Abstract of thesis entitled

Effective Partial Ontology Mapping in a Pervasive Computing Environment

Submitted by

Kong Choi Yu

for the degree of Master of Philosophy
at The University of Hong Kong
in November 2004

A pervasive computing environment can be full of numerous embedded heterogeneous computing devices. This study proposes using ontology to enable effective communication and information sharing between devices to achieve invisible computing. Ontology provides a formal, explicit specification of a shared conceptualization of a domain. It facilitates knowledge sharing in open and dynamic distributed systems. Using ontology, devices can understand the messages without prior knowledge about the format or content of the messages. It also allows devices not originally designed to work together to interoperate. A global ontology for all applications and devices is impossible and inflexible for knowledge sharing in pervasive computing environments, and different ontologies for the same domain may require ontology mapping mechanisms to bridge their knowledge gaps.

Because of limited memory, mobile and wearable devices embedded in the pervasive computing environment can only afford to store partial ontology that extracts the frequently used concepts defined in the ontology. This implies that only partial information is available, which makes traditional ontology mappings not suitable in the pervasive computing environment.

This study proposes an online partial ontology mapping mechanism to meet the new challenges in ontology mapping in pervasive computing environments. Our proposed design takes similarities of the names, properties and relationships of concepts into consideration during mapping. It outperforms the previous source- and instance-based approaches in matching accuracy. It can also use history records to store the relevant information about particular instances instead of all information on all instances, which is more space efficient than traditional instance-based ontology mapping. Partial ontologies are cached to provide the missing knowledge during mapping.

An abstract of exactly 262 words

Laurel

Effective Partial Ontology Mapping in Pervasive Computing Environment

by

Kong Choi Yu

Temporary Binding for Examination Purpose

A thesis submitted in partial fulfillment of the requirement for
the Degree of Master of Philosophy
at The University of Hong Kong.

November 2004

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Signed: . . . .

Acknowledge

I would like to express my deepest thanks to my supervisors Dr. Francis Chi Moon Lau and Dr. Cho-Li Wang for their valuable support and guidance for my research. I must also thank my family for their understanding and support for my study.

Table of Content

List of Figures

List of Tables

INTRODUCTION

Pervasive computing is an environment saturates with computing and communication capabilities, yet so gracefully integrated with users that it becomes a "technology that disappears" [Satyanarayanan, 2001]. Satyanarayanan identified three main types of challenges in the design and implementation problems raised in pervasive computing environment [Satyanarayanan, 2001]. The first type is related to hardware and system designs, such as high-level energy management, client thickness, balancing proactivity and transparency. The second type is security issues like privacy and trust. The last type is about generating and exchanging information in pervasive computing paradigm, like user's intent (or preferences) and context information (state of environment), to achieve cyber foraging and adaptation strategy.

From the great variety of computing devices in pervasive computing environment, to list a few: handheld and mobile computers, smart and wearable devices, sensors and surrogates, etc. a language is needed to provide a common representation of information across them.

Nowadays, eXtensible Makeup Language (XML) [XML] is widely used for representing information interchange between applications through Internet. XML provides a set of semantics to represent data and allows data content to have arbitrary structure. It is human readable, flexible, easy to use and create; however, it does not define the meaning of the tags. The same XML tag can have different meanings in different data structure. On the other hand, different XML tags can have the same meaning. For example, an XML describing email structure contains a "SendTo" tag while another email structure in XML contains a tag named "Recipient". It is obvious to human that "SendTo" and "Recipient" have the same meaning, but for computers that do not understand English, "SendTo" and "Recipient" are meaning totally different as they have different syntax.

To solve the above problem, communicating parties can either use the same set of XML tags or manually handle the ambiguities on the meanings of XML tags before communications are taken place. Both these solutions are only possible when there are a few and static parties. Global XML tag is certainly not feasible. Due to the dynamic nature of pervasive computing environment: users move from place to place to get their tasks done and hence negotiations beforehand is impossible as well. Another language to represent the meanings of the XML tags is needed such that they can be interpreted by the computers is essential in pervasive computing environment. Ontology, hence, is being adopted.

Ontology represents the semantics of different concepts. It provides a formal, explicit specification of a shared conceptualization of a domain that can be communicated between people and heterogeneous and widely spread application systems [Gruber]. It is a formal explicit description of concepts (also called *classes*) in a domain of discourse, properties of each concept describing various features and attributes of the concept (also called slot), together with restrictions on concepts [Noy & McGunness, 2001]. Once computers can understand the information being exchanged, they can make decisions and responses intelligently, so they can communicate with neither prior knowledge about the format nor the content of the message. This enables knowledge sharing in open and dynamic distributed systems. Devices and agents do not need to be designed to work together can interoperate.

However, using ontology to represent information partially solve the problem of numerous and dynamic communications in pervasive computing environment. We require systems to provide instant knowledge reasoning for efficiently mapping between user queries and domain knowledge. Considering an analogy in human society, two people from different countries cannot communicate in an efficient manner if they simply look up for the meaning of words in dictionary during the communication. Effective communication requires them to have the knowledge of each other's background and customs. In computing environment, we need to bridge the knowledge gaps between different devices when two devices communicate. This is especially important in pervasive computing environment because we cannot provide a global meaning for a concept.

Ontology mapping helps for bridging knowledge gaps. Given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept, relation or instance) in Ontology O_1, a corresponding entity, which has the same intended meaning, in Ontology O_2 is found [Su, 2002]. Ontology mapping is significantly important in pervasive computing environment because devices are moving from place to place and they will come across different domains.

This study aims to find an effective way to bridge the knowledge gap between the devices. Four new challenges have been identified: (1) online mapping, (2) efficiency in mapping, (3) space limitation of devices and (4) knowledge propagation to support user mobility.

This thesis is divided into five sections. The next section provides background and the literature review about pervasive computing, ontology and ontology mapping. Section 3 presents the overview of the proposed design and technical details of each component. Section 4 evaluates the proposed design. The factors that are evaluated, the tools used, the implementation details and results are included in Section 4. Conclusion is made in the last section together with a discussion of future work.

BACKGROUND AND LITERATURE REVIEW

This study aims to provide effective internal and external communications within and across smart spaces. Smart spaces are computing spaces envisioning pervasive computing. Their related researches are firstly introduced in this chapter as the background. Effective internal communications refer to the presence of a standard language for services and resources lookup. The possibilities of adopting the information representation used in existing services discovery protocols and smart space researches are investigated. The widely accepted representation language is called ontology. The details of ontology and the reasons why it is suitable in smart spaces are studied.

Effective external communications require bridging knowledge bridges across different smart spaces. Ontology mapping tools and similarity functions are discussed in later part of this chapter.

SMART SPACE

A smart space is a logically boundary for different computing spaces in pervasive computing environment. For example, a meeting room, an office and a university campus are smart spaces. A smart space is embedded with heterogeneous computer devices that have various sizes and computation powers. There are many different terminologies refer to smart spaces, for example, intelligent environment [Oxygen], interactive space [Interactive Workspace], etc. In this research, we choose the term “smart space”.

Project Oxygen [Oxygen] proposed Agent-based Intelligent Environments (AIRE) [AIRE]. AIRE embed with different devices like cameras, microphones, lighting, door locks, etc. To let users to communicate with computing devices in the same ways as they are communicating with people, knowledge is shared between users and computers. A prototype intelligent environment called Intelligent Room [Brooks, 1997] is built. Stanford Interactive Workspaces [Johanson et al., 2002] explores new possibilities for people to work in technology-rich meeting spaces that consists of computing and interacting devices on various scales. It has contributed with a great effort in solving problems in switching displays between different sizes of screens. iRoom is the prototype of Interactive workspaces. Microsoft Easyliving project [Brumitt et al., 2000] explores the architecture and technologies for intelligent environments that contains many different types of devices and support rich interactions with users. Easyliving aggregates diverse devices into a coherent user experience.

It is believed that users are not stationary in one smart space. They move across different spaces. Existing smart spaces researches focus on the design of infrastructure and explore new technologies such as switching display from small to large screens, migrating tasks to more powerful devices and detecting the environment states using sensors. Their lackage about the interoperability between different smart spaces makes this study differentiates from them. Our design allows each smart space to have its own infrastructure such that tailor-make infrastructure can be designed to suit different needs.

KNOWLEDGE REPRESENTATION

Within a smart space, devices cooperate and users lookup computing services and resources. A representation is needed for exchanging information. The information representation strategies used in existing services discovery mechanisms and in pervasive computing researches are investigated.

Service Requesting and Discovery

1. Jini

In traditional computing paradigm, users need to install device drivers before using hardware like printers and scanners. Jini [Arnold et al., 1999] aims to allow everybody in the network to access device services without installing drivers. Devices register their services in the nearby proxies by specifying their capabilities and service interfaces using Java. Services are looked up based on the device capability specifications. Services are invoked using Java Remote Method Invocation (RMI). The workflow of Jini is shown in Figure 1. Client devices must have a priori knowledge about the service interfaces before services lookup. There are some attempts to standardize service interfaces such that no priori knowledge is required. However, it is extremely difficult to achieve a real practical standard because it lasts long for all devices manufactures to negotiate a standard. Moreover, a single device provides more than one services. For example, a mobile phone can also behave as a MP3, a digital camera, a radio and a personal digit assistant. It is hard to have a universal classification of the devices.

Jini only grantees there exists a service matches the service interface specified in the request. Services are chosen by random if they have identical service interfaces. The goal of pervasive computing is not simply providing service to users. It aims to provide the service that best suits the users' needs. For example, a printer service that is nearest to the user is chosen. The knowledge representation in Jini only focuses on the implementation level of services lookup. It is, definitely, not desirable in pervasive computing environment.

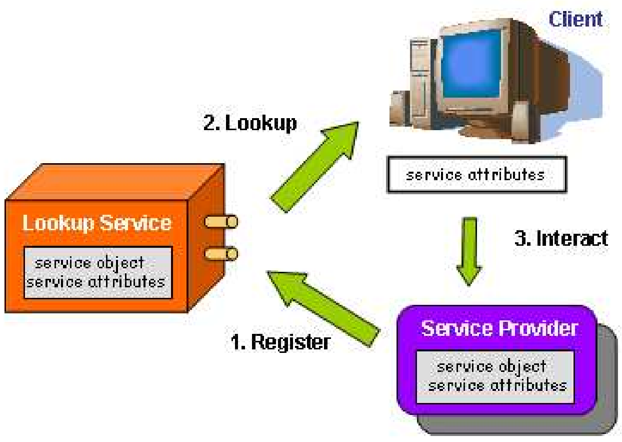


Figure 1 Workflow of Jini

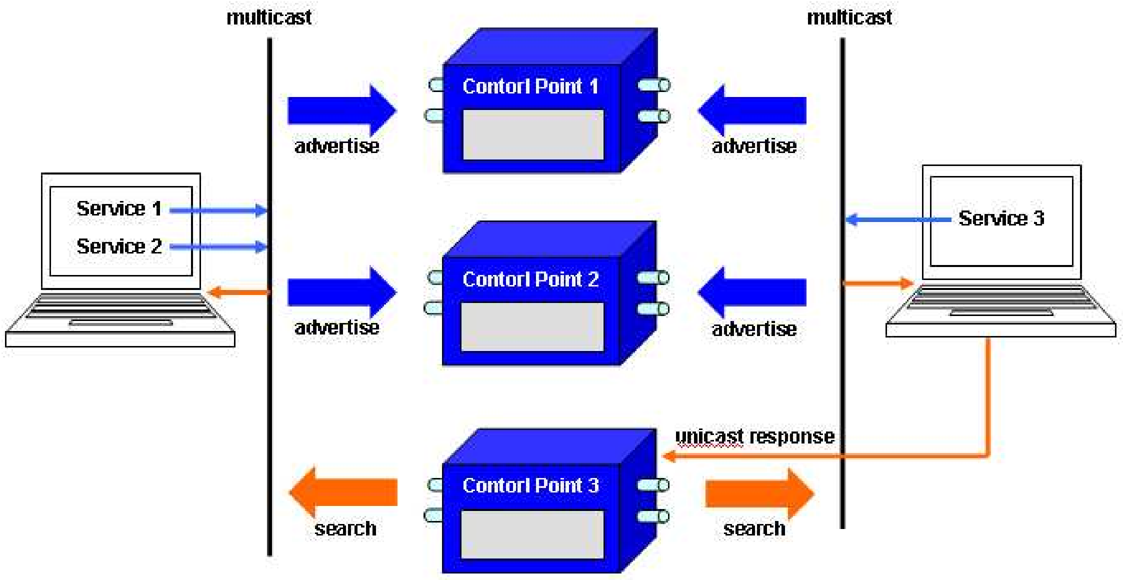


Figure 2 Architectur of UPnP

2. UPnP

UPnP (Universal Plug and Play) [UPnP] helps to allocate devices such that XML-based service requests can be received by the correct devices in the network. Devices use XML to describe their services using a list of actions the each service responds to and a list of variables that model the state of the service at run time. Service specifications are broadcasted to service proxies for other devices to lookup and retrieve. Similar to Jini, UPnP lookup services by the implementation details i.e. requests are matched based on the exact matching of the service interfaces. There are no mechanisms to automate choosing a better service for users. The architecture of UPnP is shown in Figure 2.

3. Web Services

Web services [Austin et al, 2004] are software components resident in servers. They provide online services to users so that client devices can keep as small as possible. Users send service requests to servers and get the execution results in return. Web services suit in smart spaces because they help small-embedded devices like mobile devices, wearable devices and sensors, etc. to complete computation expensive tasks. Moreover, as services are executed in service severs, users can do their task regardless their location. Web services use Simple Object Access Protocol (SOAP), XML and Web Service Definition Language (WSDL) to define service requests and descriptions. Requests are transmitted in SOAP messages enveloped in XML. Descriptions that prescribe how application or systems interact with the web service are defined using WSDL. Web services looked up using Universal Description, Discovery and Integration (UDDI). The architecture of the web services [captured from Champion et al., 2002] is shown in Figure 3.

In the introduction, the problems of representing knowledge using XML are addressed. XML tags defined using human language have potential ambiguity in their meanings. Human negotiation on the meanings of the tags is not scalable in distributed environment. An open standard for web services XML tags can partially solve the problem because it is inflexible for application and system design. Additions of new XML tags, deletions and modifications of existing XML tags affect the existing applications and systems. As a result, Ontology Web Language-based ontology for web service (OWL-S) [Martin et al., 2004] is proposed to allow automatic discovery, composition and invocation of web services.

