations and 25 axioms. The results presented in the section 5 are based on re-engineering the complete SWRC. Here we present an excerpt that is also used in order to elucidate the assembling process with $\odot$.

The SWRC ontology contains different types of axioms, namely, transitive, inverse, symmetric and general, being that, any axiom that does not fall in the first three types is called 'general'.

\begin{align}
\text{isa}(\text{Student}, \text{Person}) & \quad \text{studiesAt}(\text{Student}, \text{University}) \\
\text{isa}(\text{Employee}, \text{Person}) & \quad \text{member}(\text{Person}, \text{Project}) \\
\text{isa}(\text{Male}, \text{Person}) & \quad \text{isAbout}(\text{Project}, \text{Topic}) \\
\text{isa}(\text{Graduate}, \text{Student}) & \\
\forall p,t : \text{isAbout}(p,t) & \leftrightarrow \text{dealsWithIn}(t,p) \quad (11.1) \\
\forall p, t, \text{top} : \text{worksOn}(p, \text{top}) & \iff (11.1) \\
\exists \text{proj} : \text{instOf}(p, \text{Person}) & \wedge (11.2) \\
\text{instOf}(\text{proj}, \text{Project}) & \wedge (11.3) \\
\text{instOf}(\text{top}, \text{Topic}) & \wedge (11.4) \\
\text{isAbout}(\text{proj}, \text{top}) & \wedge (11.5) \\
\text{member}(p, \text{proj}) & \\
\end{align}

Figure 2: Excerpt of SWRC ontology

4 The Assembly Process

The assembly process comprises two main building blocks. First, the specification of temporal aspects for a time-less domain ontology remains dependent on the conceptualization of the ontology engineer. Therefore, it is very important that the engineer may interactively influence the process. Second, in order to facilitate and accelerate the assembly of time-less domain concepts with temporal notions, the interactive process is supported by heuristics asking and pointing the engineer.

The assembling process runs as depicted in Figure 3: It starts by an Initial Setup. Some basic operations are performed, namely loading the ontologies to be assembled, loading a set of rules to drive the process and initializing some process parameters. The rules and parameters are defined separately from the tool in order to allow for adaptations to the particular needs of different temporal ontologies. However the rules and parameters do not change when a new domain ontology is to be assembled. The Target Ontology initially corresponds to the union of the time-less domain ontology, $O1$, and the temporal theory, $O2$.

In the Analyze Structure step a set of tests are performed that restrict the set of possible task instances to plausible ones, which are then proposed by insertion into the Task List. As more information becomes available in subsequent iterations, the usefulness of results provided by the structure analysis improves.

In every iteration the engineer decides whether to accept an automatically proposed task instance from the Task List. Alternatively, the user may take the initiative and assemble a new task instance from scratch. Then a set of logical tests (Validate) are performed in order to detect the existence of any knowledge anomalies (e.g. circularity or redundancy [Preece and Shinghal, 1994]). In contrast, the acceptance of a proposed task instance does not require further checks as the checks are tested for validity before the user sees them.

By the Execute Task step the corresponding changes are made to the target ontology. Thereafter, the user decides either to pursue another iteration or to go to Conclude Process and accept the current Target Ontology as the final version.

4.1 Data Structures

In the remainder of Section 4, we first present the principal data structures we use for defining the heuristics, before we elaborate on its application in three subsections for assembling concepts, frame-like relations and axioms, respectively (Sections 4.2 to 4.4).

Task. We have already informally used the notion of task in order to refer to an action template (i.e. a generic task) that may be instantiated and executed in order to modify a current target ontology. A task is defined by its procedure and a task question. The task code uses a set of keywords with the commonly expected semantics of structured programming (e.g. if, then, else) and some special keywords, do and propose, the semantics of which we provide subsequently.

Task instance. A task instance is fully identified by its head
and creates a corresponding entry on the History List (TaskInstance).

Task question. Before the execution of a task, the system asks a task question in natural language to the engineer in order to determine if the proposal should really be accepted or not and in order to ask for additional constraints that the user might want to add. The task question is defined by a List of words and parameters used to compose a sentence in natural language.

For instance, the following task question exists for the previous example:

create-role-of(#arg1,#arg2):
[‘Define’,#arg1,’as role of’,#arg2,’?’]

It implies that the question “Define Student as role of Person?” would be posed before executing the example task instance.

In order to manage various task instances, the assembling algorithm uses the following data structures:

Task List. This is a list of tuples (TaskInstance,ListOfTriggers,Weight) storing proposed task instances together with the triggers that raised their proposal and their weight according to which they are ranked on the task list. Thereby, TaskInstance has been defined before.

ListOfTriggers denotes the list of items that have triggered the proposal. A trigger is a pair 

\{(TriggerType,TriggerId)\}

where TriggerType has one of the values concept, relation or axiom and the TriggerId is the item identifier. For instance, the pair \{concept, Person\} is a valid trigger. The list is useful for queries about what proposals by a specific item or a certain TriggerType:

Weight. Since competing task instances may be proposed, Weight is used to reflect the strength of the proposal on the TaskList.

Task History is the list of all tasks that were previously performed. This list is useful to allow the undo operation and to provide statistics about the assembly process.

do(TaskInstance). The function do executes a task instance and creates a corresponding entry on the History List.

propose(TaskInstance,Trigger,Weight). The function propose creates a proposal and asserts the corresponding tuple in the Task List.

4.2 Assembly of Concepts

As mentioned before system proposals are generated based on rules and constraints. Typically, in the initial phase only few interactions between the initially time-less domain ontology and the temporal theory are known and, hence, few proposals are generated. Furthermore, the assembling of concepts with temporal attributes needs to fulfill fewer constraints than the assembly of relations and far less than the assembly of axioms. Thus, proposals for modifications with concepts are typically made first — and elaborated in this subsection.

For the running example here, we assume that a user defines and executes a task instance of assemble-concepts that subclasses a concept C1, viz. person, from a C2, viz. TimedConcept:

\[
\text{procedure assemble-concepts}(C1,C2) \\
\text{if } (C2=’\text{TimedConcept’ or C2=’Role’) do (create-relation(isa(C1,C2)))} \\
\text{assemble-related-axioms}(C1) \\
\text{assemble-related-relations}(C1) \\
\text{end-if}
\]

The corresponding procedure assemble-concepts creates a new isa relation between the Person and TimedConcept and then proposes further assembling tasks for related concepts, relations and axioms. Tracing the changes that may be proposed to related concepts in assemble-related-concepts, we find that it proposes the definition of Employee, Student and Male as possible roles of Person.\(^2\)

\[
\text{procedure assemble-related-concepts}(C) \\
\text{foreach S do} \\
\text{if S is a sub-concept of C} \\
\text{W=calculate-weight(S,C)} \\
\text{T=the trigger for this proposal} \\
\text{propose(create-role-of(S,C),T,W)} \\
\text{end-if} \\
\text{end-do}
\]

Later, if definition of Student as role becomes accepted, recursively Graduate will be proposed to become a role of Student utilizing create-role-of:

\[
\text{procedure create-role-of(S,C)} \\
\text{%delete isa(S,C) if true; if not, no effect } do (delete-relation(isa(S,C))) \\
\text{do(create-relation(roleOf(S,C)))} \\
\text{do(assemble-concepts(S,’Role’))} \\
\text{end-procedure}
\]

with temporal-role-constraint (S, C) defined by

\[
\forall o,p1,p2:\text{containedBy}(p1,p2)\rightarrow instOfDuringAt(o,S,p1) \land instOfDuringAt(o,C,p2).
\]

Assuming that Male were not accepted as role of Person in the further course of assembly, the result depicted in Figure 4 would be obtained.

The reader may note that this result crucially depends on our temporal theory, but that rules could be easily modified to accommodate other theories (e.g., ones without roles).

\(^2\)The character ‘%’ indicates a line remark.
Person knowsAbout a Topic ever after he has encountered it on a Project, i.e. intersecting the lifetimes of axiomWorksOn. axiomWorksOn defines that a Person who positions in first-order horn logics with function symbols.

For instance, let us consider the axiom defined in Fig- ure 2 by lines A1.1 through A1.6 and let us name it axiomWorksOn. axiomWorksOn defines that a Person who works in a Project that is dealing with a Topic, worksOn this topic.

In order to assemble time into the axiom representation we must consider the constraints available for the instances of participating concepts and, furthermore, the ontology engineer must define which one of these constraints is used in which way. For instance, for the relation worksOn it may be adequate to say that the Person worksOn the Topic as long as he is a member of the Project and as long as the Project isAbout the Topic, i.e. intersecting the lifetimes of relations in lines A1.5 and A1.6. For an analogous structure where knowsAbout(pers, top) appears in the head instead of worksOn(pers, top), however, the conclusion might be that a Person knowsAbout a Topic ever after he has encountered it in a Project until he dies, i.e. restraining the knowsAbout-relation only to the earliest time-point of the encounter and the life-time of the Person.

Since, the only difference between the two example structures lies in the naming of the relations and the intentions associated with their names, it is necessary to involve the user for defining additional temporal constraints.

The task assemble-axiom asserts an additional temporal precondition for each concept that is temporally quanti- fied. In our running example, this affects the instances of Person (A1.2) and Project (A1.3). Furthermore, it also updates the preconditions of the axiom involving relations that need to be temporally modified. In the running example, this affects, e.g., member (A1.6). Finally, the user is asked to define one (or several) constraint(s) that relates all the timed variables (intervals or instants) for the pre- or the postconditions of the axiom.

procedure assemble-axiom(A)  
% for each concept present in the axiom  
% LC is a list of all new constraints  
foreach C partOf(A)  
  if isa(C,’Eventuality’) or  
    isa(C,’TimedThing’)  
    then add-concept-constraint(C,A)  
    LC=concat(LC,C)  
  end-if  
end-do  
% for each relation present in the axiom  
% LR is a list of modified constraints  
foreach R partOf(A)  
  if isa(R,’TimedRelation’)  
    then modify-relation-constraint(R,A)  
    LR=concat(LR,R)  
  end-if  
end-do  
% select a temporal constraint over  
% all the timed variables  
if (LC not empty) or (LR not empty)  
  then select-temporal-constraint(A,LC,LR)  
end-if  
end-procedure

Then the updated axiom includes previously existing pre-conditions (A1.2 to A1.5) and further ones about Person (A2.1), Project (A2.2) and member (A2.3) — as well as a temporal constraint (e.g.intersection) over them (A2.4). Hence the structure of the modified axiom looks like:

\[
\forall p, top, t: \text{worksOn}(p, top, t) \leftarrow \text{(A1.2)...(A1.5)}
\]

\[
\exists t1, t2, t3: \text{duringAt}(p, t1) \land (A2.1)
\]

\[
\text{duringAt}(prj, t1) \land (A2.2)
\]

\[
\text{member}(p, prj, t3) \land (A2.3)
\]

\[
\text{tempRelation}([t1, t2, t3], t) \land (A2.4)
\]

5 User Interaction

FONTE uses an approach that relies on iterative interaction with the user. All the assembly operations as well as interactions with the Task List and the History List can be performed using both the graphical drag-and-drop tool or a command line interface (please refer to appendix A).

In order to evaluate the effectiveness of FONTE, we have numerically evaluated the assembly tasks proposed and executed for the ontologies presented in Sections 2 and 3. An ontology engineer interactively assembled the two ontologies and by our process log we could evaluate the success of our approach by listing the numbers from Table 1:

The reader may note that the engineer initiated only very few tasks (10). From this initial structure a large number of tasks were proposed and accepted by the user (135). Of these
note that F ONTE implements an iterative and interactive approach which was previously successfully adopted in sophisticated tools for merging ontologies [Noy and Musen, 2000; Mitra and Wiederhold, 2001; McGuinness et al., 2000]. Also, F ONTE does not substitute these other methodologies, rather we envision that one wants to separate the target ontology to be built into different (possibly overlapping) domains as well as into time-less and temporal subontologies. The two ways of carving up the engineering task need different, complementary methodologies.

7 Conclusions and Future Work

We have proposed a method, named F ONTE, for engineering complex ontologies with time by assembling them from time-less domain ontologies and a temporal theory. F ONTE provides an operator ⊠ that operationalizes the assembly in an interactive way. ⊠ combines two independently developed ontologies into a target ontology.

Though, so far, we have only studied the assembly of time into a given ontology, we conjecture that F ONTE may also be applied to integrate other important concepts like space, trust, or user access rights — concepts that pervade a given ontology in intricate ways such that a method like F ONTE is needed in order to factorize engineering complexity out leading to more consistent and cheaper target ontologies.

Acknowledgments

This work was partially supported by a Marie-Curie Fellowship and by FCT (Portuguese Science and Technology Foundation) with programs ONTOMAPPER (POSI-41818) and SANSKI (POCTI-41830). We are also in indebted to our colleagues in LS3, particularly to Gerd Stumme and Julien Tame and Nuno Silva in ISEP.

References

A User’s Choices

A command line interface was developed for the tool that implements the FONTE method. Table 2 describes commands used on the execution of assemble tasks both are available through the acceptance of automatic produced proposals or by user driven operations.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link(C1,C2)</td>
<td>Assembles concept C1 from domain ontology with concept C2 from general ontology</td>
</tr>
<tr>
<td>link(R,C1,C2)</td>
<td>Defines the relation R between concepts C1 and C2 as a temporal relation</td>
</tr>
<tr>
<td>acptnextprop</td>
<td>Accepts the next proposal in the Task List for execution</td>
</tr>
<tr>
<td>acptpropnum(N)</td>
<td>Accepts proposal N from Task List for execution</td>
</tr>
<tr>
<td>acptprops(L)</td>
<td>Accepts a list of proposals L from Task List for execution</td>
</tr>
<tr>
<td>undo</td>
<td>Undo last performed assemble task</td>
</tr>
</tbody>
</table>

Table 2: Assemble Tasks

Table 3 describes commands used for handling Task List and History List. Provided facilities are showing and sorting list contents.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shownextprop</td>
<td>Show detailed information about next task on Task List, including a question in natural language</td>
</tr>
<tr>
<td>showprop(N)</td>
<td>Show detailed information about task N on Task List, including a question in natural language</td>
</tr>
<tr>
<td>sortproplist(F)</td>
<td>Sorts tasks in Task List by field F (e.g., Weight or Trigger)</td>
</tr>
<tr>
<td>showproplist</td>
<td>Shows content of Task List</td>
</tr>
<tr>
<td>showhistlist</td>
<td>Shows content of History List</td>
</tr>
</tbody>
</table>

Table 3: Historic and Task Lists Handling

Finally, table 5 describes some auxiliary commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd(C)</td>
<td>Performs command C, allowing to evoke any assemble task not available through command line</td>
</tr>
<tr>
<td>help</td>
<td>Shows help texts about available commands</td>
</tr>
<tr>
<td>exit</td>
<td>Finishes program execution</td>
</tr>
</tbody>
</table>

Table 5: Auxiliary Commands
The Sigma Ontology Development Environment

Adam Pease
Teknowledge
1800 Embarcadero Rd
Palo Alto CA 94303
650 424-0500
650 493-2645
apease@teknowledge.com

keywords: natural language, ontology

Abstract

Abstract: We discuss the development of an environment for formal knowledge engineering. The Sigma system is an advance over previously developed systems in that it integrates a number of modern ontology development tools, which has motivated a number of research issues. Primary components include an ontology browsing and editing environment, a first order logic inference system and a natural language to logic translator. Although largely independent of any particular ontology, it supports a number of publicly available formal ontologies.

Introduction

Human knowledge is complex, and human language is very expressive. Computer systems that process natural language have been limited to a shallow level of understanding by the very complexity of the information that they are expected to handle. Conversely, systems that manage information in databases have been left to reason either in very narrow and pre-determined ways, or to perform processing tasks that must be interpreted by humans.

It is desirable to employ computer languages for representing knowledge that are as expressive as possible, and as close to the expressiveness of human language. Expressiveness does have a computational burden. In the opinion of the author, the best compromise available for many knowledge representation efforts is first order logic.

While there are other choices, such as description logic and higher order logic, they represent departures from a middle ground. First order logic is undecidable, but can be made tractable for certain kinds of reasoning, which will be detailed later. Description logic (Baader, et al, 2003) has some very attractive formal properties, but is significantly less expressive than first order logic, and therefore limits the knowledge engineer’s ability to express the semantic content of statements found in normal human discourse. Higher order logic (Carreno et al, 2002) on the other hand is much harder to reason with.

The Sigma system was designed to integrate several different kinds of tools for working with knowledge in first order logic.

Browsing and Editing

The Sigma system contains several different sorts of browsers for viewing formal knowledge bases. The most basic component is a term browser, which presents all the statements in which a particular term appears. The statements are sorted by argument position and then the appearance of the term in rules and non-rule statements are then shown. All the statements are hyperlinked to the terms that appear in them.

Two types of tree browser are provided. One provides an automatic graph layout. Another shows a textual hierarchy. The user can chose the term to start with, the number of “levels” that should be presented from the term, and the binary relation to chose as the predicate that links the different nodes in the graph. For example, if the user asks for a graph of the term IntentionalProcess and for the subclass relation, the system will go “up” the graph to display the term Process, and down the graph to display the terms Keeping, Guiding, Maintaining etc. The user can ask for more levels up or down the graph. Note that any relation can be chosen so for example a presentation of partonomies, or attribute hierarchies is also supported by choosing the relations part and subAttribute respectively.

The editing system is currently rather rudimentary, consisting of the ability to type formulae and have some simple syntax and other error checking performed. We plan to offer a frame-based editing system similar to, or incorporating open-source components from, the Protégé (Gennari et al, 2002) system.

We have developed some more sophisticated tools for collaborative knowledge engineering. The System for Collaborative Open Ontology Production (SCOOP) (Pease & Li, 2003) supports an automated workflow process for development of ontologies and resolution of conflicts among ontologies. The SCOOP system employs a theorem prover to detect inconsistencies among ontologies.

SCOOP addresses several types of inconsistencies. The first we term vertical inconsistency. This refers to when a
set of ontologies are loaded and one ontology has a contradiction with another ontology on which it depends. Inconsistency of this type is never allowed, because from a contradictory knowledge base, any proposition can be concluded to be true. SCOOP enforces a workflow process that requires a resolution to such a situation.

The second kind of inconsistency we term horizontal inconsistency. This refers to a situation in which knowledge products created by different knowledge engineers, and which do not have a mutual dependency, are mutually contradictory. SCOOP alerts developers to such a situation, but does not require that it be eliminated. It is possible for two knowledge bases to represent different perspectives, which are contradictory, but locally valid.

We employ the KIF (Genesereth, 1991) language and have a translator to and from DAML (Hendler & McGuinness, 2000). A translator to and from Protégé (Gennari et al, 2002) is partially completed at this time.

Language Generation

Each statement can also be presented as a natural language paraphrase similar to (Sevcenko, 2003). Note that we also use Sevcenko’s language templates for English and Czech. Each term in an ontology can be given a natural language presentation form, indexed by the human language. For example, the SUMO term DiseaseOrSyndrome can be presented as “disease or syndrome” and an Italian presentation would be shown as “malattia o sindrome”. Combined with the English understanding mechanisms described below, this gives us a very rudimentary language translation capability.

Although presentation of terms is straightforward, presentation of statements is more complicated, and roughly patterned after C-language printf statements. For example the relation “part”, which states that one object is a part of another, has the corresponding language generation template “%1 is %n a part of %2”. The first argument to the logical relation is substituted for %1 etc. Note that this substitution is recursive, so that complex statements with nested formulæ can be translated effectively. The %n signifies that if the statement is negated, that the negation operator for the appropriate human language should be inserted in that position.

We currently have language templates for English, Czech, German and Italian. Some work has been done on Hindi, Telugu, Tagalog and Russian.

Natural Language Understanding

We take a restricted language approach to natural language understanding. Deep understanding of unrestricted human languages is too hard with present technology. Our approach is to create a restricted version of English, which is grammatical but more limited than true natural language. Statements must be present tense singular. Although different tense and number can be accepted, the system paraphrases such sentences into a present tense singular form. Ambiguous word choices default to the most popular sense of the word, with an escape mechanism for the user to select a different sense. Anaphoric references are handled with a simple mechanical algorithm. Some kinds of quantification (every, some, all) can be handled.

The current system is capable of taking simple English sentences and translating them into KIF, using terms from our SUMO upper ontology (described below). We have mapped all 100,000 WordNet (Miller, et al, 1993) word senses (synsets) to SUMO, one at a time, by hand over the period of a year (Niles & Pease, 2003). This endows our natural language translation system with a very large vocabulary.

Our approach is presented more fully in (Murray, Pease & Sams, 2003).

Inference

One basic issue is that of allowing variables in the predicate position. While the general case of this results in expressions being beyond first order logic, there is a slightly more restricted case, which is first order. A statement with a variable in the predicate position is only higher order if a quantifier ranges of all possible relations, rather than the finite set of relations in a particular knowledge base. Since prepending another relation in front of every statement makes everything first order for such a restricted practical case, this therefore must mean that the statement wasn’t really higher-order in the first place.

There is also an issue with prohibiting statements as arguments to relations. When arguments to relations are statements, then such a statement is higher order. We could either have a quote operator or define a Statement class and require all predicates have their argument types defined (which would allow us to know in advance whether an argument is a Statement, and therefore needs to be implicitly quoted). A statement that has this issue is (believes John (likes Bob Mary))

Currently, such statements are quoted, and therefore become opaque first-order objects rather than statements.

Another issue is row variables, for use in variable arity relations. One proposed version of KIF (Hayes & Menzel, 2001) has what are alternately called row, or sequence variables. They are denoted by the '@' symbol in KIF statements. They are analogous to the lisp @REST variable. This is not first order if the number of arguments it can "swallow up" is infinite. However, if row variables have a definite number of arguments, it can be treated like a macro, and becomes first order. For example,

(=> (and
     (subrelation ?REL1 ?REL2)
     (holds ?REL1 (@ROW))
     (holds ?REL2 @ROW)))
would become

\[
\begin{align*}
\Rightarrow \quad & \\
& \begin{cases}
(subrelation \ ?R1 \ ?R2) \\
(\text{holds} \ ?R1 \ ?A1)
\end{cases} \\
& \begin{cases}
(\text{holds} \ ?R2 \ ?A1)
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\Rightarrow \quad & \\
& \begin{cases}
(subrelation \ ?R1 \ ?R2) \\
(\text{holds} \ ?R1 \ ?A1 \ ?A2)
\end{cases} \\
& \begin{cases}
(\text{holds} \ ?R2 \ ?A1 \ ?A2)
\end{cases}
\end{align*}
\]

etc.

Also note that this "macro" style expansion has the problem that unlike the true semantics of row variables, that it is not infinite. If the macro processor only expands to five variables, there is a problem if the knowledge engineer uses a relation with six. Because of that, Sigma's syntax checker must prohibit relations with more arguments than the row variable preprocessor expands to.

**Default reasoning**

Given the example KB:

\[
\begin{align*}
\Rightarrow \\
& \begin{cases}
\text{instance} \ ?X \ \text{Bird} \\
\text{canFly} \ ?X)
\end{cases} \\
\Rightarrow \\
& \begin{cases}
\text{instance} \ ?X \ \text{Penguin} \\
\text{not} \\
\text{canFly} \ ?X)
\end{cases} \\
\text{subclass} \ \text{Penguin} \ \text{Bird}
\end{align*}
\]

We would like the second rule to override the first. The general case of default reasoning is very difficult (Schlechta, 1997). However, we believe that there is an easy way to address the most common case and get the utility we need. Given a relation

\[
\begin{cases}
\text{exception} <\text{formula-specific}> <\text{formula2-general}>
\end{cases}
\]

we can implement a macro that modifies the first formula above to exclude the exception specified and generate the KB

\[
\begin{align*}
\Rightarrow \\
& \begin{cases}
\text{instance} \ ?X \ \text{Bird} \\
\text{not} \ (\text{instance} \ ?X \ \text{Penguin})
\end{cases} \\
\Rightarrow \\
& \begin{cases}
\text{instance} \ ?X \ \text{Penguin} \\
\text{not} \\
\text{canFly} \ ?X)
\end{cases} \\
\text{subclass} \ \text{Penguin} \ \text{Bird}
\end{align*}
\]

Of course, this is only easy in the specific case where two rules have opposite conclusions and the antecedent of one subsumes the other. It's possible that in easy cases we might be able to have Sigma even generate the (exception... clause that the macro will use.

The algorithm for the macro simply would be, given the (exception... statement, to negate the antecedent of the <formula-specific> and add it as a conjunction to the <formula2-general>. Like other macros, this has implications for proof presentation that are also unresolved.

**Proof Presentation**

Sigma includes several options for proof presentation. Despite the fact that most textbooks present proofs as linear structures, proofs are trees. The same sub-proof may be used several times in reaching different intermediate conclusions. While this is a byproduct of automated reasoning, it is rarely useful to the human user. Therefore, Sigma eliminates redundant paths of reasoning in order to create a linear proof.

Sigma also has options for how it presents multiple proofs. Often, the same proof, with the same proof steps, can be reached via a different search order. This means that although two tree-structured proofs may be different, they may work out to be identical when redundant paths are removed and a linear proof structure generated. Sigma supports an option for hiding such duplicate proofs.

On additional option is to suppress proofs that may have different steps but which lead to the same answer. Sometime it is useful to a knowledge engineer to see these alternate proofs, and sometime not.

**Reasoning with Equality**

One problem in a first order theorem prover with procedural attachments for arithmetic is that the equality operator masks normal unification and inference. Currently Sigma converts the SUMO term 'equal' to '=' when sending assertions and queries to the theorem prover and then converts back for proof formatting. We are experimenting with changing the name to 'eval' and mapping that string to the theorem prover’s '='.

Our current idea is to have three predicates, one, which handles evaluation, one, which handles inference, and another, which combines the two.

\[
\begin{align*}
\text{equalAsserted} & \quad \text{eval} \\
\text{equal} & \\
\text{equalAssigned} & \quad \text{eval}
\end{align*}
\]

The following examples of assertions and queries should make clear how this will work.

assert:

\[
\begin{align*}
(\text{equal} \ (\text{CardinalityFn} \ \text{Continent}) \ 7)
\end{align*}
\]

macro expands to:

\[
\begin{align*}
(\text{eval} \ (\text{CardinalityFn} \ \text{Continent}) \ 7) \\
(\text{equalAssigned} \ (\text{CardinalityFn} \ \text{Continent}) \ 7)
\end{align*}
\]
Ontology

The Sigma system has been designed to be as independent of a particular ontology as possible, but there are many features that are available when a standard ontology is used. Current features are primarily to do with error checking. Knowing a standard method for defining argument type restrictions, class-subclass relations and documentation allows us to alert the user when such statements are conflicting or not present.

We employ the Suggested Upper Merged Ontology (SUMO) (Niles & Pease, 2001) as the system’s standard ontology. SUMO has been released to the public for free since its first version in December of 2000. Now in its 47th version, the SUMO has been reviewed by hundreds of people and subject to formal validation with a theorem prover, to ensure that it is free of contradictions. The SUMO contains roughly 1000 terms and 4000 statements, of which 750 are rules. As mentioned above, it has been mapped by hand to the WordNet lexicon, which has acted as an additional check on completeness and coverage.

A number of domain ontologies have been created that extend SUMO and can be used in the Sigma system. They include

- A Quality of Service ontology, covering computer systems and networks
- An ecommerce services ontology
- An ontology of biological viruses
- A financial ontology
- An ontology of terrain features
- Ontologies of weapons of mass destruction and terrorism
- An ontology of government and governmental organizations
- Taxonomies of the periodic table of the elements and industry types

Related Work

There have been a number of ontology editing environments created including Ontolingua (Farquhar et al, 1996), Ontosaurus (Swartout et al, 1996), Protégé (Gennari et al, 2002), and Cyc (Lenat, 1995). Ontosaurus and Cyc both have inference components. Cyc is the only system to contain a standard ontology. Cyc also contains a natural language component although little or no public information about that work is available.

The GKB editor (Paley et al, 1997) is an example of a graphical ontology editor. SNARK (Stickel et al, 1994), and Otter (Kalman, 2001) are examples of theorem provers. Many theorem provers have been tested on the CADE/TPTP (Sutcliffe et al, 2002) competitions. However, limited efforts have been made to apply formal first order theorem proving to expert system and common sense reasoning on large knowledge bases. The Sigma system brings all these types of components together, and in the process, addresses a number of research issues that have not been evident in use of these separate components.

Acknowledgements

The author would like to thank Randy Schulz for his efforts on early versions of this system.
References


B. Reasoning and Learning
On ontology-based querying

Troels Andreasen and Henrik Bulskov and Rasmus Knappe
Department of Computer Science,
Roskilde University,
P.O. Box 260, DK-4000 Roskilde, Denmark
{troels,bulskov,knappe}@ruc.dk

Abstract
In this paper we introduce an approach to exploit knowledge represented in an ontology in answers to queries to an information base. We assume that the ontology is embedded in a knowledge base covering the domain of the information base. The ontology is first of all to influence ranking of objects in answers to queries as measured by similarity to the query. We consider a generative framework where an ontology in combination with a concept language defines a set of well-formed concepts. Well-formed concepts is assumed to be the basis for an indexing of the information base in the sense that these concepts appear as descriptors attached to objects in the base. Concepts are thus applied to obtain a means for descriptions that generalizes simple word-based information base indexing. In effect query evaluation is generalized to be a matter of comparison at the level of concepts rather than words.

1 Introduction
The approach presented here concerns ontology-based querying. For the information base targeted for querying we assume an ontology (probably embedded in a knowledge base) covering the domain of the information base.

The aim is to utilize knowledge from a domain-specific ontology to obtain better and closer answers on a semantical basis comparing concepts rather than words. Better answers are primarily better ranked information base objects which in turn is a matter of better means for computing the similarity between a query and an object from the base.

The ontology plays its role behind the scenes – it defines and relates the concepts that are the basis for comparing queries and answers. However, even though it may for other reasons be relevant, it is not essential that the ontology and the concepts and relations it encloses are revealed to users. For this reason issues on editing, browsing and visualization of the ontology become subordinate and the problem of representation of ontology can be dealt with in a different perspective.

Our claim is that when the ontology is no longer the primary base in focus, more restrictive language with less expressive power is more suited in the present context. The main argument for this is that we can do with an incremental volume of knowledge represented in the ontology. Even very small fragments from a domain, such as a few related concepts, makes sense as an ontology if only there are queries with answers that can be improved from this. There is no need at all to insist on completeness on the coverage of a domain or a subdomain.

We consider a generative framework where an ontology in combination with a concept language defines a set of well-formed concepts. Well-formed concepts is assumed to be the basis for an indexing of the information base in the sense that these concepts appear as descriptors attached to objects in the base. Concepts are thus applied to obtain a means for descriptions that generalizes simple word-based information base indexing. In effect query evaluation is generalized to be a matter of comparison at the level of concepts capturing fragments of meaning rather than words.

The goal is thus a semantic basis for querying in text retrieval environments. In this context, one of the major problems is to determine the similarity between the semantic elements. It is no longer only simple match of keywords in the text objects, but also the meaning of them, we have to take into consideration when we calculate the similarity between queries and objects in the base.

The foundation of this paper is our previous work [Bulskov et al., 2002] and our affiliation to the interdisciplinary research project ONTOQUERY(Ontology-based Querying) [Andreasen et al., 2000; 2002b; OntoQuery, 2002].

An Environment for Ontology-based Querying
As introduced in the following section we consider a generative ontology that defines a set of well-formed concepts from a basis ontology. This basis ontology defines a vocabulary of concepts and situates these in a concept inclusion lattice (a taxonomy). We assume an environment where queries as well as objects from the base are attached descriptions formed from descriptors which basically are well-formed concepts. Query evaluation is then a matter of comparison of descriptions.

The environment for this type of querying may be a system that automatically can produce conceptual descriptions (conceptual indexing) of text objects and support textual/word list queries by initial transformation into descriptions.
2 A generative ontology

The purpose of the ontology is to define and relate concepts that can be used in descriptions. The ontology framework is generative in the following sense. A basis ontology defines a set of atomic concepts and situates these in a concept inclusion lattice, which basically is a taxonomy over single or multi-word concepts that are treated as atomic in the modelling of the domain. In combination with a given basis ontology, a concept language (description language) defines a set of well-formed concepts.

The concept language in focus here, ONTOLOG[Nilsson, 2001], defines a set of semantic relations which can be used for “attribution” (feature-attachment) to form compound concepts. The suitable number of available relations may vary with different domains, but among the more important relations that probably will be present in most domain models are WRT (With-respect-to), CHR (Characterized-by), CBY (Caused-by), TMP (Temporal), LOC (Location).

Expressions in ONTOLOG are descriptions of concepts situated in an ontology formed by an algebraic lattice with concept inclusion (ISA) as the ordering relation.

Attribution of concepts – combining atomic concepts into compound concepts by attaching attributes – can be written as a feature structures. Simple attribution of a concept $c_1$ with relation $r$ and a concept $c_2$ is denoted $c_1[r : c_2]$.

We assume a set of atomic concepts $A$ and a set of semantic relations $R$, as indicated with $R=\{\text{WRT, CHR, CBY, TMP, LOC, \ldots}\}$. Then the set of well-formed terms $L$ of the ONTOLOG language is recursively defined as follows.

- if $x \in A$ then $x \in L$
- if $x \in L$, $r_i \in R$ and $y_i \in L$, $i = 1, \ldots, n$, then $x[r_1 : y_1, \ldots, r_n : y_n] \in L$

It appears that compound terms can be built from nesting, for instance $c_1[r_1 : c_2[r_2 : c_3]]$ and from multiple attribution as in $c_1[r_1 : c_2, r_2 : c_3]$. The attributes of a multiple attributed term $T = x[r_1 : y_1, \ldots, r_n : y_n]$ is considered as a set, thus we can rewrite $T$ with any permutation of $r_1 : y_1, \ldots, r_n : y_n$.

The basis for the ontology is a simple taxonomic concept inclusion relation ISA$_{KB}$, which is atomic in the sense that it defines a relation over the atomic concepts $A$. It is considered as domain or world knowledge and may for instance express the view of a domain expert. We distinguish this (knowledge base) relation ISA$_{KB}$ because concepts are assumed to be related by specific knowledge over the domain. For this reason we cannot expect the relation to be transitively closed. The relation ISA is the transitive closure of ISA$_{KB}$, while the relation ISA$_{REDUC}$ is the transitive reduction of ISA$_{KB}$.

Based on ISA, the transitive closure of ISA$_{KB}$, we can generalize into a relation over all well-formed terms of the language $L$ by the following.

- if $x$ ISA $y$ then $x \leq y$
- if $x[\ldots] \leq y[\ldots]$ then also $x[\ldots, r : z] \leq y[\ldots]$, and
  $x[\ldots, r : z] \leq y[\ldots, r : z]$,
- if $x \leq y$ then also $z[\ldots, r : x] \leq z[\ldots, r : y]$

where repeated $\ldots$ in each inequality denotes identical lists of zero or more attributes of the form $r_i : w_i$. The purpose of the language introduced above is to describe fragments of meaning in text at a more thoroughly way than what can be obtained from simple keywords, while still refraining from full meaning representations which is obviously not realistic in general search applications (with a huge database).

Take as an example the sentence: “the black dog is making noise” which can be translated into this semantic expression noise[CBY: dog[CHR: black]].

Descriptions of text expressed in this language goes beyond simple keyword descriptions partly due to formation of compound terms and to the reference to the ontology. A key question in the framework of querying is of course the definitions of similarity or nearness of terms, now that we no longer can rely on simple matching of keywords.

3 From Ontology to Similarity

In building a query evaluation principle that draws on an ontology, a key issue is of course how the ontology influence the matching of values, that is, how the different relations of the ontology may contribute to similarity.

We have to decide for each relation to what extent related values are similar and we must build similarity functions, mapping values into similarities, that reflect these decisions.

We discuss below how to introduce similarity upon an ontology. We introduce firstly a shortest-path approach to similarity based on the key ordering relation in the ontology, ISA. Based on a definition for atomic concepts of the basis ontology we discuss how to extend the notion of similarity to cover general compound concepts as expressions in the language ONTOLOG. Secondly we introduce an alternative approach for devising a similarity measure based on the notion of shared nodes corresponding to lattice join in the lattice of the arguments of the similarity function. This approach can be considered as taking into account not only the shortest path but in principle all possible paths connecting two concepts.

3.1 Shortest-path similarity on atomic concepts

The concept inclusion relation plays a central role as the ordering relation that bind the ontology in a lattice. Concept inclusion intuitively imply strong similarity in the opposite direction of the inclusion (specialization), but also the direction of the inclusion (generalization) must contribute with some degree of similarity. Take as an example the small fraction of an ontology in figure 1. With reference to this ontology the atomic concept dog can be directly expanded to cover also poodle and alsatian.

This expansion respects the ontology in the sense that every instance of the extension of the expanded concept dog (that is, every element in the union of the extensions of dog, poodle and alsatian) by definition bear the relation ISA to dog. The intuition is that to a query on dog an answer including instances poodle is satisfactory (a specific answer to a general query). Since the hyponymy relation obviously is transitive we can by the same argument expand to further specializations e.g. to include poodle in the extension of animal.
However similarity exploiting the lattice should also reflect 'distance' in the relation. Intuitively greater distance (longer path in the relation graph) corresponds to smaller similarity.

Further also generalization should contribute to similarity. Of course it is not strictly correct in an ontological sense to expand the extension of dog with instances of animal, but because all dogs are animals, animals are to some degree similar to dogs. This substantiates that also a property of generalization similarity should be exploited and, for similar reasons as in the case of specializations, that also transitive generalizations should contribute with decreasing degree of similarity.

A concept inclusion relation can be mapped into a similarity function in accordance with the described intuition as follows.

