Web-Based Semantic Pervasive Computing Services

Yugyung Lee, *Member, IEEE,* Soon Ae Chun, *Member, IEEE,* and James Geller

***Abstract*— Pervasive Computing refers to a seamless and invisible computing environment which provides dynamic, proactive and context-aware services to the user by acquiring context knowledge from the environment and composing available services. In this paper, we demonstrate how heterogeneous Web services can be made interoperable and used to support Pervasive Computing. We present an architecture how a service flow can be automatically composed using syntactic, semantic and pragmatic knowledge. Thus, this paper addresses three problems: (1) How do heterogeneous Pervasive Computing services interoperate in a Pervasive Computing service flow, composed by using syntactic, semantic and pragmatic knowledge; (2) How do we define, distinguish between, and justify the need for these three different kinds of knowledge to be used in service descriptions; and (3) How can we perform ontology integration to enable the automatic composition of Web services into a service flow. A Pervasive Computing prototype system, based on this architecture, has been implemented as a proof-of-concept.**

***Index Terms*— Ontology, Semantic Web Services, Service Discovery and Composition, Pragmatic Knowledge, Pervasive Computing**

I. Introduction

There are reasons to believe that Pervasive Computing may be the next frontier of computing after the Internet revolution. Pervasive Computing aims to revolutionize the current paradigm of human-computer interaction. Computers have been used in various aspects of human life, but in most cases human beings have had to adapt their behavior to existing systems. Pervasive Computing, as envisioned by Weiser [49], is a computing environment in which computing systems weave themselves into the fabric of everyday life and become invisible. Invisibility is the most important aspect of Pervasive Computing. The user is exposed to a few sets of services available to him/her and is oblivious to the complex system implementing those services [38]. This takes the human-computer interaction into a whole different dimension, where the user is surrounded by a complete smart environment with devices/sensors communicating with each other and aggregating their functionalities to provide a set of consolidated services.

In order to build a Pervasive Computing environment, existing methodologies use smart devices, which have some processing power and are specialized to perform a set of specific tasks. Usually the user needs to carry these devices with her/him as s/he moves either within or across Pervasive Computing environments. However, we present an alternate approach and use Semantic Web technologies for Pervasive Computing environments. This allows context information to be stored on the Web, Pervasive Computing services to be dynamically composed based on Web Services, and then shared across Pervasive Computing environments via the Web to provide Pervasive Computing services.

Y. Lee is with the University of Missouri - Kansas City.

There are several challenges that we are facing in Pervasive Computing. First, it requires acquiring context from the environment and dynamically building computing models dependent on context. Context-awareness is a pivotal aspect of Pervasive Computing. Dey and Abowd [10] defined the concept of context as a piece of information that can be used to characterize the situation of a participant in an interaction. Brown [3] defined the context as location, environment and/or identity of people and time. By sensing context information, context enabled applications can present context relevant information to users, or modify their behavior according to changes in the environment. Context is however very subjective, in the sense that it can include any factors that may affect a users interaction with the system. This makes the modeling of context extremely difficult especially because we have to capture abstract and subjective factors.

In the past few years, the WWW has changed from being nothing more than an indexed repository of documents towards being a repository of interconnected services and documents. Web users are now routinely checking the Web for services such as currency converters, mortgage calculators, shortest driving distance with directions generators, etc. Unfortunately, not every required service is available on the Web, and if it is, it might be hidden at position 1921 of 2000 search engine hits. Therefore Web research has turned to the time-honored approach of its parent discipline and attempts to provide complex services by, in effect, combining simple services in the way of a workflow of services, what we call a *service flow*. However, the problem of creating a service flow for a given specification is difficult, and it is a part of the vision of the Semantic Web [2] to let roaming agents perform this difficult task. For that purpose, (simple) services need to be described in an agent-readable form.

The automatic composition of services requires more than descriptions of service capabilities and input/output parameters. Rather, a service should also indicate in what situations and in what ways it should be used. This is comparable to the manual of an electronic device that provides a service. For example, a cell phone manual describes “use cases” of the services that the cell phone offers: Making phone calls, playing games, maintaining a calendar, etc. In case of an emergency, most cell phones allow a 911 call without the payment of a fee. While it is obvious that this kind of knowledge needs to be provided and bundled with the device itself, it is only recently becoming clear that Web services need to have the same kind of knowledge attached to them.

We call this additional level of description of Web services *pragmatic* or contextual knowledge. A service should be described by a pragmatic annotation that represents this pragmatic knowledge, in addition to the semantic and syntactic knowledge that describes the necessary parameters and functionalities of the service. We propose an ontology as a model for representing knowledge to describe services. Specifically, we use ontologies to represent syntactic, semantic and pragmatic knowledge about services.

Clearly, the service composition faces an immediate problem when every service is described using terms from its own underlying domain. The pragmatic and semantic knowledge ontology may contain a collection of these terms [2]. Therefore, the discovery of correct component Web services will often require additional preliminary steps to integrate the ontologies used to describe these Web services. In many cases it will be necessary to integrate the ontology of an agent, searching for a service, with an ontology describing a service. This will have to be done on the fly and at great speed to decide whether a specific service is a possible candidate for the desired service flow.

In this paper, we demonstrate how heterogeneous Web services can be made interoperable and used to support Pervasive Computing. We present an architecture how a service flow can be automatically composed using syntactic, semantic and pragmatic knowledge. Thus, this paper addresses three problems: (1) How do heterogeneous Pervasive Computing services interoperate in a Pervasive Computing service flow, composed by using syntactic, semantic and pragmatic knowledge; (2) How do we define, distinguish between, and justify the need for these three different kinds of knowledge to be used in service descriptions; and (3) How can we perform ontology integration to enable the automatic composition of Web services into a service flow.

The three different types of compositional knowledge are expressed by compositional rules that a software agent can use for the automatic generation of a service flow. We present an ontology for these compositional rules, applying them to the description of Web services. OWL-S and Jena rule (HP Jena [19]) are used as formats for compositional knowledge [9]. The paper also illustrates one approach how to integrate terms from several ontologies in an efficient manner, using the framework of Terminological Knowledge Bases [13]. These are two-level ontologies where the semantics of concepts at the lower levels are constrained by assigning concepts to semantic types at the upper level.

The paper is organized as follows. In Section II, we introduce our motivating application. In Section III, we discuss different types of compositional knowledge, followed in Section IV by a semantic methodology for heterogenous service composition. In Section V, we present an overall system architecture and implementation for semantic Pervasive Computing services. Relevant work and conclusions are presented in Section VI and Section VII, respectively.

II. Motivation

Existing methodologies for implementing a Pervasive Computing environment use smart devices, which have some processing power and are specialized in performing a set of specific tasks. Usually the user needs to carry these devices with her/him as s/he moves either within or across Pervasive Computing environments. These devices are not readily available and are often difficult to build. The issues that limit fabrication of such personal devices are limitations like battery power, shape and weight, making practical use of such devices extremely difficult. The advantage of using smart devices is their ability to communicate with each other by building and storing contextual information, which may be used by the Pervasive Computing environment to offer services based on the stored information. In addition, current devices are costly, and thus it is difficult to replace all current devices with smart devices to implement Pervasive Computing environments. Finally, smart devices need to have functionality beyond what they are expected to do, because they are integral to the environments.