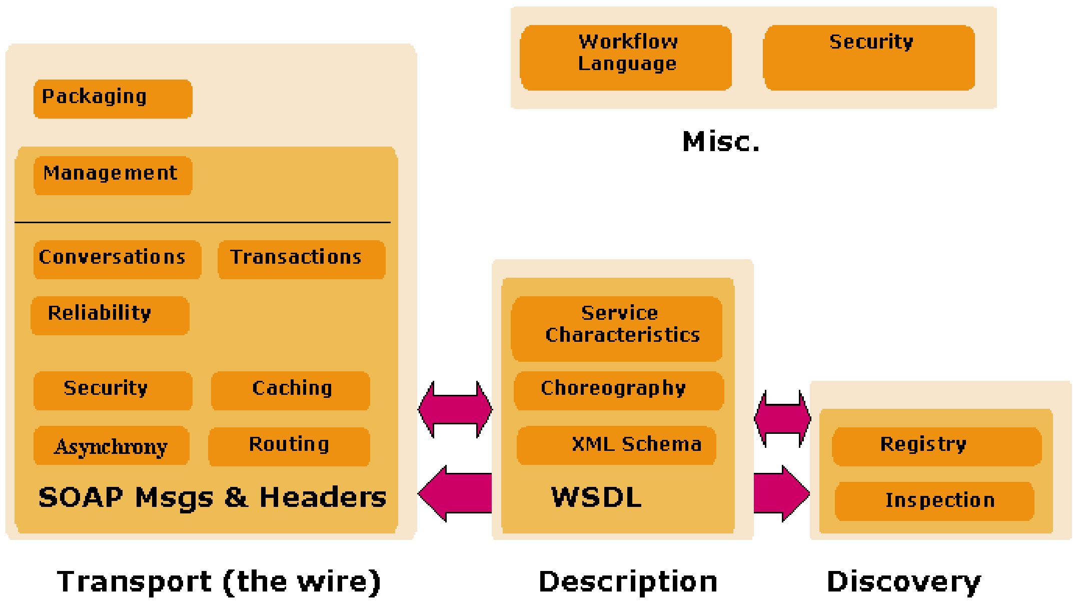


Figure 3 Architecture of Web Service

Smart Spaces

1. Project Oxygen

Project Oxygen [Oxygen] has a subsystem that supports accessing the users' personal knowledge in the same natural way as people access information. "*The subsystem stores information encountered by its users using an extensible data model that links arbitrary objects via arbitrarily named arcs. There are no restrictions on object types or names. Users and the system alike can aggregate useful information regardless of its form (text, speech, images, video). The arcs, which are also objects, represent relational (database-type) information as well as associative (hypertext-like) linkage. For example, objects and arcs in A's data model can represent B's knowledge of interest to A—and vice versa.*" [Peters and Shrobe, 2003]. In simpler words, the subsystem uses ontology written in Resource Description Framework (RDF) to represent knowledge. Project Oxygen has developed Haystack [Quan et al., 2003] to let users to manage their information. The semantic network of an Intelligent Environment defined by Peters and Shrobe [Peter and Shrobe, 2003] is shown in Figure 4.

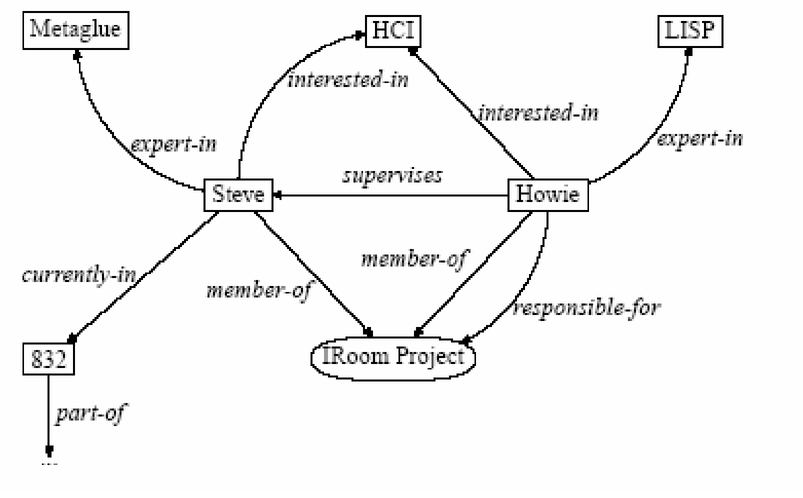


Figure 4 Semantic Network for an Intelligent Environment

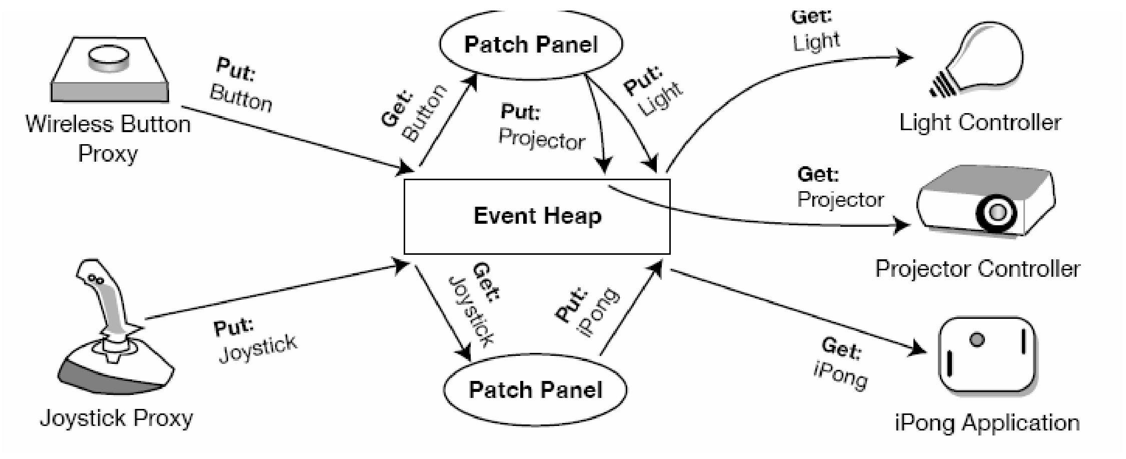


Figure 5 Interactions between Patch Panel and devices in an Interactive workspace

2. Interactive Workspaces

Interactive spaces [Johanson et al., 2002] project focuses on the interaction between the human and the devices. Users communicate with devices by triggering event changes such as button presses. Each device expresses its state changes as Finite State Machine (FSM). Patch Panels [Ballagas et al, 2004] assigns a unique identifier for each event change. Patch Panels collects event changes and send them to the corresponding devices according to the unique identifier. The interaction between the Patch Panel and the devices in an Interactive workspace is shown in Figure 5. Interactive workspace only suits for environments with stationary devices because operations done during state changes are carried by more than one devices and they are defined by applications at design time. In order to allow users to interactive with dynamic devices and enable devices to behave according to the changes in the environment, a more expressive language is required.

3. Other Smart Spaces projects

"Context Toolkit" [Dey, 1999] used similar approach as the one used in Interactive Workspace. An input event detected by the smart spaces changes the context-sensitive applications. One of the usages of "Context Toolkit" is keeping track of users who are still present in the smart space in location-based applications. Their working philosophy is that a user leaves the smart space will cause an event change. Open Agent Architecture (OAA) [Martin et al., 1999] also based on action triggering on contextual information. EasyLiving [Brumitt et al., 2000] attempted to represent geometric information. However, a more expressive and formal language is required in order to unify the geometric information into a grander vision of knowledge.

Ontology

Among all the knowledge representation strategies, we think that using ontology is most suitable in pervasive computing environment. Ontology represents the semantics of different concepts. It provides a formal, explicit specification of a shared conceptualization of a domain that can be communicated between people and heterogeneous and widely spread application systems [Gruber]. It is a formal explicit description of concepts (also called *classes*) in a domain of discourse, properties of each concept describing various features and attributes of the concept (also called slot) and restrictions on concepts [Noy & McGunness, 2001]. Ontology has been represented in different forms and languages. The most widely accepted form is semantic network because it is language independent. Below are the different forms of ontologies present in the literature. The logic form of ontology is shown in Figure 6. Knowledge Interchange Format (KIF) [KIF] is an example of logic language for ontology. Figure 7 – Figure 9 (captured from [Horrocks]) are the graphical representations of ontology. The topic map form is shown in Figure 7. Topic Map Constraint Language (TMCL) [TCML] is an example of topic map language. Semantic network is shown as Figure 8. A node in the network represents a concept. An arc connecting two nodes is their relationship. The label of the arc is the name of the relationship. Resource Description Framework (RDF) [RDF] is an XML-based language for ontology. Its graphical representaion is shown in Figure 9.

```
Everyone's age must be greater than 0.
(forall (?x) (greater (age ?x) 0))

The father of Charles is either John or Peter.
(forall (?x) (=> (father Charles ?x) (or  (= ?x John) (= ?x Peter))))

Nobody can be both a brother and a sister.
(forall (?x ?y) (=> (bother ?x ?y) (not (sister ?x ?y))))

Every person has a father.
(forall (?x) (=> (person ?x) (exists (?y)
        (and (person ?y) (father ?x ?y)))))
```

Figure 6 KIF-Logic Representaion of Ontology

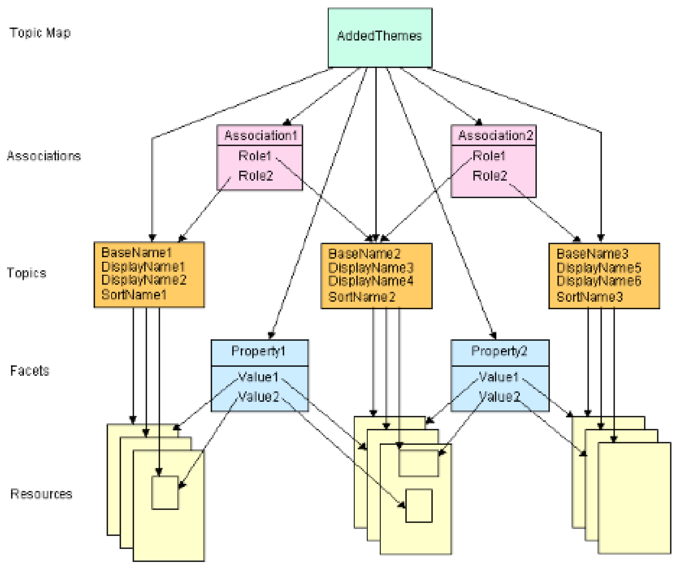


Figure 7 Topic Map Representation of Ontology

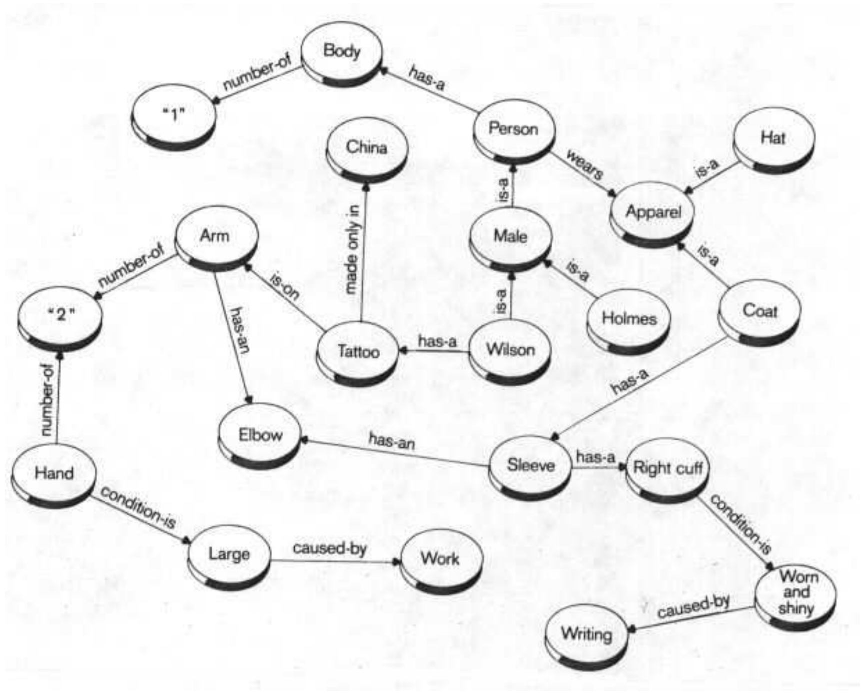


Figure 8 Semantic Network

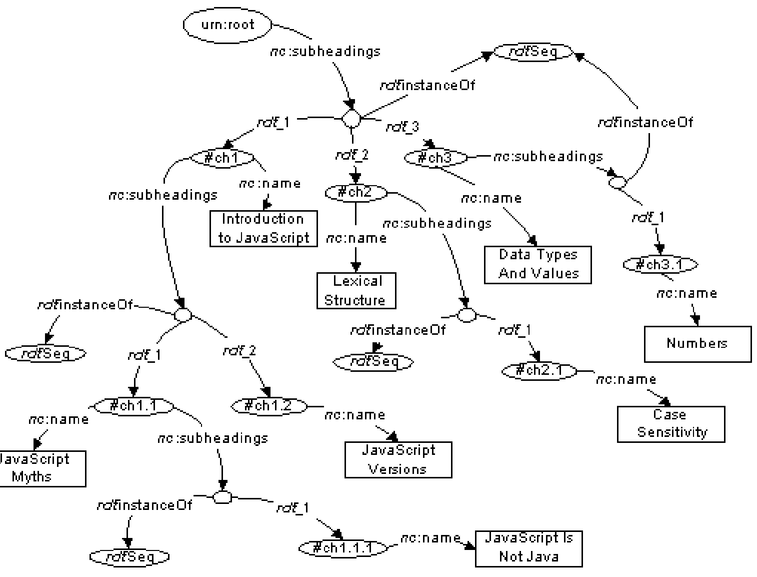


Figure 9 RDF

Ontology was firstly introduced in the computer science literature for representation information in artificial intelligence and knowledge representation ([Davis et al, 1993], [Davis and Shrobe, 1983] and [Doyle and Patil, 1991]) because it represents machine-interpretable information that let machines to act intelligently. It is now a hot topic in semantic web researches

[Davies et al., 2003], [Fensel et al., 2003] and [Horrocks and Schneider, 2003]. Semantic web, entitled as the future we, aims to provide intelligent web searching by providing meanings to homepages instead of using machine un-understandable HTML tags. When a user types in the keywords about the information that he/she wants to search in the existing web search engines, keyword matching with the content of the homepages is performed. Since a word can have different meanings, the accuracies of web search engines vary. A well-known example is the word "Apple". It is a kind of fruit and it is also a computer brand name. Semantic web proposed an additional semantic layer for the existing web to facilitate web searching. This layer uses ontology to represent information [Lee, 2000]. The proposed layer structure of the semantic web from Lee [Lee, 2000] is shown in Figure 10.

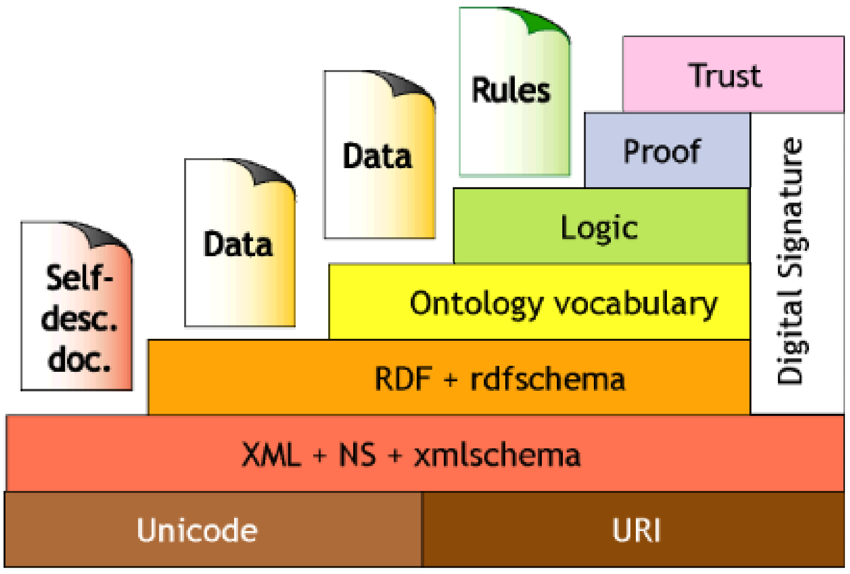


Figure 10 Semantic Web architecture

Semantic web searching is similar to service or resource lookup in pervasive computing environment because they aim to provide an intelligent matching. Hence, ontology has been adopted in projects such as Project Oxygen [Oxygen], CoBrA [Chen et al., 2003] and GAIA [Roman et al., 2002]. Oxygen is discussed in the previous sections. Ontology used in Oxygen is shown as semantic network in Figure 4. Context Broker Architecture (CoBrA) defines a set of Ontology Web Language (OWL) [OWL] ontologies to allow devices (or called agents in their term) to acquire, reason about and share context knowledge. GAIA is a middleware that enables embedded devices to aware of the context in smart spaces (Smart spaces are called "Active spaces" in their term). Similarly to CoBrA, it also defines a set of ontologies about the smart spaces such as entity and context information. Ontologies are used in a very restricted way in Oxygen, CoBrA and GAIA because devices communicate using the pre-defined ontologies. Usage of ontologies across different smart spaces is not addressed.

