WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D1.2

Personal Knowledge Management Technologies

Dissemination level	Public
Contractual date of delivery	Month 18, 30-09-09
Actual date of delivery	13-10-09
Workpackage	WP1, Personal Intelligence
Task	T1.1, T1.2
Type	Report
Approval Status	Approved
Version	0.8
Number of pages	34
Filename	D1.2_2009-10-13_v08_USFD

Abstract:

This deliverable describes the technology developed to support facilitating personal knowledge management.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.0	01/09/2009	TOC and Outline	USFD
0.1	20/09/2009	Adding the introduction	USFD
0.2	21/09/2009	Incorporating the remaining sections	USFD
0.3	22/09/2009	Added work on the interaction model	USFD
0.4	24/09/2009	Finished first integration	USFD
0.5	26/09/2009	Internal Review	TID
0.6	26/09/2009	Addressing the Reviewer comments	USFD
0.7	05/10/2009	Restructuring on basis of internal review	USFD
0.8	13/10/2009	Address internal reviewer comments	USFD

Author list

Organization	Name	Contact Information
USFD	Simon Tucker	s.tucker@dcs.shef.ac.uk
USFD	Grégoire Burel	g.burel@dcs.shef.ac.uk
USFD	Elizabeth Cano	a.cano@dcs.shef.ac.uk
USFD	Alfonso Sosa	a.sosa@dcs.shef.ac.uk
TID	Manuel Escriche	mev@tid.es
USFD	Neil Ireson	n.ireson@dcs.shef.ac.uk

Timeplan before delivery

Next Action	Deadline	Care of
Circulate outline	01/09/2009	USFD
Integrate contributions	20/09/2009	USFD
Complete version ready for internal review	24/09/2009	USFD
Internal Review	26/09/2009	TID
Make changes suggested by internal review	29/09/2009	USFD
Restructuring on the basis of internal review	05/10/2009	USFD
Completing restructured version	10/10/2009	USFD
Submit final restructured version	13/10/2009	USFD

Executive Summary

This deliverable describes the technologies and methodologies that have been implemented by WeKnowIt within T1.2 and following on from T1.1. The deliverable reports how technologies implemented by WeKnowIt meet the multi-user, multi-modal and multi-visual requirements of WeKnowIt applications.

WeKnowIt is exploring Collective Intelligence as a construct of multiple layers of intelligence cooperating as a single whole. Personal Intelligence is one such layer and is primarily concerned with allowing individual users to provide knowledge and access intelligence using WeKnowIt applications.

The previous deliverable, D1.1 (Report on Interaction Model), outlined the requirements of technologies to support the management of Personal Intelligence. The Personal Intelligence layer should support users when uploading knowledge from a variety of devices with a range of capabilities. Thus users should be able to upload knowledge as effectively from a mobile phone as they would be able to from a more powerful desktop machine. Thus WeKnowIt applications should orient themselves towards the modalities of the devices by which they are being accessed.

In addition to the modality requirement there is also a need to account for the context of the user – for example, the needs of a user when accessing WeKnowIt in an emergency situation are vastly different from the same user accessing WeKnowIt whilst on holiday.

This deliverable describes how the technologies were developed to meet these requirements. It defines an Interaction Model which allows WeKnowIt to record and track the upload of information in the form of multimedia resources (such as images and videos) as well as annotations on those resources (such as comments and tags) and a system for defining users to allow them to be connected with such resources.

At the User Interface level it describes technologies to support data driven interfaces through a semantically aware visualisation framework. It additionally describes how WeKnowIt applications are able to compensate for multimodal access and authentication.

The technologies described in this document are necessarily generic in that they are intended to support any WeKnowIt application – discussions about how specific instances of such applications additionally address the requirements (for example in the case of the demonstrator applications) will be left for their respective deliverables.

