Semantic web technologies to reconcile privacy and context awareness

Fabien Gandon, Norman Sadeh

Semantic Web Technologies to Reconcile Privacy and Context Awareness

Fabien L. Gandon and Norman M. Sadeh

School of Computer Science - Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891, USA
{Fabien.Gandon, Norman.Sadeh}@cs.cmu.edu
http://www-2.cs.cmu.edu/~sadeh/

Abstract. Increasingly, application developers are looking for ways to provide users with higher levels of personalization that capture different elements of a user's operating context, such as her location, the task that she is currently engaged in, who her colleagues are, *etc.* While there are many sources of contextual information, they tend to vary from one user to another and also over time. Different users may rely on different location tracking functionality provided by different cell phone operators; they may use different calendar systems, *etc.* In this article, we describe work on a Semantic e-Wallet aimed at supporting automated identification and access of personal resources, each represented as a Semantic Web Service. A key objective is to provide a Semantic Web environment for open access to a user's contextual resources, thereby reducing the costs associated with the development and maintenance of context-aware applications. A second objective is, through Semantic Web technologies, to empower users to selectively control who has access to their contextual information and under which conditions. This work has been carried out in the context of *my*-Campus, a context-aware environment aimed at enhancing everyday campus life. Empirical results obtained on Carnegie Mellon's campus are discussed.

1 Introduction

Increasingly, application developers are looking for ways to provide users with added levels of convenience and ease of use through functionality that is capable of capturing the context within which they operate. This may involve knowing where the user is located, the task she is currently engaged in, her eating preferences, who her colleagues are as well as a variety of other contextual attributes. While there are many sources of contextual information, they tend to vary from one user to another and also over time. Different users may rely on different location tracking functionality provided by different cell phone operators; they may use different calendar systems, *etc.* Traditionally, context-aware applications and services have been hardwired to predefined sources of contextual information (e.g. relying on a particular set of sensors and protocols to track a user's locations). As a result, they remain prohibitively expensive to build and maintain and are few and between. We argue that what is needed is a more open environment, where context-aware applications can automatically discover and access a user's personal resources such as her calendar or location tracking functionality. This can be done by viewing each source of contextual information (or personal resource) as a Web service. Unfortunately, current Web Services standards such

as UDDI [15] or WSDL [25] are not sufficient when it comes to describing a user's personal resources and to enabling automated access to them by context-aware applications. Another challenge, as we move towards more open platforms for access to a user's personal information, revolves around privacy issues. Users should be able to retain control over who has access to their personal information under different conditions. For instance, I may be willing to let my colleagues see where I am or access my calendar activities between 8am and 5pm on weekdays but not over the weekend. In addition, I may want to fine tune the granularity of the answer provided to a given query, depending on the context of that query. For instance, I may be willing to disclose the room that I am in to some people but only the city where I am to others In fact, I may even want to give different answers to different people, telling my secretary I am off to see my dentist, while telling my customers I am busy in a meeting.

In this paper, we introduce a Semantic Web architecture aimed at supporting the automated discovery and access of personal resources in support of a variety of context-aware applications. Within this architecture, each source of contextual information (e.g. a calendar, location tracking functionality, collections of relevant user preferences, organizational databases) is represented as a Semantic Web service. A central element of our architecture is its *semantic e-Wallet*, which acts as a directory of contextual resources for a given user, while enforcing her *privacy preferences*. Privacy preferences enable users to specify what information can be provided to whom in different contexts. They also allow users to specify what we call *obfuscation rules*, namely rules that control the accuracy or inaccuracy of the information provided in response to different queries under different conditions.

We have validated our architecture in the context of *my*Campus, a context-aware environment aimed at enhancing everyday campus life at Carnegie Mellon University (CMU). The environment revolves around a growing collection of task-specific agents capable of automatically accessing a variety of contextual information about their users (e.g. context-aware restaurant concierge, context-aware message filtering agent, *etc.*). This includes accessing their locations, calendar activities as well as a variety of other attributes and preferences. Students access the environment from PDAs over the campus's 802.11 wireless LAN. Empirical results obtained with a group of students over a period of several days are briefly summarized at the end of this article. While, in this paper, we focus on scenarios involving individual users, it should be noted that our architecture extends to scenarios where users are entire organizations. In this context, both organizations and individual users could each have one or more Semantic e-Wallets capable of leveraging a variety of individual or organizational knowledge subject to a rich set of privacy/confidentiality constraints.

The remainder of this article is organized as follows. Section 2 provides a brief overview of the state of the art in context-awareness, privacy, Web Services and the Semantic Web, emphasizing limitations of the work reported so far in the literature. In Section 3, we provide an overview of our Semantic Web environment for context-awareness and privacy. Section 4 focuses more specifically on the Semantic e-Wallet and includes a high-level scenario outlining its operation in response to a query about the current location of a user. Section 5 introduces the three layers of knowledge required to support the e-Wallet functionality. Section 6 discusses the e-Wallet's current implementation, which is based on OWL Lite [24], XSLT transformations [27], and

JESS [10]. Sections 7 and 8 provide further details on the e-Wallet's three layers. Section 9 discusses the interfaces to the e-Wallet and section 10 briefly describes some of our experiments carried out in our *my*Campus environment. Finally, section 11 summarizes what we view as the main contributions of our work along with some concluding remarks.

2 Prior Work

Prior efforts to develop context aware applications are many. Early work in context awareness includes the Active Badge System developed at Olivetti Research Lab to redirect phone calls based on people's locations [23]. The ParcTab system developed at the Xerox Palo Alto Research Center in the early nineties relied on PDAs to support a variety of context-aware office applications (e.g. locating nearby resources such as printers, posting electronic notes in a room, *etc*.) [20, 21]. Other relevant applications that have emerged over the years range from location-aware tour guides to context-aware memory aids. More recent research efforts in context awareness include MIT's Oxygen [6], CMU's Aura [11] and several projects at Berkeley's GUIR (e.g. [14]) to name just a few.

While early context-aware applications relied on *ad hoc* architectures and representations, it was quickly recognized that separating the process of acquiring contextual information from actual context-aware applications was key to facilitating application development and maintenance. Georgia Tech's Context Toolkit represents the most significant effort in this direction [7, 8]. In the Context Toolkit, widgets act as wrappers that provide access to different sets of contextual information (e.g. user location, identity, time and activity), while insulating applications from context acquisition concerns. Each user (as well as other relevant entities such as physical objects or locations) has a context server that contains all the widgets relevant to it. This is similar to our notion of e-Wallet, which serves as a directory of all personal resources relevant to a given user (e.g. relevant location tracking functionality, relevant collections of preferences, access to one or more calendar systems, *etc*.). Our Semantic e-Wallet however goes one step beyond Dey's Context Toolkit. It makes it possible to leverage much richer models of personal resources - what personal information they give access to, when to access one rather than the other, how to go about accessing these resources. In addition, it includes access control and obfuscation functionality to enforce user privacy preferences. This richer model is key to supporting automated discovery and access of a user's personal resources by agents. In other words, while the Context Toolkit focuses mainly on facilitating the development of context-aware applications through off-line, re-use and integration of context-aware components (i.e. widgets), our architecture emphasizes real-time, on-the-fly queries of personal resources by context-aware agents. These queries are processed through several layers of functionality that support automated discovery and access of relevant personal resources subject to user-specified privacy preferences.

The notion of e-Wallet as introduced in systems such as Microsoft's .NET Passport is not new. However current implementations have been limited to storing a very small amount of information and offer very restricted control to the user when it comes to specifying what information can be made available to different services. For instance,

in Passport, users can specify whether or not they are willing to share parts of their profiles with all participating sites but cannot distinguish between different participating sites. Our notion of Semantic e-Wallet lifts these restrictions and allows users to control access to any of their personal resources. It also allows for multiple sources of similar information (e.g. multiple calendars or multiple location tracking functionality) and for functionality that can dynamically select which of these resources to tap based on the context and the nature of the query at hand (e.g. using your car's GPS system when you are driving and your cell phone operator's location tracking functionality when you are not).