Assume an ontology given as a domain knowledge relation $\text{ISAKB}$. Figure 1 shows an example. The corresponding transitive closure relation $\text{ISA}$ includes for instance also poodle $\text{ISA}$ animal. To make 'distance' influence similarity we need to consider the transitively reduced relation $\text{ISA\ REDUC}$. Similarity reflecting distance can then be measured from path-length in the graph corresponding to the $\text{ISA\ REDUC}$ relation. A similarity function $\text{sim}$ based on distance in $\text{ISA\ REDUC}$ $\text{dist}(X, Y)$ should have the properties:

1. $\text{sim}: U \times U \rightarrow [0, 1]$, where $U$ is the universe of concepts
2. $\text{sim}(x, y) = 1$ only if $x = y$
3. $\text{sim}(x, y) < \text{sim}(x, z)$ if $\text{dist}(x, y) > \text{dist}(x, z)$

By parameterizing with two factors $\delta$ and $\gamma$ expressing similarity of immediate specialization and generalization respectively, we can define a simple similarity function: If there is a path from nodes (concepts) $x$ and $y$ in the hyponymy relation then it has the form

$$ P = (P_1, \cdots, P_n) $$

where

$$ P_i$\text{ISA\ REDUC}P_{i+1}$ or $P_{i+1}$\text{ISA\ REDUC}P_i$

for each $i$ with $X = P_1$ and $Y = P_n$.

Given a path $P = (P_1, \cdots, P_n)$, set $s(P)$ and $g(P)$ to the numbers of specializations and generalizations respectively along the path $P$ thus:

$$ s(P) = |\{i | P_i$\text{ISA\ REDUC}P_{i+1}\}|$ 

$$ g(P) = |\{i | P_{i+1}$\text{ISA\ REDUC}P_i\}|$

and

$$ g(P) = |\{i | P_{i+1}$\text{ISA\ REDUC}P_i\}|$

If $P^1, \cdots, P^m$ are all paths connecting $X$ and $Y$ then the degree to which $Y$ is similar to $X$ can be defined as

$$ \text{sim}(X, Y) = \max_{j=1,\ldots,m} \{\sigma^{(P^1)}\gamma^{(P^j)}\} \quad (1) $$

This similarity can be considered as derived from the ontology by transforming the ontology into a directional weighted graph, with $\sigma$ as downwards and $\gamma$ as upwards weights and with similarity derived as the product of the weights on the paths. Figure 2 shows the graph corresponding to the ontology in figure 1. An atomic concept $T$ can then be expanded to a fuzzy set, including $T$ and similar values $T_1, T_2, \ldots, T_n$ as in:

$$ T^+ = 1/T + \text{sim}(T, T_1)/T_1 + \cdots + \text{sim}(T, T_n)/T_n \quad (2) $$

Thus for instance with $\sigma = 0.9$ and $\gamma = 0.4$ the expansion of the concepts dog, animal and poodle into sets of similar values would be:

$$ \text{dog}^+ = 1/\text{dog} + 0.9/\text{poodle} + 0.9/\text{alsatian} + 0.4/\text{animal} $$

$$ \text{poodle}^+ = 1/\text{poodle} + 0.4/\text{dog} + 0.36/\text{alsatian} + 0.16/\text{animal} + 0.144/\text{cat} $$

$$ \text{animal}^+ = 1/\text{animal} + 0.9/\text{cat} + 0.9/\text{dog} + 0.81/\text{poodle} + 0.81/\text{alsatian} $$

### 3.2 General shortest-path similarity

The semantic relations, used in forming concepts in the ontology, indirectly contribute to similarity through subsumption. For instance noise[$\text{CBY}$: dog [CHR: black]] is subsumed by - and thus extensionally included in - each of the more general concepts noise[$\text{CBY}$: dog] and noise. Thus with a definition of similarity covering atomic concepts, and in some sense of reflecting the ordering relation (concept inclusion), we can extend to similarity on compound concepts by a relaxation, which takes subsumed concepts into account when comparing descriptions.
The principle can be considered to be a matter of subsumption expansion. Any compound concept is expanded (or relaxed) into the set of subsuming concepts, thus

\[ \text{noise}[\text{CBY}: \text{dog}][\text{CHR}: \text{black}]] \]

is expanded to the set

\[
\{ \text{noise}, \text{noise}[\text{CBY}: \text{dog}], \text{noise}[\text{CBY}: \text{dog}][\text{CHR}: \text{black}] \}
\]

One approach to query-answering in this direction is to expand the description of the query along the ontology and the potential answer objects along subsumption.

For instance a query on dog could be expanded to a query on similar values like:

\[ \text{dog}^+ = 1/\text{dog} + \ldots + 0.4/\text{animal} + \ldots \]

and a potential answer object like

\[ \text{noise}[\text{CBY}: \text{dog}][\text{WRT: black}] \]

would then be expanded as exemplified above.

While not the key issue here, we should point out the importance of applying an appropriate averaging aggregation when comparing descriptions. It is essential that similarity based on subsumption expansion, exploits that for instance the degree to which \( c[r_1: c_1] \) is matching \( c[r_1: c_1][r_2: c_2] \) is higher than the degree for \( c \) with no attributes is matching \( c[r_1: c_1][r_2: c_2] \). Approaches to aggregation that can be tailored to obtain these properties, based on order weighted averaging (Yager, 1988) and capturing nested structuring (Yager, 2000), are described in (Andreasen, 2002a; 2002b).

An alternative to the above described subsumption expansion is to include edges corresponding to semantic relations in the computation of shortest path similarity as a generalization of the principle of aggregating weights by multiplying cost factors described in the previous subsection. While the similarity between \( c \) and \( c[r_1: c_1] \) can be claimed to be justified by the ontology formalism (subsumption) or simply by the fact that \( c[r_1: c_1] \text{ ISA } c \), it is not strictly correct in an ontological sense to claim similarity likewise between \( c_1 \) and \( c[r_1: c_1] \).

For instance \( \text{noise}[\text{CBY}: \text{dog}] \) is conceptually not some kind of a dog. On the other hand it would be correct to claim that \( \text{noise}[\text{CBY}: \text{dog}] \) in a broad sense has something to do with (and thus has similarities to) dog (simply supported by the fact that concept \( \text{noise}[\text{CBY}: \text{dog}] \) is present in the base). Most examples tend to reveal the same characteristics and this phenomenon is one good explanation for the comparative success of conventional word-based querying approaches. Basically the (incorrect) assumption of no correlation between words in NL phrases, which is underlying any strictly word-based approach, does not lead to serious failure because the correlation that appears is not dominating.

This could of course be an argument for not looking at compound concepts at all, but rather these considerations points in the direction of redrawing some of the importance of correlation in NL phrases when developing similarity measures.

Consider figure 3. The solid edges are ISA references and the broken are references by other semantic relations – in this example CBY and CHR are in use. Each compound concept has broken edges to its attribution concept. Strictly the spelling out of the compound concept expression as the label of a node is redundant since the concept expression can be derived from the connecting edges.

The principle of weighted path similarity can be generalized by introducing similarity factors for the semantic relations. The extensional arguments used to argue for differentiated weights depending on direction does not apply to semantic relations and seemingly there is no obvious way to differentiate based on direction at all. Thus one approach in the generalization is simply to introduce a single similarity factor and to transform to bidirectional edges.

Assume that we have \( k \) different semantic relations \( R^1, \ldots, R^k \) and let \( p_1, \ldots, p_k \) be the attached similarity factors. Given a path \( P = (P_1, \ldots, P_m) \), set \( r^j(P) \) to the number of \( R^j \) edges along the path \( P \) thus:

\[
r^j(P) = \left| \left\{ i \mid P_i = R^j, P_{i+1} \right\} \right|
\]

If \( P^1, \ldots, P^m \) are all paths connecting \( c_1 \) and \( c_2 \) then the degree to which \( Y \) is similar to \( X \) can be defined as:

\[
sim(X, Y) = \max_{j=1,\ldots,m} \left\{ \sigma(X, Y) \gamma(P^j) \rho_1 r^1(P^j) \cdots \rho_k r^k(P^j) \right\}
\]

The result of transforming the ontology in figure 3 is shown in 4. Here two semantic relations CHR and CBY are in use. The corresponding edge count functions are \( r^{\text{WRT}} \) and \( r^{\text{CBY}} \) and the attached similarity factors are denoted \( \rho_{\text{WRT}} \) and \( \rho_{\text{CBY}} \). The figure shows the graph with the attached similarity factors as weights. Again the degree to which a concept \( c_1 \) is similar to a concept \( c_2 \) is based on shortest path (and derived as the maximum of the products of edge weights over the set of paths connecting \( c_1 \) and \( c_2 \)).

For instance we can derive from figure 4 that \( \sim(\text{cat}, \text{dog}) = 0.9 \times 0.4 = 0.36 \) and \( \sim(\text{cat}[\text{CHR: black}], \text{color}) = 0.3 \times 0.4 = 0.12 \).

\[
\text{Figure 3: An ontology where attribution with semantic relations is shown as dotted edges.}
\]

The principle of weighted path similarity can be generalized by introducing similarity factors for the semantic relations. The extensional arguments used to argue for differentiated weights depending on direction does not apply to semantic relations and seemingly there is no obvious way to differentiate based on direction at all. Thus one approach in the generalization is simply to introduce a single similarity factor and to transform to bidirectional edges.

Assume that we have \( k \) different semantic relations \( R^1, \ldots, R^k \) and let \( p_1, \ldots, p_k \) be the attached similarity factors. Given a path \( P = (P_1, \ldots, P_m) \), set \( r^j(P) \) to the number of \( R^j \) edges along the path \( P \) thus:

\[
r^j(P) = \left| \left\{ i \mid P_i = R^j, P_{i+1} \right\} \right|
\]

If \( P^1, \ldots, P^m \) are all paths connecting \( c_1 \) and \( c_2 \) then the degree to which \( Y \) is similar to \( X \) can be defined as:

\[
sim(X, Y) = \max_{j=1,\ldots,m} \left\{ \sigma(X, Y) \gamma(P^j) \rho_1 r^1(P^j) \cdots \rho_k r^k(P^j) \right\}
\]

The result of transforming the ontology in figure 3 is shown in 4. Here two semantic relations CHR and CBY are in use. The corresponding edge count functions are \( r^{\text{WRT}} \) and \( r^{\text{CBY}} \) and the attached similarity factors are denoted \( \rho_{\text{WRT}} \) and \( \rho_{\text{CBY}} \). The figure shows the graph with the attached similarity factors as weights. Again the degree to which a concept \( c_1 \) is similar to a concept \( c_2 \) is based on shortest path (and derived as the maximum of the products of edge weights over the set of paths connecting \( c_1 \) and \( c_2 \)).

For instance we can derive from figure 4 that \( \sim(\text{cat}, \text{dog}) = 0.9 \times 0.4 = 0.36 \) and \( \sim(\text{cat}[\text{CHR: black}], \text{color}) = 0.3 \times 0.4 = 0.12 \).
3.3 Shared Nodes Similarity

The shortest path approach described above is straightforward and does not entail computational problems. However one aspect that must be assumed to contribute to similarity is ignored. When two concepts are connected by multiple paths only the shortest contribute to similarity. Considering the ontology in figure 4 the similarities between \(\text{cat}[\text{CHR} : \text{black}], \text{dog}[\text{CHR} : \text{black}]\) and \(\text{dog}[\text{CHR} : \text{black}, \text{CHR} : \text{brown}]\) will not be greater than the similarity between \(\text{cat}\) and \(\text{dog}\). This example shows that other connections than the shortest path in some cases should contribute and it also indicates that similarity should be proportional to the number of possible paths connecting two concepts. Obviously a similarity measure that takes into account all possible paths will impose increased computational complexity and calls for considerations on possible optimization approaches.

In this direction we suggest a similarity measure that utilizes a well-defined subset of all possible paths. The goal then is to encircle a basis in the form of a subontology for measuring similarity.

In broad terms our simplified “all-possible-paths” approach is a “shared nodes” approach, where shared nodes between two concepts are nodes that are “upwards reachable” from both concepts and where the similarity is proportional to the number of shared nodes.

To this end we define first the term-decomposition \(\tau(c)\) and the upwards expansion \(\omega(c)\) of a concept term \(c\). The term-decomposition is defined as the set of all subterms of \(c\), which thus includes all concepts subsuming \(c\) and all attributes of subsuming concepts for \(c\). The term-decomposition is defined as follows:

\[
\tau(c) = \{x|c \leq x \leq y[r : x], x \in L, y \in L, r \in R\}
\]

As an example the term \(\text{noise}[\text{CBY} : \text{dog}[\text{CHR} : \text{black}]]\) decomposes to resulting in the set containing the following concepts:

\[
\tau(\text{noise}[\text{CBY} : \text{dog}[\text{CHR} : \text{black}]]) = \{\text{noise}[\text{CBY} : \text{dog}[\text{CHR} : \text{black}]], \text{noise}[\text{CBY} : \text{dog}], \text{noise}, \text{dog}[\text{CHR} : \text{black}], \text{dog}, \text{black}\}
\]

The upwards expansion \(\omega(C)\) of a set of terms \(C\) is the transitive closure of \(C\) with respect to \(\text{ISA}_{\text{KB}}\).

\[
\omega(C) = \{x|x \in C \lor y \in C, y \text{ISA} x\}
\]

This expansion thus only adds atoms to \(C\).

We define further the upwards spanning subgraph (sub-ontology) \(\gamma(C)\) for a set of concepts \(C = \{c_1, \ldots, c_n\}\) as the graph that appears when decomposing \(C\) and connecting the resulting set of terms with edges corresponding to the \(\text{ISA}_{\text{KB}}\) relation and to the semantic relations used in attribution of elements in \(C\). We define the triple \((x, y, r)\) as the edge of type \(r\) from concept \(x\) to concept \(y\).

\[
\gamma(C) = \bigcup \{(x, y, \text{ISA})|x, y \in \omega(\tau(C)), x \text{ISA}_{\text{REDUC}} y\}
\]

\[
\{(x, y, r)|x, y \in \omega(\tau(C)), r \in R, x[r : y] \in \tau(C)\}
\]

Figure 5 shows an example of such an subontology spanned by the two terms.

Now a shared node between concepts \(c_1\) and \(c_2\) is a node that is reachable from both \(c_1\) and \(c_2\). With the example in figure 5 both \text{Animal} and \text{Black} are shared nodes for \text{Cat}[\text{CHR} : \text{Black}] and \text{Dog}[\text{CHR} : \text{Black}].

![Figure 4: The ontology of figure 3 transformed into a directional weighted graph with the similarity factors for specialization: \(\sigma = 0.9\) for generalization: \(\gamma = 0.4\) for \text{CBY}: \(\rho_{\text{CBY}} = 0.3\) and for \text{CHR}: \(\rho_{\text{WRT}} = 0.2\).](image)

![Figure 5: An example of an upwards spanning subgraph for the concepts \text{Cat}[\text{CHR} : \text{Black}] and \text{Dog}[\text{CHR} : \text{Black}] where \text{Animal} and \text{Black} (and \text{Anything} and \text{Color}) are shared nodes.](image)
It appears that this measure has good properties in sustaining to the intuition behind the ontology. First of all we can see that the similarity \( \text{sim}(\text{cat}[\text{CHR} : \text{black}, \text{CHR} : \text{brown}], \text{dog}[\text{CHR} : \text{black}, \text{CHR} : \text{brown}]) \) is increased as compared to the similarity \( \text{sim}(\text{cat}, \text{dog}) \). We have that when \( c_1 \leq c_2 \) then \( \text{sim}(c_1, c_2) \) is smaller than \( \text{sim}(c_2, c_1) \) (a general concept is not as good as replacement for a specific as vice-versa). Furthermore it follows that steps along edges become more expensive when the edges are closer to the top of the ontology.

However it appears that \( \text{sim}(x, y) \) is independent of nestings of \( x \). For instance consider the following example where the similarity between \( \text{dog}[\text{CHR} : \text{white}, \text{LOC} : \text{tarmac}[\text{CHR} : \text{black}]] \) and \( \text{cat}[\text{CHR} : \text{black}] \) is equal to the similarity between \( \text{dog}[\text{CHR} : \text{white}, \text{LOC} : \text{tarmac}[\text{CHR} : \text{black}]] \). Obviously there is a need for refinement of the similarity measure, that takes into consideration the nesting of \( x \) in \( \text{sim}(x, y) \). This is subject for further investigation.

The inclusion of non-minimal paths in the computation of the similarity measure has as a consequence, that the locations of the concepts in the ontology, influences the measure. This is due to the fact that concepts in the upper part of the ontology have fewer potential paths than concepts in the lower part. One could argue, from a pragmatic point of view, that concepts with longer common paths to the top of the ontology are stronger connected, which substantiate the intuition of increased similarity.

4 Conclusion

We have described different principles for measuring similarity between both atomic and compound concepts, all of which incorporate meta knowledge.

- Similarity between atomic concepts based on distance in the ordering relation of the ontology, concept inclusion (ISA\_REDUC).
- Similarity between general compound concepts based on subsumption expansion.
- Similarity between both atomic and general compound concepts based on shared nodes.

The notion of measuring similarity as distance, either in the ordering relation or in combination with the semantic relations, seems to indicate a usable theoretical foundation for design of similarity measures.

The purpose of similarity measures in connection with querying is of course to look for similar rather than for exactly matching values, that is, to introduce soft rather than crisp evaluation. As indicated through examples above one approach to introduce similar values is to expand crisp values into fuzzy sets including also similar values. Expansion of this kind, applying similarity based on knowledge in the knowledge base, is a simplification replacing direct reasoning over the knowledge base during query evaluation. The graded similarity is the obvious means to make expansion a useful - by using simple threshold values for similarity the size of the answer can be fully controlled.

Acknowledgments

The work described in this paper is part of the OntoQuery project supported by the Danish Technical Research Council and the Danish IT University.

References


\[1\] The project has the following participating institutions: Centre for Language Technology, The Technical University of Denmark, Copenhagen Business School, Roskilde University, and the University of Southern Denmark.
Research Presentation from the OntoQuery Project, in [Jensen et al., 2001].


[OntoQuery, 2002] ONTOQUERY project net site: www.ontoquery.dk


Statistics Gathering for Learning from Distributed, Heterogeneous and Autonomous Data Sources

Doina Caragea, Jaime Reinoso, Adrian Silvescu, and Vasant Honavar
Artificial Intelligence Research Laboratory,
Computer Science Department, Iowa State University,
226 Atanasoff Hall, Ames, IA 50011-1040, USA,
{dcaragea, jreinoso, silvescu, and honavar }@cs.iastate.edu

Abstract

With the growing use of distributed information networks, there is an increasing need for algorithmic and system solutions for data-driven knowledge acquisition using distributed, heterogeneous and autonomous data repositories. In many applications, practical constraints require such systems to provide support for data analysis where the data and the computational resources are available. This presents us with distributed learning problems. We precisely formulate a class of distributed learning problems; present a general strategy for transforming traditional machine learning algorithms into distributed learning algorithms based on the decomposition of the learning task into hypothesis generation and information extraction components; formally defined the information required for generating the hypothesis (sufficient statistics); and show how to gather the sufficient statistics from distributed, heterogeneous, autonomous data sources, using a query decomposition (planning) approach. The resulting algorithms are provably exact in that the hypothesis constructed from distributed data is identical to that obtained by the corresponding algorithm when in the batch setting.

1 Introduction

Development of high throughput data acquisition technologies in a number of domains (e.g., biological sciences, environmental sciences, space sciences etc.) together with advances in digital storage, computing, and communications technologies have resulted in unprecedented opportunities for scientists, to utilize, at least in principle, the wealth of information available on the Internet in learning, scientific discovery, and decision making. In practice, effective use of the growing body of data, information, and knowledge to achieve fundamental advances in scientific understanding and decision making presents several challenges [Honavar et al., 1998; 2001]:

- In such domains, data repositories are large in size, dynamic and physically distributed. Consequently, it is neither desirable nor feasible to gather all the data in a centralized location for analysis. Hence, efficient distributed learning algorithms that can operate across multiple autonomous data sources without the need to transmit large amounts of data are needed [Caragea et al., 2001b; Davies and Edwards, 1999; Kargupta et al., 1999; Prodromidis et al., 2000; Provost and Kolluri, 1999].
- Data sources of interest are autonomously owned and operated. Consequently, the range of operations that can be performed on the data source (e.g., the types of queries allowed), and the precise mode of allowed interactions can be quite diverse (e.g., PROSITE repository of protein data limits queries to those that can be entered using the forms provided on the web). Hence, strategies for obtaining the required information within the operational constraints imposed by the data source are needed [Levy, 2000].
- Data sources are heterogeneous in structure (e.g., relational databases, flat files) and content (names and types of attributes and relations among attributes used to represent the data). For example, data about proteins include the amino acid sequences of proteins, multiple sources of 3-dimensional structures of proteins, multiple sources of structural features of proteins, multiple sources of protein-protein interaction data, multiple sources of functional annotations for proteins (according to different notions of protein function), among others.
- The ontologies implicit in the design of autonomous data sources (i.e., assumptions concerning objects that exist in the world, which determine the choice of terms and relationships among terms) often do not match the ontologies of the users of those data sources. In scientific discovery applications, because users often need to examine data in different contexts from different perspectives, methods for context-dependent dynamic information extraction from distributed data based on user-specified ontologies are needed to support information extraction and knowledge acquisition from heterogeneous distributed data ([Honavar et al., 2001; Levy, 2000]).

Against this background, our main goal is to develop efficient strategies for extracting the information needed for learning (e.g., sufficient statistics) from heterogeneous, au-
tonomous, and distributed data sources, under a given set of ontological commitments in a given context.

The rest of the paper is organized as follows: In Section 2, we precisely formulate a class of distributed learning problems and present a general strategy for transforming traditional machine learning algorithms into distributed learning algorithms based on the decomposition of the learning task into hypothesis generation and information extraction components. The resulting algorithms are provably exact in that the hypothesis constructed from distributed data is identical to that obtained by the corresponding algorithm when in the batch setting. In Section 3, we formally define the sufficient statistics of a data set D with respect to a learning algorithm L and show how we can obtain these statistics from distributed data sets using a query decomposition (query planning) approach, assuming that data is presented to the algorithm as a table whose rows correspond to instances and whose columns correspond to attributes. Section 4 shows how heterogeneous data sources can be integrated and made to look as tables. Section 5 concludes with a summary and a brief outline of future research directions.

2 Distributed Learning

The problem of learning from distributed data sets can be summarized as follows: data is distributed across multiple sites and the learner’s task is to discover useful knowledge from all the available data. For example, such knowledge might be expressed in the form of a decision tree or a set of rules for pattern classification. We assume that it is not feasible to transmit raw data between sites. Consequently, the learner has to rely on information (e.g., statistical summaries such as counts of data tuples that satisfy particular criteria) extracted from the sites.

Definition: A distributed learning algorithm LD is said to be exact with respect to the hypothesis inferred by a learning algorithm L, if the hypothesis produced by LD, using distributed data sets D1 through Dn is the same (in terms of error) as that obtained by L when it is given access to the complete data set D, which can be constructed (in principle) by combining the individual data sets D1 through Dn (i.e., |error(LD(1, Dn)) − error(L(D1 ∪ · · · ∪ Dn))| = 0).

Definition: A distributed learning algorithm LD is said to be approximate with respect to the hypothesis inferred by a learning algorithm L, if the hypothesis produced by LD, using distributed data sets D1 through Dn is a good approximation (in terms of error) of that obtained by L when it is given access to the complete data set D (i.e., |error(LD(1, Dn)) − error(L(D1 ∪ · · · ∪ Dn))| < ε, ∀ε > 0).

Our approach to the exact/approximate distributed learning is based on a decomposition of the learning task into a control part which drives the execution of the algorithm toward the generation of a hypothesis and an information extraction part which is triggered by the control part whenever the algorithm requires statistics about the available data in order to generate the hypothesis (Figure 1).

In this approach to distributed learning, only the information extraction component has to effectively cope with the distributed and heterogeneous nature of the data in order to guarantee provably exact or approximate learning algorithms. The control component doesn’t access data directly, but just the statistics extracted from data. These statistics are obtained from queries that the control part asks in the process of the hypothesis generation.

A query answering engine, which acts like a planner, decomposes the queries in terms of operators available to the data sources and returns the answers to the control part. The results of the queries can be seen as a statistics oracle which responds according to the needs of the control part (i.e., according to the requirements of a particular algorithm at each step). The classical example oracle, which here is distributed into a network, is used to provide sufficient statistics (e.g., counts for decision tree) to the centralized statistics oracle. The statistics oracle acts as a buffer between the data and the hypothesis corresponding to that data (Figure 2).

Our strategy for the distributed learning can be used to transform any batch learning algorithm into an efficient distributed learning algorithm, once the sufficient statistics with respect to that particular learning algorithm have been identified and a plan for gathering them has been found.

3 Sufficient Statistics

In order to be able to define sufficient statistics [Casella and Berger, 1990] in the context of learning, we look at a learning algorithm as search into a space of possible hypotheses. The hypotheses in the search space can be thought as defining a parametric function. A particular choice for the set of parameters will give us a particular hypothesis. In particular, those parameters can be estimated through learning based on the given data.

Definition: Let F be a class of functions that a learning algorithm is called upon to learn. A statistic σ(D) is called a sufficient statistic for learning a function f ∈ F given a data set D = {(x1, y1), . . . , (xm, ym)}, if there exists a function g such that g(σ(D)) = f(D).

The particular learning algorithm considered determines a particular class of functions F (e.g., decision trees in the case of the decision tree algorithm, hyperplanes in the case of the SVM algorithm), which, in turn, determines a particular class of statistics (e.g., counts satisfying some criteria in the case of the decision trees). However, a sufficient statistic is defined with respect to a learning algorithm and a training data.
set. Note that a data set \( D \) is a trivial sufficient statistic for \( D \). However, whenever possible, we are interested in finding minimal sufficient statistics.

**Definition:** The sufficient statistic \( s^*(D) \) is called minimal sufficient statistic if for any other sufficient statistic \( s(D) \), \( s^*(D) \) is a function of \( s(D) \), i.e., there exists \( h \) such that \( s^*(D) = h(s(D)) \) for any data set \( D \).

For some learning algorithms (e.g., Naive Bayes), the sufficient statistics have to be computed once in order for the learning algorithm to generate the hypothesis. However, for other algorithms (e.g., Decision Trees) the statistics gathering process and the hypothesis generation are interleaved. In this case, the target function cannot be computed in one step; the learner has to go back and forth between the data and the partially learned hypotheses several times. Instead of dealing with sufficient statistics, we use partial sufficient statistics here.

Examples of sufficient statistics for several classes of learning algorithms are shown below:

- Naive Bayes algorithm (NB) - counts of the instances that match certain criteria (e.g., attribute-value-class counts) represent minimal sufficient statistics.
- Decision Tree algorithm (DT) - counts of the instances that match certain criteria (e.g., attribute-value-class) are minimal partial sufficient statistics for computing one level of the tree. Subsequent counts that depend on the current built hypothesis (tree) are needed in order to find the final decision tree [Bhatnagar and Srinivasan, 1997].
- Support Vector Machines algorithm (SVM) - the weight vector which determines the separating hyperplane, can be considered sufficient statistics for the separating hyperplane.

### 3.1 Gathering Sufficient Statistics from Homogenous Data

To keep things simple, in what follows, we assume that regardless of the structure of the individual data repositories (relational databases, flat files, etc.) the effective data set for learning algorithm can be thought of as a table whose rows correspond to instances and whose columns correspond to attributes. We will show in Section 4 how heterogeneous data can be integrated and put into this form.

In designing distributed learning algorithms using our decomposition strategy, we assume that the identification of the sufficient statistics is done by a human expert, who also designs the control part of the algorithm (function \( g \) in the definition of the sufficient statistics). However, the gathering of the sufficient statistics necessary for learning (construction of the statistics oracle) is done automatically every time when the control part needs some statistics about data, by the query answering engine. The query answering engine receives as input a query, and builds a plan for this query according to the resources and the operators available to each data set (Figure 2).

The operators associated with a data source can be primitive operators (such as selection, projection, union, addition etc.), or they can be aggregate operators (e.g., counts or other database built-in functions). If the data source allows, the user can define specific operators (functions of the primitive operators). The set of primitive operators should be complete with respect to the set of learning tasks that needs to be executed (i.e., the application of these operators or functions of these operators is enough for gathering the information necessary for a particular learning task considered).

**Definition:** We call learning plan for computing the statistics \( s \) required by a learning algorithm \( L \), from the distributed data sets \( D_1, \ldots, D_n \), a procedure \( P \), which transforms a given query into an execution plan. An execution plan can be seen as an expression tree, where each node corresponds to an operator and each leaf corresponds to basic statistics that can be extracted directly from the data sources. All the statistics that cannot be extracted directly from the data sources, should be replaced by their definitions recursively, until we obtain a plan in which only basic statistics appear as leaves.

Each of the operators available at the distributed data sources has a cost associated with it. Based on these operators and their costs, the query answering engine, which plays...
the role of the planner, finds the best execution plan for the current query and sends it to the distributed data sources for execution. Each data source returns the statistics (answers to queries) extracted from its data to the query answering engine, which sends the final result to the learning agent. If the algorithm needs more information about data in order to finish its job, a new query is sent to the query answering engine, and the process repeats.

**Definition:** We say that two learning plans $P_1$ and $P_2$ are equivalent if they compute the same set of statistics. If we consider the costs associated with the operators included in a plan, we say that a learning plan $P_1$ is more efficient than an equivalent learning plan $P_2$ if the cost of the first plan is smaller than the cost of the second plan.

The job of the query answering engine is to find the best learning plan for a query, given a set of primitive operators, aggregate operators and user defined functions that can be applied to these operators, and their associated costs.

### 4 Gathering Sufficient Statistics from Heterogeneous Data Sources

In the previous section, we assumed that data is presented to the distributed algorithms as tables whose rows correspond to instances and whose columns correspond to attributes. However, in a heterogeneous environment, it is not trivial to get the data into this format. The differences in ontological commitments assumed by autonomous data sources present a significant hurdle in the flexible use of data from multiple sources and from different perspectives in scientific discovery.

To address this problem, we developed the INDUS software environment [Reinoso-Castillo, 2002] for rapid and flexible assembly of data sets derived from multiple data sources. INDUS is designed to provide a unified query interface over a set of distributed data sources which enables us to view each data source as if it were a table. Thus, a scientist can integrate data from different sources from his or her own perspective using INDUS. INDUS builds on a large body of work on information integration, including in particular, approaches to querying heterogeneous information sources using source descriptions [Levy et al., 1996; Levy, 2000; Ullman, 1997], as well as distributed computing [Honavar et al., 1998; Wong et al.,].

The input from a typical user (scientist) includes: an ontology that links the various data sources from the users point of view, executable code that performs specific computations, needed if they are not directly supported by the data sources, and a query expressed in terms of the user-specified ontology. In this case, the query answering engine, receives this query as input, finds the best execution plan for it, translates the plan according to the ontologies specific to the distributed data sources, and sends it to the distributed data sources. Each data source returns answers to the queries it receives and the planner translates them back to the user (learning agent) ontology. Thus, the user can extract and combine data from multiple data sources and store the results in a relational database which is structured according to his or her own ontology. The results of the queries thus executed are stored in a relational database and can be manipulated using application programs or relational database (SQL) operations and used to derive other data sets (as those necessary for learning algorithms).

More precisely, INDUS integration system is based on a federated query centric database approach [Mena et al., 2000]. It consists of three principal layers which together provide a solution to the data integration problem in a scientific discovery environment (Figure 3):

- The physical layer allows the system to communicate with data sources. This layer is based on a federated database architecture (data is retrieved only in response to a query). It implements a set of instantiators which allow the interaction with each data source according to the constraints imposed by their autonomy and limited query capabilities. As a consequence, the central repository can view disparate data sources as if they were a set of tables (relations). New iterators may be added to the system when needed (e.g. a new kind of data source has become available, the functionality offered by a data source has changed).

- The ontological layer permits users to define one or more global (user/learning agent) ontologies and also to automatically transform queries into execution plans using a query-centric approach. More specifically, it consists of the following:
  - A meta-model, developed under a relational database, allowing users to define one or more global ontologies of any size. New statistics can be defined in terms of existing statistics using a set of compositional operators. The set of operators is extensible allowing users to add new operators as needed.
  - An interface for defining queries based on statistics in a global ontology.
  - An algorithm, based on a query-centric approach, for transforming those queries into an executable plan. The query-centric approach allows users to define each compound statistics in terms of basic statistics, (i.e. statistics whose instances are directly retrieved from data sources), using a predefined set of compositional operations. Therefore, the system has a description of how to obtain the set of instances of a global statistic based on extracting and processing instances from data sources. The plans describe what information to extract from each data source and how to combine the results.
  - An algorithm that executes a plan by invoking the appropriate set of instantiators and combining the results.
  - A repository for materialized (instantiated) plans which allow users to inspect them if necessary.

- Finally, the user interface layer enable users to interact with the system, define ontologies, post queries and receive answers. Also, the materialization of an executed plan can be inspected.

In conclusion, the most important features that INDUS offers in terms of data integration are summarized below:
From a user perspective, data accessible through INDUS can be represented as tables in which the rows correspond to instances and columns correspond to attributes, regardless of the structure of the underlying data sources.

The system can include several ontologies at any time. Individual users can introduce new ontologies as needed.

New data sources can be incorporated into INDUS by specifying the data source descriptions including the corresponding data-source specific ontology and the set of instantiators.

The main difference between INDUS and other integration systems [Garcia-Molina et al., 1997; Arens et al., 1993; Knoblock et al., 2001; Subrahmanian et al., June 2000; Draper et al., 2001; Paton et al., 1999] is that it can include several ontologies at any time. The users can introduce new ontologies or add new data sources and their associated ontologies, as needed.

INDUS has been successfully used by computational biologists (including graduate and undergraduate students with varying degrees of expertise in computing) in our lab for quickly extracting and assembling the necessary data sets from multiple data repositories for exploring and visualizing protein sequence-structure-function relationships [Wang et al., 2002; Andorf et al., 2002].

For the purpose of distributed learning, INDUS is used to execute queries whose results are tables containing data of interest (e.g., counts). Depending on the operations that are allowed at a particular data source, these tables may contain raw but integrated data (according to the global ontology that different data sources share) or counts or other statistics extracted from the data sources. Thus, if a particular data source can answer specific queries, but it does not allow the execution of any program or the storage of any data at that site, then the answer to the query will produce a table that needs to be stored locally, but closely to the original data source, in order to avoid the transfer of large amount of data. This table can be further used to obtain the statistics needed for the generation of the hypothesis. On the other hand, if the data sources support aggregate operations (e.g., those that provide statistics needed by the learning algorithm) or allow user-supplied programs to be executed at the data source, we can avoid shipping large amounts of data from distributed repositories.

5 Summary and Future Work

The approach to the distributed learning taken in this paper is a generalization of a federated query centric database approach [Mena et al., 2000]. In the case of distributed learning the set of operators is usually a superset of the operators used in classical databases. Besides, here the whole query answering process is just one step in the execution of the learning algorithm during which the statistics required by the algorithm are provided to the statistic oracle.

Assuming that the points of interaction between a learning algorithm and the available training data can be identified, the distributed learning strategy described here can be easily used to transform any batch learning algorithm into an exact (or at least approximate) distributed learning algorithm.

Future work is aimed at:

• Experiment with the decomposition strategy for various classes of learning algorithms and prove theoretical results with respect to the exact or the approximate quality of the distributed algorithms obtained.

• A big variety of data mining algorithms, such as decision trees, instance-based learners, Bayesian classifiers, Bayesian networks, multi-layer neural networks and support vector machines, among others, will be incorporated in our distributed learning system. Some of these algorithms can be easily decomposed into hypothesis generation and information extraction components according to our task decomposition strategy (e.g., Naive Bayes, decision trees), while others require substantial changes to the traditional learning algorithm (e.g., Support Vector Machines), resulting sometimes in new learning algorithms for distributed learning [Caragea et al., 2001a; 2000].

• Formally define the set of operators for the algorithms that will be included in our distributed learning system and prove its completeness with respect to these algorithms.

• Extend the formal definitions for plans, formulate properties of these plans and prove these properties under various assumptions made in a distributed heterogeneous environment.

Acknowledgments

This work has been supported in part by grants from the National Science Foundation (#9982341, #9972653, #0219699), Pioneer Hi-Bred, Inc., Iowa State University Graduate College, and IBM.

References


[Caragea et al., 2001b] D. Caragea, A. Silvescu, and V. Honavar. Invited chapter. toward a theoretical framework towards a theoretical frame-


Engaging Prolog with RDF

Borys Omelayenko
Department of Computer Science
Vrije Universiteit, De Boelelaan 1081, 1081hv,
Amsterdam, the Netherlands
borys@cs.vu.nl

Abstract
Prolog has been often used to represent the axioms and inference over RDF data models often by converting all the data to plain-text Prolog facts and programs. In this paper we present the PRODeF infrastructure for using Prolog for inferring over RDF data on the Web by representing Prolog programs in RDF, allowing them to be distributed over the Web and even incomplete, and represent reasoning results in a form suitable for further automatic processing.

1 Introduction
RDF and RDF Schema lack the means for representing axioms and rules, which are still necessary to build any kind of applications and different approaches originating from different motivations and requirements have been proposed. The same time Prolog [Bratko, 1990] has been extensively used to inference over data models represented in RDF, however, mostly RDF and Prolog have been connected in an ad-hoc manner, primarily by converting everything to plain-text Prolog facts and programs.

We intend to build the infrastructure depicted in Figure 1, where a special wrapper connects the Prolog engine to the Web. It parses a Prolog program represented in RDF and downloads the RDF data modules to be processed with the program. The program itself is distributed to several locations and the predicates used in one location may be defined in other places. The predicates, especially resource-critical or performing some specific function, may even be implemented in other languages and accessible as web services. Finally, the inference results are represented in RDF for further automatic processing.

We believe that such an infrastructure should possess the following basic properties:

1. The language should have clear semantics, sufficient and stable tool support, and existing expertise in terms of available literature, courses, and skills

2. The Prolog programs should be available on the Web in a distributed manner possibly decomposed into pieces and represented in RDF according to a certain RDF Schema

3. RDF data should be interpreted in Prolog

4. Program execution should assume that some parts of the program may require some time to download from different locations or being not available at the moment

5. A clear algorithm for converting plain-text Prolog programs to and from their RDF representation should be provided

6. The reasoning results should be represented in RDF and should allow updating the data

7. The rule language should allow representing the constraints at the RDF Schema level and smoothly link them to instance data.

We try to meet these requirements in the Prolog wrapper discussed in this Chapter.

1.1 Where are the Limits of Ontology Languages?
The ontology languages for the Semantic Web incorporate certain means for representing axioms that can be then used without any additional rule language.