Our solution reduces the need for smart devices by using the Semantic Web to build dynamic service composition knowledge (context) models as a user moves from one environment to another. We can achieve dynamic building of contexts by sharing knowledge and context information between local Pervasive Computing environments through the Semantic Web. Furthermore, Pervasive Computing services can be dynamically composed by considering the contexts determined by the Pervasive Computing framework. In this approach we can utilize currently available resources (data, information, services, devices, etc), letting the devices do their basic tasks without saddling them with any pre-requisites to participate in Pervasive environments. Also we believe that this approach will help us quickly implement Pervasive Computing, since we can use currently available resources and do not need specialized devices.

III. Definitions: Syntactic, Semantic, Pragmatic Knowledge

In a previous publication we have introduced the use of syntactic, semantic and pragmatic knowledge for services and workflows [42]. Of these, syntactic and semantic knowledge are well known in computer science, but this is less so for pragmatic knowledge. Pragmatic knowledge has been an issue mostly in philosophy of language and some branches of linguistics, such as discourse understanding [25]. Giving a general definition of pragmatic knowledge and distinguishing it from semantic knowledge is difficult. However, by limiting ourselves to the fairly well defined environment of services, the distinction becomes easier.

Instead of jumping directly into a set of definitions, we will clarify our distinctions between syntactic knowledge, semantic knowledge and pragmatic knowledge by the example of a cellular phone. Our basic approach is to observe the different mistakes that users of a cell-phone may make. Every user that does NOT make those mistakes appears to have some knowledge on how to correctly use a cell phone.

This approach is metaphorically related to Cognitive Science methods that study the working brain using data from aphasia patients. By linking observable damage to certain areas of the brain with observable performance failures, it becomes possible to hypothesize which part of the brain is responsible for which cognitive activity. Instead of looking for physical damage linked to performance failures we are looking for presumed missing knowledge items that would lead to performance failures.

The mistakes that a cell phone user can make vary widely, and therefore different kinds of mistakes give rise to observations of different kinds of knowledge.

We will now turn to syntactic errors. If a person types in a 6 digit phone number in the continental United States, this should (after some waiting period) result in a voice saying "your call cannot be completed as dialed." Attempting to dial a six digit phone number is a syntactic error. In programming languages, syntactic errors can (usually) be detected by a compiler, and similarly, dialing too few digits can be detected by the phone SW. Thus, the essence of a syntactic error is that it violates simple ("knowledge free") rules that can be checked mechanically without reference to any additional knowledge. Thus, syntactic knowledge is knowledge which can be expressed in rules that do not refer to any outside database. As usual in computer science, syntactic knowledge is easier to understand than semantic knowledge.

We next turn to semantic errors and semantic knowledge. A person attempting to call a friend, who has memorized her phone number incorrectly, is making a semantic error. He will end up dialing a different person, or possibly a non-existent phone number. To verify that a number is non-existent, it is necessary, at least in theory, to have a knowledge base of all existing phone numbers. Thus, this cannot be checked by a set of self-contained rules and requires a substantial data base. To detect that a user is calling a wrong phone number, the phone would need to "know" who the user is trying to call and also need to know the phone number of that person. Again, this requires a database and cannot be done with self-contained rules alone.

In some cases, a database is not sufficient for semantic knowledge. The process of reasoning is essential to meaningful knowledge processing. For example, a person might have the phone number of a friend, but without the area code. Making a phone call without area code would result in a call to the given phone number in the area of the caller. If the caller and the callee live in the same area code, this would be a successful call, but otherwise it would result in a failure by calling a different person than the one intended.

If the caller lives in New Jersey and knows that his friend lives in Manhattan, he can still place a successful call. By knowing the area code for manhattan, he will be able to reason out and construct a complete and correct phone number. Thus, besides simple rules and a database we need to assume a knowledge base with some reasoning abilities for semantic knowledge.

However, semantic knowledge clearly does not avoid all errors. One should not call a friend at 1:00 AM. Doing it would make the friend upset and would be a pragmatic error. On the other hand, if the house of the friend is on fire, one should call him at any time, even at 1:00 AM. If one has an emergency, he should call 911. If one has an emergency and is in Austria, he should call 112.

If one has a phone that does not work, he should call 611 (presumably using another phone that works). If one does not know the phone number of a friend, one should call 411 to get it. If one has no money to call, he should only call 888 and 800 numbers or attempt to make a collect call. All these rules describe pragmatic knowledge that links situations with the actions that should be taken. They do not just require knowledge as it is stored in a database or reasoning that involves a knowledge base, they require situational awareness.

In the simplest cases, situational awareness deals with time (Is this a reasonable time to phone?) and space (In which country am I? In which area code am I?) Sometimes complex combinations of time and space need to be reasoned about. If your friend just left your house and lives an hour's drive away, it makes no sense to call him at his home phone number after 5 minutes. Determining what constitutes an emergency that would allow one to call at all hours, or to call 911 requires even more complex knowledge of ownership and values of objects. Pragmatic knowledge also includes social relationships and authorities of people. Thus, the essence of pragmatic knowledge for services is that it incorporates some kind of knowledge of the context (or situation) in which a service should be used. A subset of this kind of knowledge may be expressed in terms of time and space, which themselves are already (for time) or in the foreseeable future (GPS systems for space) integrated into all computer systems. This is the point where Pervasive Computing becomes important.

Social situations may be incorporated into services, in organizations with well defined hierarchies such as in the military. A service may provide more information for privileged users ("super-users").

Thus, the border line between semantic knowledge and pragmatic knowledge in our approach is that semantic knowledge relies on the retrieval or the reasoning with knowledge from a knowledge base, while pragmatic knowledge requires retrieval of situational information from an outside source. Thus, we have formulated a border line between semantic knowledge and pragmatic knowledge, limited to our services domain.

We will now carry over these general remarks to actual services. If we view a service as a procedure or function that takes certain inputs and produces certain outputs, then we can require the same syntactic constraints as on functions:

For functions, the number, order, directionality (in/out), data type and optionality (mandatory/optional) of inputs and outputs need to be correct. Otherwise there is a syntactic mismatch. Our view of syntactic knowledge is guided by this idea.

Syntactic knowledge for a service consists of self-contained rules that can automatically determine whether the input parameters received by the service are correct in number, order, directionality, data type and optionality.

Semantic knowledge for a service consists of rules that describe how to correctly use the service. These rules may access an outside database to retrieve information and/or an outside knowledge base to reason with information. We call them semantic rules.

Pragmatic knowledge for a service consists of rules that describe in what situations to use the service. These rules may access the same information as the rules of semantic knowledge. In addition, these rules may access information about the current situation, such as time, location of the service requester and the service provider, hierarchical ("privilege") status of the service requester and the service provider, etc. We call them pragmatic rules.

IV. A Methodology for Heterogenous Service Composition

In this section we address the issue of the heterogeneity of Web services that were developed independently, using terms from different ontologies, and present the methodology to match those concepts from different ontologies, called OnInt (Ontology Integration).

A. Ontology Integration

Every realistic service model consist necessarily of two kinds of elements. On one hand there are elements that are specific to OWL-S itself. These elements correspond to what would be called "reserved words" in a programming language. The number of types of these elements is strictly limited, but the elements are composable, in the same way in which FOR loops in a programming language may be composed by nesting. On the other hand there are elements that are specific to the service domain itself. These elements would correspond to the variable names, constant names, function names, type names and module names of a program. Every good program closely mirrors its domain by the choice of meaningful function and variable names. Therefore, there are as many different (sets of) function names as there are domains. The same applies to services. Thus, a good service description will need to use terms from its underlying domain, and the number of terms available will be as unlimited as the domains themselves.