Our notion of semantic e-Wallet extends recent efforts to develop rich languages for capturing user privacy preferences such as P3P's APPEL language [26]. It does so by making it possible to leverage any number of domain ontologies and by allowing for preferences that relate to any number of contextual attributes. In addition, it allows users to specify obfuscation rules through which they control the level of accuracy (or inaccuracy) at which their contextual information is disclosed to different parties under different conditions. This includes telling some people which room you are in, while simply telling others whether you are at work or not, or whether you are in town or not. It also includes scenarios where you might want to pretend you are in one place, while you are really elsewhere.

Last but not least, while the security community has developed powerful languages to capture access control privileges such as the Security Assertion Markup Language (SAML) [16], the XML Access Control Markup Language (XACML) [17] and the Enterprise Privacy Authorization Language (EPAL) [22], these languages do not take advantage of Semantic Web concepts. Our work builds directly on recent efforts aimed at moving the Web from an environment where information is primarily made available for human consumption to one where it is annotated with semantic markup that makes it understandable to software applications. These efforts are part of a long-term vision generally referred to as the *Semantic Web* [1, 13]. They have already resulted in a succession of semantic markup languages [4, 24] as well as early efforts to define Web Service ontologies and markup in the context of languages such as DAML-S [5]. In our work, we have relied on the use of DAML+OIL [4] and more recently OWL [24] to represent contextual information (e.g. location, calendar activities, social and organizational relationships, *etc*.) and privacy preferences and on Semantic Web service concepts to support the automated discovery and access of personal and public resources.

3 Overall System Architecture

We consider an environment where, over time, users purchase (or subscribe to) different sets of task-specific agents. These agents are each intended to assist them in the context of different activities (e.g. scheduling meetings with colleagues, reminding them of purchases they need to make, arranging trips or filtering incoming messages). To function, each agent needs to access some information about its user as well as possibly other users. Access to a user's personal (or contextual) information is controlled by that user's e-Wallet subject to privacy (enforcing) rules. The e-Wallet Manager (or simply e-Wallet) serves as a repository of static knowledge about the user –

just like .NET Passport, except that here knowledge is represented using OWL. In addition, the e-Wallet contains knowledge about how to access more information about the user by invoking a variety of resources, each represented as a Web Service. This knowledge is stored in the form of rules that map different contextual attributes onto one or more possible service invocations, enabling the e-Wallet to automatically identify and activate the most relevant resources in response to queries about the user's context (e.g. accessing the user's calendar to find out about her availability, or consulting one or more location tracking applications in an attempt to find out about her current location). User-specified privacy rules, also stored in the e-Wallet, ensure that information about the user is only disclosed to authorized parties, taking into account the context of the query. They further adjust the accuracy or inaccuracy of the information provided in accordance with the user's obfuscation preferences.

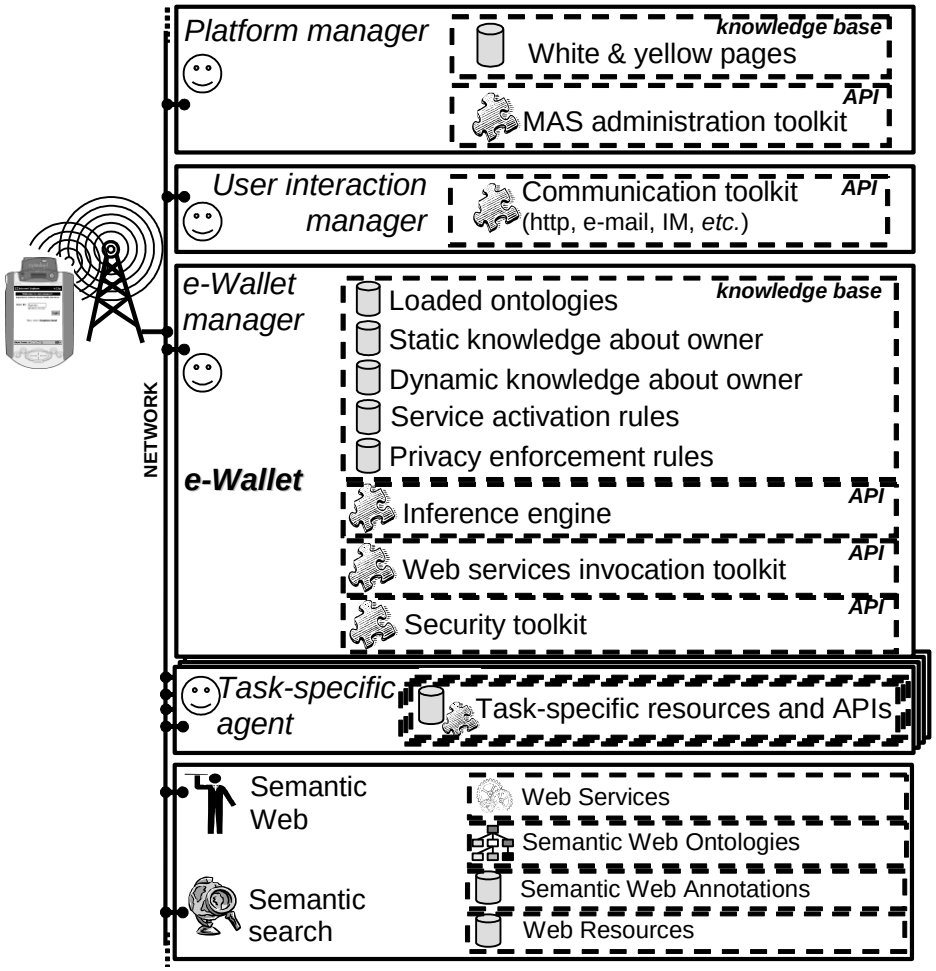


Fig. 1. *my*Campus architecture: a user's perspective - the smiley faces represent agents.

Figure 1 provides an overview of our Semantic Web environment. It illustrates a situation where access is from a PDA over a wireless network, as is the case in *my*Campus, the environment in which we have instantiated our architecture. However, our architecture extends to fixed Internet scenarios and more generally to environments where users can connect to the infrastructure through a number of access channels and devices – information about the particular access device and channel can actually be treated as part of the user's context and be made available through her e-Wallet. As can be seen in Figure 1, other key elements of our architecture include:

- One or more *Platform Managers* that build on top of Directory Facilitators and Agent Management Systems, as defined in FIPA [9]. They manage the agents running at their sites, and maintain white and yellow page directories of these agents and the services they provide.
- *User Interaction Managers* that are responsible for interactions with the user. This includes managing login sessions as well as interactions with the user's agents and her e-Wallet. Because different users interact with different sets of agents, this also includes the dynamic generation of interfaces for interacting with these agents and the customization of these interfaces to the current interaction context (e.g. particular access device). Communication with the User Interaction Manager typically takes place through a number of APIs, e.g. an Instant Messaging API, an HTTP/HTML API, *etc.*

Clearly, agents are not limited to accessing information about users in the environment. Instead, they also typically access public Web Services, Semantic Web annotations, public ontologies and other public resources. On CMU's campus, where we have deployed *my*Campus, this includes access to a variety of services such as 23 restaurant web services or a pubic weather forecasting web service.

In the following sections, we focus on the e-Wallet functionality. Additional details on *my*Campus and some of the agents we have deployed can be found in [19].

4 A Semantic E-Wallet

The e-Wallet is a central element of our Semantic Web architecture for context-awareness and privacy. It provides a unified and secure semantic interface to all the user's personal resources, enabling agents in the system, whether working for the owner of the e-Wallet or for other users, to access and, when appropriate, modify information about the user subject to that user's privacy preferences (e.g. not just determining whether the user is available between 3 and 4pm but also, possibly, scheduling a meeting at that time). The e-Wallet is *not* a static information repository. While it does contain some static information about the user, it is an agent acting as clearinghouse and gatekeeper for a user's personal resources. Its knowledge about the user, her personal resources and preferences falls into four categories:

1. *Static knowledge*. This context-independent knowledge typically includes the user's name, her email address, employer, home address as well as context-independent preferences (e.g. "I like spicy vegetarian cuisine"). This knowledge, like all other in the e-Wallet, can be edited by the user via the User Interaction Manager.