RDF and RDF Schema contain the axioms needed to form the object-attribute language for representing the conceptual models: rdfs:subClassOf and rdfs:subPropertyOf are used to organize classes.

http://www.google.com/search?q=using+prolog+rdf
and properties into hierarchies, rdfs:domain and rdfs:range specify the attachment of properties to classes. This set of axioms is often difficult to use in practice, e.g. the conjunctive semantics of multiple occurrences of rdfs:domain or rdfs:range means that a property may be attached to an intersection of one or more classes, and not a union (disjunctive semantics). This poses some problems whenever a property has to be attached to several classes.

In OWL, RDF Schema is extended and several groups of axioms are introduced. These are:

- **Equality axioms** sameClassAs, samePropertyAs, sameIndividualAs and differentIndividualFrom to denote that two classes, properties or individual are equivalent;
- **Property characteristics** are introduced to define property characteristics inverseOf, TransitiveProperty, SymmetricProperty to define inverse, transitive and symmetric properties; allValuesFrom and someValuesFrom to define property range restrictions;
- **Cardinality** constraints of properties.

This set of axioms allows modelling numerous frequently needed constraints. For example the bossOf relation is often used to represent organizational structures. To model its transitivity in RDF Schema one needs to create and interpret a special rule, while it can be directly modelled in OWL with the TransitiveProperty property. However, in many organizations the set of bosses of an employee who can actually give him the orders is limited to two levels: the immediate boss and his/her immediate boss, and not a single step further. This axiom can not be modelled in OWL directly and a rule is needed to model this two-steps transitivity.

Another sort of examples include the axioms using value constraints, e.g. to classify some offers according to price where cheap offers would assume 0 EUR < price < 500 EUR.

### 1.2 Rule Languages for RDF

Many applications require rules and axioms that can not be directly represented in RDF Schema or OWL. However, RDF and RDF Schema do not possess any rule language, that is caused by numerous difficulties that are expected in standardization of such a language at present time. However, this need has been widely understood in the Semantic Web community and several approaches for such a rule language have been proposed.

Triple [Sintek and Decker, 2001] is proposed as an RDF query and inference language, providing full support for resources and their namespaces, models represented with sets of RDF triples, reification, RDF data transformation, and an expressive rule language for RDF. The language is intended to be used with a Horn-based inference engine.

The RuleML initiative aims at defining a shared rule markup language to specify forward (bottom-up) and backward (top-down) rules in XML. The language being developed within the initiative is essentially an XML serialization for the rules, and it specifies the rules in a generic form of a head and a body consisting of atomic predicates with parameters that can be also interpreted in Prolog. RuleML is probably the only rule language for RDF that defines an RDF syntax for the rules themselves.

However, there are several differences in the goals pursued in RuleML and PRODEF. These are:

- **RDF facts.** RuleML focuses at a universal representation of the rule on the (Semantic) Web and thus makes no assumptions about the structure and arity of the facts, while any inference engine dealing with RDF naturally deals with binary RDF facts only;
- **Implementation.** RuleML is not linked to a specific inference engine and thus needs to provide its own interpretation of the rules together with a linkage to inference engines. PRODEF follows the opposite approach tightly connecting the RDF serialization to standard Prolog semantics. From the engine implementation side, PRODEF relies on the decades-long experiences in making Prolog engines;
- **RDF Schema interpretation.** The RDF serialization used in RuleML makes no commitment to RDF Schema and uses rdf:Bag and rdf:Seq and rdf:List to represent the rules that are not representable in RDF Schema.

The RuleML initiative has been hosting a workshop on rule languages for the Semantic Web where a number of initiatives have been presented.

Squish also known as RDQL is somewhat similar to SQL but is further elaborated to query RDF triples. This similarity to SQL allows seamless integration with database back-ends. However, querying in Squish is bounded to plain RDF, without any support for RDF Schema or high-level languages.

The Sesame RDF querying engine [Broekstra et al., 2002] uses the RDF Query Language RQL. Similar to Squish, RQL statements contain the select-from-where construct, however, an RQL interpreter is supposed to understand RDF Schema axioms: transitivity of the subclass-of relation, its connection to rdf:type, etc.

A comparison of different RDF query languages is published on the W3C web site together with query samples and may serve as an interesting information source.

The Object Constraint Language OCL is the expression language for the Unified Modeling Language (UML) that allows specifying constraints about the objects, links, and property values of UML models. OCL is a pure expression language and any OCL expression is guaranteed not to change anything in the model. Whenever an OCL expression is evaluated, it simply delivers a value. OCL is a modelling lan-

---

3 [http://www.soi.city.ac.uk/~msch/conf/rueml/](http://www.soi.city.ac.uk/~msch/conf/rueml/)
4 [http://swordfish.rdfweb.org/rdfquery/](http://swordfish.rdfweb.org/rdfquery/)
7 [http://www.omg.org/docs/ad/97-08-08.pdf](http://www.omg.org/docs/ad/97-08-08.pdf)
guage rather than a programming language and it is not possible to write program logic or flow-control in OCL. As a side effect, not everything in OCL is promised to be directly executable.

The Protégé axiom language PAL \(^8\) is used together with the Protégé editor\(^9\) to specify knowledge base constraints. The syntax of PAL is a variant of the Knowledge Interchange Format (KIF) and it supports KIF connectives but not all of KIF predicates and statements. PAL is not really targeted towards RDF and RDF Schema.

The RDF parser for SWI Prolog is a very relevant and popular initiative on using Prolog with RDF. The parser is capable of converting RDF documents into Prolog facts and then utilize Prolog for reasoning, but it does not address RDF representation of Prolog programs themselves.

Table 1 represents a summary showing the features possessed by the languages in respect to the desired features listed in Section ??, As we can see none of the languages fulfills all of them with RQL being the closest.

Interesting to mention the estimate of the popularity of the rule and query languages. We queried the Web for relevant documents as presented in Table 2. The table illustrates that Prolog has been frequently used with RDF, however, with a relatively small amount of publications made on that. Obviously, these results are a subject of various distortions and they do not indicate more than they do. However, it is obvious that none of the newly proposed languages has the tool support able to compete with the decades-long experience in Prolog tool development.

2 The Usage Scenario

Consider the prototypical scenario shown in Figure 2 that depicts an RDF document that has a certain constraint a attached to it with the goal property. It refers to the definition of a made in another file as a ProDIEF program. To verify the document over the constraint a Prolog parser needs to access the definition of a, that is, in turn, defined over b and c, where c is again defined in another file. In this way a parser needs to go along the rdfs:isDefinedBy links attached to the predicates and extract their definitions from different locations on the Web. At certain moment all the predicates would be collected, defined in terms of l_triple and o_triple's, and the constraint can be verified.

3 The Ontology for ProDIEF

The ontology for ProDIEF represents the syntactic structure of Prolog programs and is depicted in Figure 3. In this ontology we do not try to represent the execution semantics of the programs, but treat program text as data and encode it as data, leaving its interpretation to a Prolog engine. The modules of Prolog code are modelled with the class PrologModule that contains module's logical name represented with the rdf:id attribute and physical location of the module encoded with the rdfs:isDefinedBy attribute.\(^10\) A module may export several predicates linked with the export property, call several directives, e.g., consult, as mentioned in the calls property, and contain rule definitions. PrologModules are instantiated with RDF files with program code located somewhere on the Web.

The class PredicateName represents a predicate name that requires certain numberOfParameters. The name tag is targeted at a human user while the rdf:id's of the PredicateName instances represent their identifiers used by the parser. The ontology includes several pre-defined instances of PredicateName reserved for o_triple and l_triple that correspond to the fact names reserved in ProDIEF to represent RDF data, and bagof, setof and forall that correspond to the special Prolog constructs. The PredicateName's represent the predicate names without any connection to their possible use with different parameters.

These are represented with the ClauseWithParameters class that connects a predicate name to a list of parameters. The parameters are organized as a list of instances of the Parameter class, where each instance corresponds to one parameter (a variable or a constant) and points to the next parameter in the list.

For example, an instance of PredicateName may look like the following:

```
<PredicateName rdf:id="MyPredicate" name="myPredicate" numberOfParameters="3"/>
```

and correspond to myPredicate/3.\(^11\) and an instance of ClauseWithParameters may look like this:

```
<ClauseWithParameters rdf:id="MyPredicateClause" name="myPredicateClause" parameters=""/>
```

\(^8\)http://protege.stanford.edu/plugins/paltabs/pal-documentation/
\(^9\)http://protege.stanford.edu/
\(^10\)rdfs:isDefinedBy belongs to RDF and RDF Schema and are not presented in the figure.
\(^11\)Some of the conventions on encoding predicate names in Prolog that are lifted in ProDIEF as described later.
A certain triple Product01, price, 55 EUR (e.g. triple #23) is modelled in RDF with an instance of rdf:Statement with the property rdf:Predicate pointing to the property name, and in PRODeF – with an instance of ClauseWithParameters with the property predicate. The rdf:Subject is modelled in PRODeF with the first Parameter linked to ClauseWithParameters with the property parameters. The rdf:Object is modelled with the second Parameter linked to the previous one with the nextParameter property.

However, the modelling and interpretation of RDF triples is primarily done with the supporting tools and not by a human user. Accordingly, the different ways of modelling the triples in RDF and PRODeF may not affect the utility of the ontology.

Figure 4 illustrates how a piece of a Prolog program may be encoded in PRODeF. The figure contains the sample code defining transitive subClassOf predicate and the tree illustrating its RDF representation in PRODeF. The tree contains two branches: RULE000 and RULE001 corresponding to the two (disjunctive) definitions of subClassOf and point to the PiR:predicate name subClassOf. RULE000 comes with the list PAR000 of PiR:parameters containing PiR:parameter X and PiR:nextParameter Y. The PiR:body of the rule consists of the clause CLS000 pointing to the name o_triple and its parameters X, Y, and constant ‘...#subClassOf’. In a similar way RULE001 has its PiR:body clause CLS001 with PiR:parameters X, Z, and constant ‘...#subClassOf’, and the PiR:nextClause CLS002 pointing to PiR:predicate subClassOf.

Figure 5: The way RDF statements are aligned to PRODeF clauses.
Figure 3: The ontology for representing Prolog programs in RDF
subClassOf(X,Y) :- o_triple(X,'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subClassOf',Y).
subClassOf(X,Y) :- o_triple(X,'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subClassOf',Z),subClassOf(Z,Y).
and parameters Z and Y.

3.1 RDF facts

The facts used by the rule language should not be abstract and disconnected but need to be grounded to RDF statements.

The facts in Prolog are represented with statements of an arbitrary arity indicating that one or more string-valued concepts are in a certain relation to each other.

For example, the statement price('Product','55 EUR') denotes that something called 'Product' is in relation 'price' to something called '55 EUR'.

In RDF the facts are represented with the RDF triples. Each triple of the form (object,property,value) denotes that two objects, object and value are in relation property to each other.

For example, the previous statement can be re-written as RDF triple ('Product',price,'55 EUR'), which may be encoded in RDF/XML as the following:

```xml
<rdf:Description rdf:about="Product" price="55 EUR"/>
```

In RDF only binary facts are allowed and thus only binary facts need to be represented to Prolog. We interpreted them in a uniform way: each RDF triple (object,property,value) where value takes rdfs:Literal strings is represented with Prolog fact l_triple(object,property,value), a triple with an rdf:Resource value is translated into Prolog fact o_triple(object,property,value). No other facts are allowed.

This interpretation is similar to the one used in SWI-Prolog where the result of importing an RDF file is a list of rdf(Subject, Predicate, Object) triples, where Subject is either a plain resource (an atom), or one of the terms each(URI) or prefix(URI) with the obvious meaning. Predicate is either a plain atom for explicitly non-qualified names or a term NameSpace:Name. If NameSpace is the defined RDF name space it is returned as the atom rdf. Finally, an Object is represented by its URI, a Predicate or a term has the format literal(Value) if they take literal values.

3.2 Names for Predicates and Variables

Historically, Prolog imposes certain constraints on the names for predicates and variables that originate from the plain-text encoding used in Prolog programs. In standard Prolog predicate names are represented with the identifiers starting with a small letter, and variable names start with a capital letter. This way of name encoding looks a bit archaic from the XML and RDF perspective.

We encode both predicates and variables as RDF objects whose rdf:ID's correspond to their identifiers (or names). These objects are easily distinguished because they are defined as the instances of a certain class, either PredicateName, Variable or a Constant. Accordingly, we lift the restriction on the case for the first letter, and allow the use of different namespaces as the qualifiers to distinguish different predicates with the same names on the Web.

Opposite to the predicates, the variables are used only locally within a single rule definition and may not be accessed from the outside.

Predicate names make some sense only if there is a link to the location where they are actually defined. We use the rdfs:isDefinedBy property of a resource to denote the file where the predicate is actually defined. [?] defines rdfs:isDefinedBy as 'an instance of rdf:Property that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described' and thus is perfectly suitable for this purpose.

3.3 Namespaces

In XML and RDF namespaces are used as qualifiers for the names allowing two equivalent names to be distinguished globally by having different qualifiers. We use the namespaces as parts of predicate and variable names. Essentially the namespaces for the variables that are used only within the predicate definitions are not that important as for the predicates that may be accessed globally.

4 The Execution of PRODeF Modules on the Web

The Prolog programs on the Web are executed in a different way than in classical Prolog systems and the inference results are produced for further automatic processing rather than direct human consumption.

4.1 Modules and Goals

Prolog programs are decomposed into modules that are imported by the engine by executing the directive consult. In the Web scenario the modules may well be distributed all over the Web and consulting a module would require prior downloading of the correspondent RDF file. Accordingly, instead of a local file name consult need to receive an URL of the module.

In PRODeF each RDF data module contains two parts: RDF data itself and a possible annotation of the rdf:RDF tag with the special p1r:goal property linking it to the goal description, a set of axioms that are applicable to the data module:

```xml
<rdf:RDF p1r:goal="http://...goal.rdf"> here goes RDF data </rdf:RDF>
```

The goal descriptions are special objects that define the axioms, the applicable data modules, and the interpretation of the axioms. It consists of (Figure 6):

- axioms pointing to the axiom with the name property and its definition with the rdf:isDefinedBy property. The axioms are subclassed into PositiveAxioms and NegativeAxioms to specify whether the predicate is a positive test those results represent correct data, or a negative test that results in the incorrect data.
datasets that are applicable to the axioms. Each dataset may consist of several locations pointed with their urls or a query made in a repository.

Quite often it happens that a certain location with a piece of a program is not accessible at the moment. What should if the definition of a certain predicate cannot be found? A possible solution path is to provide the Prolog engine with a parameter specifying server's behavior: to wait, to ignore the predicate, or to fail. This failure is then included in the reasoning results.

4.2 Reasoning Results
The Prolog engine and the wrapper return two types of information:

The list of failed locations that could not be accessed and the data or program modules could not be downloaded;

Prolog inference result: success, failure, yes, or no;

The list of solutions in form of tuples \((x_1, ..., x_n)\) that correspond to the goal predicate with arguments \((X_1, ..., X_n)\) returned in case of success.

We naturally represent the solutions as a bag of RDF objects, each of which contains \(n\) properties with the names \(X_1, ..., X_n\) and the values \(x_1, ..., x_n\).

If a goal is defined as a conjunction of several predicates \(\text{goal}(X_1, ..., X_n) : - P_1(X_1, ..., X_m), ..., P_k(X_1, ..., X_m)\) where each \(P_i\) receives some or all of the \(n\) arguments of \(\text{goal}\), then we may represent each resulting object as a set of objects \(P_1, ..., P_k\), each of which corresponds to one of the predicates defining the goal. It may then make sense to explicitly represent these predicates in the reasoning results and process them further separately.

Accordingly, the Prolog interpreter receives a parameter ‘detail level of the results’ \(1, 2, ..., \infty\) that specifies the number of objects representing each result, where \(\infty\) forces the results to be fully decomposed. However, further elaboration of this scheme is rather a subject of further research.

5 Summary
In the paper we propose a solution for the problem of representing Prolog programs on the (Semantic) Web and dealing with distributed data modules in RDF.

A number of questions remain open:

- How the language should be restricted (or better to say, which extensions to Prolog should be prohibited). Primarily this refers to the problems of batch execution of the programs that may not use any graphic user interface nor console output;
- The definition of an interface between \text{PRODEF} and the predicates implemented with the other languages and available as web services;
- A number of issues concerning distributed program execution remain open.

Extra information together with the ontologies, examples and occasional tool support is available at the \text{PRODEF} homepage."13

References


\textsuperscript{13}http://www.cs.vu.nl/~borys/PiR/
Ontologies for Learning Agents: Problems, Solutions and Directions

Bogdan Stanescu, Cristina Boicu, Gabriel Balan, Marcel Barbulescu, Mihai Boicu, Gheorghe Tecuci
4A5, Learning Agents Laboratory, George Mason University
4400 University Dr., Fairfax, VA-22030, US
{bstanesc, ccascava, gbalan, mbarbule, mboicu, tecuci}@gmu.edu

Abstract

We are developing a general end-to-end approach, called Disciple, for building and using personal problem solving and learning agents. This approach raises complex challenges related to ontology specification, import, elicitation, learning, and merging, that we have explored to various degrees, as we are developing successive versions of Disciple. This paper presents some of these challenges, our current solutions and the future directions, that are relevant for building agents in general.

1 Introduction

The long term objective of our research is to develop the science and technology that will allow typical computer users to train and use their personal intelligent assistants. Our approach to this problem is to develop a series of increasingly more capable agents from the Disciple family of learning agent shells [Tecuci, 1998; Tecuci et al., 2002]. A Disciple agent can be initially trained by a subject matter expert and a knowledge engineer, in a way that is similar to how an expert would teach an apprentice, through problem solving examples and explanations. Once trained to a significant level of competence, copies of the agent are handed over to typical computer users. These agents then assist their users through mixed-initiative reasoning, increasing their recall, speed and accuracy, without impeding their creativity and flexibility. In the same time, the assistants continue to learn from this joint problem solving experience, adapting to their users to become better collaborators that are aware of users’ preferences, biases and assumptions.

The process of building and using such problem solving and learning agents raises complex challenges related to ontology specification, import, elicitation, learning, and merging, that we have explored to various degrees, as we are developing successive versions of Disciple. The goal of this paper is to present some of these challenges, our current solutions and the future directions, that are relevant for building agents in general.

In the last three years, the development of the Disciple approach was driven by the attempt to find an automatic solution to the complex Center of Gravity (COG) analysis problem, in collaboration with the US Army War Col-

lege. The center of gravity of a force (state, alliance, coalition or group) represents the foundation of capability, power and movement, upon which everything depends [Clausewitz, 1976]. In any conflict, a force should concentrate its effort on its enemy’s center of gravity, while adequately protecting its own. As a consequence, the examples used in this paper will be from the COG domain, but they will not require an understanding of this domain.

The rest of this paper is organized as follows. The next section discusses the use of the ontology for representation, communication, problem solving, and learning, both in general, and in the context of the Disciple family. Section 3 gives an overview of the Disciple agent building methodology, stressing the ontology-related activities. Then sections 4 to 7 discuss in more details some of our main results on ontology specification, exception-based ontology learning, example-based ontology learning, and ontology import and merging. These sections will include experimental results and plans for future research.

2 Knowledge representation for problem solving and learning

A Disciple learning agent shell includes general problem solving and learning engines for building a knowledge base consisting of an object ontology that specifies the terms from a particular domain, and a set of problem solving rules expressed with these terms [Tecuci et al., 2002]. The problem-solving engine is based on the general task reduction paradigm. In this paradigm, a task to be performed is successively reduced to simpler tasks, by applying task reduction rules. Then the solutions of the simplest tasks are successively combined, by applying solution composition rules, until they produce the solution of the initial task.

The object ontology is a hierarchical representation of the objects and types of objects from the application domain. It represents the different kinds of objects, the properties of each object, and the relationships existing between objects. A fragment of the object ontology for the COG domain is shown in the bottom part of Figure 1.

The reduction rules are IF-THEN structures that express how and under what conditions a certain type of task may be reduced to simpler subtasks. The reduction
An example of a simple task reduction rule is shown in Figure 1. In this case the IF task is reduced to its solution.

The learning engines use several strategies to learn the rules and to refine the object ontology. At the basis of the learning methods are the notion of plausible version space [Tecuci, 1998; Boicu, 2002] and the use of the object ontology as an incomplete and partially incorrect generalization hierarchy for learning.

A plausible version space is an approximate representation for a partially learned concept, as illustrated in Figure 2. The partially learned concept is represented by a plausible upper bound concept which, as an approximation, is more general than the concept $E_h$ to be learned, and by a plausible lower bound concept which, again as an approximation, is less general than $E_h$. During learning, the two bounds (which are first order logical expressions) converge toward one another through successive generalizations and specializations, approximating $E_h$ better and better.

![Figure 1: Ontology based rule learning.](image1.png)

![Figure 2: A representation of a plausible version space](image2.png)
question. The answer to this question leads to the reduction of this task to a solution. As the expert types these expressions using natural language, the agent interacts with him/her to replace certain phrases with the ontology terms they designate (e.g. “will of the people of Caribbean State Union” or “strategic COG candidate”). The recognition of these terms facilitates the understanding of the expert’s phrases and the learning of a general rule from this specific example. The learned rule has an informal structure (shown in the top right part of Figure 1) and a formal structure (shown in the bottom right part of Figure 1). The informal structure preserves the natural language of the expert and is used in agent-user communication. The formal structure is used in the actual reasoning of the agent. Notice that the two plausible version space conditions from the formal structure are expressed with the terms from the object ontology. The formal tasks and their features are also part of the task ontology, and feature ontology, respectively.

As mentioned above, the object ontology has a fundamental role in learning, being used as a generalization hierarchy. Indeed, notice that the specific instances from the example (“will of the people of Caribbean State Union”, “OECS Coalition”, “people of Caribbean State Union”) are replaced in the learned rule with more general concepts from the object ontology (“will of agent”, “multi member force”, “people”), and their relationships.

While the corresponding learning algorithm is presented in [Boicu et al., 2000; Boicu 2002], it is important to stress here that the agent’s generalization hierarchy (the object ontology) is itself evolving during learning (as discussed in sections 4, 5, and 6). Therefore Disciple addresses the complex and more realistic problem of learning in the context of an evolving representation language. The next section gives an overview of the agent building methodology, stressing the ontology-related activities.

3 Agent building methodology

The Disciple learning agent shell could be used to rapidly develop a Disciple agent for a specific application domain, by following the steps from Figure 3. There are two main phases in this process: the development of an initial object ontology and the teaching of the agent. The first phase has to be performed jointly by a knowledge engineer and a subject matter expert. The second phase may be performed primarily by the subject matter expert, with limited assistance from a knowledge engineer.

During domain analysis and ontology specification, the knowledge engineer works with the subject matter expert to develop an initial model of how the expert solves problems, based on the task reduction paradigm. The model identifies also the object concepts that need to be represented in Disciple’s ontology so that it can perform this type of reasoning. These object concepts represent a specification of the ontology needed for reasoning.

During ontology import and development, this specification guides the process of importing ontological knowledge from existing knowledge repositories, such as CYC [Lenat, 1995], as discussed in section 7. However, not all the necessary terms will be found in external repositories and therefore the knowledge engineer and the subject matter expert will also have to extend the imported ontology using the ontology development tools of Disciple. For instance, Figure 4 shows the interfaces of three different ontology browsers of Disciple, the association browser (which displays and objects and its relationships with other objects), the tree browser (which displays the hierarchical relationships between the objects in a tree structure), and the graphical browser (which displays the hierarchical relationships between the objects in a graph structure).

Once the object ontology is developed, the knowledge engineer has to define elicitation scripts using the Script Editor of Disciple. The elicitation scripts will be executed by the Scenario Elicitation tool, guiding the user of Disciple to define a specific scenario or problem solving situation (e.g. the current war on terror, including the characteristics of the participating forces, such as US and Al Qaeda). This process will be described in more detail in section 4. The result of this initial KB development phase is an object ontology with instances characterizing a specific scenario.

In the next major phase, the subject matter expert will use the current scenario to teach Disciple how to solve problems (e.g. how to determine the centers of gravity of the opposing forces in the current war on terror).

First, the expert will interact with the Modeling advisor tool of Disciple. This tool will assist the expert to express his or her reasoning process in English, using the task reduction paradigm. The result of this process will be task reduction steps like the one from the upper left part of Figure 1. These steps may also include new terms that are not yet present in the object ontology of Disciple. Each such term is an example for learning a general con-
cept or a general feature using the Ontology learning method discussed in section 6. Also, each specific reasoning step formulated with the Modeling advisor is an example from which a general rule is learned using the Rule Learning tool. An example of such a rule is presented in the right hand side of Figure 1.

As Disciple learns more rules, the interaction with the subject matter experts evolves from a teacher-student type of interaction to an interaction where both collaborate in solving a problem. This interaction is governed by the mixed-initiative problem solving tool. In this case, Disciple uses the partially learned rules to propose solutions to the current problems, and the expert’s feedback will be used by the Rule Refinement tool and the Ontology Refinement tool to improve both the rules and the ontology elements involved in the rules’ applications.

There is no fixed sequence of tool invocations. Instead, they are used opportunistically, based on the current problem solving situation. For example, while the expert and Disciple are performing mixed-initiative problem solving, the expert may need to define a new reduction that requires modeling, rule learning and rule refinement.

Because the rule learning and refinement processes take place in the context of an incomplete and partially incorrect object ontology, some of the learned rules may accumulate exceptions. In such a case, the exception-based KB refinement tool may be invoked to extend or correct the object ontology and to correspondingly refine the rules. This process will be presented in section 5.

Because one of the goals of this research is the rapid development of knowledge bases, the Disciple shell also includes tools to merge the ontologies and the rules developed in parallel by the subject matter experts. Section 7 discusses this issue in more detail.

In the last three years we have performed extensive experiments with Disciple at the US Army War College, where it is used in two courses, Case Studies in Center of Gravity Analysis (the COG course), and Military Applications of Artificial Intelligence (the MAAI course). In the COG course, Disciple is used as an assistant that was trained by the instructor, helping the students to perform a COG analysis of a scenario and to generate an analysis report. Over 95% of the students from the 2002 Terms II and III sessions of this course agreed with the following statement: Disciple helped me to learn how to perform a strategic center of gravity analysis of a scenario. In the follow-on MAAI course, the students taught personal Disciple agents their own expertise in COG analysis. After the experiments conducted in Spring 2001 and Spring 2002, 19 of the 25 students agreed (and 6 were neutral) with the statement: I think that a subject matter expert can use Disciple to build an agent, with limited assistance from a knowledge engineer.

The following sections will provide more details on some of the most important ontology-related processes of the Disciple agent development methodology, as well as results from the above experiments.

4 Scenario specification

As part of the initial ontology development, the knowledge engineer uses the Script Editor to define elicitation scripts that specify how to elicit the description of a scenario from the user. These scripts are associated with the concepts and features from the ontology. Each script has a name, a list of arguments, and it specifies how to display the dialog with the user, the questions to ask the user, how to store the answers in the ontology, and what other scripts to call. Table 1 shows the script “elicit government type” associated with the concept “state government”.

The elicitation scripts are executed by the Scenario Elicitation tool. As illustrated in Figure 5, the left hand side of the Scenario Elicitation interface displays a table of contents. When the expert clicks on one of these titles, questions that elicit the corresponding description are displayed in the right hand side of the screen. The use of the elicitation scripts allows a knowledge engineer to rapidly build a customized interface for a Disciple agent, thus effectively transforming this software development task into a knowledge engineering one.

The Protégé system [Noy et al., 2000] has a similar capability of using elicitation scripts to acquire instances of concepts. However, Disciple extends Protégé in several directions. In Disciple the expert does not need to see or understand the object ontology in order to answer the questions and describe a scenario. Instead, the expert-agent interaction is directed by the execution of the scripts. Once the expert answers some questions or up-

<table>
<thead>
<tr>
<th>Table 1: Sample elicitation script.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Script:</strong> state_government.elicit government type</td>
</tr>
<tr>
<td><strong>Arguments:</strong> &lt;force-name&gt;, &lt;government-name&gt;</td>
</tr>
<tr>
<td><strong>Control:</strong> single-selection-list</td>
</tr>
<tr>
<td><strong>Question:</strong> What type of government does &lt;force-name&gt; have?</td>
</tr>
<tr>
<td><strong>Possible values:</strong> the elementary subconcepts of state_government</td>
</tr>
<tr>
<td><strong>Allow adding new subconcepts:</strong> Yes</td>
</tr>
<tr>
<td><strong>Ontology actions:</strong></td>
</tr>
<tr>
<td>&lt;government-name&gt; instance-of &lt;government-type&gt;</td>
</tr>
<tr>
<td><strong>Script call:</strong> &lt;government-type&gt;_elicits government properties</td>
</tr>
<tr>
<td><strong>Arguments:</strong> &lt;government-name&gt;</td>
</tr>
</tbody>
</table>
dates his answers, new titles may be inserted into the table of contents, as directed by the script calls. For instance, after the expert specifies the opposing forces in a scenario, their names appear as titles in the table of contents, together with the characteristics that need to be elicited for them. Experimental results show that the experts can easily use the Scenario Elicitation module [Tecuci et al., 2002]. In Protégé, each concept has exactly one script that specifies how to elicit the properties of its instances. In Disciple, a concept can have any number of scripts that can be used for any purpose. In particular, the knowledge engineer can define more scripts that specify how to elicit instances for the same concept. For instance, to elicit the military factors for a single-state force, different questions have to be asked if the force is part of an alliance, or is a standalone opposing force.

The most recent development of the Scenario Elicitation tool is to allow the user to extend the ontology with new concepts in a controlled manner. For instance when the script from Table 1 is executed, the user can specify a new type of state government (e.g., “feudal god-king government”), as illustrated in Figure 5. As a result a new concept is created under “state government”. As future developments, we plan to extend the capability of the Script Editor to facilitate the script definition task for the knowledge engineer, by taking into account the structure of the ontology and by using customization of generic scripts. We also plan to add natural language processing capabilities to the Scenario Elicitation module.

5 Exception-based ontology learning

As we have mentioned in section 2, the object ontology plays a crucial role in the learning process of the agent, as it is used as the generalization hierarchy for learning. However, this ontology is itself incomplete and partially incorrect and will have to be improved during the teaching of the agent. In this section we will briefly present an exception-based approach to ontology learning.

Because the ontology is incomplete, it may not contain the knowledge to distinguish between all the positive examples and the negative examples of a learned rule, such as the one presented in Figure 1. As a result, a rule may accumulate negative and positive exceptions.

A negative exception is a negative example that is covered by the rule because the current object ontology does not contain any knowledge that distinguishes the negative example from the positive examples of the rule [Tecuci, 1998; Boicu et al., 2003]. Therefore, the rule cannot be further specialized to uncover the negative example, while still covering all the positive examples of the rules. A positive exception is defined in a similar way.

A comparative analysis of the examples and the exceptions will facilitate identifying what distinguishes them and how the object ontology needs to be extended to incorporate the identified distinction. This is precisely the main idea behind our exception-based learning method, in which a subject matter expert collaborates closely with the agent to discover possible ontology extensions (such as new concepts, new features or new feature values) that will eliminate the exceptions.

The exception-based learning method consists of four main phases: 1) a candidate discovery phase in which the agent analyzes a rule, its examples and exceptions, and the ontology and finds the most plausible types of extensions of the ontology that may reduce or eliminate the rule’s exceptions; 2) a candidate selection phase in which the expert interacts with the agent to select one of the proposed candidates; 3) an ontology refinement phase in which the agent elicits the ontology extension knowledge from the expert and 4) a rule refinement phase in which the agent updates the rule and eliminates the rule’s exceptions based on the performed ontology extension.

As an illustration, consider the example and the corresponding partially learned rule from Figure 1. This rule is used in problem solving and generates the reasoning step from Figure 6, which is rejected by the expert because both the answer to the question and the resulting solution are wrong. However, there is no knowledge in the current ontology that can distinguish between the objects from the positive example in Figure 1 and the corresponding objects from the negative example in Figure 6. Therefore, the negative example from Figure 6 will be kept as a negative exception of the rule in Figure 1.

Figure 7 shows the interface of the exception-based learning tool in the ontology refinement phase. The upper left panel of this tool shows the negative exception which needs to be eliminated. Below are the objects that are currently differentiated: “Caribbean States Union” (from the positive example) and “USA” (from the negative exception). The right panel shows the elicitation dialog, in which the expert is guided by the agent to indicate the name and value of a new feature that expresses the difference between “Caribbean States Union” and “USA.” The expert defines the new feature “is minor member of” and specifies that “Caribbean States Union” is a minor member of “OECS Coalition,” while “USA” is not. Based on this elicitation, Disciple learns a general definition of the feature “is minor member of” and refines the ontology to incorporate this knowledge. A fragment of the refined
ontology is shown in the right part of Figure 7. Notice that both the domain and the range of the new feature are represented as plausible version spaces. The plausible upper bound domain of this feature is "single member force" and the plausible lower bound domain is "single state force."

The exception-based learning tool was evaluated during the Spring 2002 agent teaching experiment performed with Disciple at the US Army War College, as part of the "Military Applications of Artificial Intelligence" course. The tool was used by seven subject matter experts with the assistance of a knowledge engineer, to eliminate the negative exceptions of the rules. We did not expect a significant number of exceptions, because before the experiment we attempted to develop a complete ontology, which contained 191 concepts and 206 features. However, during the experiment, 8 of the learned problem solving rules have collected 11 negative exceptions, indicating that the ontology was not complete. In order to eliminate these exceptions, the experts extended the ontology with 4 new features and 6 new facts. Some of the newly created features eliminated the exceptions from several rules. As a result of these ontology extensions, the rules were correspondingly refined.

This experiment proved that the exception-based learning tool can be used to extend the object ontology with new elements that represent better the subtle distinctions that the experts make in their domains of expertise. This tool allows the elimination of the rules' exceptions and it improves the accuracy of the learned rules by refining their plausible version space conditions. It also enhances the agent's problem solving efficiency by eliminating the need to explicitly check the exceptions. We plan several extensions to the presented method: propose suggestions and help the user during the exception-based learning process; use analogical reasoning and hints from the user in the discovery of plausible ontology extensions; extend the method to discover new object concepts in order to eliminate the rules' exceptions; and extend the method to also remove the positive exceptions of the rules.

6 Example-based ontology learning

There are many situations during the agent teaching process where the subject matter expert has to specify a fact involving a new instance or a new feature. In such a case, the example-based ontology learning tool is invoked to learn a new concept or a new feature definition, from the provided fact. One such situation was encountered in the previous section where the expert indicated that "Caribbean States Union is minor member of OECS..."
Coalition.” From this specific fact Disciple attempts to learn a general definition of the feature “is minor member of.” The most important characteristics of the feature that need to be learned are its position in the feature hierarchy, its domain of applicability, and its range of possible values. First Disciple identifies the features that are most likely to be more general than “is minor member of.” This set initially includes all the features whose domain and range cover “Caribbean States Union” and “OECS Coalition,” respectively, as shown in Figure 8. This set is further pruned by applying various heuristics (for instance by eliminating the other features of “Caribbean States Union”) and by directly asking the expert:

Consider the statement “Caribbean States Union is minor member of OECS Coalition.” Is this a more specific way of saying: “Caribbean States Union is member of OECS Coalition”?

As a result of this process “is minor member of” is defined as a subfeature of “is member of.” The domain and the range of the “is member of” feature become the upper bounds of the domain and range of “is minor member of.” The corresponding lower bounds are the minimal generalizations of “Caribbean States Union” and “OECS Coalition,” respectively (see the bottom part of Figure 7).

The next step is to further refine the plausible version spaces of the domain and range. The lower bounds are generalized based on new positive examples of this feature, encountered during further teaching. However, the agent will not encounter negative examples. Therefore the specialization of the upper bounds is based on a dialog with the expert who will be asked to identify objects that cannot have this feature, or cannot be a value of this feature. There are other difficult problems related to learning and refining features: how to elicit its special characteristics (e.g. whether the feature is transitive or not), how to elicit its cardinality, or how to differentiate between required and optional features for an object.

7 Ontology import and merging

Figure 9 shows another view of the Disciple agent building methodology that emphasizes ontology reuse and parallel knowledge base development. The ontology specification that results from the domain analysis phase (see Figure 3) guides the process of importing ontological knowledge, currently from CYC [Lenat, 1995] and, in the future, also from other knowledge repositories.

Our import method consists of identifying key terms in the CYC KB that correspond to the terms from the ontology specification, extracting the knowledge related to those terms and importing it into the Disciple knowledge base. The extraction of knowledge is an automated process in which all the terms related to the start-up terms are elicited, then all the terms related to those terms, and so on until a transitive closure or a user-specified stopping criteria is met. This method extends the one of Chaudhri et al. [2000] by adding stopping criteria, by allowing taxonomy relations to be followed down the hierarchy, and by considering the feature hierarchy. The translation of the extracted knowledge into the Disciple formalism consists of a syntactic phase and a semantic one, being similar with the method used in OntoMorph [Chalupsky, 2000]. During the automatic transformation of extracted knowledge into Disciple’s knowledge representation, the system records logs with a number of decisions that require the user’s approval or refinement.

The imported ontology is further extended using the ontology development tools of Disciple, as discussed in section 3, leading to an initial knowledge denoted with KB0 in Figure 9.

Another result of the Domain analysis phase is a partitioning of the application domain into several subdomains. A team of experts can now develop separate knowledge bases for each independent subdomain. Each expert teaches a personal Disciple agent, starting from the common knowledge base KB0 and building a refined one, as indicated in Figure 9. Then, the developed knowledge bases are merged into the Final KB. This KB will contain a merged ontology, but separate partitions of rules, one for each subdomain. The ontology merging algorithm exploits the fact that the KBs to be merged share KB0 as a common ontology. It starts with one of the KBs and successively merges it with the other KBs,
one at a time. Similarly to Prompt [Noy and Musen, 2000] and Chimaera [McGuinness et al., 2000], our approach to merging is based on providing an interactive way of copying one frame from an ontology into the other. While it is acknowledged that the role of the human cannot be eliminated from this process [Klein, 2001; Noy and Musen, 2000], the goal is to provide the most assistance to the knowledge engineer. Therefore, our tool handles the low level operations, allowing the user to issue only the most general commands, and assuring that the ontology is kept consistent at all times. In addition to that, the agent makes suggestions and keeps the user focused on the part of the ontology being merged.