When an agent is looking for a service, it will carry with it a description of the kind of service that it is looking for, in terms of its underlying domain. It will encounter service descriptions using the same or different terms from the domain of the service provider. Unfortunately, even if the agent domain and the service provider domain are the same, that does not mean that the agent and the provider can smoothly interact, because there is no global shared ontology of domain terms. The situation is comparable to an Italian tourist in America that tries to order a meal from a Chinese waiter, and both know only subsets of English food language. The waiter and the tourist cannot start talking with each other directly. They need to establish a common language first, by discovering shared terms and finding mappings (hard!) between differing terms.

In ontology research this kind of process is described as a form of ontology integration. The heart of this process is to find mappings between differing terms for the same concept. This integration process has to be performed quickly, as one agent may be visiting many services with service ontologies in its attempt to construct a service flow. For any pair of sizable ontologies it is out of the question to perform a brute force attempt of matching every term in one ontology with every term in the other ontology. To overcome this problem we have developed an extensive method of semantic specification and semantic integration, using two-level ontologies, which is used as a precursor to a the actual integration algorithm [13], [16]. This semantic integration algorithm greatly limits the number of matching attempts involved in every integration task. Details of this matching and integration method would go well beyond the scope of this paper, but we summarize the basic ideas and some of the formalism in this section.

B. Two Level Ontologies for Integration

Definition: Terminological Knowledge Base. We call any structure that consists of (1) a semantic network of semantic types; (2) a thesaurus of concepts; and (3) assignments of every concept to at least one semantic type a *Terminological Knowledge Base* (TKB).

$$TKB = < \hat{\mathcal{C}}, \hat{\mathcal{S}}, \mu > \tag{1}$$

in which $\hat{\mathcal{C}}$ is a set of concepts, $\hat{\mathcal{S}}$ is a set of semantic types (i.e., high-level categories), and μ is a set of assignments of concepts to semantic types. Every concept must be assigned to at least one semantic type. The opposite condition does not hold. We will use capital letters to represent semantic types and small letters to represent concepts.[1]

$$\hat{\mathcal{S}} = \{W, X, Y, ...\}; \quad \hat{\mathcal{C}} = \{a, b, c, d, e, ...\} \tag{2}$$

Finally, μ consists of pairs (c, S) such that the concept c is assigned to the semantic type S.

$$\mu \subset \{(c, S) |\, c \in \hat{\mathcal{C}} \,\&\, S \in \hat{\mathcal{S}}\} \tag{3}$$

We define that two concepts c, d are similar, $c \simeq d$, if they are assigned to exactly the same set of semantic types of a TKB.

$$c \simeq d : \, \forall S \in \hat{\mathcal{S}}\, [(c, S) \in \mu \;\Leftrightarrow\; (d, S) \in \mu] \tag{4}$$

If two concepts c and d are assigned to the same semantic type X, then these two concepts have similar semantics. On the other hand, if a concept a is assigned to X and a concept b is assigned to Y, then a and b will have semantics that are not similar in the formal sense defined above. The case of concepts with sets of several assigned semantic types that may have a non-empty intersection is discussed in [30]. With our notion of similarity, we need to decide how strict we want to be with respect to accepting partial matches of concepts.

There is a spectrum of what requirements one could impose to accept two concepts as matching. On one extreme, one might insist that there be only perfect matches. The other extreme is to insist that all (or almost all) concepts of the smaller ontology are matched against concepts in the larger ontology, as long as there is at least some structural similarity. This extreme could be based on the assumption that both ontology designers did a reasonable job to cover the domain, and thus all fundamental concepts simply have to appear in both ontologies, no matter what each one is called, and no matter how exactly it is structured. Our solution is closer to the

[1]Both roman and italic fonts

second extreme. We are optimistic that with the development of the Semantic Web many subdomains of the world will be described by ontologies which cover their domain to a reasonably complete degree. Thus, one would expect that most concepts of one such ontology exist in the other ontologies for the same domain. We note that for people the lack of exact matches does not normally make communication impossible. Indeed, philosophers would point out that we can never be sure of how another person is thinking about a concept, a fact denoted as “solipsism.”[2]

Now we will show how the two-level structure limits the required number of matching attempts. By our construction of the Terminological Knowledge Bases, two concepts, q from TKB$'$ and r from TKB$'_2$, can only match if they are both assigned to the same semantic type. There are three cases:

(1) Assume a semantic type S exists in TKB$'$ that has assigned concepts $x, y, z, ...$ Further assume that S does not exist in TKB$'_2$ or, there are no concepts assigned to S in TKB$'_2$. Then, by the similarity definitions given above, no concepts corresponding to $x, y, z, ...$ exist anywhere in TKB$'_2$. Thus, these concepts do not need to be matched at all.

(2) The above observation applies in reverse also. If a semantic type S exists in TKB$'_2$ that does not exist in TKB$'$, then the concepts $x, y, z, ...$ assigned to S will not have corresponding concepts anywhere in TKB$'$. Thus, these concepts do not need to be matched at all.

(3) Concepts assigned to the semantic type S in both TKB$'$ and TKB$'_2$ are potentially similar ($\simeq$) and need to be matched. As mentioned above, we allow partial matches between concepts that have been determined to be similar. The exact cut-off is decided by a threshold value.

C. Scoring Concept Similarities

Now we describe details of how scores for concept similarities are computed. We use three aspects to determine whether a match exists between similar concepts. Initially we rank pairs of concepts according to their terms (or synonyms) and then according to attribute similarity. After establishing some initial matches in this way, we use relationships that point from one concept to another concept to iteratively recompute the similarity value between two concepts.

1) Ranking Concepts by their Terms: If two concepts have similar names (defined below, based on bigrams) then they are possibly matches. The existence of synonyms and homonyms causes problems for concept matching. We include the use of synonyms during the concept matching step itself. If no match is found for a concept, then it is attempted to use its synonyms for matching.

The bigram approach [23] is known to be a very effective, simply programmed means of determining a similarity measure between strings. The bigram approach consists of three steps. (1) Sequences of two consecutive letters within a string are extracted (e.g., the word “calendar” contains ’ca’, ’al’, ’le’, . . . ’ar’); (2) Two sequences of bigrams are compared, and a raw similarity score is computed; (3) A matched score is computed from the raw score, i.e., the number of the common bigrams is divided by the average number of bigrams in the two strings.

[2]IEP

2) Ranking Candidates by Attributes: Assume that we are given a pair of concepts from two different ontologies. These concepts have different terms, therefore, a priori there is no reason for a computer to assume that they are in fact describing the same concepts. In order to establish whether they are indeed the same concept, we need to compare attributes.

We assign to every pair of concepts a score as follows.

- Two concepts, that have the same number of attributes, are considered perfectly matched, with a score of 1, only if for every attribute in one concept there is an attribute in the other concept of the same name and same data type,
- If two attributes (of two concepts from two ontologies) have the same name but are of different data types, we assign them a score of k ($k < 1$, $k \gg 0$).
- Then we compute the ratio of matched attribute scores divided by the number of attributes of the concept that has more attributes.
- The final decision about similarity is made, based on a minimum threshold for the computed combined score.

3) Ranking Candidates by Relationships using Propagation: In the previous step we have established matches between concepts from two different ontologies, based on pairs of terms and attributes. However, two concepts that point to exactly the same concepts with the same relationships are presumably very similar to each other. We view the relationship targets as data types, and two concepts that point to all the same data types are likely to be quite similar. However, we would have a chicken and egg problem here, if we start with considering relationships from the beginning. That is the case because the relationships targets cannot be used for matching if they themselves have not been matched up.