2. *Dynamic knowledge*. This is context-sensitive knowledge about the user, often involving a variety of preferences such as "When driving, I don't want to receive instant messages".
3. *Service invocation rules*. These rules help leverage information resources external to the e-Wallet – both personal and public. They effectively turn the e-Wallet into a semantic directory of personal resources that can be automatically discovered and accessed to process incoming queries. Specifically, service invocation rules provide a mapping between contextual attributes and personal resources available to access these attributes, viewing each personal resource as a Semantic Web service. An example of one such mapping is a rule indicating that a query about the user's current activity can be answered by accessing her Microsoft Outlook calendar. We have developed Web Service wrappers for a variety of personal resources such as Microsoft Outlook Calendar or location tracking functionality. Service invocation rules are not limited to providing a one-to-one mapping between contextual attributes and personal resources. Instead, they can leverage rich ontologies of personal resources, enabling the e-Wallet to select among a number of possible personal resources based on availability, accuracy and other relevant considerations. For instance, in response to a query about the user's location, the rules can specify that, when the user is driving, the best method available is the GPS in her car. If she is at work and her wireless-enabled PDA is on, her location can be obtained using location tracking functionality running over the enterprise's wireless LAN. If everything else fails, her calendar might have some information about her location. Finally, it should be noted that to answer queries about the user, additional mapping rules that support automated discovery and access of public services may also be needed. For instance, a query like "Tell me whether Fabien is in a sunny place right now" will typically require accessing Fabien's location as well as a public weather service.
4. *Privacy preferences*. These preferences encapsulate knowledge about what information about herself the user is willing to disclose to others under different conditions. These preferences themselves fall into two categories:
   - *Access control rules.* These rules simply express who has the right to see what information under different conditions e.g. "My location should only be visible to members of my team during week days between 8am and 5pm".
   - *Obfuscation rules.* Often user privacy preferences are not black-and-white but rather involve different levels of accuracy or inaccuracy: *Obfuscation by abstraction* is about abstracting away some details about the user's current context such as telling people whether or not you are in town without giving your exact location. *Obfuscation by falsification* is about scenarios where the user may not want to appear as if she is withholding information but would rather provide false information. For instance, a user may not want to reveal her true email address to a web service for fear of getting spammed.

All the above knowledge (including rules) is represented in OWL. It can leverage a number of relevant ontologies (e.g. ontologies about contextual attributes, personal resources, as well as more specific knowledge such as cuisine types and food preferences or message types and message filtering preferences).

Before delving deeper into the details of the e-Wallet, a scenario will help illustrate the key steps it goes through in processing incoming queries (Figure 2). For the sake of argument, we will assume a query submitted by a user (Norman) to the e-Wallet of a second user (Fabien) inquiring about that second user's current location. The main steps are as follows:

1. *Asserting the query's context*: As a first step, facts about the context of the query are asserted – namely they are loaded into the e-Wallet's inference engine for possible use as part of inferences to be made in processing the query. In our example, one such assertion is that "the sender of the query is Norman".
2. *Asserting elementary information needs and the need to go through an authorization process*: Here the query is translated into an aggregate goal that includes (a) a combination of elementary information needs – in our example the need to find "Fabien's location", along with (b) a requirement to go through an authorization process. The authorization process, which is distributed across some of the following steps, results in the request being either denied or cleared, the latter possibly following the application of obfuscation rules. In our example, the authorization goal requires checking that Norman is entitled to having access to Fabien's location and that the level of resolution at which the query is answered is compatible with Fabien's privacy preferences.
3. *Pre-checking whether the query is allowable*: A first check is performed to see whether the query is allowable based on access rights considerations. In our example, the e-Wallet checks whether Norman is allowed to inquire about Fabien's location. Fabien's e-Wallet might include a privacy preference specifying that his colleagues at work can see the building that he is in, when he is on campus, but that no one else should be given access to his location. In this first check, the e-Wallet might be able to determine that Norman is indeed a colleague of Fabien's – e.g. based on organizational knowledge stored in the static knowledge base of Fabien's e-Wallet. At this stage, because it has not yet determined whether Fabien is on campus or not, the e-Wallet has no ground for denying the request. Therefore, it continues processing it, as detailed below.
4. *Checking the e-Wallet's local knowledge base*: Some queries can be answered in whole or in part, using facts in the e-Wallet's local knowledge base, which, as we have seen in Section 3, contains both static (namely, context-independent) and dynamic (namely, context-sensitive) knowledge about the user. In our particular example, such knowledge is not particularly helpful and the e-Wallet needs to turn to outside sources of personal information to answer the query (see next step).
5. *Invoking personal resources as Web services*: When local knowledge is not sufficient to answer a query, the e-Wallet turns to its service invocation rules to identify external resources that might help answer it. This may involve accessing one or more of the user's personal resources such as his calendar and/or one or more trusted public services. In our example, the campus where Fabien works has a wireless LAN that supports location tracking. This functionality can be invoked by the e-Wallet to obtain Fabien's location. The actual invocation takes place through the web service invocation toolkit already introduced in Figure 1.
6. *Post-checking whether the query is allowable*: armed with additional knowledge obtained by invoking one or more external resources, the e-Wallet is now in a bet-

ter position to check whether the query is allowable. In our example, colleagues of Fabien's are only allowed to see his location when he is on campus. Assuming that Fabien is on campus, the request is now deemed allowable. This does not mean however that the authorization process required as part of the goals set in step 2 has been fully completed. Obfuscation rules may still need to be applied.

7. *Application of Obfuscation Rules*: suppose that the location tracking functionality used to answer our query about Fabien's location returned the specific room he is in, while Fabien is only willing to disclose the buildings that he is in. This latter requirement is captured by the e-Wallet in the form of an obfuscation rule that returns the building in which Fabien is rather than the exact room. Application of this rule will typically involve accessing ontologies about rooms and buildings as well as annotations about the campus where Fabien works.
8. The query has now been fully processed and an acceptable answer generated. This answer (e.g. "Fabien is in Smith Hall") can be returned to Norman.

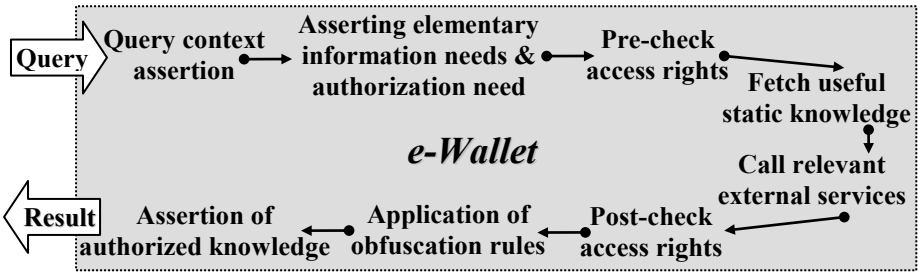


Fig. 2. Main steps involved in processing a query submitted to an e-Wallet.

5 A Three-Layer e-Wallet Implementation

As shown in Figure 3, we developed a three-layer implementation of our e-Wallet:

- *Core Layer*: At the most basic level, the e-Wallet's knowledge includes an OWL meta-model – required to interpret OWL statements. In addition, it maintains both static (context-independent) and dynamic (context-dependent) knowledge about the user. This knowledge is obtained by loading available annotations about the user along with relevant ontologies and is currently completed using forward-chaining reasoning – to avoid having to infer the same facts over and over again. Knowledge in this layer is represented using a (core) triple template:

$$(\text{predicate, subject, object})_{\text{TRIPLE}}$$

- The *Service Layer* completes the e-Wallet's core knowledge with invocation rules that map information retrieval goals about contextual attributes onto external service invocations. These are modeled as backward-chaining rules. Given an information retrieval goal such as "Give me Fabien's location", they help identify and invoke one or more relevant information resources, each modeled as a Web service, as already discussed in Section 4. Knowledge in this layer is represented using a special type of triple called "service triple" denoted:

$$(\texttt{predicate, subject, object})_{\text{SERVICE TRIPLE}}$$

Service triples reside in the service layer and are created in either of two ways. They can result from the migration of a triple from the core layer or from the activation of an invocation rule (e.g. an assertion about Fabien's location as returned by a call to a location tracking service). Migration between the core layer and the service layer is implemented by rules specifying that any (core) triple can be used to generate an equivalent service triple.