The parallel KB development and merging capabilities of Disciple were first evaluated in Spring 2002, as part of “IT 803 Intelligent Agents” course at George Mason University. The students had to develop an agent for helping someone to choose a PhD advisor. The domain was split into six parts that were developed separately by the students in the class. They started the knowledge base development with a general 23-fact knowledge base provided by the instructor and each of them had to extend it with the knowledge needed to express their own part of the domain. Each student extended its knowledge base with an average of 97 facts. Using the merging tools provided by Disciple, the students succeeded to merge all their work into a single agent with an ontology containing 473 facts. We plan to validate the entire methodology in a new experiment at the US Army War College, as part of the Spring 2003 MAAI course.

Future work includes the capability to import from OKBC knowledge servers [Chaudhri et al., 1998] and from DAML+OIL expressed ontologies [Connolly et al., 2001], and an improvement of the proactivity of the mixed-initiative ontology merging tool.

Acknowledgements. This research was sponsored by DARPA, AFRL, AFMC, USAF, under agreement number F30602-00-2-0546, by the AFOSR under grant no. F49620-00-1-0072, and by the US Army War College.

References


Abstract

It is widely accepted that the Semantic Web will be based on machine-readable metadata describing the content of resources. These descriptions are designed to enable intelligent agents to locate and filter relevant information with a higher level of accuracy. The Resource Description Framework (RDF) has been developed as universal language for encoding content-related metadata and recently a number of query languages have been proposed to extract information from metadata models. Current applications of RDF and RDF query languages only use very simple metadata like simple concept hierarchies (The Open Directory) or pre-defined attribute value pairs (Dublin Core). In this paper we address the problem of encoding and querying complex metadata using RDF models and queries. In our approach we consider ontologies with complex concept definitions in the spirit of DAML+OIL and propose a pre-processing method that enables us to access these models using RDF query languages without losing information.

1 Motivation

In todays semantic web research there is still a big gap between theoretical considerations about the use of rich background knowledge in terms of ontologies and real implementations that are actually used in applications on the web. While theoreticians claim that rich knowledge models (as supported by the ontology language DAML+OIL or recently OWL) provide the expressive power and reasoning support needed in many domains, the application of such rich models is hampered by the high complexity of reasoning that makes them unlikely to scale up to a realistic setting. As a consequence, most existing implementations of semantic web infrastructure such as JENA 1, RDFSuite 2 or Sesame 3 rely on more light-weight solutions restricting them selves to RDF, RDF schema or a small subset of the OWL language.

Our concern is now to close the gap between these views and to provide ways of exploiting the expressive power of rich ontology languages in combination with efficient implementations that rely on light weight solutions. In this paper, we concentrate on the problem of querying information on the Semantic Web. Most existing implementations support querying RDF models, some also supporting schema aware querying. However, it has been argued that the presence of rich background knowledge requires deductive reasoning for achieving complete answers [Horrocks and Tessaris, 2000]. To our knowledge, none of the existing implementations of semantic web infrastructure supports this kind of query processing 4. Therefore, if we want to make use of the expressive power of languages like DAML+OIL and the querying facilities of existing infrastructure, we have to find ways of pre-processing DAML+OIL models in such a way that they can be handled by methods developed for light-weight approaches.

The obvious way of dealing with DAML+OIL models in a lightweight setting, ignoring all language features not supported by the light weight approach has serious drawbacks as it always means loosing information. Therefore, rather than weakening the background model, our approach is based on the idea of compiling out implicit knowledge in the background model and enriching the plain RDF model with this additional information. As finding the hidden information again requires expensive logical reasoning, this compilation step is done off-line. The actual query answering is done on the enriched model already. This approach, known as knowledge compilation, is well known in Artificial Intelligence research [Cadoli and Donini, 1997]. The novelty of this work is to apply the idea in the specific context of the semantic web and its representations.

The contribution of this paper is two-fold:

- We show how existing reasoning techniques can be used to compile implicit information about named objects of a domain model into a plain RDF description.
- We propose a heuristic approach for enhancing this

---

1http://www.hpl.hp.com/semweb/jena.htm
2http://139.91.183.30:9090/RDF/
3http://sesame.aidministrator.nl/

4There is a prototypical implementation of the query answering approach of Horrocks and Tessaris, but it does not provide a real infrastructure for handling semantic web data
description with information about anonymous objects than can be derived from the background knowledge.

The paper is organized as follows. We first review some basic notions of RDF, mainly referring to its model theoretic semantics. We pay special attention to the interpretation of unlabeled nodes as existentially quantified variables referring to anonymous objects and on their role in query answering. Afterwards we review existing work on explicating implicit information contained in background knowledge. We consider the case of RDF schema and of DAML+OIL model. Finally, we discuss the issue of implicit relational information that is only partially covered by existing approaches. We show how the query answering approach of Horrocks and Tessaris [Horrocks and Tessaris, 2000] can be used to compile relational information about named objects and we sketch an approach for dealing with unnamed objects in the compilation process. We summarize with a discussion of open questions.

2 RDF Querying

The ability to query RDF models in an efficient way is an essential aspect with respect to building semantic web applications. As RDF is designed as a language for describing the content of an information source, being able to find a specific piece of information is equivalent to being able to find the corresponding RDF statement. In the following, we present a general view on answering queries based on the RDF data model introduced above and briefly review the RDF query languages.

2.1 RDF Query languages

The RDF data model described above and its associated semantics provides us with a basis for defining queries on RDF models in a straightforward way. The idea that has been proposed elsewhere and is adopted here is to use graphs with unlabeled nodes as queries. The unlabeled elements in a query graph can be seen as variables of the query. In this approach, answers to a query can be defined in two ways:

- A sub-graph of the given RDF model that is an instantiation of the query graph.
- The set of resources that if they are used to instantiate unlabeled nodes in the query graph result in a sub-graph of the given RDF model.

From a theoretical point of view these two definitions are exchangeable as one can easily be derived from the other by either extracting instantiated resources from the answer graph or by instantiating the query graph with the resources from the answer set, respectively. Due to this correspondence, we will use the definitions synonymously.

Another characteristic feature of RDF is the possibility to refer to entire RDF graphs and not only to single resources. At first sight, the ability to refer to entire statements destroys the graph. Another feature of graph-based queries is that answering queries about RDF models can be based on the same techniques that are used for reasoning about such model. In especially, graph matching is a basic technique with respect to reasoning and query answering (see e.g. [Carroll, 2002]). Adopting this technique, query answering is done by finding a match between the query graph and the RDF model, allowing unlabeled nodes in the query graph to be matched against any node in the model. Once a match is found, the query graph can be instantiated with the matching nodes on the model or those nodes can be returned as an answer.

With the increasing interest in RDF, a number of RDF query languages have been developed and integrated in the available infrastructure. One of the most widely adopted ones that is based on graphs as queries is SquishQL [Miller et al., 2002]. Different Implementations of SquishQL based languages exist as part of RDF APIs or persistent storages and can therefore be assumed to be widely used. In the following, we consider RDQL, an implementation of SquishQL that is part of the Jena RDF Toolkit and the RDF storage and query system Sesame. Following the basic definitions of SquishQL a query has the following components:

- SELECT the select clause identifies variables form the query graph that should be returned as the result of the query.
- FROM the from clause identifies the RDF model to be used as a basis for querying. The model is identified by its URI.
- WHERE the where clause describes the query graph in terms of a conjunction of RDF triples connected by variables.
- AND the and clause contains a Boolean expression that constrains possible variable instantiations in the query graph.
- USING the using clause can be used to define abbreviations used in the specification of the query.

These constructs allow the user to formulate queries in an SQL like style which is well known by most application developers.

2.2 An Example

We use a toy example from the domain of family relationships to illustrate the way queries are formulated. For the sake of simplicity, we omit the FROM, and USING clauses.

```
SELECT ?p ?c
WHERE
  (?p has-child ?c)
  (?p type ?g)
  (?p age ?a)
AND
  (?g eq #female)
```

The above query asks for mothers and their children. The result of a query will be a table with the identifiers of persons that satisfy the requirements specified in the query, namely that they are in the has-child relation where the first member of the relation is of type female. We further use the following RDF model of mothers to be queried:

```
<rdf:Description rdf:about="alice">  
  <type rdf:resource="#female">  
```
Applying the query specified above we would get the pair (alice, betty) as a result because alice is defined to be female at having a child. This result is not very satisfying as it is easy to see that all of the defined persons are actually mothers and should therefore be in the answer to the query. The reason for the incomplete result is the lack of information about the meaning of the descriptions.

3 Ontology-Aware Querying

One of the main features of the semantic web is the idea to enrich metadata descriptions with explicit models of their intended meaning. These models range from simple schema definitions to complex ontologies that define concepts and describe them by necessary and sufficient conditions thus enabling intelligent applications to reason about their members and their relation to each other. On the semantic web, we want to exploit this semantic information for query answering as it provides us with background information for computing more complete results. In the example above we also want to retrieve instances as an answer to ?p that have the following properties:

- are of type female and have a daughter or a son, because this implies having a child
- have a child and are human and not male because not being male implies being female.
- are of type mother, because this implies being female and having a child.
- are of type virgin-mary as this implies being the mother of christ.

Expressive language have been developed that can be used to encode the kind of background knowledge needed to draw the conclusions mentioned above. In especially, RDF schema provide means for define simple schematic information [Brickley and Guha, 2003]. DAML+OIL extends RDF schema towards a more expressive language for defining the meaning of classes [van Harmelen et al., 2001a]. Unfortunately, simple query language like RDQL are not able to make use of the background knowledge per se. We rather have to do some pre-processing on the RDF model to be queried in order to get all intended results. In the following, we describe how this pre-processing step can be done in order to query metadata that is based on an RDF schema and on a DAML+OIL ontology, respectively, and refer to existing approaches that use these methods to support querying. Afterwards, we point out to remaining problems that are the main motivation for the compilation approach, we propose as an extension of existing proposals.

3.1 Schemas

RDF schema provide a language for encoding structural background information about the vocabulary used in an RDF Model. This structural information provides insights into the relations between the different elements of the model and helps to draw conclusions that could not be found from the plain model. In particular, a schema defines hierarchies of classes and relations as well as restrictions on the range and the domain of relations. In the family domain, we use in our example, the schema may contain the following information about the classes and relations used in the model:

This schema defines that having a son is a special case of having a child, that mother is a subclass of female persons which is the range of the relation has-mother and the domain of the relation mother-of, which in turn is a special case of having a child. We immediately see that this information is relevant for answering our query as from it we can conclude that Jane and Eve fulfill the requirements specified in the query. Formally, this is established by the axiomatic semantics of RDF schema given in [Hayes, 2003]. Applying these axioms to our example RDF model, we can add a number of new facts that can be matched by the query engine. In particular, we get the following more complete definitions of jane, eve and betty:
Now that the information about these resources has been made explicit, posting the example query against the expanded model will also return Jane and Eve as an answer to the query. Betty, however, is now only known to be female, but there is no explicit statement about her child which disables the query engine to recognize her as matching the query. In fact, this approach of first expanding a model using schematic information and then evaluating queries against the completed model is a common approach that has for example been implemented in Sesame [Broekstra et al., 2002].

3.2 Ontologies

The example domain already shows that there are many aspects of terminological knowledge that cannot be captured by RDF schema. These aspects include the following facts that might be relevant for answering queries about our domain:

- the same person may not be male and female
- a female person automatically becomes a mother when having a child
- Virgin Mary is the mother of Christ

Including these facts in a model of information semantics requires a far more expressive language. DAML+OIL [van Harmelen et al., 2001a] is such a language that has been defined on top of RDF schema, extending it with additional operators for defining classes by constraining possible members. For our domain a DAML+OIL model might contain the following definitions:

The model adds further semantic information about the domain to our model, namely the fact that the class of all humans is exactly the union of male and female person. That male persons cannot be female at the same time, that a mother is a female person having a child and that Virgin Mary is a female person having exactly one child which is Jesus Christ.

In [van Harmelen et al., 2001b] a formal semantics for DAML+OIL is described. The semantics is based on an interpretation mapping into an abstract domain. More specifically, every concept name is mapped on a set of objects, every property name is mapped on a set of pairs of objects. Individuals (in or case resources) are mapped on individual objects in the abstract domain. Formally, an interpretation is defined as follows:

**Definition 1 (Interpretation)** An Interpretation consists of a pair \((\Delta, \mathcal{E})\) where \(\Delta\) is a (possibly infinite) set and \(\mathcal{E}\) is a mapping such that:

- \(x \in \mathcal{E}\) for every individual \(x\)
- \(C \subseteq \mathcal{E}\) for all classes \(C\)
- \(R \subseteq \Delta \times \Delta\) for all roles \(R\)

We call \(\mathcal{E}\) the extension of an individual, concept or role, respectively.

This notion of an interpretation is a very general one and does not restrict the set of objects in the extension of a concept. This is done by the use of operators for defining classes. These kinds of operators restrict the possible extensions of a concept. These kinds of restriction are the basis for deciding whether a class definition is equivalent, more specialized or more general than another. Formally, we can decide whether one of the following relations between two expressions hold:

- **subsumption**: \(C_1 \subseteq C_2 \iff C_1^\mathcal{E} \subseteq C_2^\mathcal{E}\)
- **membership**: \(x : C \iff x^\mathcal{E} \in C^\mathcal{E}\)
Based on the definitions of subsumption and membership, we can use terminological reasoners to compute these relations from a given model and the corresponding ontology. The main results for the given model are that mother is a subclass of female as all human beings that are not male are known to be female (the class human is a partition of male and female persons). This of course implies that all members of the class mother are also members of the class female. The resulting relations that correspond to the rdfs:subClassOf and the rdf:type statements can now be added to the RDF model and its schema. The result are extended descriptions of Carol, Doris and Mary as given below:

```xml
<rdf:Description rdf:about="carol">
  <type rdf:resource="#female"/>
  <has-child rdf:resource="#doris"/>
</rdf:Description>

<rdf:Description rdf:about="doris">
  <type rdf:resource="#mother"/>
</rdf:Description>

<rdf:Description rdf:about="mary">
  <type rdf:resource="#female"/>
  <type rdf:resource="#mother"/>
</rdf:Description>
```

We see that the additional information obtained by explicitly adding implicit subclass and type relations to the model extends set of answers to our query with Carol, because it is now explicitly stated that she is female and has a child named Doris. This approach goes further than the use of schema information, however, it does not solve all problems, because we still cannot find out that Doris and Mary are also answers to our query.

4 Relational Knowledge

The reason why present approaches fail to compile the ontology in such a way that all intended answers can be given from the RDF model is caused by their limited abilities to compile relational knowledge. In the example it followed from the ontology that Mary is related to jesus-christ by the has-child relation. In the case of Doris, it is implied that she is connect to some, maybe unknown object via the same relation. Description logic reasoners allow to reason with these kinds of relational information by constructing models in terms of possible objects, their type and their relations. However, this information is only used internally to establish subsumption and membership relations. In order to overcome this problem we can use techniques for deductive query answering in the off-line compilation phase and store the retrieved answers as explicit knowledge in our RDF model.

4.1 Compiling Object Relationships

Horrocks and Tessaris propose a deductive approach for answering conjunctive queries over description logic knowledge bases. The idea of the approach of Horrocks and Tessaris now to translate the query into an equivalent concept expression, classify this new concept and use standard inference methods to check whether an object is an instance of the query expression. This approach makes use of the fact that binary relations in a conjunctive query can be translated into an existential restriction in such a way that logical consequence is preserved after a minor modification of the A-Box. Details are given in the following theorem.

**Theorem 1 (Role Roll-Up (Horrocks and Tessaris 2000))**

Let \( \langle C, R, A \rangle \) be a description logic knowledge base with concept definitions \( C \), relation definitions \( R \) and assertions \( A \). Let further \( R \) be a role, \( C \) Concept names in \( T \) and \( a, b \) be individual names in \( T \). Then

\[
\langle C, R, A \rangle \models (a, b) : R \land b : C_1 \land \cdots \land b : C_k
\]

if and only if

\[
\langle C, R, A \cup \{ b : P_b \} \rangle \models a : \exists R(P_b \cap C_1 \cap \cdots \cap C_k)
\]

As DAML+OIL can be seen as a specific variant of description logics, the query answering approach can be applied to DAML+OIL ontologies (compare [Horrocks and Tessaris, 2002]). In especially, we can directly ask for objects that are related by the has-child relation using the following very simple query:

\[
Q(X, Y) \leftarrow (X, Y) : \text{has - child}
\]

Using the theorem of Horrocks and Tessaris, we can now do the translation of the conjunct \((X, Y) : \text{has - child}\) into a concept expression called role-up. In order to actually retrieve related objects, we do this translation for instantiations of the general conjunct where the \( Y \) variable is replaced by an object contained in the model. Substituting \( X \) for example by the object \( \text{jesus - christ} \) we get the conjunct \((X, \text{jesus - christ}) : \text{has - child}\) that translates into the concept expression \(\exists \text{has - child}.P\text{jesus-christ}\). This concept specifies all objects that are related to the object \( \text{jesus - christ} \) by the \( \text{has - child} \) relation. We can use existing description logic reasoners in order to retrieve all objects belonging to this concept. For the example instantiation, the reasoner will return mary telling us that we can add the information that mary and jesus-christ are in the has child relation into the RDF model. The new definition of mary will be the following:

```xml
<rdf:Description rdf:about="mary">
  <has-child rdf:resource="#jesus-christ"/>
</rdf:Description>
```

In order to compile all the object relations implied by an ontology, we have to iterate this process over all instances and over all relations mentioned. The corresponding algorithm is depicted as Algorithm 1.

Given a set of relations and objects, the algorithm compiles out all relational information about them that is implied by
Figure 1: Basic algorithm for compiling object relations

4.2 Anonymous Objects

While the use of the query answering approach of Horrocks and Tessaris enables us to compile out information about relations that exists between objects in a model, there is still implicit relational information that is not captured by this approach. The reason for this is that the logical nature of DAML+OIL allows us to capture incomplete information about the domain of interest in the sense that it is not required to always name related objects in a concept definition. There are also ways of talking about the existence and the number of related objects in the domain without actually naming these objects. The approach of Horrocks and Tessaris ignore this information, because their approach aims at answering questions about named objects in a model. We think, however, that in the in the context of the semantic web there are also may situations where we are also interested in this incomplete information. This argument is based on the open world assumption. The fact that the model does not contain the name of the child of a female person in our example does not mean that there is no information about this child. It just happens not to be contained in this specific model. Therefore, the answer that doris by virtue of being a mother has a child we do not know the name of is also a valuable answer to our example question. Following this argument, we sketch an approach for also compiling relational information that contains anonymous objects. In the resulting RDF model these anonymous objects will be represented by blank nodes.

In order to determine what kind of anonymous objects we have to deal with, we have to have a look at the possibilities DAML+OIL provides us for describing relations to objects that are not mentioned themselves. There are two operators that are directly connected with anonymous objects (compare figure 2). First of all, there is the daml:hasClass operator claiming that all objects of a class are related some object of a certain type. Further, there are cardinality statements claiming that all objects of a class are related to at least, at most or exactly a certain number of objects of a certain type. The case where no special type of the related objects is required is a special case and can therefore also be treated by the approach. The same holds for the first mentioned daml:hasClass operators, because it is equivalent to stating that the minimal number of object of certain type that is required equals one. Further, requiring an exact number of related objects (daml:CardinalityQ) can be expressed by requiring a minimal and a maximal number of related objects of a certain type where both numbers equal the required number of objects. Therefore, the main relational construct, we have to focus on when investigating the problem of compiling relational knowledge are the following constructs:

\[
\text{<daml:restriction daml:minCardinalityQ="n">}
\text{<daml:onProperty rdf:resource="#p"/>}
\text{<daml:hasClassQ rdf:resource="#C"/>}
\text{</daml:restriction>}
\]

\[
\text{<daml:restriction daml:maxCardinalityQ="m">}
\text{<daml:onProperty rdf:resource="#p"/>}
\text{<daml:hasClassQ rdf:resource="#C"/>}
\text{</daml:restriction>}
\]

In the next section we introduce an extension of the compilation algorithm that deals with anonymous objects making use of these constructs.

4.3 Compilation with Anonymous Objects

As motivated above, our approach for compiling relational information with anonymous objects relies on the use of qualified number restrictions. We assume that all existential restrictions and unqualified number restrictions have been translated into qualified number restrictions in the way sketched in the last section. Further, we assume that the background knowledge is consistent in itself and that the information model is consistent with this background model. In especially, this guarantees that there is no conflict between the upper and lower bounds of related objects with respect to the qualified number restrictions we use for compilation. This requirement can be checked using existing reasoning systems. For every object o and relation r in the model we now perform a four step compilation process:

Step 1: Collect Bounds In the first step, we determine the possible range of number of objects related to o by the relation r. We collect all class descriptions o can be proven to be a member of. From these description, we extract all qualified number restrictions that refer to r. Following this argument, we sketch an approach for also compiling relational information that contains anonymous objects. In the resulting RDF model these anonymous objects will be represented by blank nodes.

Step 2: Verify Bounds In the second step, we check whether the bounds determined for the individual classes
in the hierarchy are consistent with the bound determined for their its subclasses. For every class c in the hierarchy starting at the bottom of the hierarchy, we sum up the lower bounds determined for all direct subclasses. We then check, whether this sum is smaller than the upper bound determined for that class.

**Step 3: Adjust Bounds** In this step, we adjust the lower bound on related objects in the light of collected information about more specific information about related objects that imply some of the information contained in the current bounds.

a) As a first step, we set the lower bound to the current upper bound if the upper bound is lower is lower than the sum of the lower bounds determined for all direct subclasses.

b) As a second step we subtract the number of all known and already compiled anonymous objects of this type related to o. This ensures that only the most specific available information is actually compiled out and prevents us from adding redundant information.

**Step 4: Compile Relations** In the last step, we add information about relations of the object o to anonymous objects to the RDF model by adding the corresponding triples. For each class name, we look up the finally determined lower bound of objects related to o via r. This bound is an natural number l that specifies the number of necessarily existing relations excluding already known relational information and necessary relations to objects of a more specific type (compare step 2). We generate l anonymous objects ai and add the triples (o r ai) and (ai type c) for each of these objects.

It is important to recall, that different from the other compilation methods described in this paper, this compilation process is of a heuristic nature and does not claim to produce logically sound results. The reason is that the current version of the compilation approach does not take into account all information contained in the background knowledge.

**4.4 A Simple Example**

We illustrate our compilation approach using a simple example. Consider the following model describing the instance betty:

```
<daml:Class rdf:ID="grandma">
  <rdfs:subClassOf>
    <daml:Class rdf:resource="mother">
      <daml:restriction>
        <daml:onProperty rdf:resource="#has-child"/>
        <daml:hasClass rdf:resource="#parent"/>
      </daml:restriction>
    </rdfs:subClassOf>
  </daml:Class>
</daml:Class>

<daml:Class rdf:ID="relaxed-parent">
  <rdfs:subClassOf>
    <daml:restriction daml:maxCarninality="2">
      <daml:onProperty rdf:resource="#has-child"/>
    </daml:restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="person">
  <rdfs:subClassOf>
    <daml:hasClassQ rdf:resource="#person"/>
  </rdfs:subClassOf>
</daml:Class>

<rdf:Description rdf:about="betty">
  <type rdf:resource="#grandma"/>
  <type rdf:resource="#relaxed-parent"/>
  <has-child rdf:resource="#peter"/>
</rdf:Description>

<rdf:Description rdf:about="peter">
  <type rdf:resource="#person"/>
</rdf:Description>
```

When we look at the definition of betty, we see that there are different sources of relational information, we are interested in. From her being of type mother (implied from the fact of being a grandmother) we derive a lower bound of one for the has-child relation with respect to the class person. Further, being a grandma also implies a lower bound of one on the has-child relation with respect to the class parent, which is a subclass of person. An upper bound of two on the relation has-child is provided by the membership to the class relaxed-parent. Finally, there is the explicitly mentioned child peter who is of type person. The result of collecting these bounds is shown in figure 3a.

After processing the class parent which is lower in the hierarchy, we see that the lower bound on the number of related objects of type parent is higher that the number of related individuals of that type (i.e. one compared to zero). As a reaction, we add an anonymous object of type parent to
RDF model (compare figure 2b). In the next iteration of the process, we process the concept person by adding the lower bound of all sub-concepts and checking it against the upper bound (figure 2c). From the resulting lower bound of two we subtract the number of all related objects of this type. In our case there are two objects of this kind: peter and the anonymous object created in the previous iteration. As the result is zero we conclude that the necessary number of related objects is already present in the model. The triples added in the first iteration of the process lead to a more complete definition of the instance elly with respect to the has-child relation as shown below:

```xml
<rdf:Description rdf:about="betty">
  <type rdf:resource="#happy-grandma"/>
  <type rdf:resource="#relaxed-parent"/>
  <has-child rdf:resource="#peter"/>
  <has-child rdf:resource="_anonX"/>
</rdf:Description>

<rdf:Description rdf:about="_anonX">
  <type rdf:resource="#parent"/>
</rdf:Description>
```

The added information can now be used in querying enabling us to derive that elly is necessarily connected to an instance of type parent by the has-child relation using a plain RDF query language. In fact, this object of type parent may be identical with peter who we already know, but the background model does not provide us with enough information to prove whether this is true or not.

5 Discussion

In this paper, we addressed the problems that arise from the existing gap between theoretical investigations and practical implementations of semantic web technology. For the specific problem of querying RDF models with background knowledge we presented an approach to enhance plain RDF descriptions by information implied by the background knowledge. To this end, we discussed some already existing approaches and presented new methods for compiling complex relational knowledge using the query answering approach proposed by Horrocks and Tessaris. All the methods mentioned produce a provably correct and complete result with respect to answering queries about named objects in an RDF model.

We argued that an open world assumption as we face to on the semantic web also requires to consider anonymous objects in answering queries. We described a first step towards an adequate treatment of these objects in the compilation step. The approach presented already produces some reasonable results, but it cannot give any soundness or completeness guarantees. We do not expect to come up with a provably complete algorithm for this problem, however, there are still many options for improving the result presented most of them concerned with features for defining complex relational structures (e.g. role hierarchies, transitivity). Further, we need more experiences with applying these methods to realistic scenarios in order to develop heuristics for improving the compilation result.

References


C. Ontologies in Distributed Systems
Abstract

Ontologies are a key component for building open and dynamic distributed pervasive computing systems in which agents and devices share contextual information. We describe our use of the Web Ontology Language OWL and other tools for building the foundation ontology for the Context Broker Architecture (CoBrA), a new context-aware pervasive computing framework. The current version of the CoBrA ontology models the basic concepts of people, agents, places, and presentation events in an intelligent meeting room environment. It provides a vocabulary of terms for classes and properties suitable for building practical systems that model context in pervasive computing environments. We also describe our ongoing research in developing an OWL inference engine using Flora-2 and in extending the present CoBrA ontology to use the DAML spatial and temporal ontologies.

1 Introduction

Computing is moving toward pervasive, ubiquitous environments in which devices, software agents, and services are all expected to seamlessly integrate and cooperate in support of human objectives – anticipating needs, negotiating for service, acting on our behalf, and delivering services in an anywhere, any-time fashion [Weiser, 1991; Finin et al., 2001]. An important next step for pervasive computing is the integration of intelligent agents that employing knowledge and reasoning to understand the local context and share this information in support of intelligent applications and interfaces. We are developing a new pervasive context-aware computing infrastructure called Context Broker Architecture (CoBrA) [Chen, 2003], to support ubiquitous agents, services and devices to behave intelligently in according to their situational contexts.

Ontologies are key requirements for building context-aware pervasive computing systems for the following reasons: (i) a common ontology enables knowledge sharing in an open and dynamic distributed systems, (ii) ontologies with well defined declarative semantics provide a means for intelligent agents to reason about contextual information, and (iii) explicitly represented ontologies allow devices and agents not expressly designed to work together to interoperate, achieving “serendipitous interoperability” [Heflin, 2003].

In the past, a number of distributed systems have been developed to support pervasive computing including the Intelligent Room [Coen, 1998], Cooltown [Kindberg and Barton, 2001], and Context Toolkit [Salber et al., 1999]. These systems have made progress in various aspects of pervasive computing but are weak in supporting knowledge sharing and context reasoning. A significant source of this weakness is their lack a common ontology with explicit semantic representation [Chen et al., 2001; Chen, 2003]. CoBrA provides better support for knowledge sharing and context reasoning using a common ontology defined using Semantic Web languages. In this paper, we describe the use of the Web Ontology Language OWL [van Harmelen et al., 2002] and tools for building an ontology foundation in CoBrA.

In the next section, we overview CoBrA and its design rationale. In Section 5, we describe the role of the Semantic Web and the OWL language in our architecture. Section 4 describes two components that we believe to be necessary for building an ontology foundation in pervasive context-aware systems (e.g., in CoBrA). After our discussion, in Section 5, we present our initial work in building an ontology called CoBrA Ontology for modeling context knowledge and enabling knowledge sharing – this is the first component of the ontology foundation in CoBrA. In Section 6, we describe our on-going research work which attempts to complete the second component of the CoBrA ontology foundation, an ontology inference engine for OWL. A brief discussion of related work and our future work are given in Section 7 and Section 8, respectively. In Section 9, we summarize this document.

2 Context Broker Architecture

CoBrA is an agent based architecture for supporting context-aware computing in intelligent spaces. Intelligent spaces are physical spaces (e.g., living rooms, vehicles, corporate offices and meeting rooms) populated with intelligent systems that provide pervasive computing services to users [Kagal et al., 2001]. By context, we mean an understanding of a location, its environmental attributes (e.g., noise level, light intensity,
temperature and motion) and the people, devices, objects and software agents it contains.

Figure 1: An intelligent context broker acquires context information from devices, agents and sensors in its environment and fuses it into a coherent model, which is then shared with the devices and their agents.

Central to our architecture is the presence of an intelligent context broker (or broker for short) that maintains and manages a shared model of contexts on the behalf of a community of agents (i.e., applications running on the mobile devices that a user carries or wears, services that are provided by devices in a room, and web services that provide web presences for people, places and things in the physical world [Kindberg and Barton, 2001]). In our system, a broker assumes the responsibility to (i) acquire contexts from heterogeneous information sources and maintain the consistency of the overall context knowledge through reasoning, (ii) help distributed agents to share context knowledge through the use of ontologies, agent communication languages and protocols, and (iii) protect the privacy of users by establishing and enforcing user defined policies while sharing sensitive personal information with agents in the community. Figure 1 shows a high-level design of the broker and its relationship with agents in an intelligent space.

In a large-scale intelligent space (e.g., a campus or a building), multiple brokers can form a broker federation. Individual broker in a federation is responsible for managing parts of the intelligent space (e.g., a room in a particular building). In a federation, brokers are related to each other in some organizational structure (e.g., peer-to-peer or hierarchical), and they can periodically exchange and synchronize context knowledge.

Our centralized broker design addresses two important issues that are key to realizing the potential of ubiquitous computing: supporting resource-limited mobile computing devices [Dertouzos, 2001; Coen, 1998; Chen and Kotz, 2000] and addressing the concerns for user privacy and security [Ackerman et al., 2001; Bellotti and Sellen, 1993]. With the introduction of a context broker that operates on a stationary computer, the burdens of acquiring and reasoning over context information will be shifted away from resource-limited mobile devices to agents running on resource-rich servers; the complications inherent in establishing, monitoring and enforcing security, trust, and privacy policies will be simplified in the presence of a centralized manager. Although the existence of context broker could bring about the above advantages, its centralized design could create a “bottle neck” in a distributed system, hindering the overall system performance. In our preliminary research work, we have not addressed this problem. However, according to Kumar and Cohen [Kumar et al., 2000], this “bottle neck” issue could be resolved through fault-tolerance by introducing a persistent broker team.

3 Rationales for Exploring Semantic Web

The responsibility of a context broker is to acquire, maintain and share a coherent and consistent model of the local context. Our approach to doing this is a knowledge-based one built on a declarative ontology of basic concepts for objects and relations in a pervasive environment. The ontology is further defined by axioms that provide additional constraints and meaning as well as rules and heuristics that can derive additional useful information. Somewhat surprisingly, we found that the languages developed for the Semantic Web are also well suited for our purpose. Key design requirements are common to the web and pervasive computing. Both are very open system with a high degree of dynamism in which independent and autonomous agents publish content and also search for information of interest.

Semantic Web is a vision of the next generation World Wide Web [Berners-Lee et al., 2001]. Research efforts in the Semantic Web are driven by the need for a new knowledge representation framework to cope with the explosion of unstructured digital information on the existing Web. The present Semantic Web research focuses on the development of ontology languages and tools for constructing digital information that can be "understood" by computers [Berners-Lee et al., 2001].

In the past few years, ontology language developments in the Semantic Web have converged to a new W3C standard called OWL. The OWL language shares the same root as its predecessor DAML+OIL[Connolly et al., 2001] (e.g., using RDF as the modeling language to define ontological vocabularies and using XML as the surface syntax for representing information [van Harmelen et al., 2002]).

We have chosen the OWL language to model context ontologies for two reasons. First, it is much more expressive than RDF or RDF-S allowing us to build more knowledge into the ontology. Second, we chose to use OWL over DAML+OIL because OWL has been designed as a standard and has the backing of a well known and regarded standards organization.

Additionally, from a system design point of view, using OWL to define context ontologies underpins two important functions of a context broker. First, it provides a means for the broker to share context knowledge with agents in an associated intelligent space. Second, it provides an ontology model which can help the broker to reason about contexts and detect knowledge inconsistency.

Knowledge sharing in pervasive context-aware systems re-
requires all agents to share a common ontology\(^1\). Using the OWL language, ontology concepts are defined independent from any agent implementations, and their semantics are captured using standard knowledge representation vocabularies. Taking this approach, independently developed agents can share context knowledge with the broker without pre-defined agreements on how they should interoperate.

Context reasoning is a key function of the broker. Context reasoning involves deducing context knowledge from acquired situational information and detecting inconsistency in the knowledge base. To reason about contexts, the broker can exploit ontology reasoning using logic inference engines (e.g., the DAMLJessKB [Kopena and Regli, 1999], TRIPLE [Sintek and Decker, 2002], FaCT [Horrocks et al., 1999], RACER [Volker Haarslev, 2001] and Bubo [Volz et al., 2003]).

## 4 CoBrA Semantic Web Ontology Foundation

An ontology-driven design methodology is one way to build a distributed intelligent system (e.g., CoBrA) that can reason about contexts and can help agents to share knowledge. Using an explicit representation of the ontology, context knowledge can be reasoned over to derive additional information [Chen and Tolia, 2001], and this knowledge can also be easily shared by distributed agents using standard communication languages and protocols (e.g., FIPA-ACL, KQML, and SOAP/XML-RPC). This approach requires a suitable ontological foundation on which CoBrA specific components can be built. We believe the followings are two necessary components in this foundation:

1. **Context Ontology**: The ontology provides a set of terms for describing context knowledge (i.e., explicit statements that describe contexts in the environment). The ontology should be developed in a language with appropriate expressive power and a well defined semantics. This ontology allows distributed agents to share a common understanding of the information that they exchange and to reason about additional information that is beyond what is already known.

2. **Ontology Inference Engine**: an ontology inference engine is a logic system that reasons over the semantic model of an ontology. To reason about our context ontologies in OWL, for example, an ontology inference engine should provide a set of rules for interpreting the semantic model of OWL [Patel-Schneider et al., 2003] and detecting inconsistency in the knowledge base.

## 5 A Walkthrough of the CoBrA Ontology

This section describes key ontology concepts in the current version of the CoBrA ontology (v0.2)\(^2\). This ontology defines a set of vocabularies for describing people, agents, places and presentation events for supporting an intelligent meeting room system on a university campus. It also defines a set of properties and relationships that are associated with these basic concepts.

Figure 2 shows a complete list of the names of the classes and properties in the CoBrA ontology. Version v0.2 includes 41 classes (i.e., RDF resources that are type of owl:Class) and 36 properties (i.e., RDF resources that are type of either owl:ObjectProperty or owl:DatatypeProperty).

Our ontology is categorized into four distinctive but related themes: (i) concepts that define physical places and their associated special relationships (e.g., containment relationship, social and organizational properties)\(^3\), (ii) concepts that define agents (i.e., both human agents and software agents) and their associated attributes, (iii) concepts that describe the location contexts of an agent on a university campus, and (iv) concepts that describe the activity contexts of an agent, including the roles of speakers and audiences and their associated desires and intentions in a presentation event. In the rest of this section, we will discuss each of these four themes.

### 5.1 Concepts Related To Places

The notion of a place in CoBrA is restricted to a set of physical locations that are typically found on a university campus. These locations include campus, building, room, hallway, stairway, restroom, and parking lot. These physical locations are all assumed have well defined spatial boundaries (e.g., all locations can be uniquely identified by geographical coordinates – longitude and latitude). In addition, all locations on a university campus have identifiable string names that are assigned to them by some official bodies (e.g., by the university administration).

When modeling physical locations, we define a class called Place which generalizes all type of locations on a campus. This abstract class defines a set of properties that are common to all concrete physical location classes, which consists of longitude, latitude and hasPrettyName.

Place classes (including subclasses) have associated containment relationships. These relationships are defined by two related object properties\(^4\) called spatiallySubsumes and isSpatiallySubsumedBy. The former describes the subject of this property spatially subsumes the object of this property (e.g., a building spatially subsumes a room in the building), and the latter describes the subject of this property is spatially subsumed by the object of this property (e.g., a room in the building is spatially subsumed by the building). In the context of the OWL language, these two properties are defined as an inverse property of each other.

Note that in the current version of the ontology, the domain and the range of both spatiallySubsumes and isSpatiallySubsumedBy properties are of the class type Place. In other word, these two properties cannot be used to make statements about the containment of a person or an agent in a physical place. However, in Section 5.2, we will describe alternative constructs for expressing this type of statements.

---

\(^1\)This sharing might, in practice, be achieved with the help of ontology translation agents.

\(^2\)A complete version of the ontology is available at [http://daml.umbc.edu/ontologies/cobra/0.2/cobra-ont](http://daml.umbc.edu/ontologies/cobra/0.2/cobra-ont) in the OWL/XML syntax.

\(^3\)In v0.2, only containment relationship is defined, additional properties will be included in the next version of the ontology.

\(^4\)This refers to the owl:ObjectProperty property.
In addition to containment relationships, physical places may be also associated with events and activities (e.g., a meeting may be taken place in a room, or an annual festive may be taken place on a university campus). To make statements about a place that is associated with some event, we introduce an object property called hasEvent, which has domain Place and range Event. Instances of Event can be associated with time intervals. We define Event-HappeningNow, a subclass of Event, to represent a set of all events that are currently happening (details of this class is discussed in Section 5.4). To make statements about a place that is associated with some event that is currently happening now, we define an object property called hasEvent-HappeningNow.