This is why we start by matching up a few concepts using terms and attributes alone. By this step, we create an initialization for matching up additional concepts by using relationships. Thus, two concepts with different names that point to several target concepts that all have been matched up between two ontologies are presumably themselves a match. We can use a similar ratio criterion as for attributes, however, now the targets carry more semantics than the undifferentiated data types of attributes. Thus, we are willing to assign a pair of relationships a high score if the targets are the same OR if the relationship names are the same. Let us assume now that a set of concept pairs has been established such that the concepts in each pair match and are from two different ontologies. Then any pair of concepts that point to these matched concepts would also be considered highly ranked for being matches. Thus, after establishing initial matches, we continue ranking concepts by similarity using a process similar to a Waltz filtering [48].

The process of finding matches needs to be recomputed until a score change of one concept pair does not result in a score change of any concepts pointing to that pair anymore. This state of equilibrium can be achieved, as we are using a threshold. If there are only changes that do not cross the threshold, the update process would terminate.

4) Combining Matching Scores: Two concepts are considered matched if their terms, their attributes and their relationships are (on average) similar. A weight is assigned to each similarity aspect of a concept (term similarity, average attribute similarity, average relationship similarity). Considering these three criteria, we now compute the degree of the similarity of concepts from two distinct ontologies. For this purpose, we use a Multiple Attribute Decision Making (MADM) approach, a simple additive weight-based solution [20]. This approach determines a combined score of concept matches between ontologies. Let C_i = $\{C_{i1}, C_{i2}, \ldots C_{im}\}$ and C_j = $\{C_{j1}, C_{j2}, \ldots C_{jn}\}$ be sets of concepts for given ontologies, and let F = $\{F_1, F_2, \ldots F_p\}$ be a set of p features (in this paper p = 3) that characterize the degree of similarity. The weight vector W reflects the importance of each attribute $W = \{W_1, W_2, \ldots, W_p\},\ where \sum W_i$ = 1. We compute the scores for each of the p features for each of l matching cases ($l \ll n$ or m) in a decision matrix $D = d_{ij}$.

The method is comprised of three steps: first, scale the scores into a range [0, 1], with the best score represented by 1, using

$$r_{ij} = (d_{ij} - d_{jmin})/(d_{jmax} - d_{jmin}) \tag{5}$$

Second, apply weights and third, sum up the values for each of the alternatives, using

$$S_i = \frac{\sum W_j r_{ij}}{l} \tag{6}$$

After a combined score has been computed, we compare the weighted sum with a given threshold α. Some matches may be lacking attributes or relationships. In this case, a weight of zero will be assigned to these aspects of a concept. All combined similarity values greater than α are stored in a matrix G_T. In this matrix, rows correspond to concepts from one ontology. Columns represent concepts from the other ontology. At each row/column intersection the similarity value of two terms is stored.

Subsequently, concept pairs with similarity values above the threshold are constructed, starting with the maximal similarity value. If there are several equal maximal similarity values, they are processed in random order. Whenever the next largest similarity value has been identified between two concepts c and d, then the complete row of c and the complete column of d in the similarity matrix G_T are set to 0. This is because c and d are not available for matching anymore. Details of this algorithm are given in [30].

Our approach to ontology integration simplifies the matching task by identifying sets of semantically similar concepts before starting with the actual matching steps. Terms from two ontologies only need to be compared for integration if they are already classified as semantically similar. Therefore, our methodology reduces the computational cost of the matching operations. Fewer pairs of terms have to be matched against each other. For more details on the Terminological Knowledge Base Framework see [13], [16].

V. Pervasive Computing Services

There are several challenges that we are facing in Pervasive Computing. The first is, how to acquire context models from the environment and dynamically build computing models dependent on context. By sensing context information, context enabled applications can present context information to users, or modify their behavior according to changes in the environment. Secondly, the environment should be flexible enough to provide composite services by incorporating existing services of the Pervasive Computing environment at run time. Here we show how the proposed approach, service composition, helps in dealing with these challenges in Pervasive Computing.

A. Context Ontologies

The context ontologies are two-part ontologies in OWL format: The upper ontology provides a description of various concepts that together characterize a particular situation. The lower ontologies describe each of these concepts in more detail. For instance, the upper ontology contains Location as one of the concepts while the lower ontology contains the description of the current location i.e. location of rooms, floors etc.

The context ontologies consist of concepts from our own User Profiling ontology and several extensions of the CONON (COntext ONtology) [50], the SOUPA (the Standard Ontology for Ubiquitous and Pervasive Computing[3]), which is the outcome of a collaborative effort between researchers in the field of Pervasive Computing and Semantic Web and the *Content Selection for Device Independence* (DISelect[4]). The upper ontology is based on the CONON ontology for context. CONON provides all the basic concepts needed to model context. This ontology however needs lower ontologies that are extensions to reflect the current domain. For instance the Location concept can be extended to model a building or a city. The descriptions of Location and Time as context elements was obtained from SOUPA. The following context elements are of specific interest.

- Individual: Representing the person whose context is represented. Our Pervasive Computing framework, called SeMEther [40], contains a user profile ontology, which was mapped to the individual concept in the CONON ontology.
- Time: The temporal features associated with the current situation. They were designed based on the specifications of the SOUPA ontology.
- Location: The spatial location features of the person involved. They were designed based on the specifications of the SOUPA ontology.
- Computing Entity used: The Computing Entity used by the person in the current context. The SeMEther in its current implementation does not have extensive device modeling. The only two existing concepts describing the devices are MobileDevice and StaticDevice which indicate whether a device has a fixed location or whether its location can change. We are currently incorporating the *Content Selection for Device Independence* (DISelect[5]) and *Foundation for Intelligent Physical Agents* (FIPA[6])

[3]http://pervasive.semanticweb.org/soupa-2004-06.html
[4]http://www.w3.org/TR/cselection/
[5]http://www.w3.org/TR/cselection/
[6]http://www.fipa.org/

Ontology Type	SeMEther Ontologies	
Upper Ontology	Context Ontology (CONON)	
Lower Ontologies	Concept	Specific Ontologies
	User context	SeMEther User Profile Ontology
	Location	Standard Ontology for Ubiquitous and Pervasive Computing (SOUPA)
	Time	Standard Ontology for Ubiquitous and Pervasive Computing (SOUPA)
	Computing Entity	W3C DISelect, FIPA Device ontology
	Activity	AIAI Activity ontology

TABLE I
SeMEther Ontologies

```
[Rule1:
   (?L
     http://SeMEther/ontology/locationcontext#UserInLocation
     http://SeMEther/ontology/location/floor1#room101)
   (?$T_s$
     http://SeMEther/ontology/timecontext#EventStartTime
     http://SeMEther/ontology/time/startTime#1100)
   (?$T_e$
     http://SeMEther/ontology/timecontext#EventEndTime
     http://SeMEther/ontology/time/endTime#1300)
      ->
  (?U
     http://SeMEther/user/usercontext#status
     http://SeMEther/user/usercontext#busy )]

[Rule2:
   (http://SeMEther/ontology/timecontext#Time
    http://SeMEther/ontology/timecontext#CurrentTime
    ?val),
    greaterThan(?val,
     http://SeMEther/ontology/timecontext#EventStartTime)
    lessThan(?val,
     http://SeMEther/ontology/timecontext#EventEndTime)
      ->
   (http://SeMEther/ontology#EventManager
   http://SeMEther/ontology#sendMessage
   http://SeMEther/PatientHabits#CallForwarding)]
```

TABLE II
SeMEther Service Composition Rules

Ontologies that provide a formal description of devices and device capabilities.