Abbreviations and Acronyms

CSG	Consumer Social Group
DCTerms	Dublin Core Metadata Initiative Terms
ER	Emergency Response
E-WKI	The WeKnowIt system as seen by users
FOAF	Friend Of A Friend
GeoOWL	W3C Geospatial Vocabulary
OPO	Online Presence Ontology
PDA	Personal Digital Assistant
SIOC	Semantically Interlinked Online Communities
WKI	WeKnowIt

Table of Contents

List of Figures

1. Introduction

The aim of the WeKnowIt project is to explore Collective Intelligence – a form of intelligence derived from a number of different layers of intelligence acting and interacting with a common knowledge store. The Personal Intelligence layer contributes to Collective Intelligence through the actions and reasoning of individuals. These individuals may be part of larger organisations or notional groups but the focus of Personal Intelligence is how the independent actions of these individuals can contribute to the overall Collective Intelligence.

The processing of the Personal Intelligence layer is summarised below:

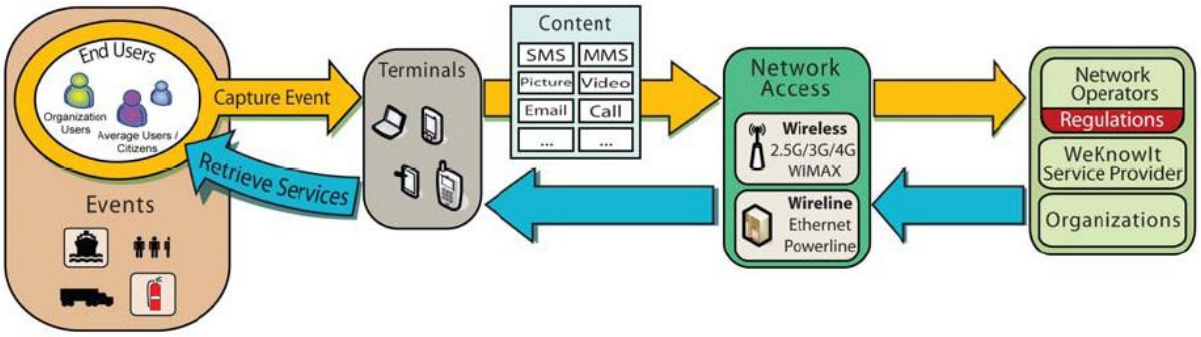

Figure 1: Personal Intelligence in WeKnowIt

As Figure 1 shows the core activity of Personal Intelligence is the provision of information to the WeKnowIt system and the retrieval of information from the system. These two processes can be considered to be occurring both sequentially (in either order), or independently of each other. For example, in an emergency situation Personal Intelligence may be used to support the access of information relating to escape routes. Or, in the case of the travel scenario, the user may upload some information relating to their travel plans at the planning phase and access this information at a later date when they are in the middle of their proposed trip.

Whilst the methods used to undertake the provision and access of knowledge will differ across different WeKnowIt applications, the generic requirements for technology that supports these processes will be relatively consistent. For example, though the specific devices may change over the time, there will be a requirement for users to be able to access WeKnowIt using multiple modalities.

The previous deliverable (D1.1 – Report on Interaction Model) outlined a set of such requirements for Personal Intelligence Management Technologies in the context of WeKnowIt. These requirements related to supporting multiple users, multiple methods for visualising knowledge and requirements related to the types of interaction that Personal Intelligence management technologies support. In addition to these generic requirements there were overall requirements for what specific services are needed in order to support the management of Personal Intelligence.

Finally some non-functional requirements were specified that related to how the expectations that the user may have of the system as a whole.

This document describes how the technologies and methodologies implemented by WeKnowIt have attempted to meet these requirements. It deals with each of the primary requirements and outlines the technology or methodology that has been developed in order to address these requirements.