**AtomicPlace**

Some of the concrete physical locations that we have mentioned (i.e., campus, building, room, hallway, stairway, etc.) usually do not contain (spatially subsume) other physical locations. For example, hallways, stairways and rooms in a building are not usually considered to be a type of physical place that contains other places.

For this reason, we introduce an abstract class called AtomicPlace to represent the set of physical places that do not contain other physical places. This class inherits all properties from its superclass Place. However, it restricts the range of the properties spatiallySubsumes and isSpatiallySubsumedBy. In the AtomicPlace class, the cardinality of the property spatiallySubsumes is 0, indicating all instances of this class do not contain any other physical places. The range of the property isSpatiallySubsumedBy is restricted to the class CompoundPlace, which is a subclass of Place.

The CompoundPlace class represents all physical places that may contain other physical places. Figure 3 shows partial representation of these classes in OWL/XML syntax.

Some subclasses of the AtomicPlace class include Room, Hallway, Stairway, Restroom, LadiesRoom, MensRoom and ParkingLot.

**CompoundPlace**

While the AtomicPlace class is introduced to represent a set of places that contains zero number of Place instances, the CompoundPlace class is defined to represent a set of places that contains at least one or more numbers of Place instances. This class is also a subclass of Place. Being a subclass of the Place class, CompoundPlace inherits all properties from its parent class. In order to express all instances of the CompoundPlace class should only be spatially subsumed by instances of other CompoundPlace, the range of this class’s property isSpatiallySubsumedBy is restricted to have class type CompoundPlace. This restriction excludes all instances of the CompoundPlace class to be spatially subsumed by instances of the AtomicPlace.

### 5.2 Concepts Related To Agents

The notion of an agent in CoBrA represents both humans agents and software agents. Human agents are simply users in an intelligent space. Software agents, on the other hand,
are autonomous computing entities that provide services to users (either directly or indirectly) in an associated space.

All agents have associated properties that describe their contact information, which includes uniquely identifiable names, URLs to their home pages, and email addresses. In addition, agents are assumed to have certain roles in different events and activities (e.g., a person can have the speaker role in a presentation event, and device agents in the close vicinity may take on the presentation assistant role during the presentation session). Different roles may give rise to different desires and intentions of an agent.

In the CoBrA ontology, the notions of desire and intention are both associated with actions. Specifically, the notion of desire is defined as an agent’s desire for some action to be achieved by some other agents (e.g., a person with the speaker role may desire some service agents to dim the lights when his presentation starts), and the notion of intention is defined as an agent’s commitment to perform some particular action (e.g., a person with the audience role may intend to download a copy of the slides after attending a presentation event).

To model ontologies for agents, we introduce a general class called Agent, which is a set of all human agents and computational agents. We define the class Person to represent human agents and the class SoftwareAgent to represent computational agents (both of which are subclasses of the Agent class and disjoints with each other). All agents in our ontology are associated with properties that describe their contact information. To generalize properties that serve as descriptions of contact information, we define an object property called hasContactInformation. From this property, we further define sub-properties of contact information, which consist of hasFullName, hasEmail, hasHomePage and hasAgentAddress.

Role
In our ontology, the class Role represents a set of all roles that are presently associated with an agent. In other words, it is an abstract class that generalizes all possible types of agent roles. In v0.2 of the ontology, pre-defined subclasses of Role are SpeakerRole and AudienceRole.

To associate roles with an agent, the object properties fillsRole and isFilledBy are defined. These two properties are inverse property of each other – fillsRole has domain Agent and range Role, and isFilledBy has domain Role and range Agent.

Intentional Actions
All actions in CoBrA are defined as instances of the class IntentionalAction. Informally, intentional actions are actions that an agent performs intentionally and with certain goals in mind. In our design, we assume domain applications will extend this class to define specialized subclasses and instances. To support the construction of intelligent meeting room system, we have pre-defined a set of concrete instances of IntentionalAction that are common in presentation events (see Figure 4).

All instances of the IntentionalAction class (or its subclasses) can be associated with either an instance of the Role class or the Agent class through object properties intendsToPerform or desiresSomeoneToAchieve. The domain of these two properties are union of the class Role and Agent (see Figure 4).

5.3 Concepts Related to Agent’s Location Context
By location context, we mean a collection of dynamic knowledge that describes the location of an agent, which is a collection of RDF statements that describes the location property of an agent. The location property of an agent is captured through the object property locatedIn. It has range Place and domain owl:Thing, indicating anything (including agents) may be located in some physical place.

Physical locations, as discussed in the previous section, are categorized into two distinctive classes: AtomicPlace (e.g., hallways and rooms) and CompoundPlace (e.g., campus and building). Following the semantics of these two classes, we can make the following reasoning: no agent can locate in two different atomic places at the same time, but an
agent can be in two different compound places at the same time just in case one spatially subsumes the other. This reasoning is important for detecting inconsistent knowledge about the current location of an agent.

To capture the notion an agent can be located in an atomic and a compound place, from the locatedIn property, we define two sub-properties called locatedInAtomicPlace and locatedInCompoundPlace. The former restricts its range to the AtomicPlace class, and the latter restricts its range to the CompoundPlace class. From these two properties, we define additional properties that further restrict the type of physical place an agent can be located in. For example, locatedInRoom, locatedInRestroom and locatedInParkingLot are sub-properties of locatedInAtomicPlace; locatedInCampus and locatedInBuilding are sub-properties of locatedInCompoundPlace.

For agents that are located in different places, we can categorize them in according to their location properties. For example, we define PersonInBuilding to represent a set of all people who are located in a building, and SoftwareAgentInBuilding to represent a set of software agents who are located in a building, respectively. The complement of these classes are PersonNotInBuilding and SoftwareAgentNotInBuilding.

5.4 Concepts Related to Agent’s Activity Context

The activity context of an agent, similar to the location context, is a collection of dynamic knowledge about certain aspects of an agent’s situational condition. While location context describes the location in which the agent is situated, activity context describes activities in which the agent participates. In our ontology, the notion of an activity is restricted to activities in which the agent participates. For example, we define PersonInBuilding to represent a set of all people who are located in a building, and SoftwareAgentInBuilding to represent a set of software agents who are located in a building, respectively. The complement of these classes are PersonNotInBuilding and SoftwareAgentNotInBuilding.

Activity events are assumed have schedules. For presentation events, we define PresentationSchedule class to represent their schedules. Presentation schedules are defined to have startTime, endTime and location properties, and each of which respectively represents the start time of a presentation, the end time of a presentation and the location of a presentation event. Each presentation event has one or more invited speaker and expected audience. These two concepts are defined using the invitedSpeaker and expectedAudience properties. In addition to start time, end time and location, the schedule of a presentation usually includes a title and an abstract of the presentations. To model these, we introduce presentationTitle and presentationAbstract properties.

The activity context of an agent is usually associated with activity events that are currently happening. For example, the activity context of a speaker includes the presentation event at which he/she is giving the presentation. To model this, we introduce the PresentationEventHappeningNow class. This class is a subclass of the EventHappeningNow class which models an event with the time predicate “now”.

For a given presentation that is currently happening, we can specialize the type of rooms at which the event takes place. For example, a room that has an on-going presentation event is defined as RoomHasPresentationEventHappeningNow, which is a subclass of Room and restricts the range of its hasEventHappeningNow property to the class PresentationSchedule. To describe people have speaker and audience roles in an on-going event, we define the SpeakerOfPresentationHappeningNow class and the AudienceOfPresentationHappeningNow class.

6 An OWL Inference Engine in Flora-2

In the last section, we have described the CoBrA ontology, which forms the first component in the ontology foundation in our system. In order for a context broker to reason about contexts, an inference engine for reasoning over OWL ontologies is required.

At the present, inference engines that can reason over the complete semantic model of the OWL language is still under development (in Section 5 we have mentioned a few of these emerging inference engines). As a part of our research, we are developing an OWL inference engine called F-OWL using the Flora-2 system in XSB. Flora-2 is a system that translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine [Yang and Kifer, 2002]. Flora-2 has a language syntax that is similar to TRIPLE [Sintek and Decker, 2002] and also allows ontology semantics to be defined using rules.

F-OWL is a rule-driven logic inference engine. Its implementation consists of four distinctive but related sets of rules: (i) rules that define the semantic model of the RDFS ontology language, (ii) rules that define the semantic model of the OWL ontology language, (iii) rules that draw inferences over the semantic model of RDFS, and (iv) rules that draw inferences over the semantic model of OWL. Inputs to F-OWL are collections of the N-Triple representation of some domain ontologies (e.g., context knowledge that are described using the CoBrA ontology), and outputs from F-OWL are ontological knowledge that can be proved by the logic inferences that are defined in (iii) and (iv). To access the output ontological knowledge, Flora-2 queries can be used.

F-OWL is still in its early stage of the development. The latest version (v0.3)7 of F-OWL support a full RDFS inferences and partial OWL inferences (limited to the OWL-Lite sub-language constructs and some OWL Full constructs). We expect to complete a full inference of the OWL language in F-OWL by late June 2003.

7 Related Work

Our work is closely related to other pervasive and context-aware computing research such as Intelligent Room [Coen, 1998], Context Toolkit [Salber et al., 1999] and Cooltown [Kindberg and Barton, 2001], One.World [Grimm et al., 2000] and Centaurus [Kagal et al., 2001]. In comparison to

---

6In v0.2 of the ontology, we have only included concepts related to presentation events. In the future version, we will extend the ontology to includes other activity events

7http://umbc.edu/~hchen4/fowl

98
the previous systems, our novel design of the context broker attempts to address challenging issues such as developing explicit ontology representations of contexts, supporting context reasoning and maintenance through logic inferences and providing user privacy protection using policies (also see discussions in Section 5).

In the previous systems, user location contexts are widely used for guiding the decision making process of context-aware applications [Salber et al., 1999; Coen, 1998; Kagal et al., 2001]. However, none of them have taken advantage of the semantics of spatial relations in reasoning about contexts (i.e., information that describes the whole physical space that surrounds a particular location and its relationship to other locations).

8 Future Work

Modeling space and time are important in CoBrA. We currently have a simple model of space and spatial relationships (see Section 5.1) and an implicit representation of time and temporal relationships (see Section 5.4). In the next version of the CoBrA ontology, we plan on using, if possible, or at least mapping to, if feasible, one of the consensus ontologies for space and time.

8.1 Adopting Spatial Ontology

At present, there are two distinctive versions of spatial ontologies namely the spatial ontology in SUO [Niles and Pease, 2001] and the upper Cyc ontology [Cyc, 1997]. Recent discussions on the daml-spatial mailing list have initiated the work to develop a Semantic Web version of the spatial ontology based these ontologies. The new spatial ontology will cover representations for dimension, shape, length, area, volume, latitude, longitude, elevation, political subdivisions, and topological relations (e.g., Relation Connection Calculus [Randell et al., 1992]). As a short term objective, we plan to investigate the applications of Relation Connection Calculus in context reasoning (e.g., detecting inconsistency knowledge about a person locating in two places that are disconnected from each other).

8.2 Adopting Temporal Ontology

In addition to the spatial ontology, the DAML community is also developing temporal ontology for expressing temporal aspects of the contents of web resources and for expressing time-related properties of web services [Hobbs, 2002]. In this ontology, interval algebra is used to define temporal relationship axioms (after, before, inside, time-between, proper-interval, etc.) and representations for clock and calendar units (i.e., year, month, day of week, etc.).

In a short term, we plan to investigate the use of interval algebra for reasoning over the temporal relationships between different context events. For example, the relation between the at-time(\(e, t\)) predicate and the during(\(e, T\)) predicate can be used determine if a person is attending a meeting at a given time interval. In an intelligent meeting room, RFID sensors periodically report the presence of a person and describe this information using the at-time predicate e.g., at 12:57 PM, they report at-time(locatedIn(harry, room201), clock_time("12:57PM")) and at 1:33 PM, they report at-time(locatedIn(harry, room201), clock_time("1:33PM")). From this knowledge, using the interval algebra, the context broker can conclude during(locatedIn(harry, room201), time_interval("1:00PM-1:30PM"))

9 Conclusion

Ontologies are key requirements for building context-aware pervasive computing systems. In this paper, we have described the use of the OWL language and other tools for building an ontology foundation in CoBrA, a new pervasive context-aware architecture. With an explicit representation of context ontologies, CoBrA will allow independently developed devices and agents to interoperate and to help them to share and reason about contexts. As a part of our long term research plan, we are prototyping an intelligent context broker. Our goal is to create and deploy a pervasive context-aware meeting room in the newly constructed Information Technology and Engineering Building on the UMBC main campus.

References


http://www.daml.org/listarchive/daml-spatial/

http://www.umbc.edu/ITE/ITE.html


A Target-Centric Ontology for Intrusion Detection

John Pinkston, Jeffrey Undercoffer, Anupam Joshi and Timothy Finin
University of Maryland, Baltimore County
Department of Computer Science and Electrical Engineering
1000 Hilltop Circle, Baltimore, MD 21250
{pinkston, undercoffer, joshi, finin} @umbc.edu

Abstract

We have produced an ontology specifying a model of computer attacks. Our ontology is based upon an analysis of over 4,000 classes of computer intrusions and their corresponding attack strategies and is categorized according to: system component targeted, means of attack, consequence of attack and location of attacker. We argue that any taxonomic characteristics used to define a computer attack be limited in scope to those features that are observable and measurable at the target of the attack. We present our model as a target-centric ontology that is to be refined and expanded over time. We state the benefits of forgoing dependence upon taxonomies, in favor of ontologies, for the classification of computer attacks and intrusions. We have specified our ontology using DAML+OIL and have prototyped it using DAMLJessKB. We present our model as a target-centric ontology and illustrate the benefits of utilizing an ontology in lieu of a taxonomy, by presenting a use case scenario of a distributed intrusion detection system.

1 Introduction

Based upon empirical evidence we have produced a model of computer attacks categorized by: the system component targeted, the means and consequence of attack, and the location of the attacker. Our model is represented as a target-centric ontology, where the structural properties of the classification scheme is in terms of features that are observable and measurable by the target of the attack or some software system acting on the target’s behalf. In turn, this ontology is used to facilitate the reasoning process of detecting and mitigating computer intrusions.

Traditionally, the characterization and classification of computer attacks and other intrusive behaviors have been limited to taxonomies. Taxonomies, however, lack the necessary and essential constructs needed to enable an intrusion detection system (IDS) to reason over an instance that is representative of the domain of a computer attack. Alternatively, ontologies provide powerful constructs that include machine interpretable definitions of the concepts within a domain and the relations between them. Ontologies, therefore, provide software systems with the ability to share a common understanding of the information at issue, in turn empowering the software system with a greater ability to reason over and analyze this information.

As detailed by Allen, et al. [2], and McHugh [22], the taxonomic characterization of intrusive behavior has typically been from the attacker’s point of view, each suggesting that alternative taxonomies need to be developed. Allen et al., state that intrusion detection is an immature discipline and has yet to establish a commonly accepted framework. McHugh suggests classifying attacks according to protocol layer or, as an alternative, whether or not a completed protocol handshake is required. Likewise, Guha [10] suggests an analysis of each layer of the TCP/IP protocol stack to serve as the foundation for an attack taxonomy.

As an alternative to a taxonomy, we propose a data model implemented with an ontology representation language such as the Resource Description Framework Schema (RDFS) [26] or the DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) [1]. We illustrate the benefits of using ontologies by presenting an implementation of our ontology being utilized by a distributed intrusion detection system. Accordingly, we have specified our target-centric ontology in DAML+OIL and have implemented it using DAMLJessKB [17], an extension to the Java Expert System Shell [7].

Because IDS’s are either adjacent to or co-located with the target of an attack it is imperative that any classification scheme used to represent an attack be target-centric, where each taxonomic character is comprised of properties and features that are observable by the target of the attack. Consequently, our ontology only defines properties and attributes that are observable and measurable by the target of an attack. As a basis for establishing our a posteriori target-centric attack ontology, we evaluated and analyzed over 4,000 computer vulnerabilities and the corresponding attack strategies employed to exploit them.

The remainder of this paper is organized as follows: Section 2 presents related work in the form of alternative attack taxonomies as well as presenting related work in the area of ontologies for intrusion detection. Section 3 presents the characteristics of a sufficient taxonomy. Section 4 details the motivation for abandoning taxonomies in favor of ontologies. Our target-centric attack taxonomy is presented in Section 5. Section 6 details our implementation and Section 7 provides an example scenario illustrating the utility of the ontology within a distributed intrusion detection system. We conclude with Section 8.

2 Related Work

As previously stated, most of the existing research in the area of the classification of computer attacks is limited to taxonomies.
Because a taxonomy is contained within an ontology we address the research in the area of defining intrusion taxonomies before we address ontologies. Accordingly, this section is subdivided, with Subsection 2.1 presenting related work in the area of taxonomies for intrusion detection and Subsection 2.2 presenting related work in the area of ontologies for intrusion detection.

2.1 Related Work: Taxonomies

There are numerous attack taxonomies proposed for use in intrusion detection research. In [19] Landwehr et al., present a taxonomy categorized according to genesis (how), time of introduction (when) and location (where). They include sub-categories of: validation errors, boundary condition errors and serialization errors, which we incorporate into our ontology as the means of an attack.

During the 1998 and 1999 DARPA Off Line Intrusion Detection System Evaluations [12] [21] [15] Weber provided a taxonomy defining the categories of consequence, to include Denial of Service, Remote to Local and User to Root, which we incorporate into our work.

Lindqvist and Jonsson [20] state that they “focus on the external observations of attacks and breaches which the system owner can make”. Our effort is consistent with their focus.

2.2 Related Work: Ontologies

There is little, if any, published research formally defining ontologies for use in Intrusion Detection.

Raskin et al. [25], introduce and advocate the use of ontologies for information security. In arguing the case for using ontologies, they state that an ontology organizes and systematizes all of the phenomena (intrusive behavior) at any level of detail, consequently reducing a large diversity of items to a smaller list of properties.

In commenting on the IETF’s IDMEF, Kemmerer and Vigna [14] state “it is a but a first step, however additional effort is needed to provide a common ontology that lets IDS sensors agree on what they observe”.

3 Characteristics of a Sufficient Taxonomy

At this point, a clear understanding of the definition, purpose and objective of a taxonomy is in order. Accordingly, a taxonomy is a classification system where the classification scheme conforms to a systematic arrangement into groups or categories according to established criteria [31]. Glass and Vessey [9] contend that taxonomies provide a set of unifying constructs so that the area of interest can be systemically described and aspects of relevance may be interpreted. The overarching goal of any taxonomy, therefore, is to supply some predictive value during the analysis of an unknown specimen, while the classifications within the taxonomy offer an explanatory value.

According to Simpson [27] classifications may be created either a priori or a posteriori. An a priori classification is created non-empirically whereas an a posteriori classification is created by empirical evidence derived from some data set. Simpson defines a taxonomic character as a feature, attribute or characteristic that is divisible into at least two contrasting states and used for constructing classifications. He further states that taxonomic characters should be observable from the object in question.

Amoroso [3], Lindqvist, et al. [20] and Krusl [18] each have identified what they believe to be the requisite properties of a sufficient and acceptable taxonomy for computer security. Collectively, they have identified the following properties as essential to a taxonomy: Mutually Exclusive, Exhaustive, Unambiguous, Repeatable, Accepted, Useful, Comprehensible, Conforming, Objective, Deterministic and Specific. Accordingly, as an ontology subsumes a taxonomy these characteristics form the underpinnings of our work.

4 From Taxonomies to Ontologies: The case for ontologies in Intrusion Detection

Ning et al. [23], propose a hierarchical model for attack specification and event abstraction using three concepts essential to their approach: System View, Misuse Signature and View Definition. Their model is based upon a thorough examination of attack characteristics and attributes, and is encoded within the logic of their proposed system. Consequently, this model is not readily interchangeable and reusable by other systems.

The Intrusion Detection Working Group of Internet Engineering Task Force (IETF) has proposed the Intrusion Detection Message Exchange Requirements [33] which, in addition to defining the requirements for the Intrusion Detection Message Exchange Format, also specifies the architecture of an IDS. The Intrusion Detection Message Exchange Format Data Model (IDMEF) and accompanying Extensible Markup Language Document Type Definition [4] is a profound effort to establish an industry wide data model which defines computer intrusions. The IDMEF, however, has its shortcomings. Specifically, it uses XML which is limited to a syntactic representation of the data model and does not convey the semantics, relationships, attributes and characteristics of the objects which it represents. This limitation requires that each individual IDS interpret and implement the data model programmatically.

According to Davis et al. [5], knowledge representation is a surrogate or substitute for an object under study. In turn, the surrogate enables an entity, such as a software system, to reason about the object. Knowledge representation is also a set of ontological commitments specifying the terms that describe the essence of the object. In other words, meta-data or data about data describing their relationships.

Frame Based Systems are an important thread in knowledge representation. According to Koller et al. [16], Frame Based Systems provide an excellent representation for the organizational structure of complex domains. Frame Based Languages, which support Frame Based Systems, include RDF, and are used to represent ontologies. Accordingly to Welty et al. [32], an ontology, at its deepest level, subsumes a taxonomy. Similarly, Noy and McGuinness [24] state the process of developing an ontology includes arranging classes in a taxonomic hierarchy.

In applying ontologies to the problem of intrusion detection, the power and utility of the ontology is not realized by the simple representation of the attributes of the attack. Instead, the power and utility of the ontology is realized by the fact that we can express the relationships between collected data and use those relationships to deduce that the particular data represents an attack of a particular type. Moreover, specifying an ontological representation decouples the data model defining an intrusion from the logic of the intrusion detection system. The decoupling of the data model from the IDS logic enables non-homogeneous IDS’s to share data without a prior agreement as to the semantics of the data. To effect this sharing, an instance of the ontology is shared between IDS’s in the form of a set of DAML+OIL (or RDF) statements. If the re-
the ontology in order to interpret and use the data as intended by its originator.

Ontologies therefore, unlike taxonomies, provide powerful constructs that include machine interpretable definitions of the concepts within a specific domain and the relations between them. In our case the domain is that of a particular computer or a software system acting on the computer’s behalf in order to detect attacks and intrusions. Ontologies may be utilized to not only provide an IDS with the ability to share a common understanding of the information at issue but also further enable the IDS with improved capacity to reason over and analyze instances of data representing an intrusion. Moreover, within an ontology, characteristics such as cardinality, range and exclusion may be specified and the notion of inheritance is supported.

5 Target-Centric Ontology Attributes of the Class Intrusion

In constructing our ontology, we relied upon an empirical analysis [30] of the features and attributes, and their interrelationships, of over 4,000 classes of computer attacks and intrusions. Figure 1, presents a high level view of our ontology. The attributes of each class and subclass (denoted by ellipses) are not shown because it would make the illustration unwieldy.

At the top most level we define the class Host. Host has the properties Current State which is defined by the class System Component and Victim of which is defined by the class Attack. As defined in Section 4 the property, also called the predicate, defines the relationship between a subject and an object.

The System Component class is comprised of the following subclasses:

1. Network. This class is inclusive of the network layers of the protocol stack. We have focused on TCP/IP therefore we only consider IP, TCP, and UDP subclasses. For example, and as will be later demonstrated, the TCP subclass includes the properties TCP_MAX which defines the maximum number of TCP connections, WAIT_STATE defining the number of connections waiting on the final ack of the three-way handshake to establish a TCP connection, THRESHOLD specifying the allowable ratio between maximum connections and partially established connections and EXCEED_T a boolean value indicating that the allowable ratio has been exceeded. It should be noted that these are only four of several network properties.

2. System. This includes attributes representing the operating system of the host. It includes attributes representing overall memory usage (MEM_TOTAL, MEM_FREE, MEM_SWAP) and CPU usage (LOAD_AVG). The class also contains attributes reflective of the number of current users, disk usage, the number of installed kernel modules, and change in state of the interrupt descriptor and system call tables.

3. Process. This class contains attributes representing particular processes that are to be monitored. These attributes include the current value of the instruction pointer (INS_P), the current top of the stack (T_STACK), a scalar value computed from the stream of system calls (CALL_V), and the number of child processes (N_CHILD).

The class Attack has the properties Directed to, Effected by, and Resulting in. This construction is predicated upon the notion that an attack consists of some input which is directed to some system component and results in some consequence. Accordingly, the classes System Component, Input, and Consequence are the corresponding objects. The class Consequence is comprised of several subclasses which include:

1. Denial of Service. The attack results in a Denial of Service to the users of the system. The denial of service may be because the system was placed into an unstable state or all of the system resources may be consumed by meaningless functions.

2. User Access. The attack results in the attacker having access to services on the target system at an unprivileged level.

3. Root Access. The attack results in the attacker being granted privileged access to the system, consequently having complete control of the system.

4. Probe. This type of an attack is the result of scanning or other activity wherein a profile of the system is disclosed.

Finally, the class Input has the the attributes Received from and Causing where Causing defines the relationship between the Means of attack and some input. We define the following subclasses for Means of attack:

1. Input Validation Error. An input validation error exists if some malformed input is received by a hardware or software component and is not properly bounded or checked. This class is further sub-classed as:

   (a) Buffer Overflow. The classic buffer overflow results from an overflow of a static-sized data structure.

   (b) Boundary Condition Error. A process attempts to read or write beyond a valid address boundary or a system resource is exhausted.

   (c) Malformed Input. A process accepts syntactically incorrect input, extraneous input fields, or the process lacks the ability to handle field-value correlation errors.

2. Logic Exploits. Logic exploits are exploited software and hardware vulnerabilities such as race conditions or undefined states that lead to performance degradation and/or system compromise. Logic exploits are further subclassed as follows:

   (a) Exception Condition. An error resulting from the failure to handle an exception condition generated by a functional module or device.

   (b) Race Condition. An error occurring during a timing window between two operations.

   (c) Serialization Error. An error that results from the improper serialization of operations.

   (d) Atomicity Error. An error occurring when a partially-modified data structure is used by another process; An error occurring because some process terminated with partially modified data where the modification should have been atomic.

6 Implementation

There are several reasoning systems that are compatible with DAML+OIL. According to their functionality, reasoning systems can be classified into two types, backward-chaining...

We have prototyped the logic portion of our system using the DAMLJessKB [17] reasoning system, an extension to the Java Expert System Shell (JESS) [7]. JESS is a Java implementation of the C Language Integrated Production System (CLIPS) [8]. DAMLJessKB is employed to reason over instances of our data model that are considered to be suspicious. These suspicious instances are constrained according to our target-centric ontology and asserted into the knowledge base.

Upon initialization of DAMLJessKB we converted the DAML+OIL statements representing the ontology into N-Triples and assert them into a knowledge base as rules. The assertions are of the form:

\[
\text{assert (PropertyValue (predicate) (subject) (object))}
\]

Once asserted, DAMLJessKB generates additional rules which include all of the chains of implication derived from the ontology.

The following series of figures illustrate the DAML+OIL encoding of selected classes, subclasses and their respective properties, of our ontology.

Figure 2 lists the DAML+OIL statements defining the class Attack and its properties Directed To, Resulting In and Effected By. These properties correspond to the edges between the node labeled Target and the nodes labeled System Component, Input and Consequence respectively, in Figure 1.

![Figure 2: DAML+OIL Statements Defining the Class Attack and its Properties: Directed To, Resulting In and Effected By](image-url)

Figure 3 presents the DAML+OIL notation for the class System Component, its subclass Network, and Network’s subclass TCP. Figure 4 lists the DAML+OIL notation for some of the attributes of the class TCP.

Figure 5 details the specification of the class Consequence while Figures 6 and 7 show similar details for the specification of the classes Denial of Service and Syn Flood. The Syn Flood class, which is not shown in Figure 1 illustrating our ontology, is a subclass of both Denial of Service and TCP and, as stated...
in the DAML+OIL notation, will only be instantiated when the threshold of pending TCP connections is exceeded.

6.1 Querying the Knowledge Base

Once the ontology is asserted into the knowledge base and all of the derived rules resulting from the chains of implication are generated, the knowledge base is ready to receive instances of the ontology. Instances are asserted and de-asserted into/from the knowledge base as temporal events dictate. To query the knowledge base for the existence of an attack or intrusion, the query could be so granular that it requests an attack of a specific type, such as a Syn Flood:

```daml
(defrule isSynFlood
  (PropertyValue
    (p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
    (s ?var)
    (o http://security.umbc.edu/IntrOnt#SynFlood))
  =>
  (printout t "A SynFlood attack has occurred."
    " with event number: " ?var))
```

The query could be of a medium level of granularity, asking for all attacks of a specific class, such as denial of service. Accordingly, the following query will return all instances of an attack of the class Denial of Service.

```daml
(defrule isDOS
  (PropertyValue
    (p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
    (s ?var)
    (o http://security.umbc.edu/IntrOnt#DoS))
  =>
  (printout t "A DoS attack has occurred."
    " with ID number: " ?var))
```

Finally, the following rule will return instances of any attack, where the event numbers that are returned by the query need to be iterated over in order to discern the specific type of attack:

```daml
(defrule isConseq
  (PropertyValue
    (p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
    (s ?var)
    (o http://security.umbc.edu/IntrOnt#Conseq))
  =>
  (printout t "An attack has occurred."
    " with ID number: " ?var))
```

These varying levels of granularity are possible because of DAML+OIL’s notion of classes, subclasses, and the relationships that holds between them. The variable ?var, contained in each of the queries, is instantiated with the subject whenever a predicate and object from a matching triple is located in the knowledge base.

7 Using the Ontology to Detect a Distributed Attack

The following example of a distributed attack illustrates the utility of our ontology.
The Mitnick attack is multi-phased; consisting of a Denial of Service attack, TCP sequence number prediction and IP spoofing. When this attack first occurred a Syn Flood was used to effect the denial of service, however any denial of service attack would have sufficed.

In the following example, which is illustrated in figure 8, Host B is the ultimate target and Host A is trusted by Host B.

The attack is structured as follows:

1. The attacker initiates a Syn/Flood attack against Host A to prevent Host A from responding to Host B.
2. The attacker sends multiple TCP packets to the target, Host B in order to be able to predict the values of TCP sequence numbers generated by Host B.
3. The attacker then pretends to be Host A, by spoofing Host A’s IP address, and sends a Syn packet to Host B in order to establish a TCP session between Host A and Host B.
4. Host B responds with a SYN/ACK to Host A. The attacker does not see this packet. Host A, since its input queue is full due to number of half open connections caused by the Syn/Flood attack, cannot send a RST message to Host B in response to the spurious Syn message.
5. Using the calculated TCP sequence number of Host B (recall that the attacker did not see the Syn/ACK message sent from Host B to Host A) the attacker sends an Ack with the predicted TCP sequence number packet in response to the Syn/Ack packet sent to Host A.
6. Host B is now in a state where it believes that a TCP session has been established with a trusted host Host A. The attacker now has a one way session with the target, Host B, and can issue commands to the target.

It should be noted that an intrusion detection system running exclusively at either host will not detect this multi-phased and distributed attack. At best, Host A’s IDS would see a relatively short lived Syn Flood attack, and Host B’s IDS might observe an attempt to infer TCP sequence numbers, although this may not stand out from other non-intrusive but ill-formed TCP connection attempts.

The following explains the utility of our ontology, as well as the importance of forming coalitions of IDSs. In our IDS model, we form coalitions of IDS services each of which is responsible for specific parts of an enterprise or domain. For example, one IDS service may be responsible for a specific host, while another is responsible for a group of hosts, while yet another is responsible for monitoring network traffic. The IDS’s all share a common ontology and utilize a secure communications infrastructure that has been optimized for IDS’s.

We present such an infrastructure in [13, 28, 29].

Consider the case of the instance of the Syn Flood attack presented in Section 6 and that it was directed against Host A in our example scenario. As the IDS responsible for Host A is continually monitoring for anomalous behavior, asserting and de-asserting data as necessary, it will detect the occurrence of an inordinate number of partially established TCP connections, and transmit the instance of the Syn Flood to the other IDS’s in its coalition.

That instance is repeated below:

```
<IntrOnt:Intrusion rdf:about="&IntrOnt;00035">
  <IntrOnt:IP_Address="130.85.112.231">  
    <IntrOnt:Resulting_In rdfs:label="DoS"/>
    <IntrOnt:Exceed_T="true"/>
    <IntrOnt:int_time="20021212 154312"/>
  </IntrOnt:IP_Address>
</IntrOnt:Intrusion>

<IntrOnt:Syn_Flood rdf:about="&IntrOnt;00038">
  <IntrOnt:Exceed_T="true"/>
  <IntrOnt:int_time="20021212 154312"/>
</IntrOnt:Syn_Flood>
```

This instance is converted into a set of N-Triples and asserted into the knowledge base of each IDS in the coalition. Those same N-Triples will be de-asserted when the responsible IDS transmits a message stating that the particular host is no longer the victim of a Syn Flood attack.

In order to detect an Mitnick type attack, we include the following DAML+OIL statements describing connections:

```
<IntrOnt:Connection rdf:about="&IntrOnt;00038"/>
<IntrOnt:IP_Address="130.85.112.231"/>
<IntrOnt:conn_time="20021212 154417"/>

<IntrOnt:Connection rdf:about="&IntrOnt;00101"/>
<IntrOnt:IP_Address="202.85.191.121"/>
<IntrOnt:conn_time="20021212 151221"/>

<IntrOnt:Connection rdf:about="&IntrOnt;00102"/>
<IntrOnt:IP_Address="68.54.101.78"/>
<IntrOnt:conn_time="20021212 150152"/>
```

In order to detect a Mitnick type attack, we include the following DAML+OIL statements that partially specify an ontology of the Mitnick attack (the class is identified as P_Mitnick for partial):

```
<daml:Class rdf:about="&Intrusion;P_Mitnick">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="&Intront;DoS"/>
    <daml:Class rdf:about="&Intront;Connection"/>
  </daml:intersectionOf>
</daml:Class>
```
The ontology is partial because the Mitnick attack has the additional property that the connection time with the victim must be greater than or equal to the time of the denial of service attack. An instance of this ontology will be instantiated provided that there exists an instance of a denial of service attack that has the same unique identifier as that of an established connection. In fact there will be an instance created in each case where this condition holds. In our prototype, we check each instance to determine if the time of the connection is greater than or equal to the time of the attack.

The following rules are used to check each instance:

(defrule isMitnick

(PropertyValue
  (p http://security.umbc.edu/IntrOnt#Mitnick )
  (s ?eventNumber) (o "true"))

(PropertyValue
  (p http://security.umbc.edu/IntrOnt#Int_Time)
  (s ?eventNumber) (o ?Int_Time))

(PropertyValue
  (p http://security.umbc.edu/IntrOnt#Conn_Time)
  (s ?eventNumber) (o ?Conn_Time))

=>
(if (>= ?Conn_Time ?Int_Time) then
  (printout t "event number: " ?eventNumber " is a Mitnick Attack: crlf())))

this rule will fire and event number 00038, the instance of the intersection of the connection and the denial of service attack, will be displayed.

At this point it is important to review the sequence of events leading up to the discovery of the Mitnick attack. Recall, that the IDS responsible for the victim of the Syn Flood attack queried its knowledge base for an instance of a DoS the IDS responsible for the victim of the Syn Flood attack leading up to the discovery of the Mitnick attack. Recall, that

The ontology specifying the Mitnick class states that it is a precursor to a more insidious attack, instances of established and pending connections were asserted into the knowledge base. As the state of the knowledge base is dynamic due to the assertions and de-assertions, the rule set of each IDS is continually applied to the knowledge base.

The ontology specifying the Mitnick class states that it is the intersection of both the DoS and Connection classes. Because each IDS instantiates an instance when this constraints imposed by intersection is true, we need to examine each instance to ensure that Connection Time ≥ Intrusion Time.

8 Conclusion and Future Work

We have analyzed vulnerability and intrusion data derived from CERT advisories and NIST’s ICAT meta-base resulting in the identification of the components (network, kernel, application and other) most frequently attacked. We have also identified the most common means and consequences of the attack as well as the location of the attacker. Our analysis shows that non-kernel space (non operating system) applications, running as either root or user, are the most frequently attacked and are attacked remotely. The most common means of attack are exploits. According to the CERT advisories issued in response to severe vulnerabilities, root access is the most common consequence of an exploit whereas the ICAT data shows denial of service to be the most common consequence.

Our analysis was conducted in order to identify the observable and measurable properties of computer attacks and intrusions. Accordingly, we have developed a target-centric ontology characterized by System Component, Means of Attack, Consequences of Attack and Location of Attacker. We have stated the case for replacing simple taxonomies with ontologies for use in IDS’s and have presented an initial ontology specifying the class Intrusion. Our ontology is available at: http://security.cs.umbc.edu/Intrusion.

We have prototyped our ontology using the DAMLJessKB, which has some limitations. We intend to either modify DAMLJessKB in order to make it a full and complete reasoner or use Stanford’s Java Theorem Prover [6] or Rename ABox and Concept Expression Reasoner [11].

References


Ontologies in a Pervasive Computing Environment
Anand Ranganathan, Robert E. McGrath, Roy H. Campbell, M. Dennis Mickunas

Department of Computer Science
University of Illinois, Urbana-Champaign
rangana@uiuc.edu, mcgrath@cs.uiuc.edu, rhc@uiuc.edu, mickunas@cs.uiuc.edu

Abstract
Ontologies are entering widespread use in many areas such as knowledge and content management, electronic commerce and the Semantic Web. In this paper we show how the use of ontologies has helped us overcome some important problems in the development of pervasive computing environments. We have integrated ontologies and Semantic Web technology into our Pervasive Computing infrastructure. Our investigations have shown that the Semantic Web technology can be integrated into our CORBA-based infrastructure to augment several important services. This work suggests a number of requirements for future research in the development of ontologies, reasoners, languages and interfaces.

1 Introduction
There is growing evidence for the potential value of Semantic Web technology for Web Services and other open, distributed systems [Goble et al., 2002; Peer, 2002]. This paper presents a case study of the use of Semantic Web technology in a Pervasive (or Ubiquitous) Computing Environment, GAIA [Roman et al., 2002].