- Activity Performed: The activity being performed by the person. This activity can be deduced (e.g. from his schedule) or explicitly specified.

B. Rules for Pervasive Services Composition

The service composition knowledge (context) model can be used to infer new knowledge about a user's situation. For instance, consider a messaging service that sends messages to the users in their current locations. The messages are sent depending on the devices that the user currently uses. A user having a cell phone may receive SMS messages while a user having a laptop may get an e-mail. Depending on the framework events these messages are sent to the devices, however context reasoning may affect this service.

Consider user A whose schedule indicates that he will be in a meeting from 11:00 AM to 1:00 PM in Room 101. At 12:00 PM a buddy_in_environment event arrives that informs the user that his buddy has entered the environment. The Context Reasoner however reasons that the user's scheduled activity indicates he is in a meeting and the user must be busy at this moment. It then asserts the fact in the knowledge base that the user is currently busy. This suppresses the message service from sending messages to the user. Rather, the message will be forwarded to his/her secretary or converted to an email message depending upon his/her profile.

SeMEther makes extensive use of such reasoning. The domain-specific inferencing requires explicit rules. RULEML[7] is an XML-based rule language and the current Semantic Web efforts[8] include building an RDF-based RuleAxiomLogic layer over the current OWL-based Ontology layer in the Semantic Web stack. Thus, we adopted Jena rules (an RDF based language). Jena Rules, written in the Jena rule syntax similar to the one described above, can be used to direct the services provided to the user. An example rule shown below indicates that if a user is in room 101 (conference room) during the meeting time then his status must be set to busy. Such composition rules can be added to guide the situation.

For pursuing advanced context reasoning, the location context manager, which is a specialized reasoner, was developed to manage the location context. As the user's location changes in an environment, this component tracks his/her spatial position. Events are generated if the user changes floors (floor change event) or rooms (room change event) etc. It reasons with the current spatial position of the user and the spatial model stored in the knowledge base in the form of a lower context ontology described above. The events generated by the location manager are used to trigger location-based services. For instance a music service makes use of the room change event to continue playing the user's music preference, switching it from his old room to the new room.

To illustrate the working of the location context manager consider the floor change event. To generate this event the location context manager performs the following query in RDQL [36], which is a query language for RDF in Jena models, to find out whether the user's old location "oldlocationuri" is within the same region as the user's new location "newlocationuri":

The location manager can verify whether the region is within the same floor or not. It then compares to see if the two rooms are on the same floor or not and generates the floor change event accordingly.

[7]http://www.ruleml.org/
[8]http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

```
[Query 1
  Select ?a where
  (<oldlocationuri / newlocationuri>,
    <http://a.com/ontology#inRegion>,
    ?a)]
```

TABLE III
SEMETHER CONTEXT QUERY EXAMPLE

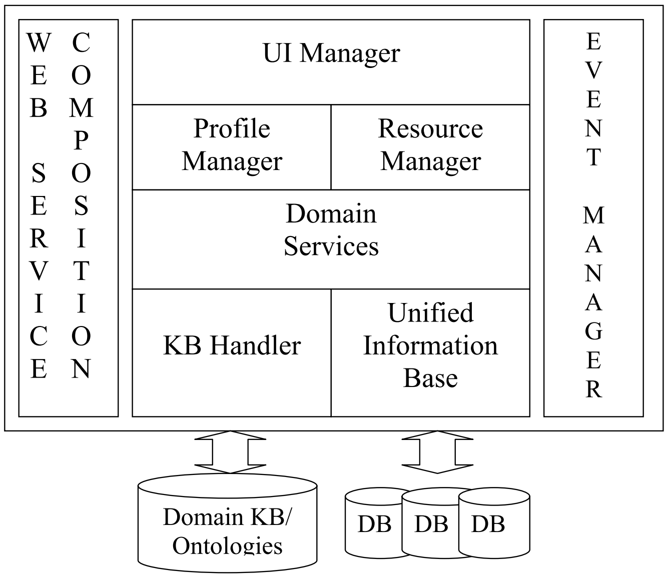


Fig. 1. The SeMEther Framework Architecture

C. SeMEther Framework: Service Composition Subsystem Architecture

As a proof of concept, the Web Service Composition Subsystem (WSCS) has been implemented on top of a Pervasive Computing framework called SeMEther [40]. The SeMEther framework provides an efficient infrastructure for the WSCS design and execution. We are in the process of developing a set of applications, tailored to meet the needs for developing Semantic Pervasive Computing Services. In this paper, we mainly highlight the WSCS. Before we demonstrate the WSCS, we briefly introduce the architecture of the SeMEther framework (Figure 1).

The Event Manager manages event-based communication between components of the framework. Components in the SeMEther framework communicate with each other by throwing and listening to events at the Event Manager. The Event Manager ensures that all services registered for an event receive that event and don't receive any duplicate events. The KB Handler maintains a Knowledge Model which reflects the current context. This context is acquired by listening to events and converting them to knowledge facts which are added or removed from the Knowledge Base (KB). The Resource Manager manages the resources available in the environment. It maintains the status of each resource and also takes care of scheduling resources for different activities. In SeMEther, a resource is anything that can be scheduled, including human actors and devices. Each resource has a semantically annotated schedule, which describes when a particular resource is available. When there is a request for a certain resource, the system will look for its availability and then schedule that resource.

The Profile Manager is responsible for fetching user profile information from the Web. We assume a centralized server, where one can request user profiles or parts of them. The SeMEther communicates with this server to get the user profile or extract specific information about the user from his/her profile.

The framework is designed to provide a dynamic user interface (UI), that is, the interface can be changed depending on the user role and the requirement of the service as well as the device used to communicate with the user. For example, the system generates different UI screens, corresponding to a desktop or a PDA, or sends an SMS over a cell phone, depending on what device the user is currently on. Thus, the framework provides pervasiveness in the sense that it uses an appropriate device to communicate with the user, depending on the context. The Domain Services provide functionalities specific to a given domain. These services use the framework to communicate with each other, and the external environment, and also to access the knowledge base of the system.

The Unified Information Base (UIB) integrates information from disparate data sources present in the environment and presents it at a more conceptual level by linking it to a local domain ontology. For example, a database field 'BP' in a hospital setting can be linked to the concept "BloodPressure" of a standard medical ontology like the UMLS. This creates an abstraction of a single homogeneous data source for other services, which need to refer to the data in terms of domain concepts. The UIB thus allows linkage to a data element and fetch or update of the same. The idea of a unified information base is an extension to our previous work [8].

The proposed system architecture for the Web Service Composition Subsystem (WSCS) is shown in Figure 2.