Specifically it describes the development of an Interaction Model and how this model addresses the requirement of multiple users. It also describes how WeKnowIt has implemented a federated identity server in order to identify users when accessing WeKnowIt applications. It also describes how the core personal intelligence management services implemented in WeKnowIt meet the requirements outlined in the previous document. Finally it describes how some of the non-functional requirements were met.

2. Support for Multiple Users

A key requirement for a system to mange personal intelligence is that it should be able to support multiple users; each with different preferences, profiles and with a different history of interaction with WeKnowIt applications. How WeKnowIt can take advantage of user modelling and profiling will be explored as part of future work. Here the focus is on how it is possible to build a record of the users' interaction with WeKnowIt and how this interaction connects with the other models present in WeKnowIt and the various pieces of information stored as a part of that interaction. To meet this requirement an interaction model was developed which explicitly models the connection between the user and their involvement with WeKnowIt.

1.1. Interaction Model

When describing the interaction model it is important to be clear about the different kinds of *events* that are modelled by WeKnowIt and how this model connects to these events. There is a generic event itself which occurs at some fixed point and unfolds over a period of time. For example, in the ER case study an event could be a tree falling over and blocking a road. The event starts at the point when the tree falls over and continues over time until the emergency is over.

There are then other, more specific, events which are connected to this generic event in the context of WeKnowIt. These are such events as an individual making a record of the emergency, for example taking a photo of the tree, uploading the image to WeKnowIt and then annotating the photo. Each of these actions could also be considered to be an event but these events are inherently associated with the generic tree falling event. Additionally it should be noted that the generic event effectively occurs outside of WeKnowIt (since the tree falling over is unrelated to WeKnowIt, unlike the specific events of uploading information to the system).

Given this structure of events, one approach to modelling the interaction between users and the WeKnowIt system would have been to extend the core Event Model to account for the hierarchical nature of the events. The Event Model is, however, a generic model for representing real world events and, as highlighted above, makes no reference to the specific nature of WeKnowIt. Therefore, a problem with extending this model to account for the types of interaction found in WeKnowIt would be that the model would get too detailed and would therefore lose its expressive power.

The approach taken, therefore, was to develop an explicit interaction model which meant that the interaction between the user and the information processed and stored by WeKnowIt could be modelled outside

of the core Event Model. Future work will examine the best approach for linking the two models together.

Thus the interaction model aims to connect users (or content providers) and resources together. In the system, we define a resource as any piece of information added to the system: it could be and image, a video, a description of an event and so on. The representation of a resource in this way enables the separation of the resource from the user as defined by the model

The model not only provides a general user representation based on standard ontologies such as SIOC [6] and FOAF [5], it represents the relation that users and their communities hold with any system resources such as access policies, subscriptions, authoring and event participation.

Since the WeKnowIt system models events, the interaction model supports the association of *real* events with document resources. This pattern generates the simple union between the concepts of *virtual* resource and *real* resource (event) that is required by the WeKnowIt system.

For each resource (event or document), a set of standard metadata is provided as an attachment. This attachment represents the generic information that can any define arbitrary resources and support the collective intelligence dimension of the system:

Tagging

Information uploaded to the system can be tagged with both standard and semantic tags. Standard tags are generally single words or a small phrase that is used to describe an object. Two problems with standard tags are that multiple tags can be used to point to the same concept and in addition tags can be ambiguous in terms of what they refer to. Semantic Tags are used to overcome these limitations by linking objects to single well-defined concepts[3].

Reviews and Rating

Each piece of information uploaded to the knowledge base can be both reviewed, in the form of a comment, and rated. This process supports the collaborative discussion of a resource and additionally provides a form of search criteria for locating suitable resources.

Roles and Policies

The Interaction Model also defined roles for each user as well as permissions for that user role. This supports a simple management framework to ensure that the privacy and access rights of the information that is uploaded by users is maintained.

Identification of Resources

The Interaction Model specifies that each resource has a corresponding properties relating to the user that uploaded that knowledge. Specifically, the date, the author, the title and the content itself is represented in the

Interaction Model. These properties support the retrieval of information from the knowledge base.