ONTOLOGY MAPPING

A smart space is called a domain and information in a domain is defined as domain knowledge. Smart spaces have different domain knowledge because they represent information using different ways. When users and devices move across smart spaces, knowledge gaps exist because of the differences in domain knowledge. Just like Alice travels from her home country to another country and communicates with Bill who lives in that country. Alice and Bill have knowledge gaps because of differences in culture, education background, etc. Ontology mapping is the mechanism to bridge knowledge gaps. Ontology mapping is defined as: *Given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept, relation or instance) in Ontology O_1, we try to find a corresponding entity, which has the same intended meaning, in Ontology O_2* [Su, 2002]. Ontology mapping between ontology O_1 and ontology O_2 is shown in Figure 11.

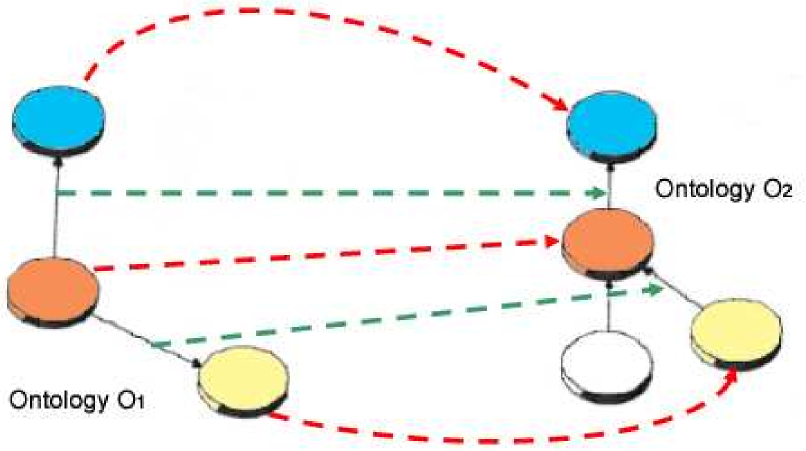


Figure 11 Ontology Mapping

Ontology mapping has been widely researched in semantic web. Most ontology mapping tools developed are to find a one-to-one corresponding mapping between concepts in two ontologies [Doan et al., 2002], [McGuinness et al., 2000], [Mitra et al., 2000], [Noy & Musen, 2000], [Stumme & Madche, 2001]. These mapping tools can be classified into two types: source-based and instance-based.

Source-based mapping tools compare the similarity of the concepts based on the properties of the concepts and the structure of the ontology defined in the source ontologies. Examples of source-based mapping tools are PROMPT [Noy & Musen, 2000], Chimaera [McGuinness et al., 2000], and ONION [Mitra et al., 2000]. PROMPT and Chimaera merge two source ontologies into a new one. The merged ontology contains concepts from both sources. They compare similarity of concept names to generate a match list of concepts. Users decide which concepts should be mapped based on the match list. ONION results in a set of mappings (articulation rules using their terms) between two ontologies. It transforms source ontologies into graphs. The nodes and the edges are used to match two graphs. Nodes are matched based on their names and a set of user-defined synonyms words. PROMPT, Chimaera and ONION use similarity between concept names for mapping. They work well for ontologies having a specialized terminology like medical ontology where each concept is a disease and each disease has a unique name. Their matching accuracy decreases when mapping ontologies with more general terminologies.

Instance-based ontology mapping tools compare the similarity of the concepts based on the source ontologies and their data instances. Examples of instance-based ontology mapping tools are FCA-Merge [Stumme & Madche, 2001] and GLUE [Doan et al., 2002]. FCA-Merge merges two source ontologies into a new ontology. FCA-Merge generates a pruned concept lattice by analyzing the frequencies of usage of concepts. Merging decisions are made based on the pruned concept lattice. FCA-Merge suits best the mapping of text documents: it requires a set of common instances for the mapping ontologies. For example, the instances are in the form of documents or homepages. GLUE gives a set of pairs of related concepts with some certainty factor associated with each pair. It analyzes the distributions of the concepts in data instances of the source ontologies and uses joint probability distribution to calculate the similarity between two concepts. GLUE, however, does not consider the structure of the ontologies (i.e., the relationships between concepts) during mapping.

Our proposed design uses the source ontologies and their instances. It calculates the similarity based on the similarities of the concept names, similarities of properties of the concepts and their relationships. It works for many different types of ontologies since it can assign a different weighting to each similarity calculation. Compared with those mapping tools that find a one-to-one corresponding mapping of concepts between two ontologies, our mechanism is more efficient since it is a partial mapping of the source ontologies. Compared with traditional instance-based ontology mapping tools, our mechanism is more space efficient since it uses history records to store the information about the instances instead of storing all the instances.

SIMILARITY FUNCTIONS

Mapping a concept in ontology O_1 to a corresponding concept in ontology O_2 is the key process in ontology mappings. Machine learning mechanism helps to classify a new data into an existing group of data. This is also applicable in ontology mapping in pervasive computing environment because ontology mapping can be viewed as to classify a concept in ontology O_1 into an existing grouping of concepts defining in ontology O_2. There are many machine learning techniques: Laplace Estimation, Decision Tree [Breiman et al., 1984], [Quinlan, 1993], k-nearest neighbour [Duda and Hart, 1973] and Naïve Bayes' rule [Mitchell, 1997].

1. Laplace Estimation

Laplace Estimation is commonly used for matching words. It is used in PROMPT [Noy & Musen, 2000], Chimaera [McGuinness et al., 2000], and ONION [Mitra et al., 2000]. Laplace Estimation is defined as:

$$P(w_i \mid c_j) = \frac{N(w_i, c_j)}{N(c_j)}$$

Laplace Estimation, denoted by $P(w_i \mid c_j)$, finds the number of times a word w_i appears in the category c_j.

2. Decision Tree

Decision Tree [Breiman et al., 1984], [Quinlan, 1993] classifies domain knowledge using tree structure. It is built before similarity comparison begins using a set of sample instances. A node in a decision tree is a discrete class. An edge between two nodes is the classification rule for the class (i.e. the content of the sample instance that led to the class). To determine a particular data belongs to which class, user needs to transverse down the decision tree based on the instance content. Decision Tree calculates similarities efficiently and is good for classification problems. Figure 12 shows a decision tree about the classification of diamonds. For example, an internal flawless (IF) class D five carats diamond is a good diamond.

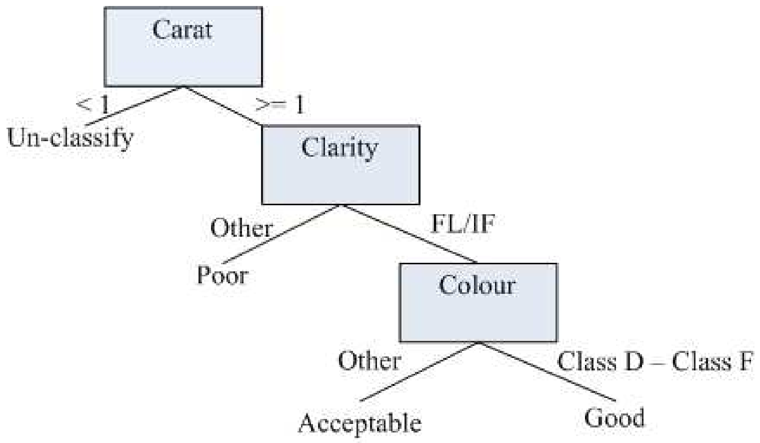


Figure 12 Decision tree for diamond classificaiton

3. K-nearest neighbour

K-nearest neighbour [Duda and Hart, 1973] converts the features of all the concepts in an ontology into vectors. It constructs an n-dimensional lattice if there are n distinct features. Sample instances are located in the lattice based on their content. The class for a new data is determined by the distance between the new data and the k-nearest neighbours in the n-dimensional space. Figure 13 (captured from [Duda and Hart, 1973]) shows the k-nearest neighbour using 2 dimensional spaces and k equals to 3. X_q belongs to class "●" as it has 2 "●"s and 1 "x" as its neighbour. K-nearest neighbour is used in FCA-Merge [Stumme & Madche, 2001].

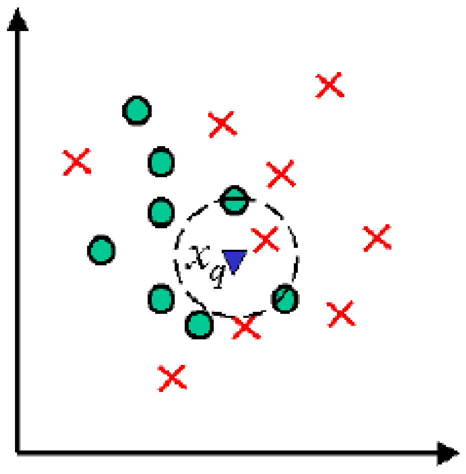


Figure 13 K-nearest neighbour

4. Naïve Bayes' rule

Naïve Bayes' rule [Mitchell, 1997] is a probability theorem that helps to determine the class c_j given the features, d, of the new data. It is used in GLUE [Doan et al., 2002]

$$P(c_j \mid d) = P(d \mid c_j) \times \frac{P(c_j)}{P(d)}$$

The probabilities $P(d \mid c_j), P(c_j)$ are obtained from the sample data. Naïve Bayes' rule gives the same value of $P(d)$ for all classes as it assumes that all features are independent. To illustrate the usage of Naïve Bayes' rule, a fruit concept is used. A fruit has properties: color and shape. Naïve Bayes' rule helps to determine what kind of fruit it is if a fruit has color orange and round in shape. Naïve Bayes classifier is the name for the Bayes' rule applied in document classification.

It is a long debate about which similarity function supervisors the others. However, there are still no winners because researchers evaluate similarity functions using different sample instance. We decided to adopt the one that is most suitable for pervasive computing environment in our design.

Laplace Estimation is, definitely, not sufficient as ontology does not simply contain string descriptions. Restrictions on properties and relationships are not compared. Translating ontology to decision trees is difficult because it is hard to determine the root node. A root node implies that this concept is more important than other concepts in the tree which contradicts to all concepts are equally important in semantic network. K-nearest neighbour constructs an n-dimensional space for features of the concepts. It is effective to measure the similarity if all concepts share the same set of features such as in FCA-merging because the number of dimensions are fixed. There are too many dimensions in pervasive computing environment that it will be time-consuming to compute the vector location for an instance in pervasive computing environment. Scaling the feature dimension is another problem using k-nearest neighbour. Taking a color feature as an example, the methodology to locate a "red" and a "green" in the dimension is critical. If Laplace Estimation is used, the distance between "red" and "green" is simply their syntactical difference rather than their semantical difference. If the meanings of the words are used as the scale, a general mechanism for scaling all the dimensions is required. Naïve Bayes' rule has the assumption that all features are independent. On the other hand, it is allowed to define sub-properties of a property in ontology. Naïve Bayes' rule is also based on the string descriptions that does not taken the restrictions into consideration.

A single similarity function does not fit the needs for the similarity measure in the pervasive computing environment. Our design proposes to use a combination of them. For string comparison like string type properties, Naïve Bayes' rule and Laplace Estimation are adopted. For relationship similarity comparison, k-nearest neighbour is used. As a result, our design is able to compute similarity using the properties and relationships of concepts.

PROPOSED DESIGN

RESEARCH CHALLENGES

From the previous chapters, the importance of ontology mapping in pervasive computing environment is addressed. Four new challenges on ontology mapping in pervasive computing environment that differentiate it from other related literatures are identified. These challenges are (1) online mapping, (2) efficiency in mapping, (3) space limitation of devices and (4) knowledge propagation to support user mobility.

Online mapping

Existing ontology mapping tools discussed in the literature review aimed to provide offline ontology mapping. They help to map ontologies during ontology design phase, for example, merging two ontologies to create a new one so that existing ontologies can be reused. They believed that ontology design is time-consuming because a single ontology is designed thoroughly to capture all concepts in a domain by domain experts. Re-using existing ontologies helps to reduce design time. It is no doubt that reusing existing ontologies is necessary. However, it is hard to include the world’s knowledge about a domain in an ontology. It is also inflexible for application and system designs when global domain is used. Changes in an ontology will cause a series of changes in worldwide applications that discourages adding new concepts. New changes cannot be reflected in previously mapped ontologies unless they are being mapped again.

The presence of smart spaces in pervasive computing environment allows us to logically group the knowledge inside it to form domain knowledge. Domain knowledge is represented in domain ontology. Communication is more effective if devices working closely to each other share the same domain language. Different smart spaces are allowed to have different domain knowledge. This is similar to the case that people in different countries have different mother languages.

In real life, people travel from countries to countries. In pervasive computing environment, devices move across smart spaces. When a device moves from smart space A to smart space B, it comes across different domain knowledge. It is obvious that ontology mapping in smart space A and smart space B cannot be done at design time because such movement is difficult to predict at design time. Online ontology mapping, therefore, is proposed. Online ontology mapping is defined as ontology mapping which is preformed at real-time instead of design time.

Efficiency

As our proposed ontology mapping is preformed at run-time, efficiency is important because proactive service requests may become out-dated services if large latency exists during analyzing service requests. For example, a compute screen provides personal advertisements when a user approaches. If it takes too long to analyze the user's preferences, he/she walks pass the screen before the advertisements are displayed. Accuracy in mapping results is equally important. Otherwise, devices cannot fully realize users' needs. There is trade-off between efficiency and accuracy. Our design should try to balance between these two.

Space limitation

Space limitation is related to the place where ontologies are store. Certainly, it is ideal that each device stores a local copy of the ontologies so that they can retrieve the concepts efficiently. In reality, memory limited mobile machines, wearable devices and sensors are unable to do so. Our proposed architecture, therefore, consists of resource-rich computer(s) called smart space monitor(s). It helps devices to obtain contextual information about the smart space and to allocate resources and functions. Complete ontologies are stored and mapping processes are performed at it. Frequently used part of an ontology named partial ontology is cached in devices. Bridging knowledge gaps means that partial ontologies are mapped to the complete ontologies stored in the smart space monitors in new smart space. The challenge of such mapping is how to increase the mapping accuracy due to partial information is given.

Knowledge propagation

Users move across smart spaces and leave a lot of mapping results in different smart spaces. It will be useful for mapping if we can analyze and reuse these results so that a user's knowledge can propagate to help another user.

SCENARIOS

The following daily life scenarios are helped to realize the usage of online partial ontology mapping.

Scenario 1

Alice is traveling to another country on the plane. This is her first time to go outside her home country and she does not understand the language spoken in the destination country. As usually, she brings her smart phone that stores her personal details such as her identity and daily schedule. Alice gets off the plane and arrives the immigration building located in the airport. Once she steps in the building, the sensors immediately inform the smart space monitor about the arrival of Alice and the smart space monitor forwards the information to the immigration department computers. In this country, every tourist needs to fill in a declaration form with his/her personal information, the purpose of the travel, the place where he/she is going to stay and the departure day. Smart space monitor forwards the form to Alice's smart space when it gets response from immigration department computers. The smart phone asks Alice whether her personal details are allowed to disclose when it receives the form. Alice clicks 'OK'. As the terminologies used to represent the personal information in Alice's smart phone are different from those required by the immigration departure, online ontology mapping is performed. When mappings completes, Alice's smart phone automatically fills in the form. Besides, the smart phone retrieves Alice's schedule and fills in the name of the hotel that Alice is going to stay and the departure day. After that, the form is sent back to the immigration department. The immigration department computers collect the form and verify whether Alice is legal to go inside the country.