- **Service Editor:** A platform to model and create Semantic Pervasive Computing Services over existing legacy applications (Figure 3). The editor allows mapping of service parameters (input, output, preconditions, effects) to concepts in predefined ontologies. New ontologies can be loaded into the editor. The editor has an ontology search component which performs keyword based search in the ontologies. The user can map resultant concepts to service parameters. To facilitate faster development and ease of use, we concentrated on development of atomic services, hence making the model simpler and easier to implement. The editor parses the WSDL (Web service Definition Language) documents discovered by the Service Crawler and creates the service grounding descriptions. One interesting feature is the possibility of plugging in of new context ontologies (required for mapping service parameters). Once stored in the composition rule KB, these services are used by the Service Matcher.
- **Service Crawler:** The Service Crawler is crawling the Web for services (Web form, WSDL, or OWL-S). A multi-threaded Service Crawler crawls multiple URLs in parallel. For efficiency, the Crawler crawls the Web

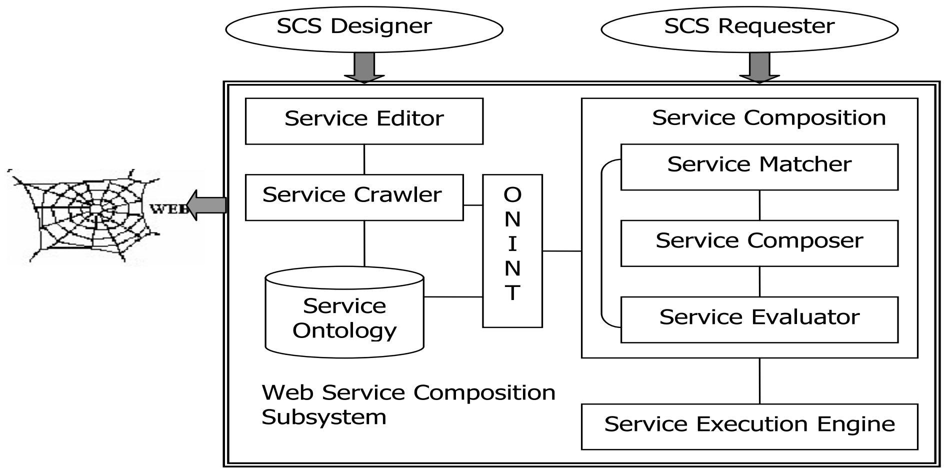


Fig. 2. The Architecture of the Web Service Composition Subsystem

using URLs taken from DMOZ[9], classifies any pages containing such service descriptions into that particular category/domain and stores them in a temporary database.

- **Service Ontology:** The Service Ontology is a repository of services either developed or discovered from the Web. Each service in the Service Ontology is semantically annotated (OWL-S) according to its respective category/domain. Thus, the services discovered by the Service Crawler are transformed into Semantic Web Services (OWL-S).
- **Service Matcher:** The Service Matcher matches a service request to existing services available in the Service Ontology. For this service matching, we could apply the Ontology Integration (OnInt) methodology to each OWL-S profile. In the profile, each service is represented by a type of service and an IOPE (Input, Output, Precondition, Effect) tuple. For two given service descriptions which are the service parameters (IOPE), the Sevice Matcher tries to match them.
- **Service Composer:** Using the Ontology integration (OnInt) methodology as described in Section IV, this module performs semantic matching of concepts. For two given concepts which are service parameters, the component tries to establish a match between them. The service composition required an iterative approach: matching of concepts followed by pragmatic evaluation.
- **Service Execution Engine:** Once the services have been discovered and composed to satisfy the goal, this module executes the services. We used the Taverna service execution tool.[10] This tool mandates the process specification in a specific format. In our case, the process specifications are generated as the result of refinement of the composition process.
- **Service Evaluator:** This component performs evaluation of a service, based on the pragmatics defined for selection of a particular service. We perform evaluation based on some simple evaluation metrics, which are similar to match algorithm described in [34]. In order to select appropriate services, we need to evaluate whether they satisfy the syntactic, semantic and pragmatic requirements of the desired composite service.

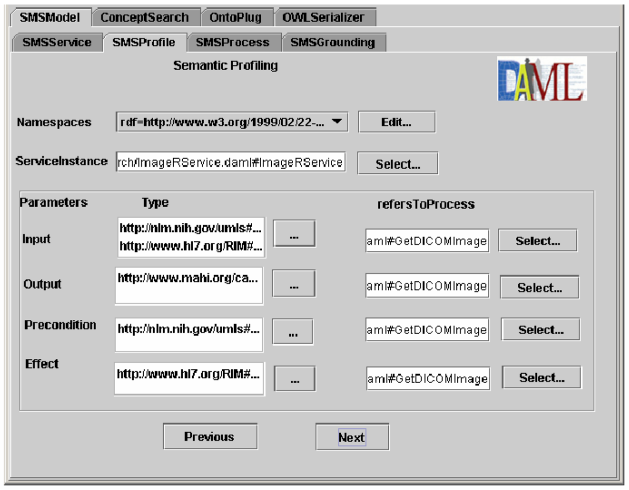
[9]http://www.dmoz.org
[10]http://taverna.sourceforge.net/

D. Implementation

We have implemented a prototype of SeMEther that demonstrates the intended goals and shows the feasibility of the proposed approach. Integrating some common computing devices such as PDAs, cell phones and personal computers, we show how the system actually functions and interacts seamlessly with the user. To bring out the effectiveness of the framework, we have implemented the Pervasive Service Compositions for several scenarios. One simple, yet powerful, service that we have implemented is the “Buddy” service. This service detects buddies of a given user, determines if they are in the vicinity, and in that case contacts them by the best possible means available. The user is detected by a Bluetooth enabled device such as a PDA or Cell Phone. Messages are delivered based on the type of device he is carrying, like a dialog box for the PDA or an SMS for the cell phone. Several other services can be run concurrently on this framework. All these services are pervasive in the sense that a user doesn’t depend on any specific device to get that service. The environment proactively detects the user, and based on his/her preferences, adapts these services and provides him/her via the best available service.

Through the implementation we have verified the viability of: (1) Generating service flow specifications in OWL-S to model context-aware services; (2) Achieving dynamic service matching and binding to the service flow; (3) Performing dynamic service composition using the compositional knowledge we specified in Section 3.

We implemented an OWL-S Editor to assist the service flow designer in modeling OWL-S specifications for the service flow, which in addition provides a graphical interface to model the abstract concepts. The Service Matching Agent (SMA) handles the task of matching the Web Services for given specifications and the service matching rules. The task execution engine was constructed as Java implementation (HP Jena Toolkit [19]) and reads the process specifications in OWL-S to execute the appropriate Web Service from a service pool associated with each task.



Semantic Profiling

Namespaces	rdf=http://www.w3.org/1999/02/22-...	
ServiceInstance	rch/ImageRService.daml#ImageRService	
Parameters	Type	refersToProcess
Input	http://nlm.nih.gov/umls#... http://www.hl7.org/RIM#...	aml#GetDICOMImage
Output	http://www.mahi.org/ca...	aml#GetDICOMImage
Precondition	http://nlm.nih.gov/umls#...	aml#GetDICOMImage
Effect	http://www.hl7.org/RIM#...	aml#GetDICOMImage

Fig. 3. The WSCS Editor

The context visualization interface provides a Touchgraph interface to visualize changes in context. As changes in the knowledge base occur due to changes in context, the Touchgraph morphs to reflect the changes. The graphl library[11] was used for the visualization. A Java based client has been developed for the visualization. The client makes an HTTP connection to the SeMEther Autonomous System (AS) head to download the latest context model. This model is in the form of an RDF document that is generated every time the service composition knowledge model changes in the knowledge base. The client can be configured to poll the server for new models at specific intervals of time. A screen shot of the context visualization tool is given in Figure 4.