Situational Context

Each resource is associated with both an event and is localised with reference to an event [13]. Thus each resource is given an explicit situational context within the knowledge base.

Subscriptions

Users are able to subscribe to any resource, which allows them to track any changes or updates made to the resource over the course of time.

User Profiling

Finally, the Interaction Model supports user profiling and personalisation by enabling the archiving of profiles and resource history. This mechanism provides identity snapshots that can be used for tracking users' behaviours. The concept is also extended to resources for enabling the tracking of the changes information over time.

The interaction model has been implemented reusing existing ontologies for facilitating the interoperability of the system and creating knowledge that can be reused across the boundaries of the project. Currently the model reuses five different ontologies including Semantically-Interlinked Online Communities (SIOC) [6] (with SIOC/Types and SIOC/Access), Online Presence Ontology (OPO) [12], Dublin Core Metadata Initiative Terms (DCTerms) [4], W3C Geospatial Vocabulary (GeoOWL) [13] and CommonTag [3]. The overall model diagram is given in the appendix.

1.2. Identifying Users

A first step in enabling the management of personal intelligence is identifying the user that wishes to access the system and ascribing them with the appropriate access rights. This process enables each user to carry out tasks which are appropriate to their status within the system.

To facilitate this process of identifying users, WeKnowIt has developed a federated identity[10] service called AsterID. The interface for AsterID is shown below:

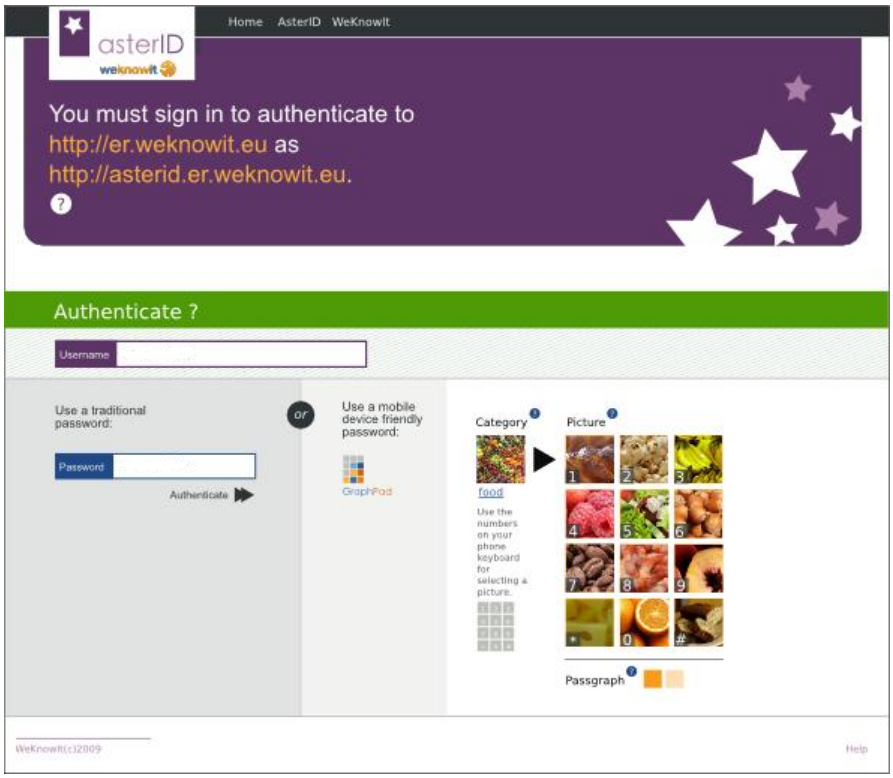

Figure 2: The AsterID login screen

Notionally logging in via AsterID is similar to standard login procedures. The user provides a username (In this case the username is a URI which references the AsterID server) and a password.