Scenario 2

Following scenario 1, Alice arrives at the hotel. The method about how the sensors located in the hotel gets Alice's personal information is described in Scenario 1. The hotel smart space monitor retrieves the reserved room location and tells Alice's smart phone. Alice's smart phone informs Alice about her room location. After a long journey, Alice's smart phone notices that Alice is hungry and wants to have some snacks. Mostly likely, Alice would like to eat fish ball. In Alice's home country, fish balls are popular snacks and there are fish ball machines everywhere. Alice's smart phone looks up the fish ball machine in the hotel through the room's smart space monitor. The room's smart space monitor does not know what fish ball is. Using Alice's partial user ontology in the smart phone, the smart space monitor knows that fish ball is a kind of food which is made from fish and is spicy. The smart space monitor then try to locate the fish ball selling machine in the hotel by connecting to the hotel's smart space monitor. Since this hotel does not have fish ball selling machine, the room's smart space monitor advices Alice to order other foods by popping up the hotel restaurant menu in the display screen inside Alice's room.

Scenario 3

Alice bought an electronic pet in the shopping arcade near the hotel. The shopkeeper helps Alice to load her partial user ontology into the pet so that it can understand more about Alice. For example, the electronic pet sings when Alice is unhappy. It clamps hands when Alice finishes playing piano. Online ontology mapping is performed to bridge the knowledge gap between the electronic pet (To be precise, it should be the software installed inside the electronic pet) and Alice's partial user ontology during loading the user ontology. Once the electronic pet gets the context information about Alice, it reasons it and behaves accordingly.

OVERVIEW

Figure 14 shows the architecture overview of our proposed design. A pervasive computing environment is composed by smart spaces. A smart space is a logical boundary for a computing and resources rich area. A smart space does not need to be distinct. It can be a large smart space consists of many smaller smart spaces or can overlap its boundary with another smart space. Users and devices can move freely across different smart spaces. Smart spaces can communicate with each other using the smart space monitors through network.

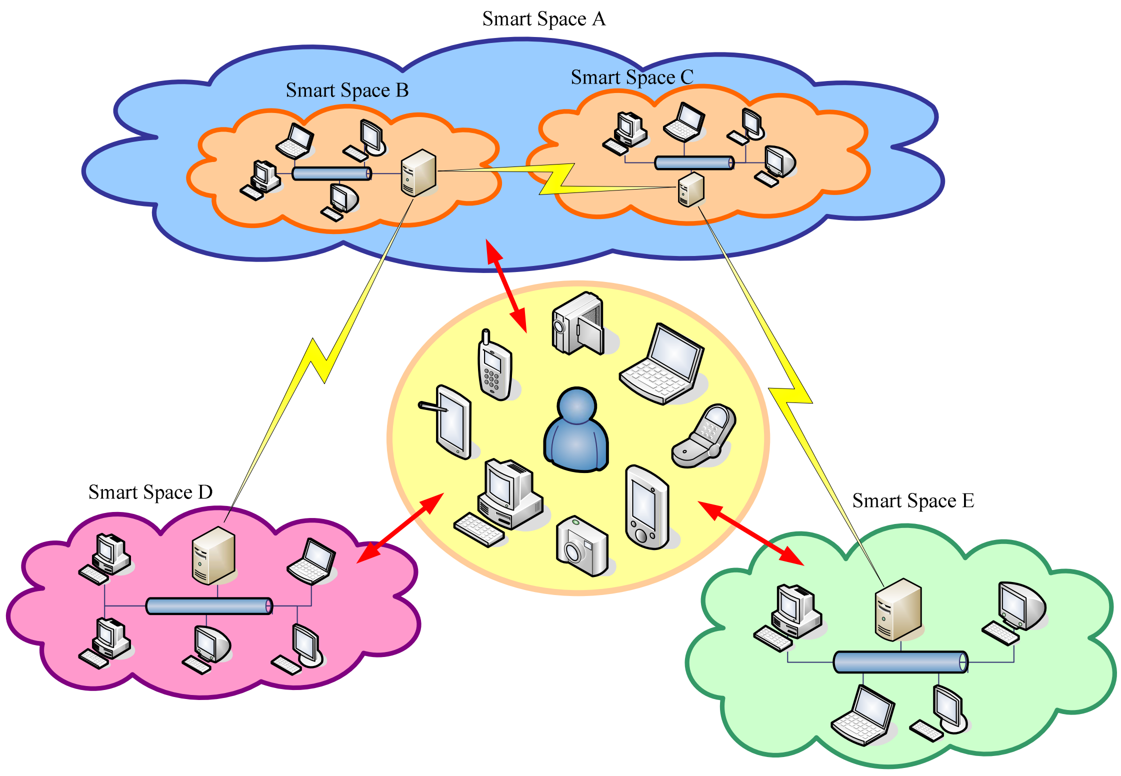


Figure 14 Architecture overview

Three types of ontologies and three types of mappings have been classified. The first type of ontology is domain ontology. Domain ontology is the ontology about the smart space, for example, the environment context, resources, activities done and people present in the smart space. Domain ontology is defined and managed by domain administrators. Smart space monitors are used to store the domain ontology. It also helps to coordinate the services and resources. There is only one domain ontology in each smart space. Application ontologies are the second type ontology. They store the concepts used in applications like device configuration, application parameters and service descriptions. They are defined and managed by application developers or application vendors. A smart space can have various application ontologies being used. Complete applications ontologies are stored inside application servers managed by application vendors. It is believed that a vendor will try to re-use its application ontologies to reduce development and maintenance costs. At run-time, application ontologies are cached in smart space monitors. The total number of application ontologies worldwide should be limited. The last type of ontology is user ontologies. User ontologies consist the users' knowledge such as user identity, social and mental status and user preferences. Complete user ontologies are resident somewhere in the Internet (For example, there can be tools that help users to define their user ontology and store them in servers. Or, users store their user ontology in their resource-rich machines). Devices store frequently used user concepts, named partial ontologies, in their local memory for fast access and efficient retrieval. Less frequently used concepts are retrieved through network and are cached in smart space monitor(s). They are numerous user ontologies in pervasive computing environment as each user can have his/her own ontology.

Mappings are required between (1) application ontology and user ontology; (2) user ontology and domain ontology and (3) application ontology and domain ontology. When users specify to use a particular service, mappings between application ontology and user ontology are required. Using the scenario as example, Alice's smart phone realizes that Alice is hungry and locates the fish ball selling machine for her in the hotel. Mapping between Alice's partial user ontology and smart phone application ontology help to determine Alice is hungry. Alice's smart phone locates the fish ball machine by mapping the description of the fish ball to the domain ontology. The last type of mapping happens when Alice tries to locate resources and devices in the hotel instead of using the proactive services provided by the smart phone. In our design, all three types of mappings are done at the smart space monitor. After mapping is completed, results are recorded and application and user ontologies are cached in smart space monitors.

Figure 15 shows the internal architecture of a smart space. A smart space consists of users, devices, sensors, resources and smart space monitors. For illustration purpose, figure 15 only shows one monitor. Smart space monitor stores domain ontology and caches application ontologies retrieved from application servers. It is responsible to coordinate devices and resources, lookup services and resources and perform ontology mappings. It also stores mapping results to achieve knowledge propagation.

Among all the functionalities provided by a smart space monitor, we think that ontology mapping is the most fundamental one because effective communicates that help to avoid ambiguity during information exchange are essential to coordinate resources and devices. Services lookup is based on the ability to keep track of the presences of the devices and services and the interpretation of the lookup queries. Again, ontology mapping is required to interpret the queries intelligently. As ontology mapping is the basic for all pervasive computing services we decided to focus on it such that a good and strong foundation is provided for the healthy growth and development of pervasive computing literature.

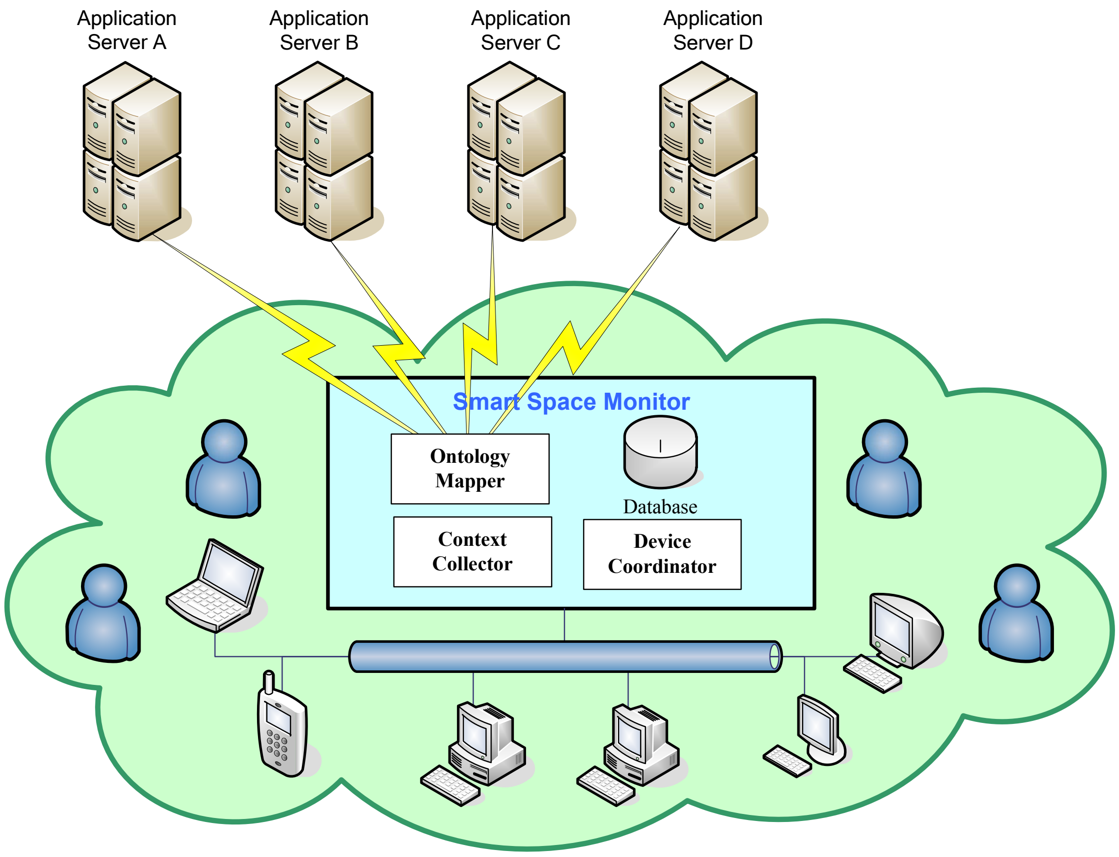


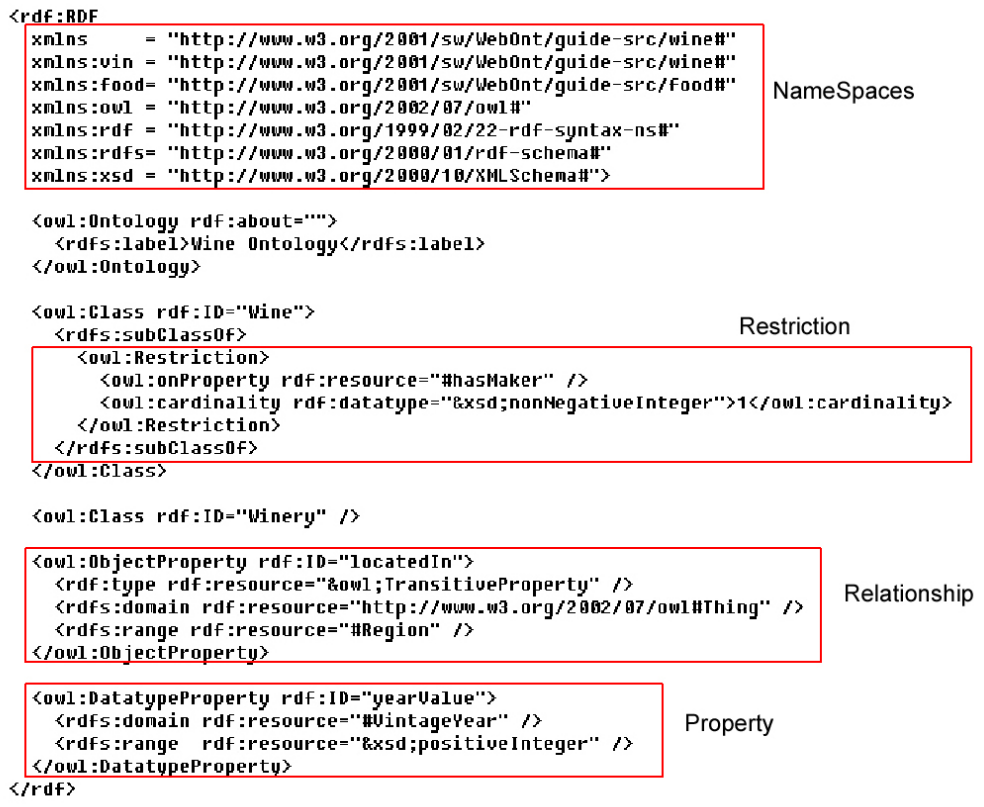
Figure 15 Architecture of a smart space

TERMINOLOGY AND NOTATION

Before discussing the details, the terminologies and notations used are explained. Table 1 shows the terminology table and table2 shows the notation table.

Term	Meaning
Ontology	Ontology is a formal specification of concepts in a domain closure.
Source ontology	Source ontology is an ontology represented by an ontology language. Ontology languages can be classified into logic-based and XML-based. Logic based languages such as Knowledge Information Interchange (KIF) [KIF] can be interpreted and reasoned by machines directly. It, however, is more difficult to write and not user-readable. XML-based languages require parsers to parse the information. It is easy to learn, write and read. XML-based ontologies allow more flexible structure. Examples of XML-base languages are Resource Description Framework (RDF) [RDF], DAML [DAML+OIL] and Ontology Web Language (OWL) [OWL]. This research is based on the usage of XML-based ontology languages.
Instance	Instances are the actual data information defined in the source ontology. Figure 16 shows a source ontology represented by OWL and Figure 17 shows an instance.
Request instance	Request instance is the service request instance.
Concepts, relationships and properties	Elements insides a source ontology are concepts, relationships and properties. To be simple, concepts are the XML tags used to define information. “Country”, “Social Status”, “Memory” are concepts. In languages like OWL, a concept is also called a class. Relationship is the semantic linking between two concepts for example, “is a” and “has” are relationships. Property (or called attribute) is the semantic description of a concept. Figure 16 shows an OWL ontology that defines concepts, relationship and property.