- The outer layer is referred to as the *Privacy Layer*, as this is where privacy (enforcing) rules are applied. Assertions in this layer are represented as another special type of triple called "authorized triple":

$$(\texttt{predicate, subject, object})_{\text{AUTHORIZED TRIPLE}}$$

Only authorized triples can be sent in response to queries. Authorized triples are generated by applying privacy enforcing rules to service triples, thereby ensuring that information about the user is only disclosed to authorized parties and in accordance with relevant obfuscation rules.

Privacy enforcing rules are encoded as backward-chaining rules. These rules map needs for authorized triples onto needs for service triples to be post-processed subject to the privacy enforcing rules. Upon receiving an incoming query, the e-Wallet generates a need for one or more authorized triples. This need in turn typically triggers needs for service triples and core triples, eventually resulting either (a) in the generation of authorized triples that can be returned in response to the query or (b) in an exception, if the query is found unallowable (e.g. an unauthorized party requesting your location or trying to schedule a meeting in your calendar). In summary, security in our architecture is directly enforced through typing.

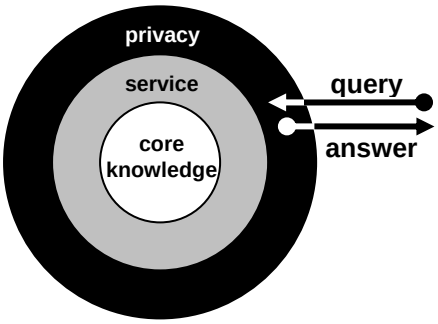


Fig. 3. e-Wallet 3-layer implementation

6 Additional Implementation Considerations

The current implementation of our e-Wallet is based on JESS, a high-performance Java-based rule engine that supports both forward and backward chaining – the latter by reifying "needs for facts" as facts themselves, which in turn trigger forward-chaining rules. The e-Wallet's knowledge base is initialized with: (a) a model of RDF [28] triples as a template for unordered facts, (b) a model of specialized triples used in our three layers (core triples, service triples and authorized triples) along with associated migration rules between the layers, and (c) an OWL meta-model.

Additional knowledge is loaded into the e-Wallet by translating OWL input files into JESS assertions and rules, using a set of XSLT stylesheets [27] (figure 4). The OWL input files include ontologies and annotations that are transformed into (core) triple assertions, forward-chaining rules (used to complete knowledge at the core layer) as well as service invocation rules and privacy enforcing rules – both

represented as backward-chaining rules. The XSLT templates act as meta-rules that generate the body, the head and typing used by the JESS rules (e.g. query transformation stylesheet in figure 5).

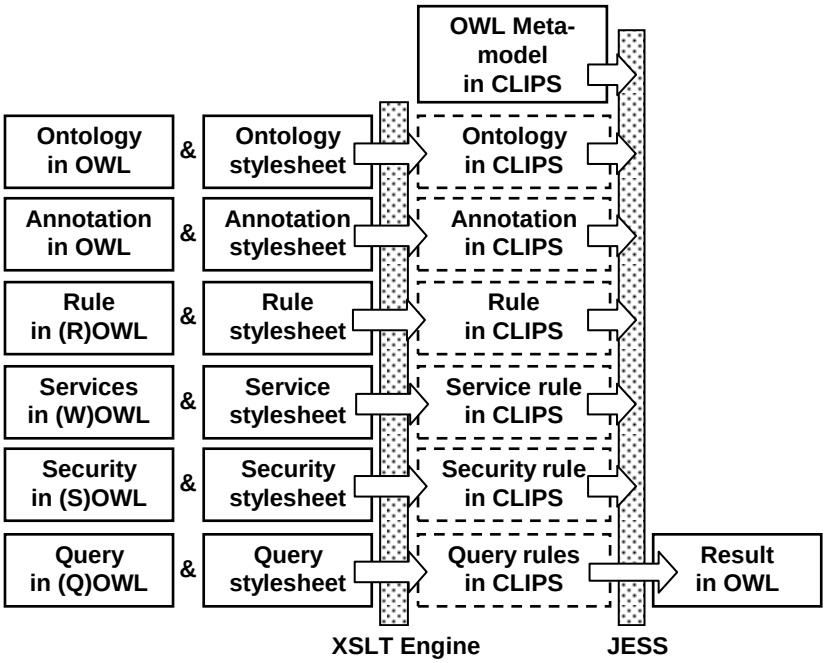


Fig. 4. High-level flows and processes in the e-Wallet

```
(…)
<xsl:template match="/rdf:RDF">
 (defrule query (declare (salience 0))
  <xsl:for-each select="*[not(self::qowl:Query)]">
   <xsl:call-template name="process-class-instance"/>
  </xsl:for-each>
 =>
 (store-result<xsl:call-template name="variable-list"/>)
)
</xsl:template>

<xsl:template name="process-class-instance" >
(authorized_triple
 (predicate "&rdf;#type")
 (subject   <xsl:call-template name="local-ID">
             <xsl:with-param name="id">
              <xsl:value-of select="@rdf:about"/>
             </xsl:with-param>
            </xsl:call-template>)
 (object    "<xsl:value-of select="concat(namespace-uri(.),local-name(.))"/>")
)
 <xsl:for-each select="*">
  <xsl:call-template name="process-property-instance"/>
 </xsl:for-each>
</xsl:template>

<xsl:template name="process-property-instance" >
  <xsl:choose>
    <xsl:when test='count(*)=1'> <!-- has an element for child -->
      <xsl:call-template name="process-objectproperty-instance"/>
    </xsl:when>
    <xsl:when test='count(text())=1'> <!-- has an text for child -->
      <xsl:call-template name="process-dataproperty-instance"/>
    </xsl:when>
    <xsl:when test='@rdf:resource'> <!-- has a reference -->
      <xsl:call-template name="process-referenceproperty-instance"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>
(…)
```

Fig. 5. Fragment of the query transformation stylesheet

```
<qowl:Query rdf:ID="">
  <qowl:sender rdf:resource="http://cs.cmu.edu/~nsadeh"/>
</qowl:Query>
<mc:Person rdf:about="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://sadehlab.cs.cmu.edu/Variable#location" />
</mc:Person>
```

Fig. 6. Query issued by the user 'nsadeh' requesting the location of user 'fgandon'

Once all this knowledge has been loaded and the forward-chaining rules have been applied to complete the core knowledge base, the e-Wallet is ready to process incoming queries. A query is transformed into the need for an authorized triple. This in turn triggers privacy enforcing rules and generates needs for service triples. The service triples are generated by either migrating core triples or activating service invocation rules or a combination of both. This is further detailed below:

1. Queries have two components (see Figure 6): (a) an annotation about the query providing its context (e.g., who the sender of the query is), and (b) the query itself in the form of a pattern using a special namespace to identify variables. The context of a query is asserted for the time it takes to process it – later, clean up rules take care of removing all assertions created while processing it. We assume that security protocols (e.g., using digital signatures) are used to verify assertions about the query's context (e.g. verifying the identity of the sender). The query itself is transformed into a set of authorized triples in the privacy layer. These authorized triples form the body of a backward-chaining rule, whose head is a function that stores the results each time the rule is triggered and that generates OWL results in XML syntax ("pretty printing").
2. The need of the query for authorized triples triggers privacy enforcing rules. As illustrated in Figure 8, these rules have two roles. First, they check that the sender of the query has the required access rights. In addition, they also apply obfuscation rules to triples to ensure that the level of accuracy or inaccuracy provided in answers to queries is compatible with the user's privacy preferences. The need for authorized triples in combination with privacy enforcing rules generates a need for service triples.
3. The need for service triples in turn triggers service rules. First a generic service rule is applied that checks whether the needed service triple is not already available as a core triple. If this is the case an equivalent service triple is simply created. If there is no equivalent core triple, the e-Wallet looks for matching rules that trigger internal function calls (e.g. getting the current time and date). If that fails too, it looks for matching (external) Web Services. To support this, we have extended the Jess library with internal functions (e.g. time) and functions to call external services. An example of a service invocation rule is given in Figure 9.