Pervasive (or Ubiquitous) Computing Environments are physical environments saturated with computing and communication, yet gracefully integrated with human users [Lyytinen et al., 2002]. These environments involve the construction of massively distributed computing systems that feature a large number of autonomous entities (or agents). These entities could be devices, applications, services, databases, users or other kinds of agents. Various types of middleware (based on CORBA, Java RMI, SOAP, etc.) have been developed that enable communication between different entities. However, existing middleware have no facilities to ease semantic interoperability between the different entities.

Ontologies have been widely used in many areas such as knowledge and content management, electronic commerce and the semantic web. In this paper we show how the use of ontologies has helped us overcome some of the challenges in constructing and managing a pervasive computing environment. Of course, these problems are not unique to pervasive computing, but are faced by any multi-agent software system. We believe that our solutions to some of these issues can be extended to any multi-agent system.

This work has considered three major issues that confront the development and deployment of Pervasive Computing Environments. These are:

- Discovery and Matchmaking
- Inter-operability between different entities
- Context-awareness

This section briefly describes these three tasks in the domain of a pervasive computing environment.

In the future, we will extend this work to augment the configuration and management of multiple Spaces, and to augment additional services, such as the Quality of Service infrastructure [Wichadakul et al., 2002].

1.1 Discovery and Matchmaking in a Pervasive Computing Environment

A Pervasive Computing Environment has one or more registries to keep a real time state of the system, i.e., the entities currently present and available; and should have a protocol for discovering the arrival and departure of mobile entities, for advertising current availability, and for notifying interested parties of changes. A registry with these protocols is termed a “Discovery Service” [McGrath, 2000]. Matchmaking [Trastour et al., 2001] uses the Discovery Service to discover not only what entities are available, but what sets or combinations meet certain criteria, i.e., the requirements and preferences of the parties.

In the Discovery Service, standard schemas are needed to describe many kinds of entities, including people, places, and things. Furthermore, the system has policies, constraints, and relationships which may need to be discovered as well. For a robust system, it is necessary to have a flexible mechanism for exchanging descriptive information of many kinds.
1.2 Inter-operability between different entities

New entities may enter the environment at any time; these new entities have to interact with existing entities. The interaction must be based on common, well-defined concepts, so that there is no misunderstanding between the entities. The entities must have a common understanding of the various terms and concepts used in the interaction.

For autonomous entities to interact with one another, they need to know, beforehand, what kinds of interfaces they support and what protocols or commands they understand. In a truly distributed scenario, such as a pervasive computing environment, it may not be reasonable to assume that such agreement exists.

Similar mechanisms are needed for humans to interact with different entities. Humans need to understand what various entities do, and they need to understand the relationships between such entities. It is essential for humans to form an accurate conceptual model of the environment so that they can interact with the environment easily.

1.3 Context-Awareness

Applications in pervasive and mobile environments need to be context-aware so that they can adapt themselves to rapidly changing situations. Applications in pervasive environments use different kinds of contexts (such as location of people, activities of individuals or groups, weather information, etc.).

The various types of contextual information that can be used in the environment must be well-defined so that different entities have a common understanding of context. Also, there needs to be mechanisms for humans to specify how different applications and services should behave in different contexts. These mechanisms need to be based on well-defined structures of different types of context information.

1.4 Ontologies in Pervasive Computing Environments

In order to tackle the problems described above, we apply the technologies from the emerging “Semantic Web” [Berners-Lee et al., 2001; W3C, 2003a]. While the Semantic Web was designed to enhance Web search and agents, we show that it is well suited to some of the requirements of Pervasive Computing Environments.

We have incorporated the use of ontologies in our prototype pervasive computing environment, GAIA [Roman et al., 2002]. The ontologies are written in DAML+OIL [daml.org, 2003], describing various parts of the GAIA environment. An Ontology Server manages a composite ontology that describes the entities of the system and performs operations on the ontologies. This composite ontology is built by composing different ontologies specified in DAML+OIL files. The ontologies are validated into a Knowledge Base (KB), built on the CORBA FaCT Server [Bechhofer et al., 1999; Horrocks et al., 1999].

Ontologies are used for describing various concepts in the GAIA Pervasive Computing Environment. We have developed ontologies that describe the different kinds of entities and their properties. These ontologies define different kinds of applications, services, devices, users, data sources and other entities. They also describe various relationships between the different entities and establish axioms on the properties of these entities that must always be satisfied.

A second use of ontologies is to describe different types of contextual information in GAIA. The ontology defines standard descriptions for locations, activities, weather information, and other information that may be used by context-aware applications.

The ontologies that describe the pervasive environment greatly help in the smooth operation of the environment. Some of the ways in which we use ontologies in our pervasive environment are:

- Checking to see if the descriptions of different entities are consistent with the axioms defined in the ontology. This also helps ensuring that certain security and safety constraints are met by the environment
- Enabling semantic discovery of entities
- Allowing users to gain a better understanding of the environment and how different pieces relate to each other
- Allowing both humans and automated agents to perform searches on different components easily
- Allowing both humans and automated agents to interact with different entities easily (say, by sending them various commands)
- Allowing both humans and automated agents to specify rules for context-sensitive behavior of different entities easily
- Enabling new entities (which follow different ontologies) to interact with the system easily.

In the following sections, we describe how ontologies are used within our Pervasive Computing Environment, GAIA. We first introduce GAIA; we then describe how the Ontology Server fits into the GAIA framework. We then briefly describe the kinds of ontologies that have been developed and how they are used within GAIA. Finally, we evaluate our approach and suggest important areas for future research.

2. GAIA: A Pervasive Computing Environment

GAIA is an infrastructure for Smart Spaces, which are pervasive computing environments that encompass physical spaces [Roman et al., 2002]. GAIA converts physical spaces and the devices they contain into a pro-
grammable computing system. It offers services to manage and program a Space and its associated state. GAIA is similar to traditional operating systems in that it manages the tasks common to all applications built for physical spaces. Each Space is self-contained, but may interact with other Spaces. GAIA provides core services, including events, entity presence (devices, users and services), discovery and naming. By specifying well-defined interfaces to services, applications may be built in a generic way so that they are able to run in arbitrary Smart Spaces. The core services are started through a bootstrap protocol that starts the GAIA infrastructure. GAIA uses CORBA to enable distributed entities to communicate with one another. GAIA has served as our test-bed for the use of ontologies in Pervasive Computing Environments.

We have used GAIA to manage rooms in our Computer Science building. GAIA helps make these rooms smart and responsive to the needs of different users. There are a wide variety of devices that exist in these rooms. These include authentication devices like fingerprint sensors and smart card readers, display devices like large plasma screens, video walls, handheld devices, wearable devices like smart watches and smart rings, various input devices such as touch screens and microphones, etc. In addition, there are large number of applications and services like music-playing applications, presentation applications and drawing applications. Ontologies provide a very easy way to manage this diversity in our environments.

Unlike many uses of ontologies, we have tightly integrated Semantic Web services into the infrastructure. We implemented an Ontology Server, which is a standard system service that provides a generic interface to manage DAML+OIL ontologies, a Knowledge Base, and logic queries. Any entity in the system can use the Ontology Server.

We have created ontologies (in DAML+OIL) to classify and describe many concepts of the Pervasive Computing Environment. The ontologies include entities of the system (not limited to software) and context information.

The ontologies and Ontology Server are used to augment system services, including:

- Configuration management
- Discovery and matchmaking
- Human Interfaces
- Interoperation of components
- Context Sensitive behavior

The ontologies and Knowledge Base are tightly integrated into the whole system.

2.1 The Ontology Infrastructure in Gaia

We have integrated the use of ontologies in our smart Spaces framework, GAIA. (Figure 1) All the ontologies in GAIA are maintained by an Ontology Server [McGrath et al., 2003]. The Ontology Server asserts the concepts described in the ontologies in the CORBA FaCT Reasoning Engine [Bechhofer et al., 1999; Horrocks et al., 1999] to make sure that they are logically consistent and for answering logical queries. Other entities in GAIA contact the Ontology Server to retrieve descriptions of entities in the environment, meta-information about context or definitions of various terms used in GAIA. It is also possible to support semantic queries (for instance, classification of individuals or subsumption of concepts). Such semantic queries are resolved using the FaCT Reasoning Engine. The Ontology Server registers with the CORBA Naming Service so that it can be discovered by other entities in the environment.

One of the key benefits in using ontologies is that they aid interaction between users and the environment since they concisely describe the properties of the environment and the various concepts used in the environment. With that aim in mind, we have developed an Ontology Explorer, which is a graphical user interface that allows users to browse and search the ontologies in the Space. The Ontology Explorer also allows users to interact with other entities in the Space through it. This interaction with other entities is governed by their properties as defined in the ontology. The Ontology Explorer and Ontology Server are described in more detail in [McGrath et al., 2003].

2.2. Kinds of Ontologies in Gaia

We use ontologies to describe various parts of our pervasive environment, GAIA. In particular, we have ontologies that have meta-data about the different kinds of entities in our environment. We also have ontologies to describe the different kinds of contextual information in our environment. The ontologies used in GAIA are described in more detail in [McGrath et al., 2003].

2.2.1. Ontologies for different entities

Pervasive computing environments have a large number of different types of entities. There are different kinds of devices ranging from small wearable devices and handhelds to large wall displays and powerful servers. There are many services that help in the functioning of the environment. There are different kinds of applications such as music players, slide show viewers, drawing applications, etc. Finally, there are the users of the environment who have different roles (student, administrator, etc.).

Ontologies help formalize and make available the informal and implicit taxonomy of the different kinds of entities in the system. We have developed ontologies that define the different kinds of entities, provide meta-data about them and describe how they relate to each other. These ontologies are written in DAML+OIL.

A Pervasive Computing Environment is very dynamic; new kinds of entities can be added to the environment at any time. The Ontology Server allows adding new classes and properties to the existing ontologies at any time, by merging new concepts into the system ontology. To do this, a new ontology describing the new entities is first
developed. The new ontology is then added to the shared ontology using bridge concepts that relate classes and properties in the new ontology to existing classes and properties in the shared ontology. These bridge concepts are typically subsumption relations that define the new entity to be a subclass of an existing class of entities. For example, if a new kind of fingerprint recognizer is added to the system, the bridge concept may state that it is a subclass of "AuthenticationDevices". These bridge concepts are written by the developer of the ontology manually. We have no way at present of automatically generating these bridge concepts, although that would be very useful.

2.2.2. Ontologies for context information

GAIA has a context infrastructure [Ranganathan et al., 2003] that enables applications to obtain and use different kinds of contexts. This infrastructure consists of sensors that sense various contexts, reasoners that infer new context information from sensed data and applications that make use of context to adapt the way they behave. We use ontologies to describe context information. This ensures that the different entities that use context have a common semantic understanding of contextual information.

There are different types of contexts that can be used by applications. These include physical contexts (location and time), environmental contexts (weather, light and sound levels), informational contexts (stock quotes, sports scores), personal contexts (health, mood, schedule, activity), social contexts (group activity, social relationships, whom one is in a room with), application contexts (email, websites visited) and system contexts (network traffic, status of printers). We represent contexts as predicates. We follow a convention where the name of the predicate is the type of context that is being described (like location, temperature or time). An example of a context predicate is Location(Chris, in, Room 2401). This predicate is true if Chris is indeed in Room 2401. Ontologies essentially define the vocabulary and types of arguments that may be used in the predicates.

The use of ontologies to describe context information is helpful in checking the validity of context information. It also makes it easier to specify the behavior of context-aware applications since we know the types of contexts that are available and their structure. We can thus easily construct rules governing application behavior using these well-defined context predicates.

2.3. Uses of Ontologies in a Pervasive Computing Environment

The Ontology Server can be used by any application, component, or service in the GAIA environment. The ontologies that describe entities and context information are used to enable different parts of the pervasive environment interact with each other easily. The ways in which ontologies are used in GAIA are described in more detail in [McGrath et al., 2003].

2.3.1. Configuration Management

A pervasive computing environment is very dynamic, the configuration must change as activities change, and as people and devices enter and leave. Configuration management is very challenging, especially because:

- New entities, never before seen, may enter
- Components need to automatically discover and collaborate with other components
Without ontologies, the GAIA environment is configured with scripts and ad hoc configuration files [Roman et al., 2002]. Ontologies can replace these mechanisms with a standard, formal XML language.

Each entity is associated with an XML file that describes its properties. When a new entity is introduced into the system, its description is checked against the existing ontology to see whether it is satisfiable. If the description is not consistent with the concepts described in the ontology, then either the description is faulty (in which case the owner of the entity/context has to develop a correct description of the entity/context), or there are safety or security issues with the new entity or context.

For example, the ontology may dictate that all electrical and electronic devices that are to be introduced in an environment (like a smart room) must accept 110V AC power. In that case, if somebody tries to install a new TV that is made for Europe and only takes 220V power, then the description of the new TV would be inconsistent with the ontology and a safety warning may be generated.

Formal ontologies also increase the capability to use descriptions from different, autonomous sources. The DAML+OIL ontologies can be published, to enable autonomous developers and service providers to describe their products with the correct vocabulary. Conversely, autonomous entities can specify the correct formal vocabulary to be used to interpret their descriptions by referring to the relevant DAML+OIL ontology. These actions require more than the URL: the formal semantics defined for DAML+OIL ensures that ontologies from different sources can be used together.

2.3.2. Semantic Discovery and Matchmaking

In our environment, the Ontology Server performs the tasks of semantic discovery and matchmaking. It poses logical queries involving subsumption and classification of concepts to the FaCT Server [Bechhofer et al., 1999; Horrocks et al., 1999], which has knowledge of all concepts used in the environment. Such queries are useful in finding appropriate matches. Other entities in the environment query the Ontology Server to discover classes of components that meet their requirements.

2.3.3. Improved Human Interfaces

Ontologies can be used to make better user interfaces and allow these environments to interact with humans in a more intelligent way. Ontologies describe different parts of the system, the various terms used and how various parts interact with each other. All classes and properties in the ontology also have documentation that describes them in greater detail in user-understandable language. Ontologies enable semantic interoperability between users and the system.

For example, we have defined the term “meeting” as a subclass of “GroupActivity”. A meeting is defined to have a location, a time, an agenda (optional) and a set of participants. It also has the following human-understandable comment: “A meeting is an activity that is performed by a group of people. A meeting involves different people coming together at a particular time or place with a common purpose in mind”.

Thus, both humans and automated entities in the environment can get a clear understanding of the term “meeting” by looking it up in the ontology.

We have developed a GUI called the Ontology Explorer that allows users to browse the ontology describing the environment. A user can search for different classes in the ontology. He can then browse the results — for example, he can get documentation about the classes returned, get properties of the class, etc.. The Ontology Explorer is similar to a class browser, but it may browse information about all concepts in the system (like context information, applications, services, terms), not just the software objects.

2.3.4. Improved Inter-operability between entities

The description of the properties of different classes of entities thus allows both users and other automated agents to interact with them more easily by performing searches on them or sending them various commands. This has proved to be one of the major advantages to using ontologies in a pervasive computing environment since it helps simplify the user’s and the agent’s interaction with such complex systems.

Entities that support searches have their schemas described in the ontology. The ontology also specifies which fields in the query are required and which are optional. Thus any other entity (including users) can browse the ontology to learn the schema and query formats supported by the searchable entity. They can then frame their query and get the results. For example, we have an MP3 Server that exposes a query interface for searching for MP3 files. The schema for querying contains fields like artist name, length of song, etc. This schema is described in the ontology. Other entities, thus, know how to query the MP3 Server.

The same idea is used to let entities interact with one another i.e. by sending commands. Different entities allow different types of actions to be performed on them. For example, the MP3 Server described above allows different commands to be sent to it — start, stop, pause, change volume, etc. In our framework, entities specify the commands they support and the parameters of these commands in an ontology. Other entities can thus send commands to these entities.

2.3.5 Context-Sensitive Behavior

An ontology can improve the robustness and portability of context-aware applications. It is not possible to design in all possible contexts — or even to know what contexts may be used. Ontologies for context information are an important mechanism for adapting to environments. The application specifies rules for context-sensitive behavior
using a specific set of context concepts and events (a vocabulary). When the application moves to a new space, the context may be different. This might be due to different sensors, different versions of services, or localizations. If the differences are terminological, an ontology may allow the rules to be “translated” and then work correctly in the new environment.

Context-aware applications in GAIA have rules that describe what actions should be taken in different contexts. In order to write such a rule, an application developer must know the different kinds of contexts available as well as possible actions that can be taken by the application. We have ontologies that describe the different kinds of context information – location, time, temperature, activities of people; and also different applications and what commands can be sent to them.

These ontologies greatly simplify the task of writing rules. We have developed a tool that uses these ontologies to allow a developer to write rules easily. The tool allows him to construct conditions out of the various possible types of contexts available. It then allows him to choose the action to be performed at these contexts from the list of possible commands that can be sent to this application as described in the ontology. Developers can, thus, very quickly, impart context-sensitive behavior to applications by associating context expressions (involving context predicates) with actions. An example of such a rule for a context-sensitive application is:

\[
\text{IF Location(Manuel, Entering, Room 2401) AND Time(morning) THEN play a rock song.}
\]

3. Lessons Learned

We have integrated ontologies and Semantic Web technology into our Pervasive Computing infrastructure. This work suggests a number of requirements for future research and development of ontologies.

Our investigations have shown that the Semantic Web technology can be used with CORBA-based infrastructure to solve some important problems for a pervasive computing environment. Our Ontology Server provides a standard interface to a Knowledge Base and logic engine. Ontologies for descriptions of entities and relationships are developed within a Knowledge Engineering Environment and stored as DAML+OIL XML files. Components of the system use the CORBA-based infrastructure to update and query the Ontology Server. In the future, we will extend this work to augment the configuration and management of multiple Spaces, and to augment additional services, such as Quality of Service infrastructure [Wichadakul et al., 2002].

Our system has integrated semantic services to an unprecedented degree. This was enabled partly by the availability of the CORBA FaCT server [Bechhofer et al., 1999] and the Java classes from OILed [Bechhofer et al., 2001; OilEd, 2002]. These packages are a model for what is needed in future software standards:

- A standard API for DAML+OIL (or, more likely, OWL [W3C, 2003b])
- A standard interface for generic Knowledge Base services

Alternative logic engines and Knowledge Bases should be wrapped in a generic interface, so they can be plugged into infrastructure services. For example, the Open Knowledge Base Connectivity (OKBC) [Chaudhri et al., 1998] could be extended to support DAML (OWL). The Java Theorem Prover (JTP) [Fikes et al., 2003] is a promising step in this direction.

Ontologies and semantic services will play a key role as we develop more sophisticated tools to construct and manage multiple Spaces. It will be important to simplify the construction and maintenance of ontologies, perhaps with a repository of standard ontologies. It will also be important to integrate ontologies with the software generation and management, perhaps using ontologies to semi-automatically generate interfaces. In short, it will be necessary to incorporate successful developments of Knowledge Engineering Environments into the software engineering and configuration management tools of the pervasive computing environment.

A standard upper ontology for services, such as DAML-S [Ankolekar et al., 2002] is a good first step. We foresee the need for standard ontologies for many aspects of the Pervasive Computing Environment, including devices, software services, events, people, places, and things.

Merging (composing) ontologies from multiple autonomous sources is critical for the Pervasive Computing Environment. In particular, it is necessary to incorporate descriptions of new classes of entities (devices, services, components, and so on) and new types of context information (e.g., new sensors) as they are introduced. It should be possible to develop frameworks and editors to assure that the creators of new entities can create descriptions that can be easily merged into system ontologies. Successful research results in merging ontologies must be implemented in standard services and libraries, in order to be integrated into the infrastructure.

A Pervasive Computing Environment poses significant challenges for the architecture and implementation of reasoners and query engines. DAML+OIL (and in the future, OWL [W3C, 2003b]) has proven to be quite useful, especially in combination with a programming interface. However, it seems clear that the DAML and the Description Logic (DL) underlying DAML are necessary but not sufficient for ubiquitous computing applications. Specifically, Description Logics are not suited for some critical aspects of pervasive computing: DL does not deal well with quantitative concepts; including order, quantity, time, or rates. Unfortunately, this kind of reasoning is essential to certain aspects of ubiquitous computing, including, for instance, Quality of Service management [Wichadakul et al., 2002], resource scheduling, and location tracking. Ontologies for pervasive computing environments will require logical models that include spatial
and temporal logic, geometry, and other quantitative reasoning. It is unclear whether DAML+OIL should be extended to include additional logical concepts, or whether other kinds of markup languages should be developed for expressing concepts involving these quantitative aspects.

The Pervasive Computing Environment is a long-running, open, real-time system. Maintaining an ontology in real-time as the system evolves presents important challenges for the design and implementation of ontologies and Knowledge Bases. In particular, the system needs to address the issues of:

- Large scale (many thousands of concepts and relations), many hundreds of services using the ontology and KB).
- Incremental updates (add, delete, or modify a few concepts in a large, active KB).
- Persistence and fault-tolerance
- Federation of multiple local Knowledge Bases

Another area that requires investigation is security, privacy, and access control. The Semantic Web as a whole is largely conceived as a completely open system, in which everything is published for everyone to see. It is far from clear how access control could or should be applied, e.g., to the information in an ontology or a KB. Reasoning engines typically can’t enforce security policies, and the DAML language, for instance, has no facility to limit visibility of concepts or attributes. This topic must be addressed in future research.

Acknowledgements

This research is supported in part by the National Science Foundation grant NSF 98-70736, NSF 9970139, and NSF infrastructure grant NSF EIA 99-72884. Important aspects of this study used software from Iona [IONA Technologies Inc., 2001] and University of Manchester [Bechhofer et al., 1999; OilEd, 2002]

References


Ontologies for World Modeling in Autonomous Vehicles

Mike Uschold, Ron Provine, Scott Smith  
The Boeing Company  
P.O. Box 3707, m/s 7L-40  
Seattle, WA USA  
98124-2207  
michael.f.uschold@boeing.com

Craig Schlenoff, Stephen Balikirsky  
NIST  
Maryland  
schlenof@cme.nist.gov

Abstract

We are in the initial stages of a collaboration between Boeing and NIST. We are exploring the hypothesis that it is beneficial to use ontologies to augment traditional world modeling technologies for autonomous vehicles. Our approach is to develop a theory of obstacles represented as an ontology. It will provide the basis for identifying and reasoning about potential obstacles in the vehicle environment in order to support navigation. We will develop a prototype implementation that incorporates the obstacle ontology and an associated reasoner into an existing autonomous system infrastructure. This infrastructure is based on the 4D/RCS architecture developed at NIST.

1 Introduction

We are in the initial stages of a project that is testing the hypothesis that ontologies can provide benefits in the context of autonomous vehicle navigation. This is a collaboration that leverages and applies ontology expertise at the Boeing Company to an existing autonomous vehicle effort at the National Institute of Standards (NIST). In order to get early feedback from the community, we describe our plans for this collaboration and our progress to date.

A major challenge in autonomous vehicle navigation is the ability to maintain an accurate representation of pertinent information about the environment in which it operates. The inability to do this well hinders effective task planning and execution, especially navigation. Efforts on-going at NIST are applying the 4D/RCS reference model architecture [Albus, J. et.al. 2002] to control an autonomous High Mobility Multipurpose Wheeled Vehicle (HMMWV). An explicit component of the 4D/RCS architecture is a world model which represents the vehicle environment (see figure 1). While the need for ontologies for world modeling is acknowledged, it has not been addressed.

The overall goal of this work is to explore the hypothesis that ontologies can play a significant role in enhancing the capabilities and performance of autonomous vehicles, particularly in the area of navigation planning. To support navigation, an ontology needs to include:

- various objects that an autonomous vehicle is expected to encounter in its environment, and their important characteristics;
- factors that affect the motion of objects, for example: obstacles, road networks, rules of the road;
- actions that an autonomous vehicle is able to perform.

By introducing an ontology (or set of ontologies) into an autonomous vehicle’s knowledge base, we can achieve many potential benefits. One is the potential for reuse and modularity. A general theory of obstacles, for example could apply in a broad range of autonomous vehicles, adapted to the special circumstances of each. Also, for a given vehicle system, an ontology provides the opportu-
nity for a more centralized approach for representing and reasoning with information about the environment. Different modules could query the ontology, rather than having different pieces of the problem scattered across different modules. This has a corresponding benefit in cheaper and more reliable maintenance. An ontology can also extend the range of important questions that can be answered to support navigation planning. For example:

- Based upon sensor data, what are the objects we perceive in the environment at a given time?
- To what extent is a particular object a potential obstacle?
- What is our risk of colliding with the object assuming the motion patterns do not change?
- What are the appropriate actions in a given situation?

Finally, there is potential for increased flexibility of response for the autonomous vehicle. Methods that rely on pre-classification of certain kinds of terrain in terms of their traversability [Donlon & Forbus 1999; Malyankar 1999] are important, but do not support reasoning with obstacles in a more dynamic context.

This effort will focus on assisting in vehicle navigation. For successful navigation planning, an autonomous vehicle is required to know the extent to which a given object may impede its progress. We will develop a theory of obstacles represented as an ontology to determine this for a variety of objects, vehicles and situations. Necessarily, it will be tightly integrated with the ontology of objects in the environment. This will complement other work at NIST that is addressing the representation of rules of the road and road networks.

2 Approach

Our primary focus is the role of obstacles in navigation planning. A long term aim is to develop a comprehensive and reusable ontology of obstacles that can be used in a wide variety of contexts, different vehicle types and environments. We will implement a proof of concept demonstrator which incorporates an obstacle ontology of modest scope with an associated inference engine into an existing autonomous vehicle infrastructure currently being developed at NIST. We will use a simulation tool to generate object data that would otherwise be obtained from processing sensor data from an actual vehicle.

The work will proceed in three phases.

1. Identify the requirements;
2. Create an obstacle ontology related to the objects in the vehicle environment;
3. Implement the proof of concept demonstrator.

We will use a scenario-driven spiral development method starting with a small set of initial requirements, and then repeating the steps adding new requirements and functionality. We now elaborate on these steps, summarizing our progress to date.

3. Identify Requirements

The requirements phase involves the following:

1. identify a scenario;
2. identify competency questions;
3. scope the ontology;
4. identify the representation and inference requirements.

3.1 Scenario

First we identify a scenario which demonstrates the utility of reasoning with an ontology of obstacles. For example, a simple initial scenario could involve a single vehicle driving down a road. We will consider different objects that may be on the road, e.g. small cardboard box or a crate of oranges. We will also consider different traffic conditions. The appropriate action with a small cardboard box in the vehicle's lane is to drive around it. The same situation in heavy traffic, might require going over the box which will is unlikely to cause damage to the vehicle. The same scenario with crate of oranges is more complex. The risk of damaging the vehicle by running into the object must be balanced by the risk of an accident causing damage to one or more vehicles on the road. If the other vehicles are driving in a predictable manner, it may be safe to swerve to avoid the object, if there is a nearby vehicle that is driving erratically, it will be less safe. The navigation planner will view this as a different costs presented by the existence of the obstacle.

In later cycles of the spiral, we will elaborate on the initial scenario, and/or identify a set of related scenarios that could affect the context-specific characteristics of an obstacle (e.g., the speed of a vehicle could affect the damage that could be done by colliding with the obstacle).

3.2 Competency Questions

Next we identify specific competency questions [Gruninger and Fox 1994] that the reasoner must answer using the ontology to support the scenario. In the initial scenario, there would be a small number of simple questions. Here we include a broader range of questions that might come up in more complex scenarios.

1. If I see an object with certain properties,
   a. what is it? what is it not?
   b. at what level of detail can I determine what it is? (e.g. is it a vehicle, a four-wheeled vehicle, a van, a minivan),
   c. is that level of detail enough to determine whether it is an obstacle, and to what extent?
d. how confident am I that the object is what I determine it to be based on sensor input and other reasoning?

e. how do I determine that confidence?

2. What other information do I know about the object once I identify it? Does it have ammunition and am I in its range, is it friend or foe, etc.

3. If I seen a object with certain properties and I’m not sure what it is, what additional information should I gather so that I will be better able to identify the object? This information could be used to task sensors for gathering further information.

4. If I am going a certain speed in specific terrain and I see an obstacle of a particular type, what is the cost of running into it, or of avoiding it?

5. If I see a group of objects that seem to form a particular situation (e.g. a “MEN WORKING” scenario) what additional objects should I be on the lookout for? (e.g. men walking around).

6. If I see an object of type X, then
   a. what is the range of possible speeds that it can be going?
   b. What are its possible directions of travel?
   c. What is the possible rate of change of direction of travel, at a given speed?

   This could depend on context. If a man is standing holding a lollipop sign with slow/stop on either side, then he is unlikely to move into traffic.

3.3 Scope the Ontology

Next we identify the scope of the ontology that is required to answer the competency questions to support the identified scenario. The questions listed above are broader in scope than would be required to support the small initial scenario.

3.4 Representation and Inference

Finally, we will identify the representational and inference requirements needed to answer the identified set of competency questions. This will be the basis for selecting ontology development and inference tools for subsequent phases.

4 Create Ontology

The ontology creation phase involves the following:

1. Literature search on obstacle ontologies;
2. Select appropriate ontology representation, inference and development tools;
3. Create formal representation of obstacles and objects to meet requirements from phase 1;

4.1 Literature Search

We have begun to perform a literature survey to determine relevant work that can be leveraged in the development of a general theory of obstacles. Google returns no hits on obvious search patterns such as: “ontology of obstacles” or “obstacle ontology” (except for the authors’ prior work). The pattern, “theory of obstacles” returns many and only false hits. When this is conjoined with “navigation” or “robot” or “autonomous” there are no hits at all. This is an indicator that the idea of having an explicit theory or ontology of obstacles for autonomous system navigation purposes may be relatively new.

The most closely related work we found is in the area of determining ‘trafficability’. This is defined to be: “a measure of the capability for vehicular movement through some region” i.e. specific kinds of terrain [Donlon & Forbus 1999]. This work is being done in the context of traditional GIS algorithms that may be used for route planning. They are being augmented with qualitative reasoning techniques. Terrain is regarded as being in one of three categories: unrestricted, restricted, or severely restricted. The idea is to pre-classify certain kinds of terrain in terms of its traversability. Slope, hydrography, vegetation and other things are taken into account. For example, if the slope angle is greater than 45 degrees, this would be severely restricted for most 4-wheel vehicles.

Similar work is reported in [Malyankar 1999]. The creation of “navigation ontology” in a marine environment is discussed. It is also set in the context of GIS. These and other sources will be studied and mined for ideas that we hope to generalize and apply to create an ontology of obstacles. Neither work addresses the issue of reasoning about obstacles in real-time from sensor data, it is all based on pre-classifying known terrain. These approaches therefore would not be able to handle our crate of oranges example. Such dynamic capability will be a focus of our research.

4.2 Select Ontology Tools

We will then select an appropriate ontology language, inference engine, and development tools. There is a wide variety of tools to select from. This will be performed by 1) analyzing and determining an appropriate formalism (or set of formalisms) in which to represent the ontology of obstacles, 2) analyzing and determining an appropriate formalism (or set of formalisms) for inference engines, 3) identifying suitable formalism/inference engine combinations, 4) selecting the best combination, and 5) selecting a development tool (e.g. OilEd, Protégé). This decision will also be affected by system requirements arising from the NIST software infrastructure.

4.3 Create Formal Representation

We consider two aspects of creating a formal representation of the ontology:

   1. conceptual analysis
   2. formalization
The first entails identifying the important objects and relationships and finding a way to think about obstacles and their relationship to objects. The second is to represent the results of this analysis and design in a formal language. We report here on some early analysis. We have not begun the formalization stage.

A theory of obstacles is different from an ontology of objects per se. An object may or may not be an obstacle, and this can change over time. One of the interesting questions of our project is: what is the relationship between a theory of obstacles and ontology of objects. Some of the factors that determine whether something is an obstacle are: the vehicle, the context, and to some extent the purpose or goals of the vehicle. The same object, say a small bush will be an obstacle for a small car, but not for an army tank. For a given vehicle, say a car, the same object may be an obstacle at high speed, but not at low speed. An object’s location also determines the extent to which it will be an obstacle.

We distinguish two types of characteristics about objects:
- **static characteristics** - characteristics about object that are not a function of the context in which it is viewed (dimensions, location, velocity, armed/not armed, color, etc.)
- **inferred characteristics** - characteristics that need to be determined through reasoning (is the object of importance?, is the object a threat?, etc.). This would be a function of context, intention, environment, etc.

We will also have to determine which characteristics will be represented in the ontology, and which will be represented outside of the ontology (e.g., cost of running into obstacles?).

### 5 Implement Prototype

We will implement a proof-of-concept scenario in which the planner develops a plan around/through obstacles based on the retrieved characteristics of the obstacles from the ontology. This entails integrating the ontology of obstacles with NIST’s planner and simulation package. Initially, the simulation package will send the exact obstacle (object classification optional) that is being encountered to the ontology and the ontology is sending back the important characteristics of that object.

A cost model will be developed that represents how to respond to different obstacle characteristics. Also, we will ensure that all information provided by the ontology, and the associated inferences, are viewable by the user to allow for ‘white box’ planning and development.

### 6 Issues and Challenges

There are many open questions and technical challenges posed by this work. Some of these are listed below:

- What is the nature of a “theory of obstacles”? How will it be integrated with the ontology of objects in the vehicle’s environment?
- What existing general theories and formal ontologies can be leveraged to create a theory of obstacles?
- How can symbolic reasoning methods be used in conjunction with probabilistic reasoning for use in autonomous vehicle navigation?
- How can ontologies be linked to other types of representations, including sensor data, and other techniques for object identification (e.g. data and information fusion).
- How can we leverage and/or complement a recent effort on applying ontologies for data fusion with the work described here on using ontologies for autonomous vehicle navigation? Attendees at a recent workshop on this topic provisionally agreed that: “Good Ontologies Yield Good Fusion Systems” [Llinas and Little, 2002]. One obvious area of overlap is the object identification task in data fusion.
- Will the response times for ontology reasoning be fast enough to be useful in a real-time environment?
- To what extent can a general theory of obstacles be adapted to a wide variety of autonomous vehicle applications? Can we have a single ontology for multiple types of vehicles and contexts? How much will they have to be tailored? This is analogous to the long-time question about standard upper ontologies (SUO), but within a limited domain. Can their be a SUO of obstacles?
- What will be the best mechanisms for ontology sharing among different autonomous vehicles?
- Using formal ontologies increases the possibility of having different autonomous vehicles be able to communicate among one another with reduced ambiguity. This would be particularly useful where multiple vehicles may be working toward a common goal.
- Can semantic integration techniques using ontologies be leveraged with multiple heterogeneous autonomous vehicles working together?
- What other aspects of autonomous systems may ontologies add value besides navigation planning?
- How can one evaluate the performance of the ontology? Where does the ontology really add leverage compared to approaches not using ontologies? For ex-
ample, does the ontology really help increase the ability to deal with dynamically changing environments? When would these other approaches be preferred, and when would ontology-bases approaches be preferred?

References


[Schlenoff, 2002a] Position statement at panel discussion on the "Role of Ontologies in Intelligent Systems" at Performance Metrics for Intelligent Systems (PerMIS) 2002 held at NIST in Washington, DC

D. Meaning Negotiation
Context Matching for Electronic Marketplaces - a case study

Matteo Bonifacio$^{1,2}$, Antonia Donà$^1$, Alessandra Molani$^{1,2}$ and Luciano Serafini$^3$

$^1$ITC-IRST, Centro per la Ricerca Scientifica e Tecnologica
Panté di Povo, Trento Italy
$^2$Department of Information Technology, University of Trento
Panté di Povo, Trento Italy
$^3$ITC-IRST, Centro per la Ricerca Scientifica e Tecnologica
{antodona,serafini}@itc.it
{bonifacio,molani}@dit.unitn.it

Abstract

Matching algorithms automatically discover semantic relations between two autonomously developed conceptual representations of two overlapping domains. Typical examples of such conceptualizations are electronic market catalogues (e.g., UNSPSC and eCL@SS) and web directories (e.g., GOOGLE and YAHOO!). The objective of this paper is the description of a use case in which matching algorithm has been used to re-classify into UNSPSC the catalogue of the office equipment and accessories used by a worldwide telecommunications company to classify their suppliers. On the basis of this experience we are envisaging new application of the algorithm in the area of demand aggregation, and we will conclude the paper by briefly describing a future application in this area.

1 Introduction

In the e-Business hype, marketplaces have been proposed as the optimal solution to foster efficiency and dynamic business integration, however, reality have shown something different. The assumption that standard catalogues can substitute local business domains, there are different competing standards ones has been neglected by the simple evidence that, on similar business domains, there are different competing standards [Agrawal and Srikant, 2001]. Moreover, company buyers have difficulties in adopting classification standards that are way complex and generic from their simple and task specific ones. This is more true when considering that local conceptualizations are not just the mere result of cultural/historical differences, but rather the consequence of different, substantial, valuable ways of doing things. Furthermore, the idea of standardizing semantics seems to be conceptually wrong, more than practically unfeasible, if semantic heterogeneity is read in terms of richness to be exploited rather than in terms of noise to be reduced [Bonifacio and Molani, 2003]. In very simple words, if people call things in different ways is because they do different things, have different goals, adopt different perspectives.

In this scenario the only feasible solution to heterogeneity in e-Business is the one that admits the existence of a set of heterogeneous and overlapping products catalogues, and supports semantic interoperability between then. Semantic interoperability is reached by matching algorithms, i.e., procedures capable to find semantic relations between the categories of different products catalogues. Two examples of matching algorithms, representing two different approaches, are the CTXMATCH algorithm described in [Bouquet et al., 2003], and the GoldenBullet system described in [Ding et al., 2002]. The former is based on NLP techniques applied to the labels occurring in the classification and the transformation of the matching problem in a satisfiability problem, while the latter uses techniques for information retrieval and machine learning applied to the content of the classification. While GoldenBullet demands for a training set, i.e., it needs a set of examples of mappings between concepts which have been checked manually, CTXMATCH is completely automatic and needs as input just two classifications.

The objective of this paper is the description of an experiment of applying the CTXMATCH algorithm in a case where no training set was available. In particular we apply CTXMATCH to find mapping between the catalogue of office equipment and accessories used to classify company suppliers of a worldwide telecommunication company and the standard catalogue UNSPSC. In this sense we simulated a situation in which a network actor of a marketplace becomes able to share information about products and services with other actors without adopting a predefined ontology. This paper can be consider also as a partial answer of the call for proposal described in [Schulten et al., 2001], where the author suggest the following challenge in the e-business “[…] to come up with a generic model and working solution that can semiautomatically map a given product description between two different e-commerce product classification standards”.