The identification of users is vital in WeKnowIt for supporting the profiling and modelling of users (as will be described in D1.3 and D1.4). It also enables the connection between individuals users, the resources they provide to WeKnowIt applications and the events stored in the WeKnowIt knowledge store.

Currently the AsterID is implemented as an OpenID[8] server. AsterID extends the standard OpenID server by generating semantic identities from the basic OpenID attributes. Currently this is achieved by transforming the standard Attribute Exchange information into a FOAF[5] profile. Thus AsterID is able to not only validate and authenticate users of WeKnowIt but is also able to generate and share semantic identities for further processing and also enables this service for use by other web services.

The sharing of semantic identities also allows WeKnowIt applications to link users to external sources of information and thus enables automatic knowledge acquisition from these sources. Thus the use of AsterID allows WeKnowIt applications to uniquely identify users and their associated profile and interaction history, as well as identifying interactions the same users have had with other knowledge sources.

3. Support for Multiple Visualisations

Combined with the requirement for multiple users is the need for WeKnowIt to support multiple visualisations. It is clear that different users will have different informational needs and will, therefore, require different visualisations of the data in order to fully support those needs.

Clearly, a large number of visualisations can be developed for displaying the type of data stored in the WeKnowIt knowledge base. For example data could be displayed in terms of its temporal relationships or in terms of its geographical location. A key issue is, however, how to determine which visualisation should be presented to the user at which time. This decision depends on several factors: what data is available, the type of data that has been selected to view, the profile of the user and so on.

This requirement is addressed in WeKnowIt through the development of a semantically aware user interface framework. This framework allows decisions about how to display information to be determined by the semantic information available in a web page as well as information that can be generated by making inferences on the web page and associated data. Thus the visualisation takes into account both the user that is viewing the information and the information itself. This framework is described in detail below, following which the use of the framework is illustrated through a system for defining overlays over web pages containing semantic information.

1.3. Semantically Aware UI Framework

The Sparks framework is fully written in JavaScript and therefore does not require any particular server side technology. Additionally, the framework integrates a resource management engine that enables cross-domain libraries to be loaded on demand using different methods depending of the context (HTTPRequest, Flash XDR and ScriptTag insertion). The engine is able to manage resources dependencies and load them dynamically.

The framework uses the Design-Bid-Build Pattern for selecting the appropriate views given the available data and also for constructing the view. Prior to executing this pattern Sparks builds a local knowledge store on the client side using information selected by the user and enhanced using inferences from external entities. The pattern then acts on this local knowledge store.

The information in the local knowledge store then forms the requirements to the bidding phase – that is, the structure of the data within the knowledge store formulates the design of the data. The next phase of processing links modules with the data requirements via an efficiency function. This efficiency function measures how well each processing module is able to visualise the data contained in the local knowledge

store. The "highest bidder" is then selected and this module is used to visualise the data and present it to the user.

1.3.1. Ozone Browser

As a means of demonstrating the functioning of the Sparks framework the Ozone Browser was developed [1]. The purpose of the Ozone Browser is to generate overlays over web pages that contain RDFa data. The browser extracts the semantic content from a web document and constructs a client-side triple store which can then be used to derive inferences about the information contained within the document. Sparks is then used to bind a graphical overlay over the document which highlights all the actionable items of the page using a simple icon. This process of enhancing the page is shown below.