Figure 7 uses pseudo-code to depict the e-Wallet's overall processing flow.

```
//Load CLIPS model of RDF/S and OWL                                INITIALIZATION
{∀ triple(Ti) | Ti ∈ OWLmodel } assert (Ti) in JESS
{∀ rule(Ri) | Ri ∈ OWLmodel } defrule (Ri) in JESS
//Load ontologies
{∀ OOWLi | Ontology (OOWLi)}
  OCLIPSi:=Ontology stylesheet (OOWLi) // extract ontology triples
  {∀ triple(Ti) | Ti ∈ OCLIPSi } assert(Ti) in JESS
//Load annotations
{∀ AOWLi | Annotation (AOWLi)}
  ACLIPSi:=Annotation stylesheet (AOWLi) // extract annotation triples
  {∀ triple(Ti) | Ti ∈ ACLIPSi } assert(Ti) in JESS
//Load rules
{∀ RROWLi | Rule (RROWLi)}
  RCLIPSi:=Rule stylesheet (RROWLi) //makes forward rules producing triples
  {∀ rule(Ri) | Ri ∈ RCLIPSi } defrule (Ri) in JESS
//Load service rules
{∀ SWOWLi | service description (SWOWLi)}
  SCLIPSi:=Service stylesheet (SWOWLi)
                    //makes backward rules producing dynamic triples
  {∀ rule(Ri) | Ri ∈ SCLIPSi } defrule (Ri) in JESS
//Load privacy rules
{∀ PROWLi | Security rule (PROWLi)}
  PCLIPSi:=Privacy stylesheet (PPOWLi)
                  // makes backward rules producing authorized triples
  {∀ rule(Ri) | Ri ∈ PCLIPSi } defrule (Ri) in JESS
-------------------------------------------------------------------------⇩
                                                          QUERY PROCESSING
//Load query
CCLIPSi:=Query stylesheet (AOWLi) // extract context triples e.g.: sender
{∀ triple(Ti) | Ti ∈ CCLIPSi } assert(Ti) in JESS
QCLIPSi := Query stylesheet (QPOWLi)
                // makes one backward rule requiring authorized triples
defrule (QCLIPSi) in JESS
(Run the RETE Algorithm)
```

Fig. 7. e-Wallet's overall processing flow.

The system has been designed with efficiency in mind. For instance, query processing stops as soon as an authorization violation is detected. Also, at the service layer, rules ensure that the system first checks for available core triples before attempting to invoke external resources.

We currently use RDF-S/OWL to represent rules. In comparison to RuleML [2], we do not reify the role of the relation and its arguments. We simply use triples to represent rules and take advantage of the typing mechanism of the XML syntax. We use a special namespace to identify variables. While we are following ongoing developments in RuleML [2] and OWL Rules, our current focus is on the use of rules that apply to OWL assertions. Later we could easily extend our system, for instance using XSLT stylesheets to translate between our representation of rules in OWL and RuleML representations.

As shown in Figure 8, *privacy enforcing rules* are defined using three tags: the content of the *target tag* describes the piece of knowledge to which this rule applies; the content of the *check tag* describes the conditions under which read access is granted; the content of the *revision tag* describes the obfuscation to be applied before migrat-

ing triples to the authorized layer. Note that, at the time of writing, our e-Wallet also supports limited write access rules.

As shown in Figure 9 the *service rules* have three child tags: the content of the *output tag* describes the piece of knowledge that this rule can produce; the content of the *precondition tag* describes the knowledge needed for calling the service; the content of the *call tag* describes the function to trigger and its parameters. For reference, the CLIPS representation of this rule, following the application of our XSLT transformation, is also provided in Figure 10.

```
<sowl:ReadAccessRule>
  <rdfs:label>people can only know whether or not I am on campus</rdfs:label>
  <sowl:target>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="&variable;#location"/>
    </mc:Person>
  </sowl:target>
  <sowl:check>
   <rowl:And>
     <rowl:condition>
       <mc:E-Wallet rdf:about="&variable;#e-Wallet">
         <mc:owner>
          <mc:Person rdf:about="&variable;#owner"/>
         </mc:owner>
       </mc:E-Wallet>
     </rowl:condition>
     <rowl:condition>
       <mc:Place rdf:about="http://www.cmu.edu">
          <mc:include rdf:resource="&variable;#location" />
       </mc:Place>
     </rowl:condition>
     <rowl:not-condition>
       <qowl:Query rdf:about="&variable;#query">
         <qowl:sender rdf:resource="&variable;#owner" />
       </qowl:Query>
     </rowl:not-condition>
   </rowl:And>
  </sowl:check>
  <sowl:revision>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="http://www.cmu.edu"/>
    </mc:Person>
  </sowl:revision>
</sowl:ReadAccessRule>
```

Fig. 8. Privacy rule obfuscating the location of the owner

```
<wowl:ServiceRule wowl:salience="50">
  <rdfs:label>provide activity status for a person</rdfs:label>
  <wowl:output>
    <mc:Person rdf:ID="&variable;#person">
      <mc:has_activity rdf:resource="&variable;#activity" />
    </mc:Person>
  </wowl:output>
  <wowl:precondition>
    <mc:Person rdf:ID="&variable;#owner">
      <mc:PDA_endpoint>&variable;#endpoint</mc:PDA_endpoint>
    </mc:Person>
  </wowl:precondition>
  <wowl:call>
   <wowl:Service wowl:name="call-web-service">
     <wowl:qname>http://mycampus/PDAService#</wowl:qname>
     <wowl:endpoint>&variable;#endpoint</wowl:endpoint>
     <wowl:method>GetCurrentWeekAppointments</wowl:method>
     <wowl:user_id>&variable;#owner</wowl:user_id>
   </wowl:Service>
  </wowl:call>
</wowl:ServiceRule>
```

Fig. 9. Service rule for activity-tracking invocation in WOWL

```
(defrule provide activity status for a person (declare (salience 50))
  (need-dynamic_triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#has_activity")
   (subject   ?person)
   (object    ?activity)
  )
  (dynamic_triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?owner)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
  (dynamic_triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#PDA_endpoint")
   (subject   ?owner)
   (object    ?endpoint)
  )
  =>
    (call-web-service qname "http://mycampus/PDAService#"
                      endpoint ?endpoint
                      method "GetCurrentWeekAppointments"
                      user_id ?owner)
)
```

Fig. 10. Service rule for activity-tracking invocation translated in CLIPS

7 Static Knowledge and Domain-Specific Rules

As indicated earlier, the RDF triple meta-model is defined as a template used in forward chaining rules. The OWL meta-model is asserted as a list of unordered facts such as the one shown in Figure 11. The semantics attached to properties is translated into rules as illustrated in Figure 12.

```
(triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   "http://www.w3.org/2002/07/owl#equivalentProperty")
   (object    "http://www.w3.org/2002/07/owl#SymmetricProperty")
)
```

Fig. 11. Declare property equivalence as a symmetric property

```
(defrule equivalent-property (declare (salience 100))
   (triple
     (predicate "http://www.w3.org/2002/07/owl#equivalentProperty")
     (subject   ?p1)
     (object    ?p2))
   (triple (predicate p1?) (subject ?s) (object ?o))
   (not (triple (predicate p2?) (subject ?s) (object ?o)))
   =>
   (assert (triple (predicate p2?) (subject ?s) (object ?o)))
)
```

Fig. 12. Rule for forward-chaining completion of property equivalence

As far as the OWL meta-model is concerned, we are focusing on those aspects of OWL-Lite relevant to our application scenarios. More precisely the current system handles: `Resource`, `Class`, `Property`, `type`, `subClassOf`, `subPropertyOf`, `ObjectProperty`, `TransitiveProperty`, `SymmetricProperty`, `inverseProperty`, `equivalentProperty`, `equivalentClass`, `sameIndividualAs`, `DatatypeProperty`, `FunctionalProperty`, `InverseFunctionalProperty`. – the source code and the results obtained by running our OWL engine on the official OWL test cases are available at http://mycampus.sadehlab.cs.cmu.edu/public_pages/OWLEngine.html.