Table 1 Terminology Table



```
<rdf:RDF
  xmlns     = "http://www.w3.org/2001/sw/WebOnt/guide-src/wine#"
  xmlns:vin = "http://www.w3.org/2001/sw/WebOnt/guide-src/wine#"
  xmlns:food= "http://www.w3.org/2001/sw/WebOnt/guide-src/food#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema#">

  <owl:Ontology rdf:about="">
    <rdfs:label>Wine Ontology</rdfs:label>
  </owl:Ontology>

  <owl:Class rdf:ID="Wine">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasMaker" />
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="Winery" />

  <owl:ObjectProperty rdf:ID="locatedIn">
    <rdf:type rdf:resource="&owl;TransitiveProperty" />
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
    <rdfs:range rdf:resource="#Region" />
  </owl:ObjectProperty>

  <owl:DatatypeProperty rdf:ID="yearValue">
    <rdfs:domain rdf:resource="#VintageYear" />
    <rdfs:range  rdf:resource="&xsd;positiveInteger" />
  </owl:DatatypeProperty>
</rdf>
```

Figure 16 OWL wine ontology

```
<Winery rdf:ID="WhitehallLane" />
<Winery rdf:ID="Taylor" />
<Winery rdf:ID="Stonleigh" />
<Winery rdf:ID="SevreEtMaine" />

<Region rdf:ID="ChiantiRegion">
  <locatedIn rdf:resource="#ItalianRegion" />
</Region>
<Region rdf:ID="BeaujolaisRegion">
  <locatedIn rdf:resource="#FrenchRegion" />
</Region>

<VintageYear rdf:ID="Year1998">
  <yearValue rdf:datatype="&xsd;positiveInteger">1998</yearValue>
</VintageYear>
```

Figure 17 Ontology instance of wine ontology

Notation	Meaning
O_1	The source ontology used by the device or user for requesting resources or services. For application and user ontologies mapping, O_1 refers to the user ontology. For user ontology and domain ontologies mapping, O_1 refers to the user ontology. For application and domain ontologies mapping, O_1 refers to the application domain.
O_2	The source ontology that describes the resources or services with which the smart space monitor stores or caches. For application and user ontologies mapping, O_2 refers to the application ontology. For mappings involves domain ontology, O_2 refers to the domain ontology.
U_n	The set of instances of ontology O_n.
I_n	An instance of ontology $O_{n.}$
C_n	A concept presents in ontology O_n.
w_i	Weighting.
Sim(A,B)	The similarity between element A and element B.

Table 2 Notation table

MAPPING PROCESS OVERVIEW

In pervasive computing environment, efficient and accurate mappings for service requests are required to provide intelligent and proactive services. Three procedures have been designed for the mapping process: (1) pre-fetching; (2) similarity calculation and (3) recording results.

Pre-fectching is designed to minimize the number of concepts to be mapped between two ontologies such that efficient online ontology mapping can be achieved. It is especially important for mapping large ontologies such as the domain ontology. Pre-fetching estimates the similarity between two concepts by looking at their names and filters highly possibly un-related concepts to save mapping efforts. Our philosophy is that concepts with totally different meanings in their names are unlikely to refer to the same thing.

A concept definition in an ontology contains a concept's name/identifier, its properties and relationships with other concepts. Our design makes use of them in the similarity calculation process. K-nearest neighbour is used to calculate the semantic distance between concepts C_1 and C_2 by converting relationships in C_1 into vector space. Each relationship in C_1 is a dimension in the vector space and C_1 is located at the coordinate (1, 1, … , 1) where there are n "1"s if there are n relationship dimensions. Recall that the dynamic nature, the large number of dimensions and scaling for each dimension are the problems of using k-nearest neighbour in pervasive computing environment. We have overcome them by limiting the number of dimensions and having a dimension scale from 0 to 1. The number of dimensions are fixed to be the number of relationships in concept C_1. K-nearest neighbour locates C_2 in the vector lattice by calculating the similarity between each dimension and each relationship in C_2. The similarity formula uses the names of the relationships and their related concepts for calculations and outputs a value between 0 and 1 which limits the scale of the dimension. The smaller the distance between the concepts C_1 and concept C_2 in the vector space, the larger their relationships similarity is.

There are different types of properties: constraints (or called restrictions) and datatype properties. Constraints are the restrictions on the concepts such as the datatype range of a property or the maximum appearance of a concept in an instance. For example, a concept 'person' should have maximum one 'gender' property. Datatype properties describe the property types of the data content. For instance, string type or integer type. In traditional ontology mapping tools, all data contents are converted into strings and the similarity between the properties equal to the similarity between the strings. String "123456" is similar to string "12345". On the other hand, there is big difference between the values 123456 and 12345. Our design, therefore, recognizes the datatypes of the properites and handles them differently. For string datatype properties, we use Naïve Bayes' rule to calculate the similarities. For integer datatype properties, mathematical calculations such as addition and subtractions are used. The similarity formula is automatically chosen by our proposed mechanism at execution time based on the datatype of the properties.

The overall similarity calculation process is based on the Jaccard coefficient [Rijsbergen, 1979]. Jaccard coefficient determines the ratio of the overlapping data between two sets of data. Jaccard coefficient is defined as:

$$\frac{P(C_1 \cap C_2)}{P(C_1 \cup C_2)} = \frac{P(C_1, C_2)}{P(C_1, C_2) + P(C_1, \sim C_2) + P(\sim C_1, C_2)}$$

Jaccard coefficient is a commonly used similarity measure which is based on the joint probability distribution. It is adopted in our design because data instances should be taken into consideration during mapping because our research goal is to provide a service and resource instance to the requester. In other words, we try to find an instance from ontology O_1 that can best match an instance from ontology O_2. It is useless if we can have a prefect matching between two ontologies without finding a matched instance. Jaccard coefficient outputs similarity measure between zero and one. A "1" means the two concepts are identical and a "0" means the two concepts are totally different. $P(C_1, C_2)$, $P(C_1, C_2)$, $P(C_1, \sim C_2)$ and $P(\sim C_1, C_2)$ in the formula are computed based on the definition of the concepts (i.e. the name, relationships and properties of the concept) and their instances. It is easy to retrieve instances for the domain ontology because the smart space monitor coordinates all the instances distributed in the environment. Instances of application ontologies are also available in the smart space as there are limited application ontologies. Some of them must be present in the smart space. User ontologies are cached as partial ontologies in different devices. Users move across different smart space and partial user instances are distributed everywhere. It is time inefficient if our design locates these instances at run-time. Counters are used instead to keep trace of the appearance of the concepts in the partial user ontology instances. When the users stay long in the smart space and their partial user ontology are mapped frequently, our design can provide accuracy increasing mapping results. Moreover, instance data can keep private because its content is not stored in the monitor.

After mapping is completed, smart space monitor stores the results such that knowledge propagation is supported. We analyze the mapping results and group them into different categories. Mapping results can be re-applied to the same category of users in the future. The categories are determined based on the environment context such as time and activities. Domain administrators are allowed to predict users' behaviours and help to define rules for the categorization to fit the true needs of the users in the smart space. As space is limited, it is impossible to store infinite mapping results. Replacement mechanism is designed which is based on the usage frequency and the context information.

Other techniques are used to enhance our matching mechanism. Complete user ontology is not always available as they are either disturbed in the pervasive computing environment or they are stored far away in the network. Devices cache frequently used partial ontologies for fast access and retrieval. Our design also caches the partial ontology after mapping. This helps to construct a more complete picture of the user ontology where the mappings of the different parts of the user ontology are frequent. A replacement mechanism is also designed. It is based on the presence of the concepts in the instances and the usage frequency.

PRE-FETCH

To increase the efficiency of mapping a concept C_1 in O_1 to a concept C_2 in O_2, we filter out the highly unrelated concepts by examining the names of C_1 and C_2 and generate a set of possible candidates of C_n from O_2. Many mechanisms are proposed to compare similarity between two strings, for example, longest common substring, longest common subsequence and hamming distance. These mechanisms are based on the syntactic meaning (i.e. the spellings) of the two strings. Using the email XML structures described in the introduction as example, concept "SendTo" and concept "Recipient" are not similar as their syntactic meanings are greatly difference. However, we know that "SendTo" and "Recipient" have similar meanings in English that reflects that semantic meanings of the concept names are more important than their syntactic meanings.

To define the semantic meanings of the words, we propose to use the WordNet ontology [Halimi et al., 1998]. WordNet organizes English nouns, verbs, adjectives and adverbs as synonym sets. A synonym represents a lexical concept. Synonym sets are linked by relations. We introduce the term *Semantic Distance* as the smallest number of intermediate concepts to connect the meanings of two concepts. For example, "Send" is related to "Sendee" and the hyponym of "Recipient" is "Sendee" in the WordNet ontology. As a result, "SendTo" and "Recipient" are semantically related and their Semantic Distance (SendTo, Recipient) = 1 as there is one intermediate concept "sendee" that connects between "SendTo" and "Recipient".

To filter un-related concepts, we retrieve the semantic distance of their concept names which is smaller than or equal to the semantic distance threshold for each pair of concepts between C defined in O_1 and C' defined in O_2,.

$$\text{Semantic Distance}\left(C,\ C'\right) \le \mathit{Threshold}$$

For common similarity measures, similarity is ranged from 0 to 1. A "0" means the compared items are totally different and a "1" means they are identical. Our design follows this rule by normalizing each semantic distance that is smaller than the threshold as:

$$Sim\left(C_{name}, C'_{name}\right) = \frac{\mathit{Threshold} - \text{Semantic Distance}\left(C,\ C'\right)}{\mathit{Threshold}}$$

For the set of the first k concepts with the highest similarity degree (i.e. the k-highest $Sim(C_{name}, C'_{name})$) denoted by $S_{k\text{-high}}$, we form the possible candidates set to be compared with C by:

$$\begin{aligned}&\text{Possible candidate set}\\ &= S_{k\text{-}high} \cup \forall C_i \in S_{k-high} \text{ concepts that has relationships with } C_i \cup\\ &\forall C_i \in S_{k-high} \text{ merged concepts of } \mathrm{C_i} \text{ with each of its neighbour} \cup\\ &\forall C_i \in S_{k-high} \text{ parent (super class) of } C_i \cup\\ &\forall C_i \in S_{k-high} \text{ children (sub - class) of } C_i\end{aligned}$$

In ontology mapping, a concept in O_1 may be split into two concepts. For instance, a concept “name” in ontology O_1 may be split into two concepts, “first name” and “last name” in ontology O_2. To handle the splitting problem, our proposed mechanism merges concepts with their neighborhood concepts, parent concepts and children concepts. Merging concepts C'_1 and C'_2 of the same ontology is done by merging their concept names, attributes and relationships. To resolve naming conflict of attributes and relationships, attributes and relationships are renamed as C'_1.attribute name and C'_2.attribute name and C'_1.relationship name and C’2.relationship name respectively. Duplicated relationships are removed during merging. A relationship between C'_1 and C'_2 is converted as attribute with the name of the relationship as the attribute name.

SIMILARITY CALCULATION

Jaccard coefficient is defined as:

$$\frac{P(C_1 \cap C_2)}{P(C_1 \cup C_2)} = \frac{P(C_1, C_2)}{P(C_1, C_2) + P(C_1, \sim C_2) + P(\sim C_1, C_2)} \qquad \mathbf{(1)}$$

From the project GLUE [Doan et al., 2002], $P(C_1, C_2)$ is defined as equation (2) where U_1 and U_2 are the instance set of O_1 and O_2 respectively. N ($U_1^{C1,C2}$) is the number of instances of O_1 that contain concept C_1 and concept C_2. N ($U_2^{C1,C2}$) is the number of instances of O_2 that contain concept C_1 and concept C_2, N (U_1) and N (U_2) are the number of instances of O_1 and the number of instances of O_2 respectively.

$$P(C_1, C_2) = \frac{N\left(U_1^{C1,C2}\right) + N\left(U_2^{C1,C2}\right)}{N(U_1) + N(U_2)} \qquad \mathbf{(2)}$$

We adopt these formulae when computing similarity between concepts C_1 and C_2 so that instances are considered when mappings are performed. To calculate $P(C_1, C_2)$, we should have two instance sets U_1 and U_2. As discussed in the above chapter, it is difficult for smart space monitors to locate the instances of partial user ontologies and to store the instances. As a result, we use history records to determine the instance sets U_1 and U_2.

The smart space monitor counts the number of concepts appearing in each mapping instance. For example, an ontology O_A contains concepts C_a, C_b, C_c and C_d and a request instance contains concepts C_a and C_b. After mapping, the total number of instances of O_A and the numbers of instances that contain C_a and C_b are incremented by 1 while the numbers of instances that contain C_c and C_d remains unchanged. $N(U_1)$ and $N(U_2)$, therefore, are recorded. To get $N(U_1^{C1,C2})$, we have to estimate the number of instances of O_1 that contain concept C_1 and concept C_2. The number of instances of that contain concept C_1 in U_1 can be found from the history records. The number of concepts that contain both concept C_1 and concept C_2 in U_1 is estimated by calculating the similarity degree of the properties and relationships between concept C_1 and concept C_2. If the properties and relationships of the concepts are similar, it is likely that the instance of concept C_1 is an instance of concept C_2. For numerical property such as memory size, it is important that a mapping is found to another concept whose property contains also a numerical value; otherwise, it is difficult to satisfy functionality requests like “memory size less than 10 kbytes”. Hence, weights are added when calculating property similarity.

The proposed mechanism also matches instances content when comparing two concepts. When a concept of the request instance matches a concept of a resource instance or a function instance, it is likely that these concepts are matched. The smart space monitor uses all the present instances of O_1 and O_2 in its smart space to compare with the request instance.

Before looking at the details of our ontology mapping mechanism, the *Property Similarity* and the *Relationship Similarity* are discussed. Property similarity calculates how similar of the properties of the two concepts are and relationship similarity calculates how similar of the relationships of the two concepts are. Property similarity and relationship similarity compares all the properties and relationships exist in concept C_1 and C_2.

Property Similarity, ps

There are three elements compose a property in ontology: name, datatype and cardinality. Name is the name of the property such as “colour” and “size”. Datatype is the type of the content data. For example, property “colour” is string type. “Size” is integer type. Cardinality refers to the restrictions of the properties such as the range of the datatype, the maximum value of the datatype and the number of appearances of the property in a concept. For example, the minimum cardinality of property “Size” = 0 means “Size” must be a positive integer. As there are many different types, we handle them separately.

For numerical datatype like integer and real number, similarity of the data content of property instances p_1 and p_2 is the division of the data value.

$$Sim(\text{instance of } p_1, \text{instance of } p_2) = \frac{\text{value of instance } p_1}{\text{value of instance } p_2}$$

For string datatype, similarity of the data content of property instances p_1 and p_2 is calculated using Naïve Bayes’ rule. Data content of property instance p_2 is tokenized into words as $\{w_1, w_2, \ldots, w_n\}$. Similarity of data content can be calculated using the following equation:

$$P(p_1 \mid w_1, w_2, \ldots, w_n) = P(w_1, w_2, \ldots, w_n \mid p_1) \times \frac{P(c_1)}{P(w_1, w_2, \ldots, w_n)}$$

As Naïve Bayes' rule assumes each word is independent, $P(w_1, w_2,..., w_n)$ can be omitted. And Naïve Bayes' rule can be re-written as below where P(p_1) is the usage frequency of the property p_1 in the data instance set:

$$P(p_1 \mid w_1, w_2,..., w_n) = P(p_1) \times \prod_{i=1}^{n} P(w_i \mid n)$$

It is ideal that meaning of the strings can be considered. However, this requires English sentence interpreter that is out of scope of this research. If there exists good one in the future, it can be plugged into our design.

To compare the similarity of user defined datatype structures is within the scope of a compiler design. For simplicity, the value of each element in the structure is concatenated as a long string. And then, it's similarity is calculated as string datatype as described above.