VI. Related Work

Current Web services support a certain level of interoperability in using and accessing them. The next level of interoperability cannot be achieved by just making services available, but requires providing automatic mechanisms so that the services can be linked in appropriate and meaningful ways [14]. Semantic interoperability is essential for automated discovery, matching and composition of services. This enhancement depends on the existence of ontologies for the terms used by Web services. The Semantic Web research work, following the DARPA Agent Markup Language (DAML), includes DAML+OIL [17] for the creation of arbitrary domain ontologies and DAML+OIL/RDF(S) [14] for the semantic mediation between services and workflow logic. Some research has focused on the composition of services using workflow management. Automatic composition of Web services [28] has been achieved through automated mapping, composition and

[11]http://home.subnet.at/flo/mv/graphl/

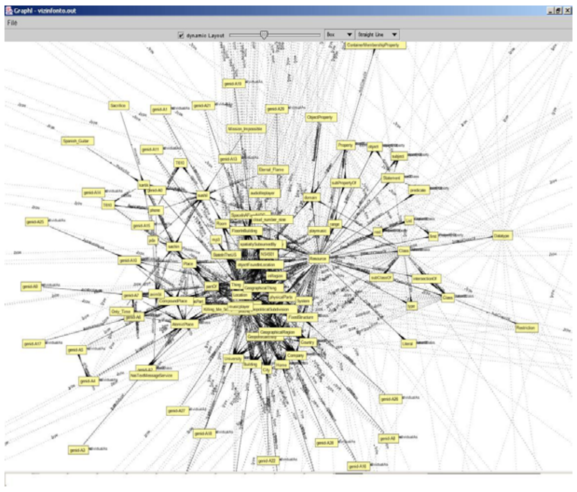


Fig. 4. Visualization of the Pragmatic Knowledge in SeMEther

interoperation of services, service verification, and execution monitoring. Process modeling languages such as PIF [29], PSL [41], and frame based models of services [12] were designed to support process management. There are other emerging relevant approaches such as indexing services based on process models [26] and reasoning and matching over service descriptions for choosing computational resources [35].

Several applications require that multiple ontologies are combined into a single coherent ontology [33]. Many lines of research have addressed ontology matching in the context of ontology construction and integration [5] and effective methodologies for automated mappings [31]. Similarity measure studies were introduced for effective ontology integration. Tversky's feature-based approach [47] is one of the most powerful similarity models, but depends on the structure of ontology features. Resnik [37] considered the extent of shared information between concepts. Lin [27] proposed an information-theoretic notion of similarity based on the joint distribution of properties. Jiang and Conrath's similarity measurement [24] is based on the conditional probability of encountering a child synonym set given a parent synonym set.

Ontologies are used for constraining the parameters of dynamic service configurations. Reasoning to ensure the semantic validity of compositions is used for automated workflows. Scientific workflow [4] is supposed to support interoperation through semantics. It may have the potential to support Web service descriptions for service discovery, invocation, activation and execution of an identified service by an agent or other service [28]. Unlike these efforts, our approach emphasizes the importance of different kinds of knowledge, especially pragmatic knowledge, and the ontological methodology for heterogeneous semantics for the automatic composition of service flows.

There have been efforts in representing business contracts for service evaluation and negotiations [15] but how to use such pragmatic knowledge for service matching remains still unresolved. We show the semantic and pragmatic representations for Pervasive Computing service flows and how the Pervasive Computing community can reap the benefits of using semantic and pragmatic rules over the Semantic Web. In other work [4], workflows in Pervasive Computing settings have been studied, but their efforts are more geared towards QoS (Quality of Service) and workflow execution aspects. We address the need to consider a broad set of pragmatic rules (including QoS) to compose a service flow of Pervasive Computing services for the Semantic Web.

Mennie et al. [32] provide an insight how to achieve dynamic service modification and up-gradation. Specifically, they describe switching or updating services without bringing down the system. A new service can be incorporated into the system by dynamic service composition. Tripathi et al. [46] define access policies for “collaboration spaces” which can also be referred to as pervasive domains. They also describe creating ubiquitous and context-aware applications from high level specifications coupled with a policy driven middleware. Their idea of a user’s ’View’ of the system, with static or dynamic binding to actual resources, governed by access policies, is highly relevant to our approach.

Amann et al. [1] focused on knowledge management in distributed environments, adaptive decision support and assistance with dissemination of relevant information and knowledge among geographically dispersed user groups. The key technical contribution is the integration of the extended tuple space concept to adapt, co-ordinate and control a set of ordered events as well as applications and devices in mobile settings. Henricksen et al. [18] introduced appropriate context modeling issues for Pervasive Computing, such as wide variations in information quality, the existence of complex relationships amongst context information, and temporal aspects of context. They provide a very good understanding and a solid model for modeling context; however they utilize a traditional database to store context information and relationships, while we think an ontology is a better structure to model context.

In Strang et al. [45] Con-text Ontology Language (CoOL) is derived from the model, which may be used to enable context-awareness and contextual interoperability during service discovery and execution in a proposed distributed system architecture. Specifically, Indulska et al. [21] present a location management system able to gather process and manage location information from a variety of physical and virtual location sensors. Their approach scales to the complexity of context-aware applications, to a variety of types and a large number of location sensors and clients, and to a geographical size of the environment. The objective of CoBrA [7] is to provide a centralized model of context that can be shared by all devices, services, and agents in the space. They acquire contextual information from sources that are unreachable by the resource-limited devices. They also reason about contextual information that cannot be directly acquired from the sensors (e.g., intentions, roles, temporal and spatial relations). The main idea in [6] is that of a central context broker which manages the context knowledge base using a context reasoning engine. This is analogous to our idea of a KBHandler. As the centralized KB approach discussed here becomes a bottleneck, this paper proposes a distributed knowledge base. For instance, the building agent maintains knowledge about a building and these agents then exchange/share knowledge. Our approach differs from CoBrA in that we propose a completely service-based architecture, in contrast to their agent-based one.

VII. Conclusions and Future Work

In this paper, we have laid out an architecture of the knowledge processing that is necessary for composing services automatically, using different kinds of knowledge. We showed heterogeneous Pervasive Computing services interoperate in a Pervasive Computing service flow, composed by using syntactic, semantic and pragmatic knowledge. We defined, distinguished between, and justified the need for these three different kinds of knowledge to be used in service descriptions. Finally, we demonstrated principles of ontology integration to enable the automatic composition of Web services into a service flow. We have developed a prototype of a Pervasive Computing service flow as a proof-of-concept. This prototype allows the routing of information to a user with the most appropriate device for a given context.

Future work includes the extension of compositional knowledge to include negotiation rules. When certain services in the process of service selection do not exactly meet the conditions of a rule, then there should be a possibility to relax the conditions to continue with the selection and integration process. This is best modeled as a form of inter-agent negotiation. We are also planning to work on a user service request model and representation that support a wide range of different services.

References

[1] P. Amann, D. Bright, G. Quirchmayr, B. Thomas: Supporting Knowledge Management in Context-Aware and Pervasive Environments Using Event-Based Co-ordination. DEXA Workshops 2003: 929-935

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, 284(5), pp. 34-43, May 2001

[3] Brown, P.J. The Stick-e Document: a Framework for Creating Context-Aware Applications. Electronic Publishing 96 (1996) 259-27.

[4] J. Cardoso and A. Sheth, Semantic e-Workflow Composition, Journal of Intelligent Information Systems, 2003. (In press)

[5] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufman, San Francisco, CA, 2000.