Likewise, triples in the ontologies and annotations loaded into the e-Wallet are asserted as unordered facts. Finally, domain-dependent rules are also loaded in the e-wallet. An example of one such rule is illustrated in Figure 13. It defines colleagues as members of the same team. Such rules can help represent and interpret context-sensitive preferences such as 'My colleagues can see my location when I am at work'. The inference engine is used to complete the base applying all the rules, thus saving time during the query solving process and providing a rollback point if needed.

OWL

```
<rowl:Rule direction="forward">
  <rdfs:label>Members of the same group means colleagues</rdfs:label>
  <rowl:head>
    <mc:Person rdf:about="&variable;#person1">
      <mc:colleague rdf:resource="&variable;#person2"/>
    </mc:Person>
  </rowl:head>
  <rowl:body>
    <mc:Team rdf:ID="&variable;#group">
      <mc:include><mc:Person rdf:about="&variable;#person1"/></mc:include>
      <mc:include><mc:Person rdf:about="&variable;#person2"/></mc:include>
    </mc:Team>
  </rowl:body>
</rowl:Rule>
```

XSLT

CLIPS

```
(defrule Members of the same group means colleagues
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?group)
   (object    "http://mycampus.cs.cmu.edu/ontology#Team")
  )
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#include")
   (subject   ?group)
   (object    ?person1)
  )
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person1)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#include")
   (subject   ?group)
   (object    ?person2)
  )
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person2)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
   =>
 (assert
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person1)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
 )
 (assert
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#colleague")
   (subject   ?person1)
   (object    ?person2)
  )
 )
)
```

Fig. 13. Rule defining colleagues as members of the same team

8 Service Layer Processing

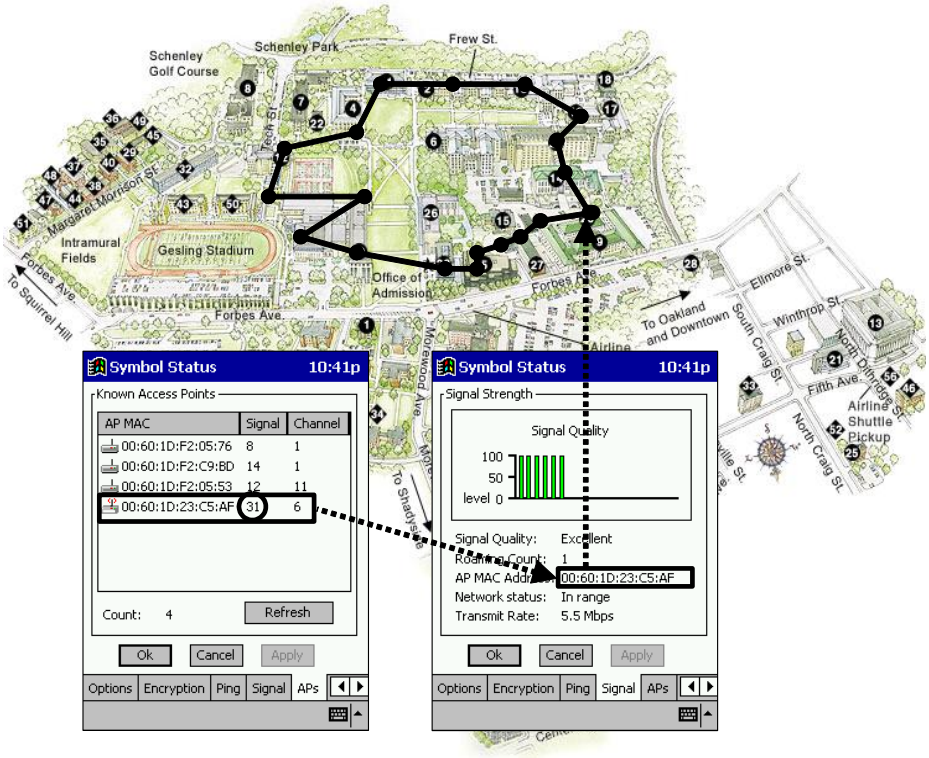
As indicated earlier, needs for service triples can be satisfied by either migrating a matching core triple or by activating a matching service invocation rule. For obvious

efficiency reasons, it makes sense to always look for core triples first. This can be enforced by assigning a high priority (also called salience) to rules that look for matching core triples and lower priority to service invocation rules. Service invocation rules are themselves given different priorities, based on the nature of the resource they invoke. This is further detailed below:

- If the needed service triple can be obtained by invoking an internal function (e.g. getting the current time), that function will be activated;
- If no internal service can provide the triple (or if internal calls have failed) but there is a personal service that can possibly provide the needed triple (e.g., obtaining the user's current activity from her personal calendar), the corresponding backward invocation rule is fired, calling that personal resource's Web Service wrapper;
- If no personal resource can provide the needed triple or if the calls failed, the engine looks for invocation rules involving public web services. This can include invoking public Semantic Web search engines (e.g. CORESE [3] or distributed search architecture such as TAP [12]) or public matchmaking services such as the one in [18] – this step is not currently implemented.
- If everything fails, the query is considered to have failed.

In summary, the body of each rule requires a need for a particular piece of information or triple (e.g. Fabien's location) along with the availability of a specific set of arguments (e.g. knowledge of the IP address of Fabien's PDA). When these conditions are matched, the rule fires and calls the service (Figure 14 depicts the semantic web service used to support location-tracking over CMU's wireless LAN).

When looking for particular piece of information, rule salience helps determine the order in which to try and invoke available services (e.g. if multiple sources of location information are available). In general, we envision having a set of rules, where, should everything else fail, the e-Wallet reverts to a low salience rule that invokes one or more semantic search engines and/or one or more public matchmaking services. Clearly, as users acquire new personal resources (e.g. a new calendar), they will have to register them with their e-Wallets (e.g. using predefined service profiles that are provided with the resource itself).



Annotation of access points on campus:

```
<mc:AccessPoint>
  <mc:MAC>00:60:1D:23:C5:AF</mc:MAC>
  <mc:location rdf:resource="http://www.cmu.edu/SmithHall" />
</mc:Entity>
```

Result of invoking the location-tracking web service:

```
<mc:Entity rdf:ID="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://www.cmu.edu/SmithHall" />
</mc:Entity>
```

Fig. 14. Semantic web service for location-tracking over CMU's wireless LAN.

9 Capturing User Preferences

As should be clear by now, our Semantic Web technologies are capable of capturing a wide variety of user preferences that may refer to any relevant set of OWL ontologies. This is true for message filtering preferences, food preferences, music preferences, privacy preferences, scheduling preferences, *etc.* One approach to capturing these preferences is to develop a variety of special-purpose editing tools that enable users to specify their preferences with regard to predefined sets of ontologies. For instance, each time a user subscribes to (or acquires) a new task-specific agent, she might be prompted by a special-purpose editor to customize a predefined set of preferences. The same could be done to capture predefined sets of privacy preferences. However, a key objective in our architecture has been to provide for an open environment, where new sources of contextual information, new contextual ontologies and new agents can be introduced over time. Supporting the capture of user privacy preferences in this broader context ideally requires a general-purpose privacy preference editor that enables users to refer to any relevant source of contextual information and any relevant contextual ontology. Figure 15 shows screenshots of such a general-purpose privacy preference editor. The editor uses XSLT stylesheets and allows users to browse (Figure 15-a) and edit their privacy rules (Figure 15 - b and c).

The editor allows users to create new rules as well as edit and delete existing ones. The editor draws directly on available ontologies (ontologies loaded into the e-Wallet), enabling users to express any privacy/confidentiality rules they want as they relate to concepts and properties defined in these ontologies. The editor takes into account the OWL meta-model as the user edits rules. For instance, it will restrict the instantiation of a given concept to be within the range of a given property, as specified using the OWL "ObjectProperty" construct [24].

Every single editing operation is specified through an external XSLT stylesheet. The stylesheets are independent of the domain ontologies and could be refined to support more specific instantiations of our rule editor. In addition, rule editing is supported through the definition of high-level functions, namely "creating", "deleting", "extracting", "updating" a rule, or "adding/deleting concepts", "adding/deleting properties", *etc.* These high-level functions are instantiated at run time, using XSLT stylesheets that perform the actual manipulation. In other words, the editor could easily be adapted to accommodate extensions to our rule syntax. As can also be seen, use of this general purpose privacy preference editor, in its current form, is best left to system administrators and advanced users.