For each pair of property/attribute in C_1 (denoted by P_{C1}) and property/attribute in C_2 (denoted by P_{C2}), compute their property similarity using equation (3). The similarity of the property names are calculated using their meanings which is similar to computing the concept name similarity. Property instance similarity is calculated by counting the number of instances of C_2 whose property has similar content as the corresponding property of the instances of C_1 and the formula is shown in equation (4).

$$\begin{aligned} & Sim(P_{C1},P_{C2}) \\ & = w_1 * Sim(P_{C1} name, P_{C2} name) + \\ & \quad w_2 * Sim(P_{C1} cardinality, P_{C2} cardinality) + \\ & \quad w_3 * Sim(P_{C1} data\ type,\ P_{C2}\ data\ type) + \\ & \quad w_4 * \text{property instance similarity} \end{aligned} \tag{3}$$

$$\begin{aligned} & \text{Property similarity, } ps \\ & = \frac{\sum (\text{frequency of properity } i \text{ in } C_2 \ * Equation(5))}{\sum \text{frequency of properity } i \text{ in } C_2} \\ & for\ i = 1\ to\ number\ of\ property\ in\ C_2 \end{aligned} \tag{4}$$

Relationship Similarity

Relationship similarity is calculated using k-nearest neighbour. For each relationship between C_1 and C_2, similarity between relationship R_{C1} and R_{C2} is computed as below. Equation (5) is used to locate the concept C_2 in the concept lattice of C_1 in dimension R_{C1}. Similarity of relationship names are calculated using their meanings which is similar to compute concept name similarity.

$$\begin{aligned} & Sim(R_{C1}, R_{C2}) \\ & = w_1 * Sim(R_{C1} name, R_{C2} name) + \\ & \quad w_2 * Sim(R_{C1}\text{cardinality}, R_{C2}\text{cardinality}) + \\ & \quad w_3 * Sim(R_{C1} type, R_{C2} type) \end{aligned} \tag{5}$$

The relationship similarity is calculated as the distance between the origin and the concept C$_2$ in the concept lattice. The longer the distance from the origin means the closer to the concept C$_1$. It is the distance between the origin instead of the distance between the concept C$_1$ because to make the formula consistent with the similarity calculations with less similar having smaller value and a equal totally identical having value “1”.

$$\begin{aligned} & \mathrm{Re}\,lationship\ similarity\ \deg ree \\ & = \sqrt{\frac{\sum (similarity\ of\ relationship\ i\ in\ concept\ C_1\ and\ C_2)^2}{3}} \\ & for\ i = 1\ to\ number\ of\ relationships\ in\ C_2 \end{aligned} \tag{6}$$

Calculation of $P(C_1, C_2)$

After knowing the property similarity and relationship similarity, we can use of them to compute the $P(C_1, C_2)$ defined in equation (2). The following shows the procedures.

1. U$_1$ is partitioned into two sets. One set contains concept C$_1$ (denoted as U_1^{C1}) while the other set does not contain concept C$_1$ (denoted as $U_1^{\sim C1}$) based on the history records.
2. U$_2$ is partitioned into two sets. One set contains concept C$_2$ (denoted as U_2^{C2}) while the other set does not contain concept C$_2$ (denoted as $U_2^{\sim C2}$) based on the history record.
3. Estimate the similarity between O$_1$ and O$_2$ with equation (7) where the denominators N(O$_1$) and N(O$_2$) are the total numbers of concepts in O$_1$ and O$_2$ respectively. Total number of similar concepts can be computed at pre-fectching when calculating maximum concept name similarity between each concept in O$_1$ and in O$_2$. If the maximum concept name similarity is larger than the threshold, the total number of similar concepts is incremented.

$$Sim(O_1, O_2) = \frac{\text{total number of similar concepts} \times 2}{N(O_1) + N(O_2)} \quad \textbf{(7)}$$

4. N ($U_1^{C1,C2}$) is found.

$$\begin{aligned} &\text{Number of instance in } U_1 \text{ that contains concept } C_2 \\ &= \text{Property Similarity} \times N\left(U_2^{\ C2}\right) \\ &= Equation(4) \times N\left(U_2^{\ C2}\right) \end{aligned} \quad \textbf{(8)}$$

5. Similarly, calculate N($U_2^{C1,C2}$), N($U_2^{C1,\sim C2}$) and N($U_2^{\sim C1,C2}$) in $P(C_1, C_2)$.

$$N\left(U_2^{\ C1,C2}\right) = N\left(U_1^{\ C1}\right) * ps = N\left(U_1^{\ C1}\right) * Equation(4)$$

$$N\left(U_1^{\ C1,\sim C2}\right) = N\left(U_1^{\ C1}\right) - N\left(U_1^{\ C1,C2}\right)$$

$$N\left(U_1^{\ \sim C1,C2}\right) = Equation(8) - N\left(U_1^{\ C1,C2}\right)$$

6. P(C_1,C_2), P(C_1,~C_2) and P(~C_1,C_2) are computed using equation (2).
7. The similarity degree using equation (1) is computed. This similarity degree is called *Instance Similarity Degree* as we use instances to calculate.
8. The relationship similarity degree for C_1 and C_2 is computed using equation (5).
9. The similarity between concept C_1 and concept C_2 is finally computed using the below equation.

$$Sim(C_1, C_2) = w_1 * \text{instance similarity degree} + w_2 * \text{relationship similarity}$$

COMPARISON BETWEEN TWO ONTOLOGIES O_1 & O_2

The detailed formulae for calculating the similarities are introduced. The overall working mechanism is discussed in this section. Below is the methodology to compare two ontologies; OntologyMapping() is our mapping function and NewMapping() is a procedure call that is invoked when mapping is performed from scratch.

```
NewMapping($C_i$)
{
   Pre-fetch the candidate concepts for $C_i$.

   For each candidate $C_k$ found,
      Computer the Jaccard coefficient $\frac{P(C_i \cap C_k)}{P(C_i \cup C_k)}$ for $C_i$ and $C_k$

   If the highest similarity degree > threshold,
      Mapping is found.
   Else
      Mapping is failed.
}
```

```
Ontology Mapping (Ontology $O_1$, Ontology $O_2$)
{
   Search history mapping record.

   If $O_1$ and $O_2$ have been mapped,
      If $O_1$ and $O_2$ have the same last modified date (i.e. the same
         version number) as the history record,

         For each concept $C_i$ in the request instance,
            If $C_i$ is mapped to a concept in $O_2$ in the record,
               Mapping is found.
            Else
               Invoke NewMapping($C_i$).
      Else
         For each concept $C_i$ in the request instance,
            If $C_i$ is mapping to a concept,$C_p$ in $O_2$ in the record,
               Compute Jaccard coefficient $\frac{P(C_i \cap C_p)}{P(C_i \cup C_p)}$ for $C_i$ and $C_k$

               If similarity degree > threshold,
                  Existing mapping is reused.
               Else
                  Invoke NewMapping($C_i$).
            Else
                  Invoke NewMapping($C_i$).
   Else
      For each concept $C_i$ in the request instance,
         Invoke NewMapping($C_i$).

   Update number of instances and concepts encountered.

   For each new mapping found,
      Add <concept in $O_1$, concept in $O_2$, similarity degree,instance
            count> in history record.
}
```

RESULT RECORDING

From the similarity calculation, it has been mentioned that the matching records are stored in triple form: <concept in O_1, concept in O_2, similarity degree, instance count>. Mapping results can be reused directly if they are from application and domain ontology mappings because they are shared by numerous devices. It is obvious that domain ontology has large instance set because there is only one domain ontology in each smart space. All environment variables such as time, place, activities, temperature and lightings are instances of the domain ontology. An application ontology also has large set of instances because a limited number of application ontologies defines many applications. Similarly, 26 alphabets generate numerous English words. Application ontology designers, application developers and vendors, may share a common application ontology or each of them define their own or a hybrid between the two methods. Application ontology mapping results can be reused directly once the same application or applications sharing the same application ontology execute. It is difficult to directly reuse the mapping results involving user ontologies because each user has its own user ontology. There are so many user ontologies that it is rare to access the same user ontology mappings frequently in a smart space. In other words, there is always a mapping miss in retrieving history records when a new user enters a smart space. In order to reduce the miss rate in our design, mapping results also categorized based on the usage pattern of different users.

Categorization

Domain administrators are allowed to predict the users' behaviors and help to define the rules for the categorization to fit the true needs of the users in the smart space. The default categorization uses date, time and activities as categorization criteria if no domain administration defined rules are found. We assumed that concepts date, time and activities are present in the domain ontology and they are further coarsely categorized. Time is catalogized into "Working days" (from Monday to Friday) and "Weekend" (from Saturday to Sunday). Time is catalogized into "Morning", "Afternoon" and "Night". The usage pattern is stored along with the mapping results. Table 3 visualizes the mapping results categorized by date, time and activities. Table 4 shows how mapping results are stored in our design. The columns "C_1", "C_2", "Similarity degree" and "Instance count" are the triple form described in the previous chapter. A smart space monitor has one and only one mapping table. Its size is limited by the domain administrations.

URI of O_1	URI of O_2	Date	Time	Activity
http://xxx.com	http://yyy.com	Working days	Morning	Watch morning news
http://abc.com	http://def.com	Weekend	Afternoon	Play computer game
http://xyz.com	http://ijk.com	Weekend	Night	Watch concert

Table 3 Mapping results categorized by date, time and activities

URI of O_1	URI of O_2	C_1	C_2	Similarity degree	Instance count
http://xxx.com	http://yyy.com	sendTo	Recipient	0.7	12
http://abc.com	http://def.com	Name	Name	1.0	40
http://xyz.com	http://ijk.com	Email	Mail	0.3	24

Table 4 Mapping records in smart space monitor

The working mechanism of how mapping records are used in explained. A new user enters the smart space with his/her partial user ontology stores in a device. He/She tries to locate a service that is described by O_{new}. The smart space monitor tries to map O_{new} to a service description defining in ontology O_2. Before ontology mapping begins, previous mapping results are retrieved. From the records in column "URI of O_1" in table 3, it is realized that O_{new} is a new user ontology to the smart space because there is no previous mappings between O_{new} with any ontologies in the smart space. Then, the mapping records of O_2 are found from column "URI of O_2". If a history record exists, a mapping record is chosen based on the current data, time and activity. If more than one records satisfy the date, time and activity, the ontology record with the source ontology O_1 is chosen if it is being used currently and the user of O_1 is nearest to the new user. Otherwise, it is chosen randomly by the smart space monitor. When a record is chosen, pre-fetching and ontology mapping begin. Our design treats O_{new} as an updated version of the ontology stated in column "URI of O_1". The methodology about how updated ontology can be mapped using history record has been discussed in the overall algorithm in previous chapter.

We use date, time and activities as criteria in categorization because we want to balance between the complexity and the accuracy. It is true that if more parameters are included in the categorization, we can get a history record that is closer to the definitions of the new partial user ontology. However, it may also cause the categorization to be too specific that there are no improvements on the high miss rate of finding mapping records. If there are too few parameters, even a mapping record is retrieved. It cannot provide a satisfactory similarity degree. In other words, the mapping records are useless for knowledge propagation to map ontologies. Overheads will be induced for finding a useless mapping record. Our design tries to include as many as parameters in the categorization such as date, time and activities to increase the probability for finding a useful mapping record. On the other hand, we provide a coarse-grain classification to each parameter to avoid a specific categorization. If more than one ontology mapping records match with the environment context, our design locates the user whom is nearest to the new user and have requested for the same service recently. Our philosophy is based on the belief that people who are having the same activity and are near to each other have a high chance that they requires similar services.

Another consideration for the categorization criteria is the availabilities of the criteria parameters. As our default categorization is used by all the smart space monitors, the parameters determining the category should be present in most smart spaces. After investigating the existing smart space researches, we found that concepts date, time and activities are always defined in a smart space. We, therefore, use them as our criteria for categorization. As we mentioned before, strict rules are not included for the categorization. Domain administrators are always allowed to modify them by adding specify parameters for categorization. At current stage, we do not provide any syntax for the rules. A simple configuration file stating the criteria concepts with their relative importance having value between 0 to 1 is used.

Replacement

A replacement strategy is designed to delete out-dated mapping results so that memories can be freed for new ones. Replacement is taken place when the smart space monitor realizes the record table is full. We base on three factors to decide the replacement. Firstly, it is the usage frequency of the ontology. Concepts from frequently used ontologies should not replaced although the concepts themselves are not used frequently because the similarity degrees of their mappings help to provide more accurate mappings for mapping their neighbour concepts with new concepts. As the ontology is used frequently, it is believed that such neighbour mappings occurs quite oven. Secondly, the usage frequency of the concepts affects the replacement mechanism. Records having low instance count mean they are rarely used in the smart space and the hit rate for retrieving history records is not affected if they are deleted. The environment context is the last factor. For example, the mapping context categorization table realizes that an activity has not performed in the previous 100 records. It can treat it as an out-dated activity and free all mapping results related to the out-dated activity.

Another technique that is similar to resulting record is called caching. Caching stores partial user ontologies and constructs a complete picture for a user ontology such that even partial information are provided in partial ontologies, our design is able to provide accurate mappings. Details of the caching mechanism is presented in the next section.

CACHING

Complete structure of the user ontology may not be available in the smart space monitor because they are either distributed in different devices or they are stored far away in the network. In order to construct a complete structure of the user ontology, our mapping mechanism caches the partial ontologies after mapping is completed. When a user performs frequent user ontology mappings, his/her partial user ontologies stored in the smart space helps devices in the smart space to provide better services to him/her because the devices know them better. Caching also improves mapping by providing the knowledge of the missing concepts.

Mechanism

Caching is taken place after the mapping results have been recorded. Application ontologies and partial user ontologies are cached. Domain ontology is stored in the smart space monitor. Application ontologies are retrieved from the application servers provided by application developers or vendors. Recently used application ontologies are cached in the smart space monitor in order to save retrieval time as application ontologies are expected to be large. The partial user ontology is merged with the existing parts cached in the smart space monitor. The size for caching the partial ontologies, which is similar to the size of the memories for recording results, is defined by domain administrators. Different users have assigned different amount of cache places for caching their user ontologies. Stationary users (i.e. users tends to stay long in the smart space) can have a large cache while guests and temporary users are allocated with a small cache. In default, users have equal amount of memories to cache their partial user ontologies. Domain administrators need to define caching rules if special handlings on the caching memories are required.

The merging process is straightforward. If the version of the partial user ontology is the same as the version in the parts caching in the smart space monitor, merging begins. Concepts are merged with the existing parts except for the overlapped concepts that are not added. If the versions of the partial user ontologies are not the same, mapping is treated as an updated version of the user ontology and the mapping records will be cached separately. The older version results are not overridden because our design wants to be compatible with different versions of ontologies. If the older version is not longer in used, its mapping results will be eventually pruned out by the caching policy specified in the record storing section. Although the process for merging is straightforward, the complexity lies in the replacement mechanism because the smart space monitor does not have unlimited space for the ontology to grow and expand.

Replacement

Replacement mechanism is required to handle the case when caching memories are full. Replacement in caching partial user ontologies also depends on the usage frequency of the user ontology, the usage frequency of a concept and the environment context. Details of the replacement mechanism can be referred to the one that is described in the result recording section.

EVALUATION

Experiments are carried out to evaluate the effectiveness of our design. We compare our mapping results with the source-based and the instance-based ontology mapping tools. To stimulate the source-based ontology mapping tools, a string comparer based on the Laplace Estimation is implemented. For instance-based ontology mapping tools, we compare our results with the GLUE project [Doan et al., 2002].

The nature of the existing ontology mapping tools is different from our matching mechanism. A fair comparison on the performance is difficult to carry out. Our mapping mechanism performs mappings for concepts appear in the request instance while existing ontology mapping tools perform mappings for all concepts. Our matching mechanism must be faster if the source ontologies are huge and only a small number of concepts are used in the request instance. Moreover, our design stores mapping history records and partial user ontologies. If a record hit is found during mapping, it will even fasten our mapping mechanism. GLUE [Doan et al., 2002] has stated that they can achieve good matching accuracy when there are around 30 - 50 instances for the source ontologies. We, therefore, provide a large instance set for GLUE.