[6] H. Chen et al., ”An Ontology for Context-Aware Pervasive Computing Environments”, Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, November 2003

[7] H. Chen, T. Finin and A. Joshi, Semantic Web in the Context Broker Architecture, IEEE Conference on Pervasive Computing and Communications (PerCom), Orlando, March, 2004

[8] Q. Chong, A. Marwadi, K. Supekar and Y. Lee, Ontology Based Metadata Management in Medical Domains, Journal of Research and Practice in Information Technology (JRPIT), 2003, 35(2), pp. 139 - 154.

[9] S. A. Chun, V. Atluri and N. R. Adam, Domain Knowledge-based Automatic Workflow Generation, in the proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA 2002), September 2-6, 2002, Aix en Provence, France.

[10] Dey, A.K. and Abowd, G.D.: Towards a better understanding of Context and Context-Awareness. GVU Technical Report GITGVU-99-22, College of Computing, Georgia Institute of Technology. 2 (1999) 2 14

[11] D.F. D’Souza and A.C. Wills, Objects, Components, and Frameworks with UML: the Catalysis Approach, Addison Wesley, 1999.

[12] M. G. Fugini and S. Faustle. Retrieval of reusable components in a development information system. In Proceedings of Second International Workshop on Software Reusability, IEEE, 1993.

[13] J. Geller, H. Gu, Y. Perl and M. Halper, Semantic refinement and error correction in large terminological knowledge bases, Data & Knowledge Engineering, 45(1), 2003, pp. 1-32.

[14] C. A. Goble. Supporting Web-based biology with ontologies. In The Third IEEE ITAB00, pages 384-390, Arlington, VA, November 2001.
[15] B. N. Grosof, T. C. Poon, SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions, In Proc of WWW 2003, May 20-24, 2003, Budapest, Hungary.
[16] H. Gu, Y. Perl, J. Geller, M. Halper, L. Liu and J.J. Cimino, ”Representing the UMLS as an OODB: Modeling Issues and Advantages.” Journal of the American Medical Informatics Association, 7(1), January 2000, pp.66-80.
[17] J. Hendler and D. L. McGuinness. DARPA Agent Markup Language. IEEE Intelligent Systems, 15(6):72-73, 2001.
[18] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In Proceedings of the First International Conference on Pervasive Computing, volume 2414 of Lecture Notes in Computer Science, pages 167-180, Zurich, August 2002. Springer-Verlag.
[19] HP Labs Jena 2 Toolkit, http://www.hpl.hp.com/semweb/index.html
[20] C.L. Hwang and K. Yoon, Multiple Attribute Decision Making, Springer-Verlag, Berlin, 1981.
[21] J. Indulska, T. McFadden, M. Kind, and K. Henricksen. Scalable location management for context-aware systems. In Proceedings of the 4th International Conference on Distributed Applications and Interoperable Systems, DAIS 2003, volume 2893 of Lecture Notes in Computer Science, pages 224-235, Paris, France, November 19-21 2003.
[22] Internet Encyclopedia of Philosophy Website http://www.iep.utm.edu/s/solipsis.htm
[23] F. Jelinek. 1990. Self-organized Language Modeling for Speech Recognition. In Readings in Speech Recognition. Edited by Waibel and Lee. Morgan Kaufmann Publishers.
[24] J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In Proceedings of International Conference on Research in Computational Linguistics, 1997.
[25] A. K. Joshi, B. J. Weber, and I. A. Sag. Elements of Discourse Understanding. Cambridge University Press, Cambridge, 1981.
[26] M. Klein and A. Bernstein. Searching for services on the semantic Web using process ontologies. In Proceedings of the International Semantic Web Working Symposium (SWWS), July 2001.
[27] D. Lin. An information-theoretic definition of similarity. In Proceedings of the 15th International Conference on Machine Learning, 1998.
[28] S. McIlraith, T. Son, and H. Zeng. Semantic Web services. IEEE Intelligent Systems (Special Issue on the Semantic Web), 16(2):46-53, 2001.
[29] J. Lee, G. Yost and the PIF Working Group, The PIF Process Interchange Format and Framework, 1994, http://ccs.mit.edu/pifmain.html
[30] Y. Lee, C. Patel, S. Chun, J. Geller, Towards Intelligent Web Services for Automating Medical Services Composition, in Proceedings of 2004 IEEE International Conference on Web Services (ICWS 2004) July 6-9, 2004, San Diego, California. pp. 384 391.
[31] Maedche. A machine learning perspective for the Semantic Web. In Proceedings of Semantic Web Working Symposium (SWWS), 2001.
[32] David Mennie, Bernard Pagurek, An Architecture to Support Dynamic Composition of Service Components, Proceeding of the WCOP2000, 2000.
[33] N. Noy and M. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2000.
[34] Massimo Paolucci, Katia Sycara, and Takahiro Kawamura, ”Delivering Semantic Web Services.” In Proceedings of the Twelves World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003, pp 111- 118 .
[35] R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. Cluster Computing: The Journal of Networks, Software Tools and Applications, 2:129-138, 1999.
[36] Query language for RDF Website: http://www.hpl.hp.com/semweb/doc/tutorial/RDQL/index.html
[37] P. Resnik. Selection and Information: A Class based Approach to Lexical Relationships. PhD thesis, University of Pennsylvania, 1993.
[38] M. Satyanarayanan, ”Pervasive computing: Vision and challenges,” IEEE Personal Communications, vol. 8, pp. 10–17, Aug. 2001
[39] D. Saha, A. Mukherjee, ”Pervasive Computing: A Paradigm for the 21st Century”, IEEE Computer, vol. 36, pp. 25-31, March 2003
[40] S. Singh, S. Puradkar, Y. Lee, Ubiquitous Computing: Connecting Pervasive Computing through Semantic Web, ISEB Journal (Accepted)
[41] Schlenoff et al. The essence of the process specification language. Transactions of the Society for Computer Simulation, 16(4):204-216, 1999. http://ats.nist.gov/psl/
[42] S. A. Chun, Y. Lee, J. Geller , Ontological and Pragmatic Knowledge Management for Web service Composition, 9th International Conference on Database Systems for Advanced Applications (DASFAA 2004) Lecture Notes in Computer Science 2973 Springer 2004, pp. 365-373.
[43] K. Supekar, The OnInt (Ontology Integration) Implementation Report, UMKC, Technical Report, 2002.
[44] W. W. Stead, R. A. Miller, M. A. Musen, and W. R. Hersh, Integration and Beyond: Linking Information from Disparate Sources and into Workflow, Am Med Inform Assoc 2000;7(2):135-145
[45] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A Context Ontology Language to enable Contextual Interoperability. LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003), pages 236-247, 2003.
[46] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka, Context-Based Secure Resource Access in Pervasive Computing Environments In 1st IEEE International Workshop on Pervasive Computing and Communications Security(IEEE PerSec’04), To Appear.
[47] Tversky. Features of similarity. Psychological Review, 84:327-352, 1977.
[48] D. Waltz, Understanding Line Drawings with Shadows, P. Winston, The Psychology of Computer Vision, McGraw Hill, New York, 1975, pp. 19-91.
[49] M. Weiser, The Computer for the 21st Century, Scientific American., 1991, pp 94-104; reprinted in IEEE Pervasive Computing, Jan-Mar 2003, pp. 19-25
[50] X. Wang, D. Zhang, T. Gu, H. Fung, Ontology Based Context Modeling and Reasoning using OWL, Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004