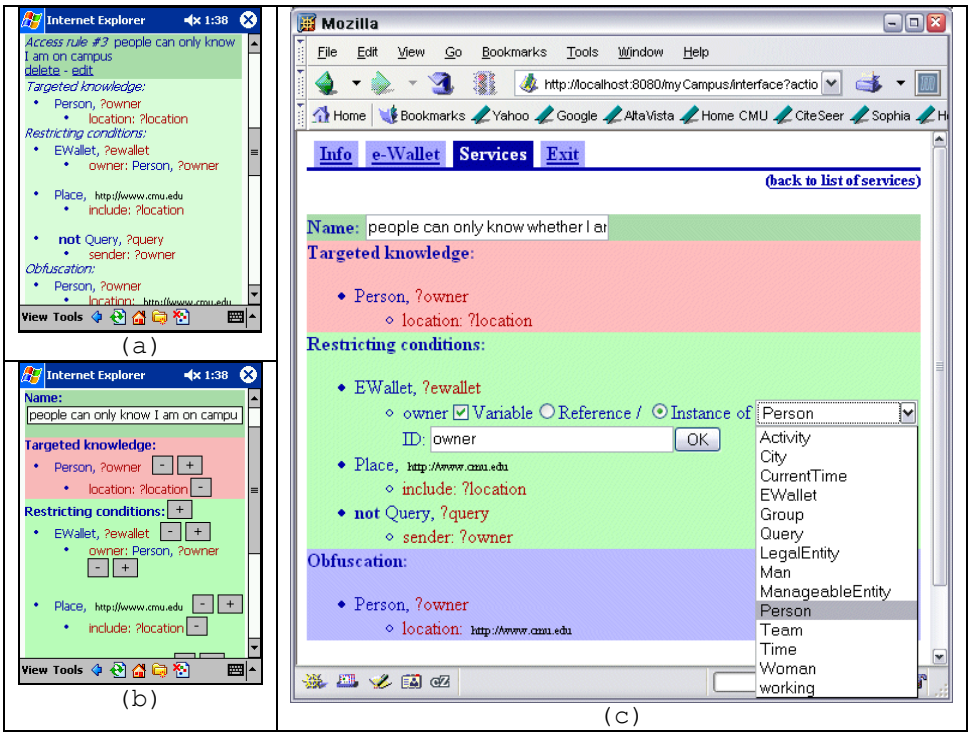


Fig. 15 Generic rule editor that enables users to (a) browse and (b) (c) edit their OWL-based privacy/confidentiality preferences.

10 Empirical Evaluation

An early version of our architecture has been validated in *my*Campus, a context-aware environment aimed at enhancing everyday campus life at CMU. The environment is accessible to members of the campus community from their PDAs over the university's wireless LAN. An example of a *my*Campus agent we have developed is a "restaurant concierge" that gives users suggestions on where to have lunch, depending on their food preferences, their location on campus and the weather. For instance, when it rains, the concierge might look for places that do not require walking outside – depending on how the user sets her preferences. Another task-specific agent that has proved particularly popular among students is a context-aware message filtering agent. The agent filters incoming alerts, taking into account a profile of topics a user is interested in as well as contextual attributes such as the user's current activities (e.g. "When in class, only show me emergency alerts" or "When I am busy, delay showing me interesting messages until my current activity is over"). Screenshots of both agents are shown in Figure 16.

Evaluation of the system by 11 users over a period of 3 days has indicated positive overall user acceptance. Among other things, the experiments required users to configure their individual preferences and use several context-aware agents, including a context-aware restaurant concierge agent and a context-aware message filtering agent. The message filtering agent was used to process a total of 44 messages for each user and the restaurant concierge was systematically used by students to decide where to eat, selecting from a total of 23 web services created for restaurants on or near campus. The context-aware functionality embedded in the agents used in the experiments proved rather successful with context awareness systematically improving performance over the use of static user profiles. For instance, detailed feedback from users indicated that over 70 percent of the 484 messages processed by the filtering agents benefited from the use of contextual information. In other words, the action taken by the message filtering agent based on contextual information was always at least as good as that taken based on static user profiles and was actually better in 70 percent of the cases.

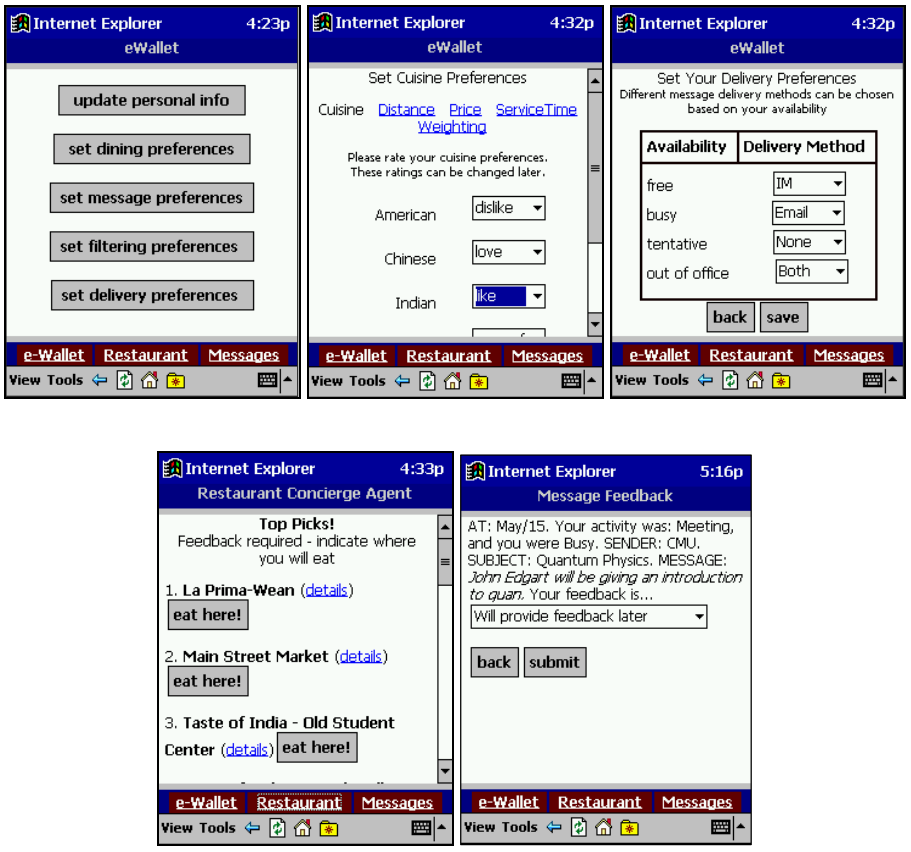


Fig. 16. Screenshots of the e-Wallet (top), restaurant recommendation from the Restaurant Concierge Agent (bottom left) and request for feedback from the Message Filtering Agent (bottom right)

Figure 17 displays additional screenshots of the *my*Campus environment, illustrating different sets of pervasive computing scenarios. The Directory Facilitator (DF) provides users with a list of available task-specific agents. The map agent is a user locator that displays the location of a user on a map, subject to that user's privacy preferences. Map (1) corresponds to a request where the user is willing to disclose the particular zip code she is in, while map (2) corresponds to a query where she is only willing to disclose her location at the level of the city she is in. Other similar agents include a location-sensitive movie recommendation agent and a location-sensitive weather forecast agent. A slide show agent enables users to access slides that other users have agreed to share with them subject to preferences specified in their e-Wallets and to display these slides on a nearby projector.