As it is difficult to perform fair comparisons, our experiments should be carefully designed. The factors that we are considered to be evaluated are listed in the following section. The implementation details of our matching mechanism are going to be discussed after that. As source ontologies are very important in our experiments, they are going to be introduced. Evaluation results are going to be present in the last part of this chapter.

EVALUATION PARAMETERS

Our objective is to design an effective partial ontology mapping for pervasive computing environment. The effectiveness can be measured by the speed, the memory consumption and the accuracy of the mapping results of the proposed matching mechanism. The speed of the algorithm is measured by the time used to complete the ontology mapping. We use seconds to measure the speed of the algorithm. Memory consumption means the memory usage requires storing the source ontologies, the instances and the history records. To measure the accuracy of mappings between two source ontologies, manually mappings are required. A mapping result is defined as the mapping from concept C_1 in ontology O_1 to concept C_2 in ontology O_2. The manual mapping results are supposed to be the most precise mappings for all the concepts. The number of mapping results manually found is denoted by $\mathrm{N}(\text{matching mechanism})$. If a mapping result found by our mapping mechanism matches with the one found manually, the mapping result is defined as correct. If a mapping result found by our mapping mechanism does not match with the one found manually, the mapping result is treated as incorrect mapping. The number of correct mappings found by our matching mechanism is denoted by $\mathrm{N}(\text{manual} \cap \text{matching mechanism})$. Mapping accuracy is defined as the ratio of the number of correct mappings to the number of mappings found by our design. In our experiment, mapping accuracy is measured in percentage. The formula below shows the percentage of mapping accuracy.

$$\text{Mapping accuracy} = \frac{\mathrm{N}(\text{manual} \cap \text{matching mechanism})}{\mathrm{N}(\text{matching mechanism})} \times 100\%$$

TOOLS

Different tools that help to construct ontologies and develop ontology-related applications have been designed in the ontology literature. For ontology and instances construction, ontology editors can be used. Different ontology editors support different ontology languages. For ontology storage and query, query tools have been designed. Some query tools are embedded in ontology editors while some are in the form of APIs. Ontology validators checks whether there exist syntax errors and conflicts in the ontologies. They are used after ontology has been constructed. Ontology-related applications can process ontology information by parsing the ontology using ontology parsers. Queries can be handled by applications through the query APIs.

There are various ontology languages like RDF [RDF], KIF [Gensereth and Fikes, 1992], SHOE [Hefline and Hendler, 2000] and OWL [OWL]. Myriam and Patricia have stated the differences between different ontology languages [Ribière and Charlton, 2000]. "*XML-based languages like RDF seems to be interesting because it allows to share ontologies on the web by using URI and namespace but it is not expressive enough. RDF-based languages like DAML+OIL are interesting because of the rich expressiveness to represent concepts and their relationships and also most common used axioms. The drawback of this language is just the readability of this language.".* Ontology Web Language (OWL) ontologies can share on the web by using URI and namespace because its vocabularies are extension of Resource Description Language (RDF). But OWL has expressive power as it is designed based on DAML+OIL

[DAML+OIL]. OWL becomes a W3C recommendation in February 2004 [McGuinness and Harmelen, 2003]. OWL provides three sublanguages OWL Lite, OWL DL and OWL Full. Each sublanguage has increasing expressive power. The differences between these three sublanguages are: "*OWL Lite supports primarily needing of a classification hierarchy and simple constraint features. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems. OWL Full has maximum expressiveness but it is unlikely that any reasoning software will be able to support every feature of OWL Full.*" [McGuinness and Harmelen, 2003]. In our implementation, OWL DL is chosen as it is a standard proposed by W3C which means it is widely accepted. It uses URI for storing and retrieving ontologies which is also suitable for pervasive computing environment as ontologies are distributed all over smart spaces.

Ontology editors help to create new ontologies and instances. Over 50 ontology editors have been designed. Michael has made comparisons between different ontology editors [Denny, 2002]. BBN OWL Validator [BBN OWL Validator] and OWL Ontology Validator [Manchester OWL Validator] from University of Manchester are tools for checking OWL ontologies. Survey about ontology storage and querying can be found at [Magkanaraki et al., 2002]. General comparisons of different kinds ontology tools can be found at [Asunción Gómez-Pérez et al., 2002]. Other resources for ontology implementation can be found at [OWL Implementations].

In existing ontology mapping tools, experimental ontologies are generated from scratch for evaluation. The advantage of using own designed source ontologies is that time can be saved for translation from one ontology language to another language and ontologies can be interpreted by the mapping tools directly. Translation was necessary in the past because there was no standard for ontology language. Ontology Web Language (OWL) has been proposed to W3C at December 2003 as a standard language for ontology. At present, OWL has been widely accepted in the literature. OWL provides a clear documentation about its syntax. There will be increasing number of OWL ontologies being constructed. These encourage us to use existing OWL ontologies for our experiments such that the practical mapping ability of our mechanism can be reflected.

SOURCE ONTOLOGIES

Four sets of ontologies are used in our experiments. East set contains two ontologies. The first and second sets of ontologies contain general daily terms. The third set of ontologies contains specialized terms. The last set contains pervasive computing environment terms. These sets of ontologies are chosen to show that our matching mechanism suits all kinds of ontologies. All of the ontologies used for evaluation are existing ontologies that can be found in the Web. Some of them are defined using OWL while others use languages like DAML. Conversions are required for ontologies not written in OWL. We use time domain and publication domain for general terms ontologies, football terminologies for specialized terms and service descriptions ontologies used in pervasive computing environment. The OWL ontology for web services (OWL-S) [Martin et al., 2004] is included in the experiment as the last set of ontologies. Table 5 - Table 8 show the detailed information about the ontologies used.

	Time.daml	Time-Entry.owl
Concepts	3	16
Properties and relationships	4	47
Language	DAML	OWL
URI	http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Time.daml	http://www.isi.edu/~pan/damltime/time-entry.owl

Table 5 General terms ontologies in time domain

	SWRC.daml (Semantic Web Research Community Ontology)	Publication-ont.daml
Concepts	56	18
Properties and relationships	143	33
Language	DAML	DAML
URI	http://ontobroker.semanticweb.org/ontologies/swrc-onto-2001-12-11.daml	http://www.daml.ri.cmu.edu/ont/homework/cmu-ri-publications-ont.daml

Table 6 General terms ontologies in publication domain

	Soccer.daml	FootballAgents.owl, FootballObjects.owl, FootballState.owl
Concepts	199	93
Properties and relationships	32	54
Language	DAML	OWL
URI	http://www.lgi2p.ema.fr/~ranwezs/ontologies/soccerV2.0.daml	http://elikonas.ced.tuc.gr/ontologies/football

Table 7 Specilized terms ontologies in football domain

	Services.owl Profile.owl (From OWL-S)	WSDL-ont.daml
Concepts	14	25
Properties and relationships	28	27
Language	OWL	DAML
URI	http://www.daml.org/services/owl-s/1.1/	http://onto.cs.yale.edu:8080/ontologies/wsdl-ont.daml

Table 8 Pervasive computing ontologies

IMPLEMENTATION DETAILS

The matching mechanism has been implemented using Java (Java SDK v.1.4.2) language and Jena API v2.1 [Jena]. Jena is a Java framework for semantic web. It consists of an OWL API for reading and writing OWL ontologies.

A normal desktop computer is used as the simulator of the smart space monitor. It executes the matching mechanism, storing domain ontology and caching mapping results and partial ontologies. The matching mechanism takes two source ontologies, the instances of the two ontologies and the request instance as inputs. The source ontologies are in the form of URIs. For domain ontology, a file URI is used to stimulate it as a local copy stores in the smart space monitor. For applications ontologies and partial user ontologies, the URIs store the place where they are stored. Partial user ontology captures part of the concepts in the original user ontology. The complete user ontology is stored in the URI specified in the partial user ontology. We define the weightings and thresholds in the configuration file. Figure 18 shows the sample configuration file. The concepts presented in the request instance are matched with the second ontology. The first ontology defines the concepts used in the request instance. A matching table with the concepts in the request instance and the concepts in the second ontology are outputted.

```
concept_extract_threshold       0.7
attribute_name_weight           0.3
attribute_candnality_weight     0.1
attribute_datatype_weight       0.2
attribute_instance_weight       0.4
relationship_name_weight        0.3
relationship_cardnality_weight  0.1
relationship_datatype_weight    0.6
instance_weight                 0.4
relationship_weight             0.6
```

Figure 18 Configuration file

RESULTS

Four experiments are carried out using four sets of ontologies. Instance-based ontology mapping requires 30-50 instances to train its learner. The learners are used as the reference of degree of similarity between two concepts. Each experiment provides 35 instances for instance-based ontology mappings to have accuracy results. The time for training the learners in the instance-based ontology mapping is counted. We provide 10 instances for each source ontology for our matching mechanism to stimulate that smart space monitor has allocated 10 instances in the smart space. Historical instance counts are initialized to 0. In other words, the smart space monitor does not have any historical mapping. In later part, we are going to discuss the results with historical records.

Experiment 1

For ontologies in time domain, it is rather small. Request instance used all the concepts in the ontology. Therefore, a full mapping for request ontology is required. Source-based ontology mapping results all incorrect mappings because the concepts with the same name do not have the semantic meanings in the other ontology. Instance-based ontology mapping and our proposed matching mechanism have 100% accuracy. However, our proposed matching mechanism has a greater similarity degree for each pair of mapped concept. Similarity degree is the confidence of the mapping result. This shows that our matching mechanism maps better than the instance-based ontology mapping. Instance-based ontology mapping depends on the string description constructed by concatenate the relationships and properties of the concepts. In the time domain ontology, concepts in the second ontology have much more relationships than the concepts in the first ontology. The string descriptions in the second ontology are much longer than the descriptions in the first ontology which causes a low similarity degree.

For the efficiency, source-based ontology mapping is the fastest because few concepts are involved in the experiment. It is the slowest for the instance-based mapping to because it takes long for training the learners as there are 30-50 instances need to be processed. Our matching mechanism has a reasonable response time. Table 9 summarizes the results of the first experiment.

	Source-based	Instance-based	Our design
Accuracy	0%	100%	100%
Speed	1s	9s	5s

Table 9 Results for experiment 1

Experiment 2

Semantic Web Research Community (SWRC) ontology is a medium size ontology that contains more than 50 concepts and over 100 relationships and properties. We create a request instance that contains 30% of the original instance (i.e. 15 concepts are in the request instance). For source-based ontology mapping, it gives 53% accuracy. Instance-based ontology mapping is 66% correct. Our matching mechanism achieves the highest accuracy because we consider the semantic meanings for the names of the concepts and we generate candidate concepts by merging a concept's parents, children and neighbours in the pre-fetching. This helps to include the right candidates during pre-fetching. While other mechanisms provide one-to-one corresponding mappings, we have intelligence to merge concepts. If partial ontology that contains only the 15 concepts of the SWRC ontology that is the same set of concepts as the request instance, our matching accuracy retains 80% accurate because SWRC ontology is highly connected and concepts have few properties. With partial ontology, the definitions of the request concepts are not affected. Our matching mechanism still has knowledge about the properties and relationships of the request concepts. We do not know the properties and the relationships of its neighbour only. As SWRC is highly connected and has few properties for each concept, the information loss about the neighbours is minimized. Hence, we retain the same accuracy but with lower similarity degrees for the mapping results.

Source-based ontology mapping is still the fastest. Only 3 seconds are required to complete the matching process. Our matching mechanism and the instance-based ontology are slower because the SWRC ontology is large. A long processing time needs to extract information from the instances. Table 10 summarizes the results of the second experiment.

	Source-based	Instance-based	Our design
Accuracy	53%	66%	80%
Speed	3s	34s	19s

Table 10 Results for experiment 2

Experiment 3

In the third experiment, the two ontologies are huge. Both are more that 100 concepts in average. Request instance contains about half of the concepts (i.e. 50 concepts) in the source ontology. All algorithms achieve very promising accuracy. It is above 80%. However, the efficiency of our design and the instance-based ontology mapping are discouraging because we need more than two minutes to finish the mapping. Source-based ontology mapping obtains a good accuracy because the terms are unique in the domain. There are rare cases for a scorer concept having two terms.

	Source-based	Instance-based	Our design

Accuracy	>80%	>80%	>80%
Speed	24s	183s	127s

Table 11Results for experiment 3

Experiment 4

The two source ontologies in this experiment have little overlapping in their domain. OWL-S ontologies describes the overall picture of the web services which WSDL ontology focuses on the terminologies inside a WSDL document. These ontologies should be work together in order to capture all the terms used in the web service domain. We expect that all mapping results have low similarity degrees. Similarity degree less than 0.2 (1 is maximum) is treated as a correct mapping for not finding an appropriate mapping between two concepts. Request instance contains 50% of the concepts in the OWL-S ontologies (i.e. 7 concepts). Source-based ontology mapping achieves 60% accuracy as the two ontologies are from different domain and their concept names are different. Instance-based ontology mapping and our design is 70% correct.

Source-based ontology mapping uses 2 seconds to complete the mapping. Instance-based ontology mapping and our design are slower. Our design is much faster than the instance-based ontology mapping because during the pre-fetching, more unrelated concepts have been pruned out.

	Source-based	Instance-based	Our design
Accuracy	60%	70%	70%
Speed	2s	22s	14s

Conclusion

Our mapping mechanism has achieved an average 82.5% accuracy that is nearly double more accurate than source-based ontology mapping. Although it is only 4% more accurate than the instance based ontology mapping, it is more than 50% faster. It is observed that our design requires more than double the time to execute than source-based ontology mapping, especially in experiment three. Optimization has been studied to make execution faster. The bottleneck for the mechanism is at parsing the ontologies. In the above experiments, source ontologies are retrieved by URIs. There is a large latency for the ontology parser to get the source ontology from the network and parse all the necessary concepts in the ontologies. The larger the source ontologies, the slower are the parsing. We have attempted to find a better parser. However, only Jena can support most of the features proposed by OWL. As source-based ontology mapping simply requires parsing the concept names, Jena can execute at a reasonable speed even thought the ontology is large. For instance-based ontology mapping and our design, the properties, structures and the instances are parsed which causes a large overhead when extracting information from large ontologies. It is recommended that the number of instances to be processed should be minimized and avoids from defining concept with many relationships and properties. The most effective solution is that a more efficient ontology parser should be designed.

The run-time memory usage mainly depends on the size of the source ontologies. During ontology parsing, Jena stores all parsed concepts in memory. For large ontologies such as the set used in experiment 3, memory usage is significantly large. As all mapping mechanisms require Jean to parse the ontologies, the memory usages for all the mechanism are not promising. For experiment 3, more than 300 mega-bytes memories have been used during run-time. And in average, 75% of the run-time memory is used for parsing ontologies. The memory usage of source-based ontology is minimum because its calculation is simple. We used about 15% less memories than source-based ontology mapping because 30-50 instances are stored to train the learners.

In order to show the effect of the historical mappings in our matching mechanism, we re-execute experiment 1 to 4 by using the previous mapping results we have generated. The average time for retrieving the ontology is 4 seconds. It is believed that if we do have some history records in our smart space monitor, it helps to improving our mapping mechanism further. And it is likely the case in the pervasive computing environment as devices keep requesting services and resources.

CONTRIBUTIONS AND CONCLUSION

Ontology mappings are essential and important in pervasive computing environment. It is impossible for ontologies to be shared by all users because it is difficult to capture all the necessary concepts. It is also impossible for ontologies to be mapped and merged at design stage so that it discourages ontology updates. Devices in pervasive computing environment have limited space for storing huge user ontologies, which can only afford the storage of the most frequently used concepts. Partial online ontology mapping, which ontology mapping is performed on partial user ontologies at execution time, is needed in pervasive computing environment. As ontology mappings are performed online, they should be efficient, space saving and accurate. Knowledge propagation should be allowed so that users can have a faster access for the services based on the usage patterns of the previous users.