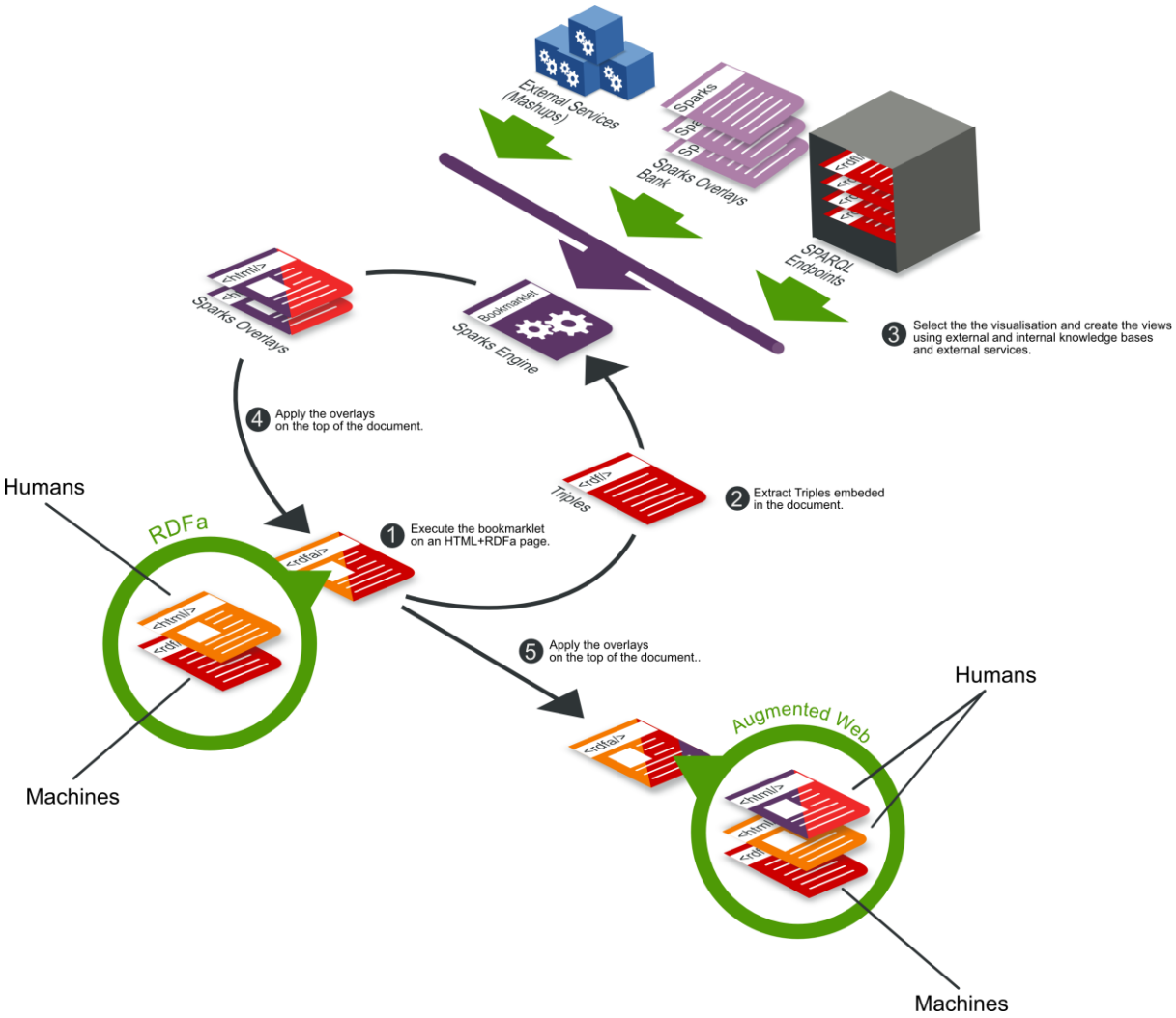

Figure 3: Using sparks to add semantic overlays to suitable web pages

The user is then able to explore these actionable items by pointing at them – this enables Sparks to generate a contextualized view of the knowledge by making inferences from inside and outside of the page. Sparks is then used to dynamically select a view appropriate to the data that has been selected and to generate this view on the page. If no such

visualisation is appropriate to the given data then a default view is generated which simply displays the underlying triple representation.