The paper is structured as follows. In Section 2 we describe the CTXMATCH algorithm; Section 3 describes the problem and the solution we have proposed in use case. Section 4 draw some conclusions and describe a future application.

2 CTXMATCH Algorithm

CTXMATCH enables semantic interoperability between overlapping concept hierarchies through mapping discovery. The algorithm [Magnini et al., 2002] takes as input two conceptual hierarchies (i.e. a source hierarchy and a target hierarchy) and returns a set of directed mappings between source and target concepts. The main features of the algorithm are the following: it does not consider concept instances (e.g., documents), so that it can be used in situations where such information is partially available or is not available at all; it returns a semantic evaluation of the mapping between two concepts (i.e. equivalence, more general than, less general than); it is context-based, in the sense that it builds a semantic representation of the meaning of a concept which depends
both on the position in which it appears in a concept hierarchy and on world knowledge available in an external resource.

The algorithm performs three main steps: (i) a linguistic analysis of the concepts without considering the hierarchical structure of the context; (ii) a logical interpretation of the concept based on the structural relations of the context; (iii) the identification of mapping relations between the logical interpretations of the concepts using SAT.

2.1 Linguistic Analysis

The first step of the procedure consists of text chunking, i.e. dividing each label into syntactically correlated parts of words. We run the standard Alembic chunker [Day and Vine, 2000], developed by MITRE Corporation as part of the Alembic extraction system [Aberdeen et al., 1995].

For example, with the label Globalization and Free Trade, the chunker first selects a part of speech for each word ('Globalization' and 'Trade' are nouns, 'Free' is an adjective, 'and' is a conjunction); then, it identifies two noun groups (NGs), i.e. 'GLOBALIZATION' and 'Free Trade' (notice that the syntactic head is marked in small capitals), and a coordinating conjunction between them: \[
\text{\{\text{GLOBALIZATION}\}_{m} \text{\{\text{Trade}\}}_{n} \text{\{\text{Free}\}}_{o} \text{\{\text{and}\}}_{p} \text{\{\text{Trade}\}}_{q}
\]

The output of the chunker is used to transform each label into a basic logical form. A noun group consisting of more than one word is interpreted as the conjunction of the head and all its modifiers; for instance, Iron Trade is interpreted as [Iron<\&>]Trade. The relations between different noun groups are interpreted on the basis of the linguistic material connecting them: coordinating conjunctions and commas are interpreted as a conjunction (e.g. Globalization and Free Trade is interpreted as [[GLOBALIZATION]< \& \& ] [[Free]< \& \& ]] [[Trade]< \& \& ]), prepositions, like 'in' or 'of', are interpreted as a conjunction (e.g. Iron Trade of Great Britain is transformed into [[Iron<\&>]Trade]<\&>][[Great<\&>]Britain]])], expressions denoting exclusion, like 'except' or 'but not', are interpreted as a negation (e.g. Garments except Skirts becomes [[Garments]<\&>&]>[[Skirts]])].

In order to perform the semantic interpretation of the labels CtxMATCH accesses WORDNET [Fellbaum, 1998]. When a word is found, all the senses of that word are selected and attached to the basic logical form.

When two or more words in a label are contained in WORDNET as a single expression (i.e. a multiword), the corresponding senses are selected and, in the basic logical form, the intersection between the two words is substituted by the multiword. In the case of [[Iron<\&>]Trade]<\&>][[Great<\&>]Britain*]], for instance, 'Great Britain' is provided in WORDNET as a single expression, so the logical interpretation is substituted by the senses of the multiword, thus obtaining [[Iron<\&>]Trade*]<\&>][[Great, Britain*]].

2.2 Logical Interpretation

The full logical form of a label is the conjunction of the basic logical forms of the label and all its ancestors. To make an example, let's take the concept hierarchy whose root is Soccer, with a descendant Leagues and a further descendant Clubs. The full logical form of the root is simply \([\text{soccer}]\), the full logical form of Leagues is \([\text{soccer}] \& [\text{league}]\) and the full logical form of Clubs is \([\text{soccer}] \& [\text{league}] \& [\text{club}]\).

As explained before, the disjunction between noun groups can be made explicit by the presence of a coordinating conjunction, but we can also have implicit disjunction between elements placed at different levels of the hierarchy. In the example above, at a deeper level of analysis there are two conflicting interpretations: from the point of view of the hierarchical structure clubs denotes a subset of leagues; on the other hand, from the point of view of the world knowledge provided in WORDNET, [club#2] and [league#1] are disjoint because they have the same hypernym, i.e. association.

In order to combine the two information sources, leagues has to be reinterpreted as if it were leagues and clubs, i.e. \([\text{league#1}] \& \& [\text{club#2}]\).

Similarly, also the negation is not always marked by expressions like 'but not' or 'except'. For instance, we can have Sociology and Science as sibling nodes classified under an Academic Study of Soccer. From the point of view of world knowledge, sociology is a science (and in fact in WORDNET sociology#1 is a second level hyponym of science#2). As a consequence, the node labeled with Science has to be interpreted as if it were Science except Sociology.

The recognition of multiwords can also be performed on different contiguous levels. For instance, in WORDNET there is a multiword 'billiard player', so in a hierarchy where Sport has Billiards as a child and Player as a further descendant, the conjunction of [billiard*] and [player*] can be substituted with the multiword, giving as a result the logical form \([\text{sports*}] < \& \& > [\text{billiard, player*}]\).

CtxMATCH performs word sense disambiguation by taking into consideration both structural relations between labels and conceptual relations between words belonging to different labels. Let L be a generic label and L.1 either an ancestor label or a descendant label of L and let s* and s1* be respectively the sets of WORDNET senses of a word in L and a word in L.1. If one of the senses belonging to s* is either a synonym, a hyponym, a holonym, a hypernym or a meronym of one of the senses belonging to s1*, these two senses are retained and all other senses are discarded.

As an example, imagine Apple (which can denote either a tree or a fruit) and Food as its ancestor; since there exists a hyponymy relation between apple#1 (denoting a fruit) and food#1, we retain apple#1 and discard apple#2.

2.3 Computing Concepts Relations via SAT

In the first two steps, CtxMATCH associates a formula \(w(k)\) (expressed in a simple description logic) to each concept k of a hierarchy. This formula is supposed to capture the semantic of this concepts. The last phase of CtxMATCH addresses the problem of discovering the semantic relationship between two concepts k and k' by reducing it to the problem of checking, via SAT, a set of logical relations between the formulas \(w(k)\) and \(w(k')\). The SAT problem is built in two steps. First, it selects the portion \(T\) of the background theory relevant to \(w(k)\) and \(w(k')\), namely the WORDNET relations involving the senses that appear in \(w(k)\) and \(w(k')\). In the second phase, we compute some of the logical relations between \(w(k)\) and \(w(k')\) which are implied by \(T\).
The background theory $T(k, k')$ relevant for computing the relation between $k$ and $k'$ is obtained by translating the WORDNET hierarchical relations on senses appearing in $w(k)$ and $w(k')$ into a subset of subsumptions in description logic.

The equivalence between $k$ and $k'$ is checked by verifying that $w(k) \subseteq w(k')$ and $w(k') \subseteq w(k)$ are both implied by $T(k, k')$. Similarly, the fact that $k$ is more specific [general] than $k'$ is checked by verifying that $w(k) \subset w(k')$ $[w(k') \subset w(k)]$ is implied by $T(k, k')$; the fact that $k$ is compatible with $k'$ is checked by verifying that $w(k) \cap w(k')$ is satisfiable in $T(k, k')$; finally the fact that $k$ is disjoint from $k'$ is checked by verifying that $w(k) \cap w(k')$ is not satisfiable in $T(k, k')$.

To each relation it is possible to associate also a quantitative measure that considers the cardinality of models satisfying $w(k)$ and $w(k')$.

3 Use case Product Re-classification

In order to centrally manage all the company acquisition processes, the headquarter of a well known wide telecommunication company has realized an eProcurement system, which all the company branch-quarters have been required to join. In order to join it, each single office was also required to migrate from the product catalogue they used to manage with, to the new one managed within the platform. This catalogue is extracted from the Universal Standard Products and Services Classification (UNSPSC), which is an open global coding system that classifies products and services. The UNSPSC is used extensively around the world in the electronic catalogues, search engines, procurement application systems and accounting systems. UNSPSC is a four level hierarchical classification; an extract is reported in the following table:

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Furniture and Furnishings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2</td>
<td>Accommodation furniture</td>
</tr>
<tr>
<td>Level 3</td>
<td>Furniture</td>
</tr>
<tr>
<td>Level 4</td>
<td>Stands</td>
</tr>
<tr>
<td>Level 4</td>
<td>Sofas</td>
</tr>
<tr>
<td>Level 4</td>
<td>Coat racks</td>
</tr>
</tbody>
</table>

The Italian office asked us to apply the matching algorithm to re-classify into UNSPSC (version 5.0.2) the catalogue of the office equipment and accessories used to classify company suppliers.

The items to be re-classified are mainly labeled with Italian phrases, but labels contain also abbreviations, acronyms, proper names, some English phrases and some typing errors. The English translation of an extract of this list is reported in the following table (the italic parts were contained in the original labels).

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENT.21.13</td>
<td>cartridge hp desk jet 2000c</td>
</tr>
<tr>
<td>ENR.00.20</td>
<td>magnetic tape cassette exatape 160m x 7,0gb</td>
</tr>
<tr>
<td>ESA.11.52</td>
<td>hybrid roller pentel red</td>
</tr>
<tr>
<td>EVM.00.40</td>
<td>safety scissors, length 25 cm</td>
</tr>
</tbody>
</table>

The item list was matched with two UNSPSC’s segments, namely: Office Equipment and Accessories and Supplies (segment 44) and Paper Materials and Products (segment 14).

3.1 Methodology

We started with the linguistic analysis (normalization phase) of the company item catalogue and of the UNSPSC segments we took into account. The linguistic analysis involves first a morphological cleaning process, then a disambiguation and enrichment process. This is performed by accessing WordNet and, for each given term, finding out all the instances of its semantic meanings (corresponding to WordNet numeric IDs) and associate them all the available synonyms. The output was two files with the semantic explicitations of the catalogue items on one hand, and of the UNSPSC nodes on the other, both in terms of IDs of WordNet. Then we went on with the matching phase by running the algorithm on the two files. The result of the matching can be clearly interpreted in terms of re-classification: if the algorithm returns that the item $i$ is equivalent to, or more specific than, the node $c_{UNSPSC}$ of UNSPSC, then $i$ can be classified under $c_{UNSPSC}$ of UNSPSC.

Notice that the company item catalogue we had to deal with, was a plain list of items, each identified with a numerical code made up of two couple of numbers the first referring to a set of more general categories (for example, in 21.13 cartridge hp desk jet 2000c- 21 corresponds to printer tapes, cartridge and toner). We first normalized and matched against UNSPSC such plain list. This did not lead us to a satisfactory result. The algorithm performed much better when we made explicit the hierarchical classification contained in the item codes. This has been done by substituting the items first numerical codes with their textual description, provided us by the company. The validation phase of our results has been made by comparing them with the results of a simple keyword based algorithm. Obviously, in order to set the correctness, in terms of precision and recall, of such results we needed a correct matching list to be used as point of reference for the validation. Then we ask a domain expert, Alessandro Cederle, Managing Director of Kompass Italia to validate a possible correct matching list we provided him with.

3.2 Results

This sections presents the results of the re-classification. Consider first the baseline matching process. The baseline has been performed by a simple keyword based matching which worked according to the following rule: for each item description (made up of one or more words) gives back the set of nodes, and their paths, which maximize the occurrences of the item words.

The following tables summarizes the results for baseline matching:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENT.21.13</td>
<td>cartridge hp desk jet 2000c</td>
</tr>
<tr>
<td>ENR.00.20</td>
<td>magnetic tape cassette exatape 160m x 7,0gb</td>
</tr>
<tr>
<td>ESA.11.52</td>
<td>hybrid roller pentel red</td>
</tr>
<tr>
<td>EVM.00.40</td>
<td>safety scissors, length 25 cm</td>
</tr>
</tbody>
</table>

An eProcurement system is a technological platform which supports a company in managing its procurement processes and, more in general, the re-organization of the value chain on the supply side.

2Kompass (www.kompass.com) is a company which provides product information, contacts and other information about 1.8 million companies worldwide. All companies are classified under the Kompass Product Classification with more than 52,000 products and services.
Given the 194 items to be re-classified, the baseline process found 1945 possible nodes, that means that for each item it found a set of 6 possible maximizing nodes by average. What is crucial is that only 75 out of the 1945 proposed nodes are correct. The baseline, being a mere simple string matching, is able to capture a certain number of re-classifications, but the percentage of error is quite high (50%), with respect to the one of correctness (39%). Such parameter shows different values for the matching algorithm, that is able not only to compare strings but also to interpret their meaning. The results of the matching algorithm are reported in the following table:

<table>
<thead>
<tr>
<th></th>
<th>Baseline classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total items</td>
<td>194</td>
</tr>
<tr>
<td>Rightly classified</td>
<td>75</td>
</tr>
<tr>
<td>Wrongly classified</td>
<td>91</td>
</tr>
<tr>
<td>Non classified</td>
<td>27</td>
</tr>
</tbody>
</table>

In this case the percentage of success is sensibly higher (70%) and, even most relevant, the percentage of error is minimal (8%). This is confirmed also by the values of precision and recall, computed with respect to the validated list:

<table>
<thead>
<tr>
<th></th>
<th>Matching classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total items</td>
<td>194</td>
</tr>
<tr>
<td>Rightly classified</td>
<td>134</td>
</tr>
<tr>
<td>Wrongly classified</td>
<td>16</td>
</tr>
<tr>
<td>Non classified</td>
<td>42</td>
</tr>
</tbody>
</table>

The baseline precision level is quite small, while the matching one is not excellent but definitely better. The same observations can be done also for the recall values.

Table 1 reports some examples where the algorithm found out a correct item for re-classification, while the baseline did not. The ability of the alghoritm to reason both on linguistic data and on structural ones, accounts for its good performance with respect to the baseline one. From the linguistic point of view, two considerations are to be done. First the possibility for our algorithm to manage with synonymous allows it, for example, to recognize that the item perforatrice\(^4\), and all its variants (perforatrice 2/4 fori, perforatrice universale, etc) have the same meaning of the UNSPSC node punzonatrice so it suggested to re-classify it under that node. Second, the fact that during the normalization phase a stemming cleaning is performed on words allows the matching phase to deal with lemmas in their basic form, without any morphological modification. (e.g. the singular or plural word). This means that in the example of the item evidenziatore\(^5\) (singular form) the algorithm is able to suggest the matching with the UNSPSC node evidenziatori, which is the plural form.

From the structural point of view, the possibility to reason on some topological properties allows the algorithm to point out a semantic relation such as More General than between the company catalogue item nastro per stampante, toner, cartuccia, testina di stampa and the UNSPSC node Cartucce d'inchiostro. In this case, the properties taken into account are two: the fact that the catalogue item has just one level higher, while the UNSPSC one has two, and the fact that the catalogue item is made up of several single items, while the UNSPSC one just of one. These two consideration are enough to set that the meaning of the first should be more general then the second’s. If there are not enough structural data to suggest a semantic relation, the alghoritm gives back at least a compatibility between the two elements, computed on some linguistic occurrences measures (see the last four lines of the table). For the Non Classified items, it should be noticed the following:

- in some cases the item to be re-classified has an incorrect position within the company catalogue, so the matching alghoritm couldn’t compute in the right way the relations with the node and its father node. Examples are the ast-ray which has been classified under tape dispenser, the wrapping paper which has been classified under adhesive labels.
- in some cases in order to understand the meaning of the item to be re-classified, more domain knowledge should be requested and then embedded within the system. An example is the case of paper for hp: in order to understand that it is printer paper, it’s necessary to know that hp stands for Helwett Packard and that this is a company which produces printers.

A final step of the use case was having a second matching between the company catalogue’s IDs and the output of the normalization of the English version of UNSPSC with the English version of WordNet. This procedure, viable because the matching computation performing on the matrix takes into account just the concepts’ IDs, allows us to find many more matching than using just one language.

More in general, this way allows us to approach and manage multilanguage environments and to exploit the richness which typically caracterizes the English version of any linguistic resources.\(^6\)

Two lessons have been learned from this experiment. First, our algorithm is good for re-classification rather than for simple classification of a plain list of items. As previously explained, the alghoritm exploits two kinds of data, linguistic and structural ones: in the case of a plain list, the second set of data are missed and this impacts on the results’ goodness. This is the reason why results definitely improve when we run the alghoritm on the company items catalogue enriched by the category structure extracted from the numerical codes.

The second lesson concerns the quality of the labels. The better the labels are written, the better is the matching obtained. In presence of meaningless labels such as short-cuts (“num” for number, “cart.” for cartridge, etc.), proper names (Duracell, Hewlett Packard, etc.), this version of the

---

\(^4\)Notice that the algorithm did not take into account only the UNSPSC 4th level category, since in some cases catalogue items can be matched with UNSPSC (3rd level) category nodes.

\(^5\)Since the whole matching work strongly depends on languages, we will present results in Italian but provided with a very literally English translation. We apologise if in some cases such translation will not completely support the reader in the comprehension of the example.

\(^6\)In English: “drilling machine.”

\(^7\)In English: “highlight.”

\(^8\)We do not report here the results of this last step, since we cannot compare them with the results of the baseline that, being a mere keyword based algorithm, could run only on homogeneous linguistic situation.
algorithm is not capable of assigning a proper semantic to the labels, decreasing the performances. Possible improvements in this direction could be reached through two ways. First, the linguistic analysis of labels can be improved by using domain-oriented linguistic resources. Several domains are developing thesauri, ontologies, standard classifications which specifically deal with their lexicons.  

These kind of resources involve also relevant proper names, company names, abbreviations and acronyms, which right now are still problematical data for the algorithm.  

The second way to be investigated is the possibility of supporting functionalities of spell checking, able to detect spelling errors and to suggest the right alternatives.

### 4 Related Work

An alternative approach to CtxMATCH, for product classification in UNSPSC, called GoldenBullet, is described in [Ding et al., 2002]. GoldenBullet is an environment that support product classification according to content standards. It applies techniques of information retrieval and machine learning. The classification is based on a training set. The classification algorithm implemented in GoldenBullet performs indeed very well when it is used in a supervised way. This approach can therefore be useful (and maybe it could perform better than CtxMATCH) in presence of a representative set of pre-classified examples. CtxMATCH, instead provides good results also without such a training set.

A relevant approach to ontology matching has been proposed in [Doan et al., 2002] and [Madhavan et al., 2002]. Although the aim of the work (i.e. establishing mappings among concepts of overlapping ontologies) is in many respects similar to our goals, the methodologies differ significantly. A major difference is that the GLUE system builds mappings taking advantage of information contained in instances, while our current version of the CtxMATCH algorithm completely ignores them. This makes CtxMATCH more appealing, since most of the ontologies currently available on the Semantic Web still do not contain significant amount of instances. A second difference concerns the use of domain-dependent constraints, which, in case of the GLUE system, need to be provided manually by domain experts, while in CtxMATCH they are automatically extracted from an already existing resource (i.e. WordNet). Finally, CtxMATCH attempts to provide a qualitative characterization of the mapping in terms of the relation involved among two concepts, a feature which is not considered in GLUE. Although a strict comparison with the performances reported in [Doan et al., 2002] is rather difficult, the accuracy achieved by CtxMATCH could be roughly compared with the accuracy of the GLUE module which uses less information (i.e. the “name learner”).

A mapping procedure based on lexical information has been proposed in [Bergamaschi et al., 2002]. No quantitative evaluation is reported. Only a qualitative exemplification, based on the task proposed in [Schulten et al., 2001], is described to show the algorithm capabilities.

Finally, the evaluation of the Anchor-PROMPT System [Noy and Musen, 2001] has been conducted on two ontologies and the mappings identified by the algorithm have been manually checked. Results are presented in term of the achieved precision.

### 5 Conclusions and Further Application

We focused on the evaluation of a context matching algorithm, which automatically generates mappings among the concepts of two overlapping hierarchies. The main features of the algorithm are the following: it does not consider concept instances, so that it can be used in situations where such information is partially available or not available at all; it returns a qualitative estimation of the mapping between two concepts (i.e. equivalence, more general than, less general than); it is content-based, in the sense that it builds a semantic representation of the meaning of a concept given both the context and the world knowledge available in an external resource (i.e. WordNet).

We have presented three empirical experiments with a twofold aim: first, we wanted to evaluate the CtxMATCH algorithm in real, large scale scenarios; second, we wanted...
to test different evaluation methodologies. In particular, we have experimented CTXTMATCH on Web directories and market place catalogues.

A number of data have been collected, which are to be considered as a first contribution toward common evaluation practices and the possibility to share resources for context matching algorithms.

Another application we are now working on is a system to manage the automatic aggregation of buyers’ demands. This is aimed to be embedded in those technological platforms (such as eProcurements system, or also marketplaces) where the possibility for buyers to aggregate their product demands could give them some advantages in terms of furnitue conditions or buying power. In order to support the aggregation process, the system should be able, first to point out groups of buyers interested in a similar category of product, then to suggest if and how each buyer should modify some requested features in order to get to more advantages. As an example consider the following case: the acquisition offices of two public universities are interested in buying 200 mobile phones, but one office prefers mobiles with the features A and B, the other office is more interested in mobile with the features C and D. Let’s suppose that a mobile seller proposes a strong discount for 400 mobiles with features A, D and E. If the two offices converged on this last kind of mobile, they would get to the discount. In order to support this process, the system should be able not only to match item at the product description level, but also at the attribute level. Attributes are used to specify product features such as color, length, size, etc, and typically they are the elements on which a negotiation process could be done. The first idea we are investigating on is to develop different attribute contexts (one for colors, one for length, one for size, ...), and a specific version of the algorithm aimed to match this kind of structures. This way would allow us to split the matching between product structures, and the one between attribute structures and to combine afterwards the two sets of results.

A step further will be to develop the capability for the system to support buyers in the negotiation on attributes, for example by providing users with simulations of different combinations of attributes and displaying the related advantages.

References


Ontologies provide potential support for knowledge and content management on P2P platform. Although we can design ontology beforehand for an application, it is argued that in P2P environments, static or predefined ontology cannot satisfy the ever-changing requirements of all users. So we propose every user should make proposals for what kind of ontology is the most up to his need. Collecting all these proposals (or votes) helps to the drift of ontology. This paper presents OntoVote, a scalable distributed votes collecting mechanism based on application-level broadcast trees and describes how OntoVote can be applied to ontology drift on P2P platform by discussing several problems involved in the voting process.

1 Introduction

With the rapid development of ontology technology and P2P computing in the past few years, it has been suggested that knowledge and content management on P2P platform make use of ontologies to provide enhanced services to users [Fensel et al., 2002]. To attain this object, some limited but beneficial attempts have been made and even more research plans are on the agenda. For example, the open source project Edutella [Nejdl et al., 2002] aims to provide an RDF-based metadata infrastructure for P2P applications building on the JXTA framework [JXTA]. [Sato et al., 2002; Nodine et al., 2000; Arumugam et al., 2002; etc] try to gather and share information and knowledge with the help of ontologies in P2P or other distributed environments.

These attempts presume that ontologies have been constructed beforehand and what they are concerned about is how to use ontologies to exchange knowledge and to enable efficient and accurate semantic search in distributed environments. In many application scenarios, such predefined ontologies cannot catch up with the ever-changing requirements of users. Instead, ontology should drift with the appearance of new application requirements. But just as [Fensel et al., 2002] has stated, one cannot expect any maintenance to happen on the ontologies in P2P environments (in fact, users will not often know what is in the ontologies on the machine, let alone that they perform maintenance on them) and as a result, we must design mechanisms that allow the ontologies to update themselves, in order to cope with ontological drift. [Fensel et al., 2002] has proposed several informal mechanisms that use metaphors from social science (opinion-forming, rumor-spreading, etc). In this paper, we propose a more formal mechanism of ontology drift that is based on every user’s participating in proposing the modification of ontology according to his demands of the application. To relieve the burden of users, proposals can be obtained by mining user activities (so called emergent semantics [Maedche and Staab, 2001], e.g., by mapping the modification of a directory name to the modification of a concept in ontology) or by providing users with a basic ontology together with visualization tools with which the users can make modifications easily. The modifications cause every user to hold a local ontology. These ontologies are characterized by:

- They are partly overlapped, but the same concepts may be expressed in different words.
- There are a lot of noisy semantics, owing to the wrong activities of users (e.g., a rookie of a domain may modify the domain ontology wrongly).
- Most of them cannot represent all aspects of the requirements of users.

In order to align concepts, to filter out noisy semantics, and to indicate the principal direction of the development of user requirements, we propose these local ontologies be combined together to construct a common ontology. With a common ontology, we can also improve the efficiency of semantic search by avoiding too many mappings between ontologies.

One possible way to combine the ontologies from all users is votes collecting: we collect the proposals of all users to make some analyses; only the semantics hold by a majority of the users (or we can set a threshold for the proportion of users) is adopted by the common ontology. The various minor semantics collected can also be treated in different ways according to its value in use, which we will describe in details afterwards.
Practically, a voting organizer (such as a chairman or a tally clerk) is needed to accomplish the voting task. This organizer can be considered as a server and serves for the common interests of a community by publishing messages to and receiving messages from all other voters. But in P2P environments, it may be hard to find any volunteer to serve the community for no evident good. Moreover, using a server to collect votes will bring about scalability and single node failure problems as discussed in many P2P researches. To get rid of such problems, we use OntoVote, a scalable distributed votes collecting mechanism based on application-level broadcast trees, to collect votes on P2P platform.

The rest of the paper is organized as follows. Section 2 describes the design of OntoVote. Section 3 applies OntoVote to the process of ontology drift. Section 4 makes a conclusion of our work and proposes our future work.

2 OntoVote

In practice, a voting process can be divided into three successive phases. The first is a preparing phase, notifying all participants to get ready their votes. The second is a collecting phase, collecting votes from all participants. And the third phase is devoted to publishing the voting results to all participants.

OntoVote realizes the voting process in a fully decentralized manner. It uses broadcasts and a reverse operation on an application-level broadcast tree to fulfill the three phases of a voting. As in Figure 1, at the first and the third phase in (a), notifying messages and voting results are published from root to leaves. At the second phase in (b), a node on the tree first collects votes from all of its children, then it sums up the votes from the children together with its own votes, and submits the total votes to its parent.

To make use of broadcast trees, OntoVote supposes there are a large number of groups on P2P platform. Every peer can choose several groups to join in. Every group forms a reliable application-level broadcast tree with mechanisms introduced in some researches [Castro et al., 2002; Zhuang et al., 2001; etc], which is out of the scope of this paper (one can simply view a broadcast tree as a tree). Votes collecting happens inside a group. OntoVote provides best-quality votes collecting service (i.e., collecting exactly one copy of votes from every participant) by extending the existing application-level broadcast mechanisms.

2.1 Basic Implementation of OntoVote

In this section we introduce the basic implementation of OntoVote, mainly discussing two important procedures: getCredential() and deliver(msg). getCredential validates a voting. deliver handles the messages on broadcast trees.

Voting Validity

Before a new round of voting in a group is initiated, the root node of the broadcast tree of the group calls getCredential to get a voting credential for the voting. A voting credential is granted only when a majority of group members are online so that they are capable to take participant in the voting, otherwise, the voting results plotted by a minority of group members will be invalid and misleading. Current version of OntoVote simply assumes that all votes are available, so a voting credential is always granted. This assumption is correct if all peers publish their votes to rendezvous, which are online most of the time, just as in JXTA [JXTA].

Voting Process

The three phases of a voting process is realized in the procedure deliver(msg), which is called whenever a node receives a message whose destination is the node itself. The parameter msg contains the received message. The pseudo code for this procedure, simplified for clarity, is shown in Figure 2.

The following variables are used in the pseudo code: msg.type is the message type, which may be PREPARE, SUBMIT or PUBLISH, corresponding to the three phases of a voting. msg.groupID indicates the group the message belongs to. groups is a set of groups that the node has joined in, groups[].children and groups[].parent are bi-directed links of the broadcast tree of the group. To avoid conflicts among different voting tasks, we treat each voting process as a transaction and use transID to distinguish them from each other globally and uniquely. trans is a set of transactions that the node is involved in, trans[].votePool is the vote pool which keeps an entry for the votes from every child (i.e., keep the votes from every child separately).

After the root node has got a voting credential, it sends PREPARE messages to its children, thus starting the new round of voting. On receiving a PREPARE message, a node sets up a new transaction environment by clearing the vote pool for the transaction (line 2). If the node is a leaf node, it sends a SUBMIT message to itself to start the collecting phase (lines 3, 4). Otherwise, it recursively passes on the message to its children (lines 5, 6).

When a node receives a SUBMIT message from a child, it adds the votes from the child to its vote pool in line 7 (If the child has submitted votes before, the old submitted votes in the vote pool will be replaced with new submitted votes). After all children of the node have submitted votes, the node adds the votes in the pool and its own votes together, then submits the total votes to its parent (lines 10 to
12). These lines may be called for several times allowing for node failures, which will be discussed in details in the next section.

After getting votes from all of its children, the root node extracts some useful knowledge from the votes and re-publishes the knowledge to its children (lines 8, 9), thus any node in the group can update its local knowledge base (lines 13 to 15).

As is seen, the basic implementation of OntoVote is very simple. But if we want to collect votes in a distributed fashion reliably and efficiently on network, there are still more things to be considered.

2.2 Reliability
On account of the unreliable nature of Internet, a broadcast tree may break at any time, including during voting process. If we collect votes from rendezvous, the rate of failure will decrease sharply, but we still cannot avoid node failures completely. Therefore, OntoVote proposes repairing the broadcast tree for the best-quality collecting purpose.

Periodically, each node in the tree sends a heart beat message to its children, if any, and the children respond with answering messages. A child suspects its parent is faulty when it fails to receive heartbeat messages and so does the parent, i.e. when two nodes lose in touch with each other, every one of them will suspect the other has failed. But in fact, any one of them may be really faulty, or none of them are faulty, but the link between them is broken.

Upon detection of the failure of its parent, a child tries to connect to a new parent. To maintain the performance of fully distributed votes collecting, the tree’s balance should be approximately retained, so the new parent is chosen from the tree nodes that are nearly at the same level with the old parent in a uniformly-random way.

If node failures occur during publishing phases (i.e., the first or the third phase in Figure 1), it is a trivial task to ensure that every node on the tree would receive the published message: the new parent sends all published messages it has received or will receive to the new child, and the child either relays the messages to it children or discards the messages, according to whether or not it has received the identical messages before.

However, if node failures occur during collecting phase, things become a bit more complicated: if a node fails at the very time that some children have submitted votes to it and some not, how about these submitted votes? If the failing node has disconnected from the network, the submitted votes will be lost. To avoid losing votes, the children of the failing node can resubmit votes to a new parent. But if the failing node has not disconnected from the network or if the failing node has submitted votes to parent before its failure, straightforward resubmitting will result in redundant votes on the broadcast tree. In the rest of this section, we first put forward the repairing protocol of OntoVote for collecting phase, then we show that this protocol satisfy our best-quality collecting purpose by leaving out as few votes as possible and by avoiding counting in the same copy of votes repeatedly.

Repairing Protocol for Collecting Phase
As in Figure 3, let $FN$ be a failing node from the aspect of its child $CN$. $CN$ reconnects to a new parent $NPN$. We use $P[X]$ to denote the parent of a node $X$. $DT$ is a configurable time limit. After $CN$ reconnects to $NPN$, the repairing protocol goes as follows:

1. If $CN$ has not submitted votes to $FN$ yet, then $CN$ will submit votes to $NPN$ after it collects votes from all of its children.

   Else if $CN$ has submitted votes to $FN$, but the interval from the submission of $CN$ to the failure of $FN$ is still within the time limit $DT$, then $CN$ will submit votes to $NPN$ immediately.
Else $CN$ submits a null vote as a placeholder to $NPN$ immediately.

(2) After $NPN$ adds the votes from $CN$ to vote pool:

If $NPN$ has never submitted votes before (that is, $NPN$ is still waiting for some other children), or if $NPN$ finds that the votes from $CN$ are null, then stop.

Else $NPN$ recounts the total votes (including that of $CN$) and resubmits the total votes to $P[NPN]$. The recounting and resubmiting process will be iterated up the tree until some ancestor of $NPN$ that has not submitted votes yet.

(3) If $CN$ has submitted votes to $FN$ and the interval from the submission of $CN$ to the failure of $FN$ is still within the time limit $DT$, then $FN$ will delete the entry of $CN$ in its vote pool. If $FN$ has also submitted votes to $P[FN]$, it will recount the total votes it has collected (excluding that of $CN$) and resubmit the total votes to $P[FN]$. The recounting and resubmiting process will be iterated up the tree until some ancestor of $FN$ that has not submitted votes yet.

Figure 3. Repairing broadcast trees for best-quality collecting

Explanation of the Repairing Protocol

To illustrate the robustness of this protocol, assume in Figure 3, $CN$ loses in touch with $FN$ and reconnects to $NPN$. If $CN$ has not submitted votes to $FN$ yet, it is all right for $CN$ to collect votes from all children and to submit the votes to $NPN$. Otherwise, $CN$ resubmits votes to $NPN$ and the previously submitted votes to $FN$ should be eliminated thoroughly. If $FN$ is not disconnected from the network, $P[FN]$ may find $FN$ is still alive, so $FN$ and its ancestors can delete the copy of votes of $CN$ from their vote pools. Assume before this deleting process is performed till some ancestor $CN'$, $CN'$ finds its parent $FN'$ is also faulty and reconnects to $NPN'$, then the deleting process will be continued on the new path. Meanwhile, $FN'$ will start a new deleting process for the votes of $CN'$, which contains the votes of $CN$. If $FN$ is disconnected from the network, then $P[FN]$ will delete votes of $FN$, which also contains the votes of $CN$. Such recursive call of the deleting process ensures the old copy of the votes of $CN$ is deleted completely.

Obviously, the resubmitting process and the deleting process for the votes of $CN$ are executed along two different paths to root, so if the processes can reach the root, they are not likely to always arrive at the same time. To avoid losing votes or introducing redundant votes, after all children have submitted votes, the root node should wait for a period of time that is long enough for the two processes to be finished. But the problem is that while the root is waiting for the repairing processes for one node, another node may happen to fail. The passive repairing processes will be called again, despite that the votes submitted by the node may have reached the root. As a result, the root will be trapped in an ever-waiting deadlock.

We adjust the passive repairing protocol by introducing some active ingredient to avoid ever-waiting: If a node has submitted votes to its parent long before it finds the parent is dead (i.e., beyond a time limit $DT$, which is much longer than the collecting time of the node), then it simply assumes that the parent has also submitted votes and there are several ancestors that have received the votes. Because the parent (if it is not disconnected from the network) and every one of the ancestors that have kept the votes longer than $DT$ will try their best (just as the node itself does) to relay the votes to the root, it is not necessary for the node to resubmit votes.

With this compromised protocol, the loss of votes will still occur if several nearest ancestors of a node disconnect from the network concurrently, but the probability of such loss is greatly less than before (the above mentioned scenario).

2.3 Efficiency

Recall that before a node submits votes to its parent, it will sum up the votes in the vote pool together with its own votes. OntoVote does not export the summation method but leaves it to application level. The implementation of the method is highly related to the efficiency of OntoVote: when more and more votes are collected, message packets that encapsulate votes will become larger and larger.
Without additional disposal, the packets will overwhelm the network and the scalability of OntoVote will be no better than that of a client-server model. So we should follow several principles in the design of this method.

To understand our principles, one can view a message packet that contains votes as a sheet. Every vote has its entry on the sheet. The number of entries a sheet can hold is limited. By combining the entries on several sheets together to form a new sheet and by relaying the new sheet to parent node, a voting participant fulfils his collecting task. Here are the design principles described with the above metaphor:

- Merge identical entries. To save on the space of a sheet, votes devoted to the same candidate should be merged together, that is, each entry should maintain a counter of the votes that fall in this entry. Actually, this is the original meaning of votes counting.
- Filter minor entries. Because the number of the entries a sheet can hold is limited, one should filter out the least counting entries when the sheet overflows. The contents of the least counting entries are likely to be noisy or unimportant to most users, so it is acceptable to filter them out.
- Choose a proper-sized sheet. Each application should choose a proper-sized sheet by simulation or by probability analysis to avoid a phenomenon that we call “Entry Jolts”: although some vote may be very large altogether, it happens to be filtered out every time before it can accumulate. “Entry Jolts” is determined by such factors as data makeup, data distribution on the network, filtering algorithms and sheet sizes, etc. To reduce the probability of “Entry Jolts”, a large-sized sheet is preferred, but we’d better get an upper bound of the size of the sheet, size larger than which brings about no evidently-good performance but more consumption of the network bandwidth.

In addition to the implementation of the summation method, other factors such as the balance of a broadcast tree, which has been addressed above, also affect the efficiency of OntoVote. But they are beyond our considerations.

3 Application of OntoVote to Ontology Drift

In section 1, we give our thought that collecting votes from all users can drive the drift of a common ontology and in section 2, we show that it is possible to collect votes in P2P environments with OntoVote. In this section, we will try to apply OntoVote to the process of ontology drift. Our implementation of ontology drift is part of the knowledge acquisition module developed under our undergoing PICQ project. PICQ is dedicated to paper sharing on P2P platform with semantic web technologies. In PICQ, users are grouped according to research interest. Every group has a common ontology. New fields or new application requirements will introduce new concepts or attributes in the ontology. The common ontology is used to provide more powerful semantic search service.

3.1 Process of Ontology Drift

We use the following process to cope with the drift of a common ontology:

1. When a new group is created, the creator provides a basic common ontology.
2. When a new user joins the group, he is provided with the currently available common ontology. If the user is dissatisfied with the common ontology, he can modify it with ontology visualization tools to create a local ontology. The visualization tools can map ontology elements to application elements (e.g., directories, bookmarks, etc) to hide ontologies from elementary users. They can also provide powerful support for expert users to modify ontologies directly and easily. In any case, the modifications of the local ontology are translated into the user’s votes on the common ontology. By tracking user modifications, system can also partly do the mapping between the local ontology and the common ontology.
3. At a proper time, all votes are collected and the common ontology is modified with the voting results.
4. After the new common ontology is published to all peers, the mapping between the local ontology and the new common ontology is adjusted again.
5. Iterate the above process.