An effective partial ontology mapping for pervasive computing environment is proposed in this thesis. This research is based on the presences of smart spaces in pervasive computing environment. Existing smart spaces researches focus on the infrastructure of the smart space. New hardware and sensors are designed to detect the environment context, new speech and other interactive recognition are proposed so that users can interact more conveniently with the devices embedded in the smart space. However, all existing researches focus on their own proposed smart space. There is no address about what will happen if the devices and applications that work perfectly in the smart space are moved outside the smart space. Our research fills in this gap by providing online partial ontology mapping. Existing online ontology mapping tools provide off-line ontology mappings. They try to find a one-to-one corresponding between all the concepts defined in the source ontologies. Their similarity formulae only compute similarity using either the instances or the source ontologies, but not both. Our design is a hybrid between them. We use both the source ontologies and the instances for our mapping. A combination of similarity formulae is adopted. Jaccard coefficient is used to taken the distribution of the instance data in consideration. Naïve Bayes' rule is used to calculate the similarity of the string description. K-nearest neighbour function is adopted to compute the similarity between the relationships of two concepts. When comparing two strings, existing mapping tools compares the syntactic similarity between two strings. Some of them have tried to attempt to add a similarity word tables so that words with similar meanings can be identified. Our string similarity measure uses the semantic meaning of the words. We adopt a well-known word ontology: WordNet in our design. The semantic distance between two words is the similarity between the meanings of the words. Pre-fetching and history records are used to improve the efficient of our mapping mechanism. Caching history records and user ontologies can improve the mapping accuracy although partial ontologies are given.

Four experiments have been carried out to evaluate our matching mechanism. Our matching mechanism is about double more accurate than traditional source-based ontology mapping and uses 50% of the time used by traditional instance-based ontology mapping. From the evaluation, it shows clearly that our design can meet the research objective – an effective ontology mapping for pervasive computing environment.

FUTURE WORKS

As pervasive computing environment keeps on evolution, modifications and enhancements of existing works should not stop. Some future works have been identified to enhance our partial mapping mechanism. Syntax for classification of mapping records is needed. Recall that mapping results are stored in smart space monitors and they are classified based on the current date, time and activities. Domain administrators can freely add new rules for the classification. Simple syntax such as weightings is supported at the current stage. An advanced language is needed so that more complicated factors and calculations can be used for classification. The categorization language should be easy to learn, human readable and machine interpreted. Using ontology can satisfy these requirements and seems to be a good choice. However, more detailed studies are required.

A smart space monitor aims to match request instances to services and resources so that better services are provided to users. Our design only focusing on ontology mapping requires an intelligent matching mechanism to provide a full picture of the design of a smart space monitor.

In the current design of our work, we have a smart space monitor to coordinate the devices and the resources. Mappings and results storing are executed in the smart space. It is believed that peer-to-peer mapping is also required when the smart space monitor is busy or absence. Peer-to-peer ontology mapping requires a lightweight mechanism that is flexible to suits in different devices. Memory and energy usage should be minimal so that small devices are able to perform ontology mapping. To make peer-to-peer mapping to be possible, new ontology development tools are required. For example, lightweight and efficient ontology parsers, query tools that can efficiently locate ontologies instances are needed.

Another alternative for peer-to-peer ontology mappings is to distribute similarity calculations to different devices embedded in the smart space. Similarity calculations during ontology mappings can be distributed to idle machines such that machines are better utilized. Distribution of calculation tasks can also help to improve the efficiency of our mapping mechanism. Recall that when parsing ontologies, Jena uses time and memories not proportional to the size of the ontologies. If source ontologies are partitioned and distributed to different devices to perform mapping, it can effectively reduce the time used for ontology parsing. And as our mapping mechanism is purely calculations, the overhead for communications between different devices is little.

Advanced syntax for categorization, intelligent matching and peer-to-peer ontology mappings are the suggested future works. It is believed that there exist many other possible enhancements for applications and systems in pervasive computing environment. With the great efforts from the researchers, pervasive computing environment is going to be reality in the very near future.

BIBLIOGRAPHY

[AIRE] MIT Computer Science and Artificial Intelligence Laboratory. *Agent-based Intelligent Environments (AIRE).* Available at: http://www.ai.mit.edu/projects/aire/

[Arnold et al., 1999] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo and A. Wollrath. *The Jini Specification.* 1999.

[Asunción Gómez-Pérez et al., 2002] Asunción Gómez-Pérez, Mariano Fernández-López, Jürgen Angele, York Sure, Arthur Stutt and Vassilis Christophides (Editors). *A survey on ontology tools*. OntoWeb Ontology-based information exchange for knowledge management and electronic commerce IST-2000-29243. May 2002.

[Austin et al, 2004] Daniel Austin, Abbie Barbir, Christopher Ferris and Sharad Garg. *Web Services Architecture Requirements*. W3C Working Group Note. February 2004.

[Ballagas et al., 2004] Rafael Ballagas (RWTH-Aachen), Andy Szybalski and Armando Fox. *The PatchPanel: Enabling Control-Flow Interoperability in Ubicomp Environments.* In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (Percom 2004). March 2004.

[BBN OWL Validator] BBN OWL Validator. Available at: http://owl.bbn.com/validator/

[Breiman et al., 1984] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone. *Classification and Regression Trees.* Kluwer Academic Publishers, ISBN: 0412048418. January, 1984.

[Brooks, 1997] Rodney Brooks. *The Intelligent Room Project.* In Proceedings of the 2nd International Cognitive Technology Conference (CT' 97). 1997.

[Brumitt et al., 2000] B.Brumitt, B. Meyers, J. Krumm, A. Kern and S. Shafer. *EasyLiving: Technologies for Intelligent Environments.* In Proceedings of Handheld and Ubiquitous Computing 2nd International Symposium (HUC 2000), Springer-Verlag, p.12-29. 2000.

[Champion et al., 2002] Michael Champion, Chris Ferris, Eric Newcomer and David Orchard. *Web Service Architecture.* W3C Working Draft. November 2002.

[Chen et al., 2003] H. Chen, T. Finin and A. Joshi. *Using OWL in a Pervasive Computing Broker.* In Proceedings of Workshop on Ontologies in Agents Systems, held in conjunction with the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems.July, 2003.

[DAML+OIL] *DAML + OIL (March 2001) Reference Description*, W3C note. December, 2001. Available at: http://www.w3.org/TR/daml+oil-reference

[Davies et al., 2003] John Davies, Dieter Fensel and Frank van Harmelen. *Towards the Semantic Web: Ontology-Driven Knowledge Management.* John Wiley & Sons, ISBN 0-470-84867-7. 2003.

[Davis et al, 1993] Davis, Randall, Howard Shrobe and Peter Szolovits. *What is A knowledge Representation?* AI Magazine 14(1) p.17 – 33. 1993.

[Davis and Shrobe, 1983] Davis and Howard Shrobe. *Representing Structure and Behavior of Digital Hardware.* IEEE Computer (Special Issue on Knowledge Representation) 16(10), p. 75–82. 1983.

[Denny, 2002] Michael Denny. *Ontology Building: A Survey of Editing Tools*. November 2002. Available at: http://www.xml.com/pub/a/2002/11/06/ontologies.html

[Dey, 1999] A. K. Dey, G. D. Abowd and D. Salber. *A context-based infrastructure for smart environments.* In P. Nixon, G. Lacey and S. Dobson, editors. Managing Interactions in Smart Environments. First International Workshop on Managing Interactions in Smart Environments (MANSE '99). 1999.

[Doan et al., 2002] A. Doan, J. Madhavan, P. Domingos and A. Halevy. *Learning to Map Between Ontologies on the Semantic Web.* In 11th International WWW Conference. 2002

[Doyle and Patil, 1991] J. Doyle and R. Patil. *Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the Utility of Representation Services.* Artificial Intelligence 48(3), p.261–297. 1991

[Duda and Hart, 1973] R. O. Duda and P. E. Hart. *Pattern Classification and Scene analysis.* John Wiley & Sons, ISBN: 0471223611. June, 1973

[Fensel et al., 2003] Dieter Fensel, James Hendler, Henry Lieberma and Wolfgang Wahlster, editors. *Spinning the Semantic Web: brining the World Wide Web to its full potential.* MIT Press, ISBN: 0-262-06232-1. 2003.

[Gensereth and Fikes, 1992] M. R. Gensereth and R. E. Fikes. *Knowledge Interchange Format (KIF) Version 3.0 Reference Manual*. Computer Science Department, Stanford University, Technical Report Logic-92-1. June 1992.

[Gruber] T. R. Gruber. *A Translation Approach to Portable Ontology Specifications.* Knowledge Acquisition 5, p. 199-220.

[Halimi et al., 1998] Reem Al-Halimi, Robert C. Berwick, J. F. M. Burg, Martin Chodorow, Christiane Fellbaum, Joachim Grabowski, Sanda Harabagiu, Marti A. Hearst, Graeme Hirst, Douglas A. Jones, Rick Kazman, Karen T. Kohl, Shari Landes, Claudia Leacock, George A. Miller, Katherine J. Miller, Dan Moldovan, Naoyuki Nomura, Uta Priss, Philip Resnik, David St-Onge, Randee Tengi, Reind P. van de Riet, Ellen Voorhees. *WordNet An Electronic Lexical Database.* Christiane Fellbaum (Editor), MIT Press, ISBN: 0-262-06197-X. May 1998.

[Hefline and Hendler, 2000] J. Heflin and James Hendler. *Searching the Web with SHOE*. In Artifical Intelligence (AAAI-2000) Workshop on AI for Web Search. 2000.

[Horrocks] Ian Horrocks. *Lecture Notes for "The Semantic Web: Ontologies and OWL*". Department of Computer Science, the University of Manchester.
Available at: http://www.cs.man.ac.uk/~horrocks/Teaching/cs646/

[Horrocks and Schneider, 2003] Ian Horrocks and Peter F. Patel-Schneider. *Three theses of representation in the semantic web.* In proceedings of the Twelfth International World Wide Web Conference (WWW 2003), p. 39 – 47. 2003

[Jena] Jena : Semantic Web Framework for Java. Available at : http://jena.sourceforge.net/

[Johanson et al., 2002] Brad Johanson, Armando Fox, Terry Wingograd. *The Interactive Workspace Project: Experiences with Ubiquitous Computing Rooms.* IEEE Pervasive Computing Magazine 1(2). April – June 2002.

[Judd and Steenkiste, 2003] Glenn Judd and Peter Steenkiste. *Providing Contextual Information to Pervasive Computing Applications.* IEEE International Conference on Pervasive Computing (PERCOM). March, 2003.

[KIF] Knowledge Interchange Format (KIF).
Available at: http://logic.stanford.edu/kif/dpans.html

[Lee, 2000] Tim Berners-Lee. *Semantic Web on XML.* Keynote speech at XML 2000. 2000. Slides available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl

[Magkanaraki et al., 2002] A. Magkanaraki, G. Karvounarakis, T. T. Anh, V. Christophides and D. Plexousakis. *Ontology Storage and Querying.* Technical Report No. 308. April 2002.

[Manchester OWL Validator] OWL Ontology Validator from University of Manchester.
Available at: http://phoebus.cs.man.ac.uk:9999/OWL/Validator

[Martin et al., 1999] D. Martin, A. Cheyer and D. Moran. *The Open Agent Architecture: a framework for building distributed software systems.* Applied Artificial Intelligence, 13(1/2), p.91–128. 1999.

[Martin et al., 2004] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila Mcllraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan and Katia Sycara. *OWL-S: Semantic Markup for Web Services Release 1.1.* 2004.

[McGuinness et al., 2000] D. L. McGuinness, R. Fikes, J. Rice and S. Wilder. *An Environment for Merging and Testing Large Ontologies.* In A. G. Cohn, F. Giunchiglia and B. Selman, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the 17th International Conference (KR2000). 2000.

[McGuinness and Harmelen, 2003] Deborah L. McGuinness and Frank van Harmelen (Editors). *OWL Web Ontology Language Overview.* W3C Proposed Recommendation. December, 2003.
Available at: http://www.w3.org/TR/2003/PR-owl-features-20031215/

[Michael et al., 2002] Michael Champion, Chris Ferris, Eric Newcomer and David Orchard. *Web Services Architecture. W3C Working Draft.* 2002.
Available at: http://www.w3.org/TR/2002/WD-ws-arch-20021114/#whatisws

[Mitchell, 1997] T. Mitchell. *Machine Learning.* McGraw Hill, ISBN: 0070428077. 1997.

[Mitra et al., 2000] P. Mitra, G. Wiederhold and M. Kersten. *A Graph-Oriented Model for Articulation of Ontology Interdependencies.* In Proceedings Conferences on Extending Database Technology 2000 (EDBT'2000). 2000.

[Noy & McGunness, 2001] N. F. Noy and D. L.McGunness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge System Laboratory Technical Report KSL-01-05 and Standard Medical Informatics Technical Report SMI-2001-0880. March, 2001.

[Noy & Musen, 2000] N. F. Noy and M. A. Musen. *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment.* In 17th National Conferences on Artificial Intelligence (AAAI-2000). 2000.

[OWL] Ontology Web Language (OWL). Available at: http://www.w3.org/2001/sw/WebOnt

[OWL Implementations] OWL Implementations.
Available at: http://www.w3.org/2001/sw/WebOnt/impls

[Oxygen] MIT Laboratory for Computing Science and MIT Artificial Intelligence Laboratory. MIT Project Oxygen: Pervasive, Human-Centered Computing.
Available at: http://oxygen.lcs.mit.edu/index.html

[Peters and Shrobe, 2003] Stephen Peters and Howie Shrobe. *Using Semantic Networks for Knowledge Representation in an Intelligent Environment.* In 1st Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '03). March 2003.

[Quan et al., 2003] Dennis Quan, David Huynh and David R. Karger. *Haystack: A Platform for Authoring End User Semantic Web Applications.* In proceedings of 2nd Semantic Web Conference. 2003.

[Quinlan, 1993] J. R. Quinlan. *C4.5: Programming for machine learning.* Morgan Kaufmann, ISBN: 1558602380. January, 1993.

[RDF] Resource Description Framework (RDF). Available at http://www.w3.org/RDF

[Ribière and Charlton, 2000] Myriam Ribière and Charlton Patricia. *Ontology Overview from Motorola Labs with a comparison of ontology languages.* December 2000.

[Rijsbergen, 1979] van Rijsbergen. *Information Retrieval*. London: Butterworths, Second Edition. 1979.

[Roman et al., 2002] M. Roman, C.Hess, R. Cerqueira, A. Ranganathan, RH Campbell, K. Nahrstedt. *Gaia: A Middleware Infrastructure for Active Spaces.* IEEE Pervasive Computing Vol. 1, No. 4, p. 74-83. October – December, 2002.

[Satyanarayanan, 2001] M. Satyanarayanan. *Pervasive computing: Vision and Challenges.* IEEE Personal Communications, p.10-17. August, 2001.

[Stumme & Madche, 2001] G. Stumme and A. Madche. *FCA-Merge: Bottom-up Merging of Ontologies.* In 7th International Conference on Artificial Intelligence (IJCAI'01), p. 225-230. 2001.

[Su, 2002] Xiaomeng Su. *A text categorization perspective for ontology mapping.* Technical report, Department of Computer and Information Science, Norwegian University of Science and Technology. 2002

[TCML] Topic Map Constraint Language (TCML).
Available at: http://www.isotopicmaps.org/tmcl/

[UPnP] University Play and Play. UPnP Forum. Available at: http://www.upnp.org

[XML] Extensible Markup Language (XML). Available at: http://www.w3.org/XML/