For example, if the user selects information relating to an event, Sparks is able to make inferences about the event and so determine the location of the event and the corresponding start and end times. From this a map visualisation is generated and overlaid on the page. This visualisation collates this information. An example of such a visualisation is shown below.

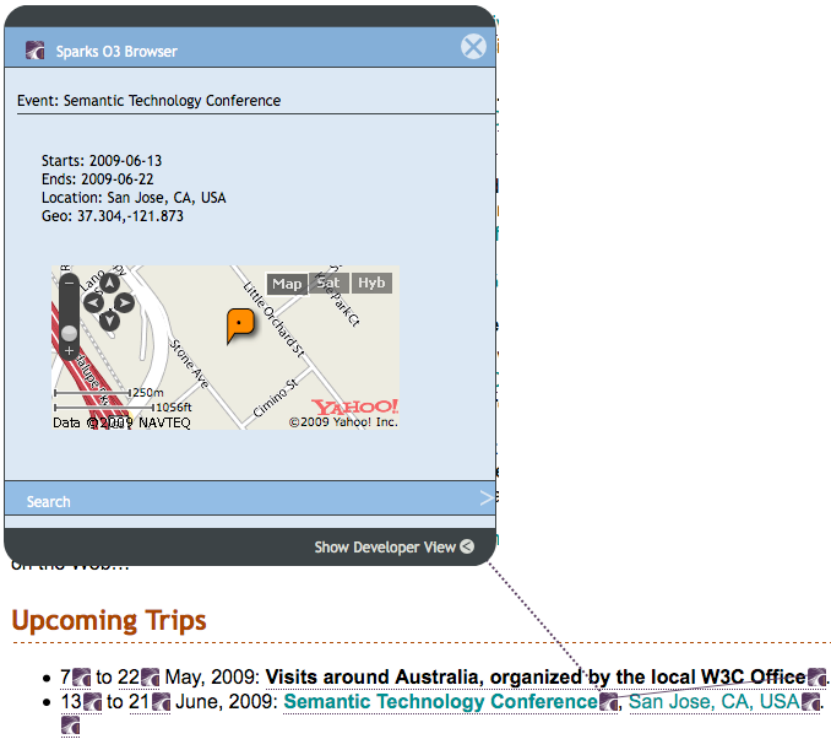

Figure 4: Example of a dynamically generated overlay

Thus it can be seen that Sparks is able to dynamically generate interface elements which are relevant to both the underlying data and any contextual information that can be generated about that data.

1.3.2. Future Work

The new architecture of Sparks (Sparks 2) is going to integrate a JavaScript templating engine based on Pure. The engine can automatically embed semantics in the user interface using the Information Store and the semantic engine (Ozone). These components use RDF JQuery for parsing on RDFa (or RDF and Sparql endpoints) on the fly and creating a client side triple store. This triple store can use local and external semantics.

Figure 5 represents the high level architecture of the reimplementation of Sparks called Sparks 2. The new implementation uses the YUI 3 core libraries and extends then for simplifying the programming task. Contrary to the previous implementation, the new framework integrates the concept of Views (additionally to Libraries and Assets) for producing

cleaner and more portable code. Similarly to the first implementation, the framework uses the concept of convention over configuration for mapping Libraries, Views and Assets together at runtime. Repositories enable the insertion of cross-domain libraries in a simple manner: when required, Sparks can fetch additional dependencies. The framework is currently divided in four sections: Core, Utils, Viz and Ozone. The first namespace manage the fundamental functionalities of the system. Utils implements a new architectural pattern called DBB (Design Bid Build) and the client side information store. Viz gives fundamental graphical libraries for drawing vectors graphics (SVG and VML) and processing canvas elements (using processing). It also provides a templating engine and widgets. Finally, the Ozone module gives additional tools for Web Augmentation.