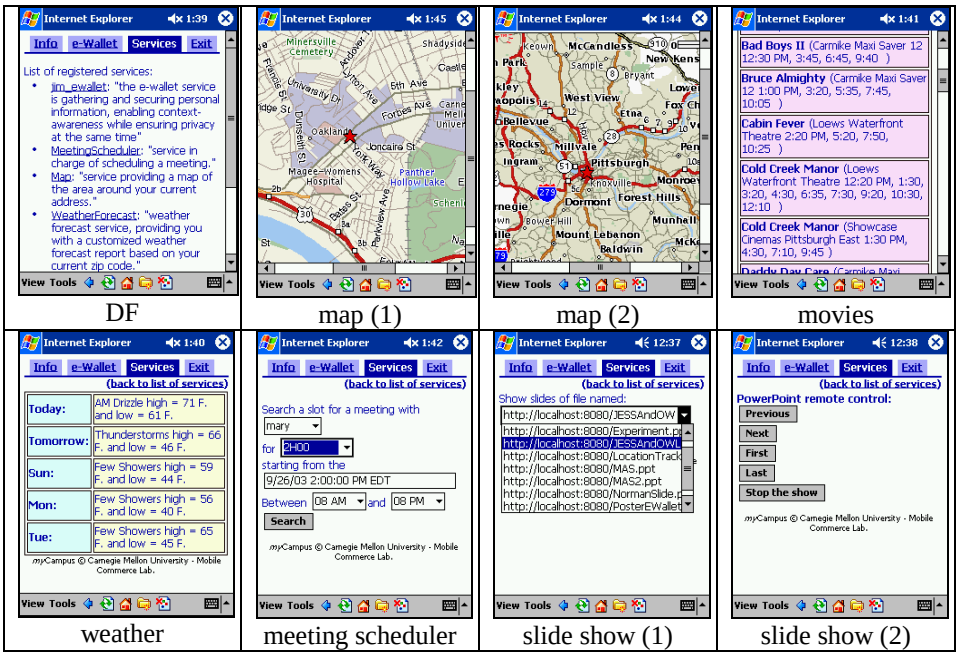


Fig. 17. – *my*Campus: Additional screenshots showcasing additional agents and pervasive computing scenarios.

11 Summary and Concluding Remarks

In this paper, we have presented a Semantic Web architecture for context-awareness and privacy. A key element of our architecture is its e-Wallet, which supports the automated discovery and access of a user's personal resources subject to user-specified privacy preferences. Personal and public resources are represented as web services. Service invocation rules along with service ontologies and service profiles enable the e-Wallet to dynamically identify the most promising resources available to answer a query. When one resource is unavailable, service invocation rules can help identify the next most relevant resource (e.g. using a calendar resource instead of a location tracking resource to estimate the user's current location). Public matchmaking services and semantic search engine functionality can also be leveraged through low salience rules that amount to reverting to these services when everything else has failed. Another innovation introduced in our e-Wallet is its support for a rich set of privacy preferences, including obfuscation rules that enable users to selectively adjust the accuracy or inaccuracy of responses they provide depending on the context of each query. We have described a three-layer implementation of our e-Wallet using JESS, OWL-Lite and XSLT stylesheets. A query to the e-Wallet successively results in the creation of needs for authorization triples and service triples. The latter can be satisfied through the identification of matching core triples and/or the activation of service invocation rules.

Experiments with *my*Campus indicate that different students are interested in different task-specific agents and that, to be effective, many agents require access to a great variety of contextual resources. Our experiments also confirmed that users are concerned about protecting access to their personal information. The need for leveraging a variety of contextual attributes and the students' demand for privacy strongly argue for Semantic e-Wallets such as the one presented here. A key challenge however remains to reconcile the power of these Semantic environments with all important usability requirements that demand systems which are flexible, yet easy to configure. We are experimenting with different approaches to editing and learning user profiles, which we hope will help alleviate this problem.

Acknowledgments. This material is based on research conducted as part of the DAML initiative and has been sponsored by the Air Force Research Laboratory under contract F30602-02-2-0035 and, earlier, by the Defense Advanced Research Project Agency under contract F30602-98-2-0135. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. This work was also in part supported by grants from IBM, HP, Symbol, Boeing, Amazon and the IST Program (SWAP project).

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Scientific American, May (2001)
2. Boley, H., Grosof, B., Tabet, S., Wagner, G.: RuleML DTDs, The Rule Markup Initiative RuleML, http://www.dfki.uni-kl.de/ruleml/indtd0.8.html
3. Corby, O., Faron-Zucker, C., Corese: A Corporate Semantic Web Engine, Workshop on Real World RDF and Semantic Web Applications, WWW Conference, Hawaii, (2002)
4. DAML Joint Committee: DAML+OIL language, 27 March 2001, http://www.daml.org/2001/03/daml+oil-index.html
5. DAML Services Coalition: DAML-S: Web Service Description for the Semantic Web, First International Semantic Web Conference, ISWC'02, Sardinia, Italy, LNCS 2342, (2002) 348-363
6. Dertouzos, M.: The Future of Computing, Scientific American, August (1999)
7. Dey, A.K., Abowd, G.D.: Toward a Better Understanding of Context and Context-Awareness, GVU Technical Report GIT-GVU-99-22. College of Computing, Georgia Institute of Technology, (1999)
8. Dey, A., Salber, D., Futakawa, M., Abowd, G.: An Architecture to Support Context Aware Computing, Technical Report GIT-GVU-99-23. College Computing, Georgia Institute of Technology, Nov. (2000)
9. FIPA, Specifications (2002) http://www.fipa.org/repository/fipa2000.html
10. Friedman-Hill, E.: Jess in Action: Java Rule-based Systems, Manning Publications Com-pany, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/
11. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste P.: Project Aura: Towards Distraction-Free Pervasive Computing, IEEE Pervasive Computing, Special Issue on Integrated Perva-sive Computing Environments, Vol. 1, Number 2, April-June (2002) 22-31
12. Guha, R., McCool, R., Miller E.: Semantic Search, Proceedings of WWW Conference, Budapest pp. 700-709 (2003)
13. Hendler, J.: Agents on the Web, IEEE Intelligent Systems, Special Issue on the Semantic Web, Vol. 16, No. 2, pp. 30-37, March/April, (2001)
14. Hong, J., Llanday, J.: A Context/Communication Information Agent, in Personal and Ubiquitous Computing Special Issue Situated Interaction and Context-Aware Computing, Vol. 5(1) (2001) 78-81
15. OASIS: Universal Description, Discovery and Integration standard, http://www.uddi.org/
16. OASIS: Security Assertion Markup Language (SAML), Technology Reports, April 14 (2003) http://xml.coverpages.org/saml.html
17. OASIS: Extensible Access Control Markup Language (XACML), Technology Reports, March 28 (2003) http://xml.coverpages.org/xacml.html
18. Paolucci, M., Kawamura, T., Payne, T., and Sycara, K.: Semantic Matching of Web Service Capabilities, Proceedings of the 1st International Semantic Web Conference, Eds. I. Horrocks and J. Hendler, LCNS 2342, Springer Verlag, 2002.

19. Norman M. Sadeh, Ting-Chak Chan, Linh Van, OhByung Kwon and Kazuaki Takizawa. "Creating an Open Agent Environment for Context-aware M-Commerce", in "Agentcities: Challenges in Open Agent Environments", Ed. by Burg, Dale, Finin, Nakashima, Padgham, Sierra, and Willmott, LNAI, Springer Verlag, pp.152-158, 2003
20. Schilit, W.: A System Architecture for Context-Aware Mobile Computing, Ph.D. Thesis, Columbia University, 1995.
21. Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Proc. of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, (1994) 85-90
22. Schunter, M., Powers, C., The Enterprise Privacy Authorization Language (EPAL 1.1), IBM Research Laboratory, http://www.zurich.ibm.com/security/enterprise-privacy/epal/
23. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System, ACM Transactions on Information Systems 10(1) (1992) 91-102.
24. W3C: OWL Web Ontology Language Reference, Working Draft 31 March 2003, http://www.w3.org/TR/owl-ref/
25. W3C: Web Services Description Language (WSDL) 1.1, Note 15 March 2001 http://www.w3.org/TR/wsdl
26. W3C: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, Recommendation 16 April 2002, http://www.w3.org/TR/P3P/
27. W3C: XSL Transformations (XSLT) Version 1.0, Recommendation 16 November 1999, http://www.w3.org/TR/xslt
28. W3C: RDF Vocabulary Description Language 1.0: RDF Schema, Working Draft 23 January (2003) http://www.w3.org/TR/rdf-schema/