To put it simply, the whole is an iterated process. Let $LO$ (Local Ontology) denote the local ontology of a user and let $CO$ (Common Ontology) denote the common ontology of a community. $LC$ is the mapping between $LO$ and $CO$. The iterated process can be described as follows:

1. $LO=CO, LC=Identity Mapping$
2. User modifies $LO$. System records user modifications and automatically adjusts $LC$.
3. At a proper time, system translates modifications into votes, collects all votes and modifies $CO$ background.
4. Publish $CO$ to every peer and adjust $LC$ again.
5. Goto (2).

There are some issues that need to be further addressed in the management of the ontologies involved in this process, e.g., how to track user modifications? Why do we keep the mapping between the common ontology and the local ontology and how to do this mapping (Obviously, it is not enough to do the mapping between two ontologies just by tracking the changes of either ontology)? How to derive a user’s modification proposals (or votes) on the common ontology from his modifications of the local ontology? What the system should do if there is a conflict of opinions? In the following sections, we will discuss these problems one by one.
3.2 Tracking User Modifications

The problem for tracking changes within ontologies or within a knowledge base has been addressed in [Kiryakov and Ognyanov, 2002]. [Kiryakov and Ognyanov, 2002] proposes using RDF statements (i.e. triples) instead of resources or literals as the smallest trackable pieces of knowledge. The two basic types of updates in a repository are addition and removal of a statement. To track series of updates that are bundled together according to the logic of the application, it uses batch update that works with the repository in a transactional fashion.

![Figure 4. Two Different Local Ontologies](image)

In regard to reflecting the modification proposals of users, we think atomic update (additions or removals of statements) plus batch update is almost appropriate and only a small change is made: because we treat modification proposals as votes and OntoVote requires identical votes to be merged, but two batch updates that represent the same proposals may not match exactly (e.g., in Figure 4, both of two users propose the sub-tree rooted at the concept “P2P” be deleted, but the two sets of removed triples can’t match exactly. One set contains two more triples than the other.), so we propose the application should induce some patterns from batch updates, which reflect the intention of a user. Two batch updates are matched if and only if both their patterns and the parameters of the patterns are matched. For instance, the deleting of the two sub-trees rooted at “P2P” in Figure 4 can be matched if a pattern for the deleting of a sub-tree is defined and the root node of the sub-tree is used as a parameter of the pattern.

![Figure 5. Tracking Modifications from Initial Common Ontology to Current Local Ontology](image)

Our approach for tracking modifications is illustrated in Figure 5. The left structure of the figure denotes the initial common ontology that the user imported from the community; the right structure is the local ontology. A history of modifications is recorded in a way something like that of [Kiryakov and Ognyanov, 2002]:

**History:**
1. remove sub-tree(C): remove <C is_a A>, remove<D is_a C>
2. add <E is_a A>

3.3 Maintenance of Ontology Mapping

In P2P applications, every user should be allowed to hold a local ontology to keep his personality. He uses this local ontology to raise queries. The queries are translated into the common ontology before being sent to other peers to improve search efficiency and the recall of search results. So it is of great importance to maintain the mapping between the local ontology and the common ontology.

By tracking the modifications of the local ontology and the common ontology in step (2) and step (3) in the process of ontology drift, we can partly do the mapping between them. For instance, if a resource in either ontology is renamed, the mapping can be adjusted. But how about adding a new resource? How can we know whether or not the new resource in one ontology can be mapped to some old resource in another ontology? This problem may be solved with emergent semantics [Maedche and Staab, 2001; Fensel et al., 2002]: after the user adds a new resource, he queries with this new resource for a period of time. During this period, emergent semantics helps to find out the mappings between the new resource and some other resources in the common ontology or other local ontologies (e.g. same file categorized to different concepts indicates alignment). The derived mappings are expressed with probabilities, based on the number of the instances that indicating alignment. Different peers may find different mappings, or same mappings with different probabilities. At a proper time, all new resources and the mappings among them are collected and coordinated with a new round of voting. Among the resources that can be mapped to each other, the one that wins the vote is adopted by the common ontology, the noisy semantics (votes that are below a threshold) is discarded and the rest are mapped to the one accepted. This process is something like the revision of a dictionary in social life: after a new word emerges, people use this word in their communication and every one gets a scrap of the meanings of the word (e.g., find synonyms of the word). When a dictionary is revised, all meanings of the word is collected and validated, and if necessary, the word is lexicalized.

In practice, the mapping results that are automatically obtained and maintained may not always be sound, so a semi-automatic ontology mapping mechanism is preferred, i.e., advanced users are allowed to manually correct the
mapping results before or after any round of votes collecting.

### 3.4 Generation of Votes

Before votes collecting, the modifications of the local ontology should be translated into the modification proposals (votes) on the common ontology. Or else, just as section 3.2 has stated, although the proposals of two users are identical, they cannot be merged.

Currently, our system generates the votes in a rather simple way, which is mainly based on the mapping between the common ontology and the local ontology. Below we discuss how our approach works in various conditions.

Firstly, if the mapping between the common ontology and the local ontology is well maintained, and the pattern of the modification has been defined by the application, then the translation is straightforward. For instance, in Figure 6, assume concept “DC” is mapped to concept “Distributed Computing” and concept “P2P” is mapped to concept “Peer-to-Peer”. If a user adds two new concepts “Pure” and “Hybrid” under concept “P2P” in his local ontology, then we think the user proposes adding these concepts under concept “Peer-to-Peer” in the common ontology; if the user deletes the tree rooted at “DC” completely, then we think the user proposes deleting the tree rooted at “Distributed Computing” completely.

![Figure 6. Common Ontology and Local Ontology](image)

Secondly, if no appropriate mapping information is obtained, then we either ignore the modification or transform the modification into a list of sub-modifications, according to the specification of the modification pattern. For example, in Figure 6, if no corresponding concept of “DC” is found in the common ontology, then we either discard the vote or split the vote to generate a new one to delete the sub-tree rooted at the concept “peer-to-peer”.

Thirdly, for some modifications (e.g., additions of statements), the mapping information is not necessary at all, i.e., additions of statements that have no relations with other resources in the common ontology are allowed in our system.

The last but not the least, advanced users are allowed to modify the local copy of the common ontology directly, if they are pleased to do that. We also allow a user to replace his local ontology with the common ontology, if he is dissatisfied with his local ontology or he does not want his opinions diverge too much from those of the masses, thus the old modification records are cleared and the new modifications of the local ontology can be more easily translated into those of the common ontology.

It should be noted that our system tries to translate every record in the modification history into a vote. If the modifications do not interact, this approach may work well. However, sometimes an appropriate combination of additions and removals may trigger complex “non monotonic” updates of the ontology, e.g., a user first adds “B” as a sub-concept of “A” and adds “C” as a sub-concept of “B”, and later, he finds that “C” is unnecessary, so he deletes it and adds “B” as a sub-concept of “A” directly. This sequence of modifications will be translated into a list of votes. If most members of the community believe that “B” is a sub-concept of “A”, it is likely that the last proper modification of the user will be accepted. However, if many users make the similar wrong modifications before, the wrong modifications may also be accepted. To filter out the wrong modifications, we’d better find out the last determination of the user (or the real intention of the user) before we construct a vote from a sequence of related modifications. Unfortunately, our system has not realized this object yet, and we will leave it to the future.

### 3.5 Resolving Conflicts

It is obvious that there are conflicts among all collected proposals, e.g., some users suggest deleting concept “Peer-to-Peer”, while others suggest adding “Pure” as a sub-concept of “Peer-to-Peer”.

One straightforward way to resolve a conflict is to choose among the conflicting proposals the one with the largest proportion. However, such simple conflict-resolving mechanism will bring about the instability of the common ontology, especially when the proportion of the adopted proposal is not overwhelming.

To understand this problem, assume there are totally 100 users in the community. At the beginning, 60 percent of them suggest adding the concept “Peer-to-Peer” (or concepts that are equivalences of “Peer-to-Peer”) to the common ontology, thus “Peer-to-Peer” is accepted in the common ontology. In the second round of voting, assume 30 users suggest deleting the concept “Peer-to-Peer” (these 30 users may come from the previously 60 percent of users, or advanced users who modify common ontology directly, or users who reimport and modify their local ontologies, etc.), while 10 users add sub-concepts “Pure” under “Peer-to-Peer” and the rest make no modifications. After the voting, if the system chooses to delete “Peer-to-Peer”, then the local ontologies that still record the addition of “Peer-to-Peer” will propose adding “Peer-to-Peer” to the common ontology again in the next round of voting. Because the proportion of this proposal is still large enough, it may be adopted again by the common ontology, which causes the instability of the common ontology.
Intuitively, we can get rid of the instability problem by tracking the history of voting. But till now, this idea has not been tested yet. Instead, we take a rather simpler measure to ease the instability problem, i.e., we set different thresholds for modifications with different patterns to be accepted in the common ontology. Because the common ontology is mainly used to improve search efficiency and the recall of search results, it is better to contain more resources than not. So we set low thresholds for additions of statements and high thresholds for deleting operations. In this way, when a conflict occurs, it is more likely that a proposal with an overwhelming vote proportion exists, and that the opposite proposals may be too trivial to bring about instability.

4 Conclusions and Future Work

Ontology drift is important in many requirement-sensitive P2P applications. We proposed collecting the modification proposals (votes) from all users to drive ontology drift. To collect votes on P2P platform, we presented OntoVote, a scalable distributed votes collecting mechanism based on application-level broadcast trees. OntoVote is reliable in that it leaves out as few votes as possible and avoids counting in the same copy of votes repeatedly. We also summarized several design principles of vote counting for OntoVote to work efficiently. And finally, we tried to apply OntoVote to the process of ontology drift with the discussion of several problems encountered in the application.

In future, we will research ontology drift further by obtaining more general modification patterns of ontology. we will also try to make the drifting process more stable. Besides, we will research some further issues of voting, such as voting security and the using of the opinions of authoritative members. There is also a problem that common ontology based on voting will neglect the views of an individual (or a small group of people) that brings real innovation and original perspectives on community’s point of view. We will find out whether intercommunication between peers helps to solve this problem.

References


Abstract

We think of \textit{match} as an operator that takes two graph-like structures (e.g., database schemas or ontologies) and produces a mapping between elements of the two graphs that correspond semantically to each other. The goal of this paper is to propose a new approach to matching, called \textit{semantic matching}. As from its name, the key intuition is to exploit the model-theoretic information, which is codified in the nodes and the structure of graphs. The contributions of this paper are (i) a rational reconstruction of the major matching problems and their articulation in terms of the more generic problem of matching graphs; (ii) the identification of semantic matching as a new approach for performing generic matching; and (iii) a proposal of implementing semantic matching via SAT.

1 Introduction

The progress of information and communication technologies has made accessible a large amount of information stored in different application-specific databases and web sites. The number of different information resources is rapidly increasing, and the problem of semantic heterogeneity is becoming more and more severe, see for instance [Halevy, 2001], [Washe \textit{et al.}, 2001], [Goh, 1997], [Giunchiglia and Zaihrayeu, 2002]. One proposed solution is matching. \textit{Match} is an operator that takes two graph-like structures (e.g., database schemas or ontologies) and produces a mapping between elements of the two graphs that correspond semantically to each other. So far, with the noticeable exception of [Serafini \textit{et al.}, 2003], the key intuition underlying all the approaches to matching has been to map labels (of nodes) and to look for similarity (between labels) using syntax driven techniques and syntactic similarity measures; see for instance [Do and Rahm, 2002], [Madhavan \textit{et al.}, 2001]. Thus for example, some of the most used techniques look for common substrings (e.g., \underline{\texttt{phone}}\underline{\texttt{e}}\underline{\texttt{e}}\underline{\texttt{e}} and \underline{\texttt{telephone}}\underline{\texttt{e}}\underline{\texttt{e}} for strings with similar soundex (e.g., \underline{\texttt{U}}\underline{\texttt{e}}\underline{\texttt{e}} and \underline{\texttt{for}}\underline{\texttt{e}}\underline{\texttt{u}} or expand abbreviations (e.g., \underline{\texttt{P}}\underline{\texttt{O}}\underline{\texttt{e}}\underline{\texttt{e}} and \underline{\texttt{Post}}\underline{\texttt{Office}}\underline{\texttt{e}}. We say that all these approaches are different variations of \textit{syntactic matching}. In syntactic matching semantics are not analyzed directly, but semantic correspondences are searched for only on the basis of syntactic features.

In this paper we propose a novel approach, called \textit{semantic matching}, with the following main features:

\begin{itemize}
  \item We search for semantic correspondences by mapping meanings (concepts), and not labels, as in syntactic matching. As the rest of the paper makes clearer, when mapping concepts, it is not sufficient to consider the meanings of labels of the nodes, but also the positions that the nodes have in the graph.
  \item We use semantic similarity relations between elements (concepts) instead of syntactic similarity relations. In particular, we consider relations, which relate the extensions of the concepts under consideration (for instance, more/less general relations).
\end{itemize}

The contributions of this paper are (i) a rational reconstruction of the major matching problems and their articulation in terms of the more generic problem of matching graphs; (ii) the identification of semantic matching as a new approach for performing generic matching; and (iii) a proposal of using SAT as a possible way of implementing semantic matching. It is important to notice that SAT is a correct and complete decision procedure for propositional logics. Using SAT allows us to find only and all possible mappings between elements. This is another major advantage over syntactic matching approaches, which are based on heuristics. The SAT-based algorithm discussed in this paper is a minor modification/extension of the work described in [Serafini \textit{et al.}, 2003].

The rest of the paper is organized as follows. Section 2 introduces some well-known matching problems and shows how they can be stated in terms of the generic problem of matching graphs. Section 3 defines the notion of matching and discusses the essence of semantic matching. Section 4 provides guidelines to the implementation of semantic matching. Section 5 overviews the related work. Section 6 reports some conclusions.
2 Matching Problems

Major data and conceptual models representing information sources across the WWW are database schemas, XML schemas, and ontologies.

2.1 Relational DB schemas

Let us consider the hypothetical relational database (RDB) BANK presented in Figure 1, storing information about the location of branches and of the staff that works at the BANK.

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>BN</th>
<th>Street</th>
<th>City</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B8</td>
<td>Piazza Venezia</td>
<td>Trento</td>
<td>38100</td>
</tr>
<tr>
<td></td>
<td>B2</td>
<td>Piazza Cordusio</td>
<td>Milano</td>
<td>20123</td>
</tr>
</tbody>
</table>

Figure 1. RDB BANK

We can represent the schema and data instances of the above database as a graph in two possible ways. In the first case, starting from the name (root), the schema is partitioned into relations and further down into attributes and data instances. See Figure 2. Arcs of Level 1 encode relations; arcs of Level 2 stand for attributes, and arcs of Level 3 specify data instances. Blank nodes stand for primary keys. Blank nodes with dashed circles stand for foreign keys. Notice that we know in advance that the maximum height of the tree is 3.

Figure 2. Tree representation 1 of the RDB BANK

In the second approach, as from [Buneman et al., 1996], starting from the root, a database is partitioned into relations, then into tuples, and finally into attributes and data instances. See Figure 3. For lack of space not all attributes and their identifiers are presented in the diagram. Notice that the maximum height of the tree is 4.

The information about the structure of the database resides only at arcs’ labels. Dashed arcs stand for primary keys. R1 and R2 denote relations of the database BANK. ROOT.RI.TJ.AK is a path to the K-th attribute of the J-th tuple of the I-th relation of the root of the tree. Data instances are presented as arcs at Level 4. Thus, the instances of the element BRANCH are represented by tuples: (BN B8 ± Piazza Venezia ± Trento ± 38100) and (BN B2 ± Piazza Cordusio ± Milano ± 20123).

Figure 3. Tree representation 2 of the RDB BANK

Which of the two representations is more preferable depends on the concrete task, but its worth to note that it’s always possible to transform one model into another.

Database schemas are seldom trees. If referential constraints are taken into account, schemas become DAGs. If we further consider recursive references we have cycles, see Figure 4. Referential constraints are shown as dashed arrows. Bold arrows represent recursive references, which appear if, for instance, we add to the relation STAFF the attribute Manager that expresses administrative relationships between employees.

Figure 4. Graph representation of the RDB BANK

2.2 OODB schemas

Let us rebuild the relational database BANK example in terms of an object-oriented approach. Now, BANK consists of the three classes, expressing the same data as above:

BRANCH(Street, City, Zip)  
PERSON(F_Name, L_Name)  
STAFF:PERSON(Position, Salary, Manager).

A graph representation of the given OODB schema is shown in Figure 5. Arcs with blank arrows stand for the use case generalization; dashed arrows play notationally the same role as associations in UML.

Figure 5. Digraph representation of the OODB BANK
The object-oriented data model captures more semantics than the relational data model. It explicitly expresses subsumption relations between elements, and admits special types of arcs for part/whole relationships in terms of aggregation and composition.

### 2.3 XML schemas

Neither the OO data model, nor the relational data model captures all the features of semistructured or unstructured data [Buneman, 1997]. Semistructured data don’t possess regular structure; the structure could be partial or even implicit. Missing or duplicated fields are allowed. Semistructured data could be schemless, or have a schema that poses only loose constraints on data. Typical examples are markup languages, e.g. HTML or XML.

XML schemas can be represented as DAGs. The graph in Figure 2 could also be obtained from an XML schema. Often, XML schemas represent hierarchical data models. In this case the only relationships between the elements are \{is-a\}. A DAG is obtained through the ID/IDREF mechanism. Attributes in XML are used to represent extra information about data. There are no strict rules telling us when data should be represented as elements, or as attributes.

### 2.4 Concept Hierarchies

A concept hierarchy is a way of defining a conceptualization of an application domain in terms of concepts and relationships expressed in a formal language. Concept hierarchies usually support \{is-a\} relations. Traditional examples of concept hierarchies are classifications, for instance, parts of Yahoo and Google electronic catalogs. Figure 6 presents a part of Google web directory devoted to business.

![Figure 6. Google web directory](image)

The concept hierarchy shown in Figure 6 consists of 11 concepts, and 10 subsumption relations, one per arc.

### 2.5 Ontologies

By an ontology we mean here a way of defining a conceptualization of an application domain in terms of concepts, attributes, and relations expressed in a formal language. Relations can be defined by the user, but there are some predefined relationships with known semantics, i.e., \{is-a; part-of; instance-of\}. A concept hierarchy is an ontology without attributes and only with \{is-a\} relations between elements.

One example of ontology can be constructed by complicating the concept hierarchy shown in Figure 6, by adding attributes to the concept Association, see Figure 7. Attributes of the concept Associations are BN, City, Street, Zip, while data instances are B8 and B2. Data instances have fixed attributes values: instance B8 has BN=B8±City=Venezia±Street=Piazza Venezia± etc.

![Figure 7. Example of ontology Business](image)

### 3 Matching

All the data and conceptual models discussed in the previous section can be represented as graphs. Therefore, the problem of matching heterogeneous and autonomous information resources can be decomposed in two steps:

1. extract graphs from the data or conceptual models,
2. match the resulting graphs.

Notice that this allows for the statement and solution of a more generic matching problem, very much along the lines of what done in Cupid [Madhavan et al., 2001], and COMA [Do and Rahm, 2002]. However, as already discussed in some detail in Section 2, each of the five matching problems presented there, has different properties and it is still an open problem whether we will be able to develop a general purpose matcher, and exploit most (all?) the problem and domain dependent analysis in step (1).

Let us define the notion of matching graphs more precisely. Mapping element is a 4-tuple \(<m_{ID}, N^1_j, N^2_i, R>\), \(i=1...h; j=1...k\); where \(m_{ID}\) is a unique identifier of the given mapping element; \(N^1_j\) is the \(i\)-th node of the first graph, \(h\) is the number of nodes in the first graph; \(N^2_i\) is the \(j\)-th node of the second graph, \(k\) is the number of nodes in the second graph; and \(R\) specifies a similarity relation of the given nodes. A Mapping is a set of mapping elements. Matching is the process of discovering mappings between two graphs through the application of a matching algorithm. There exist two approaches to
In P subgraph matching problems: find all occurrences of a pattern graph \( P \) of \( m \) nodes as a subgraph of a graph \( G \) of \( n \) nodes. In the case of exact matching we look for subgraphs \( S \) of \( G \) that are identical to \( P \). In inexact matching some errors are acceptable. For obvious reasons we are interested in inexact matching.

We classify matching into syntactic and semantic matching depending on how matching elements are computed and on the kind of similarity relation \( R \) used.

In syntactic matching the key intuition is to map labels (of nodes) and to look for the similarity using syntax driven techniques and syntactic similarity measures. Thus, in the case of syntactic matching, mapping elements are computed as 4-tuples \( < m_{ip}, L_i', \_L_j', R > \), where \( L_i' \) is the label at the \( i \)-th node of the first graph; \( L_j' \) is the label at the \( j \)-th node of the second graph; and \( R \) specifies a similarity relation in the form of a coefficient, which measures the similarity between the labels of the given nodes. Typical examples of \( R \) are coefficients in \([0,1]\), for instance, similarity coefficients [Madhavan et al., 2001]. Simularity coefficients usually measure the closeness between the two elements linguistically and structurally. For instance, based on linguistic analysis, the similarity coefficient between elements "telephone" and "phone" from the two hypothetical schemas could be 0.7.

As from its name, in semantic matching the key intuition is to map meanings (concepts). Thus, in the case of semantic matching, mapping elements are computed as 4-tuples \( < m_{ip}, C_i', C_j', R > \), where \( C_i' \) is the concept of the \( i \)-th node of the first graph; \( C_j' \) is the concept of the \( j \)-th node of the second graph; and \( R \) specifies a similarity relation in the form of a semantic relation between the extensions of concepts at the given nodes. Possible \( R \)'s between nodes are equality (=), overlapping (\( \mathcal{E} \)), mismatch (\( \ast \)), or more general/specific (\( \hat{\mathcal{E}} \)).

These ideas are schematically represented in Figure 8. It is important to notice that all past approaches to matching we are aware of, with the exception of [Serafini et al., 2003], are based on syntactic matching.

![Figure 8. Matching problems](image)

One of the key differences between syntactic and semantic matching is that in syntactic matching, when we match two nodes, we only consider the labels attached to them, independently of the position of the nodes in the graph. In semantic matching, instead, when we match two nodes, the concepts we analyse depend not only on the concept attached to the node (the concept denoted by the label of the node), but also on the position of the node in the graph. Let us consider the example in Figure 9. Numbers in circles are the unique identifiers of the nodes under consideration. \( A \) stands for the label at a node; \( A_j \) stands for the concept denoted by \( A \); \( C_i \) stands for the concept at the node \( i \) (in the following we sometimes confuse concepts with their extensions).

![Figure 9. Syntactic vs. semantic matching](image)

Let us consider for instance, the analysis carried out when the node numbered 5 is submitted to matching (against a node in another graph). In syntactic matching the matcher tries to match the label at node 5, namely \( C \). In semantic matching, instead, the matcher tries to match the concept at node 5, namely \( C_5 \), which is that subset of the extension of \( A_j \) which is also in the extension of \( C_i \). Thus, \( C_5 = A_j \mathcal{E} C_i \). A semantic matcher will therefore try to match \( A_j \mathcal{E} C_i \) and not (!) \( C \).

Let us consider some more examples, which make the consequences of the observation described in the previous paragraph clearer. For any example we also report the results produced by the state of the art matcher, Cupid [Madhavan et al., 2001], which exploits very sophisticated syntactic matching techniques. Notationally in the following we write \( A_1 \) to mean that the string \( A \) occurs in the graph on the left, and \( A_2 \) to mean that \( A \) occurs in the graph on the right. We use the same notation also for nodes of graphs, concepts denoted by labels, and concepts at nodes. Thus for instance, \( 5_1 \) stands for the node 5 in the graph on the left, \( A_5 \) is the concept denoted by the label \( A \), while \( C_5 \) is the concept at node 5.

**Analysis of siblings.** Let us consider Figure 10. Structurally the graphs shown in Figure 10 differ in the order of siblings. Suppose that we want to match node 5_1 with node 2_2.

![Figure 10. Analysis of siblings. Case 1](image)

Cupid correctly processes this situation, and as a result, the similarity coefficient between labels at the given nodes equals to 0.8. This is because \( A_1 = A_2 \), \( C_1 = C_2 \) and we have the same structures on both sides. A semantic matching approach compares concepts \( A_j \mathcal{E} C_i \) with \( A_j \mathcal{E} C_i \) and produces \( C_5 = C_5 \).

**Analysis of ancestors.** Let us consider Figure 11. Suppose that we want to match nodes 5_1 and 1_2.
Cupid does not find a similarity coefficient between the nodes under consideration, due to the significant differences in structure of the given graphs. In semantic matching, the concept denoted by the label at node $5_1$ is $C_h$, while the concept at node $5_2$ is $C_5 = A \cap \mathcal{E} \cap \mathcal{F} \cdot C_h$. The concept at the node $1_2$ is $C_{21} = C_F$. By comparing the concepts denoted by the labels at nodes $5_1$ and $1_2$ we have that, being identical, they denote the same concept, namely $C_h = C_F$. Thus, the concept at node $5_1$ is a subset of the concept at node $1_2$, namely $C_5 \subseteq C_{12}$.

Let us complicate the example shown in Figure 11 by allowing for an arbitrary distance between ancestors, see Figure 12. The asterisk means that an arbitrary number of nodes are allowed between nodes $1_2$ and $5_2$. Suppose that we want to match nodes $5_1$ and $5_2$.

Cupid finds out that the similarity coefficient between labels $C_1$ and $C_2$ is 0.86. This is because of the identity of the labels ($A_1 = A_2$ and $C_1 = C_2$), and due to the fact that nodes $5_1$ and $5_2$ are leaves. Notice how Cupid treats very different the two situations represented here and in the example above, even if, from a semantic point of view, they are similar. Following semantic matching, the concept at node $5_1$ is $C_5 = A \cap \mathcal{E} \cap \mathcal{F} \cdot C_h$; while the concept at node $5_2$ is $C_5 = A \cap \mathcal{E} \cap \mathcal{F} \cdot C_h$. Since we have that $A_h = A_B$ and $C_h = C_F$, then $C_5 \subseteq C_{12}$.

**Enriched analysis of siblings.** Suppose that we want to match nodes $1_2$ and $2_2$, see Figure 13.

Cupid without thesaurus doesn’t find a match; with the use of thesaurus it finds out that the similarity coefficient between nodes with labels Benelux$_1$ and Belgium$_2$ is 0.68. This is mainly because of the entry in the thesaurus specifying Belgium as a part of Benelux, and due to the fact that the nodes with labels Benelux$_1$ and Belgium$_2$ are leaves. Following semantic matching, both concepts Benelux$_h$ and Benelux$_F$ are subsets of the concept World$_h$.

Let us suppose that an oracle, for instance WordNet, states that Benelux is a name standing for Belgium, Neth-

erlands and Luxembourg. Therefore, we treat $C_i$ in Figure 14 as Benelux$_i$ $\in \{ \text{Netherlands}_i, \text{Luxembourg}_i \}$ $\in$ Belgium. Thus, $C_i = C_h$.

**Analysis of attributes.** Let us consider Figure 14. On the left we have a graph, which represents an ontology where State and Square are attributes of the concept Europe. State has two sets of items corresponding to Italy and Belgium. On the right we have a graph, which represents the concept hierarchy World, where the concept Italy is populated with a set of items about Italy. Attributes can be matched with attributes, but also with concepts. Suppose that we want to match nodes $7_1$ and $4_2$.

**4 Implementing Semantic Matching**

There are two levels of granularity while performing semantic (and also syntactic matching) matching: *element-level* and *structure-level*. Element-level matching techniques compute mapping elements between individual labels/concepts at nodes; structure-level techniques compute mapping elements between subgraphs.

**4.1 Element-level Semantic Matching**

Element-level semantic techniques analyze individual labels/concepts at nodes. At the element-level we can exploit all the techniques discussed in the literature, see for instance [Do and Rahm, 2002], [Melnik et al., 2002], [Rahm and Bernstein, 2001]. The main difference here is that, instead of a syntactic similarity measure, these techniques must be modified to return a semantic relation $R$, as defined in Section 3. We distinguish between *weak semantics* and *strong semantics* element-level techniques. Weak semantics techniques are syntax driven techniques: examples are techniques, which consider labels as strings, or analyze data types, or soundex of schema elements. Let us consider some examples.

**Analysis of strings.** String analysis looks for common prefixes or suffixes and calculates the distance between two
strings. For example, the fact that the string "phone" is a substring of the string "telephone" can be used to infer that "phone" and "telephone" are synonyms. Before analyzing strings, a matcher could perform some preliminary parsing, e.g., extract tokens, expand abbreviations, delete articles and then match tokens. The analysis of strings discovers only equality between concepts.

**Analysis of data types.** These techniques analyze the data types of the elements to be compared and are usually performed in combination with string analysis. For example, the elements "phone" and "telephone" are supposed to have the same data type, namely "string" and therefore can be found equal. However, "phone" could also be specified as an "integer" data type. In this case a mismatch is found. As another example the integer "Quantity" is found to be a subset of the real "Qty". This kind of analysis can produce any kind of semantic relation.

**Precompiled thesaurus.** A precompiled thesaurus usually stores entries with synonym and hypernym relations. For example, the elements "e-mail" and "email" are treated as synonyms from the thesaurus look up: syn key = "e-mail:email"=1. Precompiled thesauruses (most of them) identify equivalence and more general/specific relations. In some cases domain ontologies are used as precompiled thesauruses [Mena et al., 1996].

**WordNet.** WordNet is an electronic lexical database for English (and other languages), where various senses (namely, possible meanings of a word or expression) of words are put together into sets of synonyms (synsets). Synsets in turn are organized as hierarchy. Following [Serafini et al., 2003] we can define the semantic relations in terms of senses. **Equality:** one concept is equal to another if there is at least one sense of the first concept, which is a synonym of the second. **Overlapping:** one concept is overlapped with the other if there are some senses in common. **Mismatch:** two concepts are mismatched if they have no sense in common. **More general / specific:** One concept is more general than the other iff there exists at least one sense of the first concept that has a sense of the other as a hyponym or as a meronym. One concept is less general than the other iff there exists at least one sense of the first concept that has a sense of the other concept as a hyponym or as a holonym. For example, according to WordNet, the concept "hat" is a holonym for the concept "brim", which means that "brim" is less general than "hat".

### 4.2 Structure-level Semantic Matching

The approach we propose is to translate the matching problem, namely the two graphs and our mapping queries into a propositional formula and then to check it for its validity. By mapping query we mean here the pair of nodes that we think will match and the semantic relation between them. We check validity by using SAT. Notice that SAT is a correct and complete decision for propositional satisfiability and therefore will exhaustively check for all possible mappings. Being complete, SAT automatically implements all the examples described in the previous section, and more. This is another advantage over syntactic matching, whose existing implementations are based only on heuristics.

Our SAT based approach to semantic matching incorporates six steps. We describe below its intended behavior by running these six steps on the example shown in Figure 11 and by matching nodes $s_1$ and $s_2$ (steps 2-5 are taken from [Serafini et al., 2003]).

1. **Extract the two graphs.** Notice that during this step, in the case of DB, XML or OODB schemas, it is necessary to extract useful semantic information, for instance in the form of ontologies. There are various techniques for doing this, see for instance [Mena et al., 1996]. The result is the graph in Figure 11.

2. **Compute element-level semantic matching.** For each node, compute semantic relations holding among all the concepts denoted by labels at nodes under consideration. In this case $C_h$ has no semantic relation with $C_e$ while we have that $C_h \in C_e$.

3. **Compute concepts at nodes.** Starting from the root of the graph, attach to each node the concepts of all the nodes above it. Thus, we attach $C_o = A_h \in C_h$ to node $s_1$; $C_o = A_h \in C_h$ to node $s_2$; $C_o = C_e$ to node $s_3$. As it turns out we have that $C_o \in C_o$.

4. **Construct the propositional formula.** representing the matching problem. In this step we translate all the semantic relations computed in step 2 into propositional formulas. This is done according to the following transition rules:

\[
\begin{align*}
A_h \in C_e \quad & \quad A_h \in C_e \quad - \quad A_h \\
A_h \in C_e \quad & \quad A_h \in C_e \quad - \quad A_h \\
A_h = A_h \quad & \quad A_h \in C_e \\
A_h \cup A_h \quad & \quad A_h \cup A_h \\
A_h \cap A_h \quad & \quad A_h \cap A_h
\end{align*}
\]

Subset translates into implication; equality into equivalence; disjointness into the negation of conjunction. In the case of Figure 11 we have that $C_h \cup C_e$ is an axiom. Furthermore, since we want to prove that $C_o \in C_o$, our goal is to prove that $(A_h \cup C_h) - C_e$. Thus, our target formula is $((C_h \cup C_e) - (A_h \cup C_h))$. 

---

144
5. Run SAT. In order to prove that \((C \bar{h} \cap C\hat{h}) - (A\bar{h} \cap C\hat{h})\) is valid, we prove that its negation is unsatisfiable, namely that a SAT solver run on the following formula \((C \bar{h} \cap C\hat{h}) \otimes ((A\bar{h} \cap C\hat{h}) - C\hat{h})\) fails. A quick analysis shows that SAT will return FALSE.

6. Iterations. Iterations are performed re-running SAT. We need iterations, for instance, when matching results are not good enough, for instance no matching is found or a form of matching is found, which is too weak, and so on. The idea is to exploit the results obtained during the previous run of SAT to tune the matching and improve the quality of the final outcome. Let us consider Figure 15.

![Figure 15. Not good enough answer](image)

Suppose that we have found out that \(C_b \notin E C_b\), \(\bar{A}\), and that we want to improve this result. Suppose that an oracle tells us that \(A\bar{h} = F\hat{h} \cap G\hat{h}\). In this case the graph on the left in Figure 15 can be transformed into the two graphs in Figure 16.

![Figure 16. Extraction of additional semantic information](image)

After this additional analysis we can infer that \(C_b = C_b\). Another motivation for multiple iterations is to use the result of a previous match in order to speed up the search of new matches. Consider the following example.

![Figure 17. Iterations](image)

Suppose that \(F\bar{h} \models B\bar{h}\). Having found that \(C_s \models C_s\), we can automatically infer that \(C_s \models C_s\), without rerunning SAT, for obvious symmetry reasons.

5 Related Work

At present, there exists a line of semi-automated schema matching and ontology integration systems, see for instance \cite{Madhavan2001, DoAndRahm2002, LiAndClifton2000, Castano2000, Arens1996, Mena1996, Doan2002}. Most of them implement syntactic matching. A good survey, up to 2001, is provided in \cite{RahmBernstein2001}. The classification given in this survey distinguishes between individual implementations of match and combinations of matchers. Individual matchers comprise instance- and schema-level, element- and structure-level, linguistic- and constrained-based matching techniques. Individual matchers can be used in different ways, e.g., simultaneously (hybrid matchers), see \cite{LiAndClifton2000, Castano2000, Madhavan2001} or in series (composite matchers), see \cite{Doan2002, DoAndRahm2002}.

The idea of generic (syntactic) matching was first proposed by Phil Bernstein and implemented in Cupid system \cite{Madhavan2001}. Cupid implements a complicated hybrid match algorithm comprising linguistic and structural schema matching techniques, and computes normalized similarity coefficients with the assistance of a precompiled thesaurus. COMA \cite{DoAndRahm2002} is a generic schema matching tool, which implements more recent composite generic matchers. With respect to Cupid, the main innovation seems to be a more flexible architecture. COMA provides an extensible library of matching algorithms; a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers.

A lot of state of the art syntactic matching techniques exploiting weak semantic element-level matching techniques have been implemented. For instance, in COMA, schemas are internally encoded as DAGs, where the elements are the paths, which are analyzed using string comparison techniques. Similar ideas are exploited in Similarity Flooding (SF) \cite{Melnik2002}. SF is a hybrid matching algorithm based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs; the algorithm manipulates them in an iterative fix-point computation to produce mappings between the nodes of the input graphs. The technique uses a syntactic string comparison mechanism of the vertices’ names to obtain an initial mapping, which is further refined within the fix-point computation.

Some work has also been done in strong semantics element-level matching. For example, \cite{Castano2000} utilizes a common thesaurus, while \cite{Madhavan2001} has a precompiled thesaurus. In MOMIS \cite{Castano2000} element-level matching using a common thesaurus is carried out through a calculation of the name, structural and global affinity coefficients. The thesaurus presents a set of intensional and extensional relations, which depict intra- and inter-schema knowledge about classes, and attributes of the input schemas. All these systems implement syntactic matching and, when moving from element-level to structure-level matching, don’t exploit the semantic information residing in the

\[\text{Giunchiglia and Zaihrayeu, 2002}\] provides a long discussion about the importance of dealing with the notion of “good enough answer” in information coordination in peer-to-peer systems.
graph structure, and just translate the element-level semantic information into affinity levels.

As far as we know the only example where element-level and a simplified version of structure-level strong semantics matching have been applied is CTXmatch [Serafini et al., 2003]. In this work SAT is used as the basic inference engine for structure-level matching. The main problem of CTXmatch is that its rather limited in scope (it applies only to concept hierarchies), and it is hard to see the general lessons behind this work. For instance, the authors have made no attempt to do a thorough comparison of their approach with the other matching techniques, or to highlight its strengths and weaknesses. This paper provides the basics for a better understanding of the work on CTXmatch.

6 Conclusions and Future Work

In this paper we have stated and analyzed the major matching problems e.g., matching database schemas, XML schemas, conceptual hierarchies and ontologies and shown how all these problems can be defined as a more generic problem of matching graphs. We have identified semantic matching as a new approach for performing generic matching, and discussed some of its key properties. Finally, we have identified SAT as a possible way of implementing semantic matching, and proposed an iterative semantic matching approach based on SAT.

This is only very preliminary work, some of the main issues we need to work on are: develop an efficient implementation of the system, do a thorough testing of the system, also against the other state of the art matching systems, study how to take into account attributes and instances, analyze how to extract semantics from schemas (also taking into account integrity constraints), and so on.

Acknowledgments

Thanks to Luciano Serafini, Paolo Bouquet, Bernardo Magnini, Stefano Zanobini for many discussions on CTXmatch. Also thanks to Michail Yatskevich for his work on running SAT solvers on our matching problems.

References