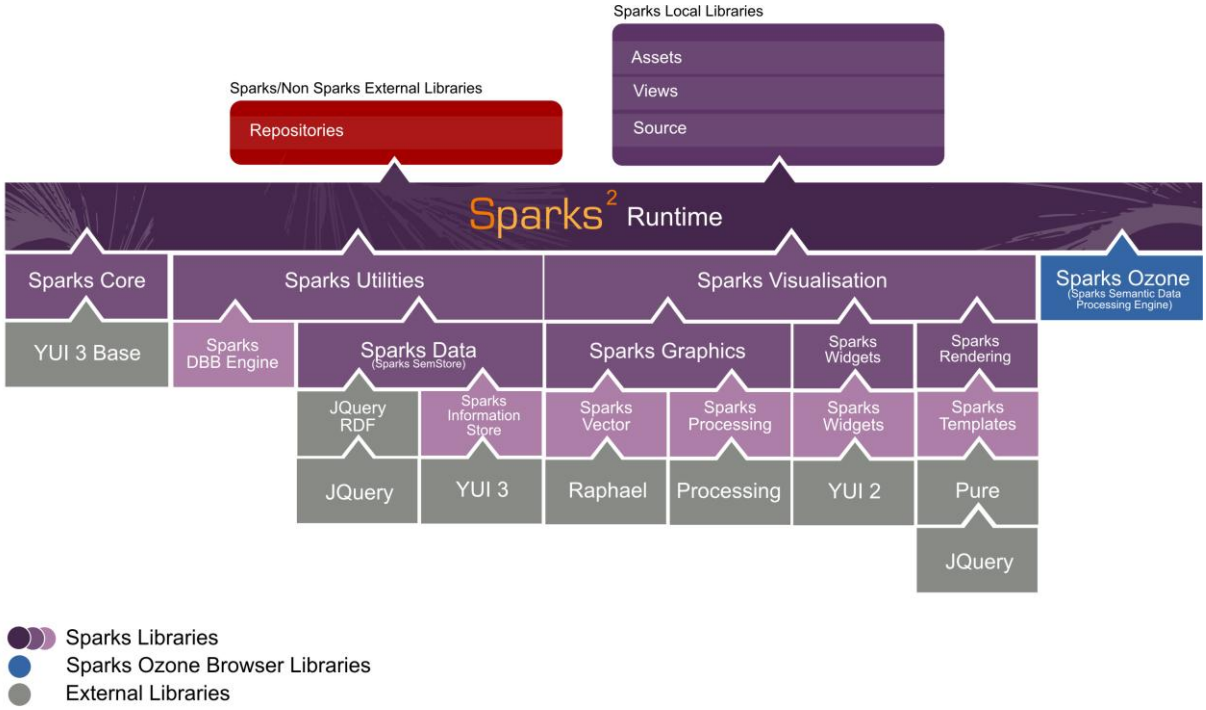

Figure 5: Sparks 2 Global Architecture

4. Technologies for Multimodal Interaction

WeKnowIt applications are designed to be accessible by multiple devices of differing capabilities. This places requirements on such applications in terms of not only how information is presented to the user

One potential approach to supporting multiple modalities for user interaction would have been to support *Migrationary Interfaces*[1]. Typically, migrationary interfaces allow the seamless transition between interface modalities – for example, the user is able to switch between a mobile and a desktop interface whilst retaining their context across both interface modalities.

This process is implicitly supported by the Personal Intelligence technologies described above since the Interaction Model explicitly retains a record of both the user and their previous interactions with the system. In this way, whilst the direct migration between interfaces is not explicitly supported, the previous interactions with WeKnowIt are taken into account when the interface is presented to the user, whether that interface is desktop or mobile based.

In addition to this, WeKnowIt takes the approach of using the browser model to deliver a user interface applicable to the relevant form of modality. WeKnowIt applications are therefore formed from two different interface views – a fully featured desktop view and a reduced mobile view. The web server then determines which modality is being used to access the site and serves the appropriate interface. This approach is used in many Web 2.0 applications to support multi-modality (see Figure 6).

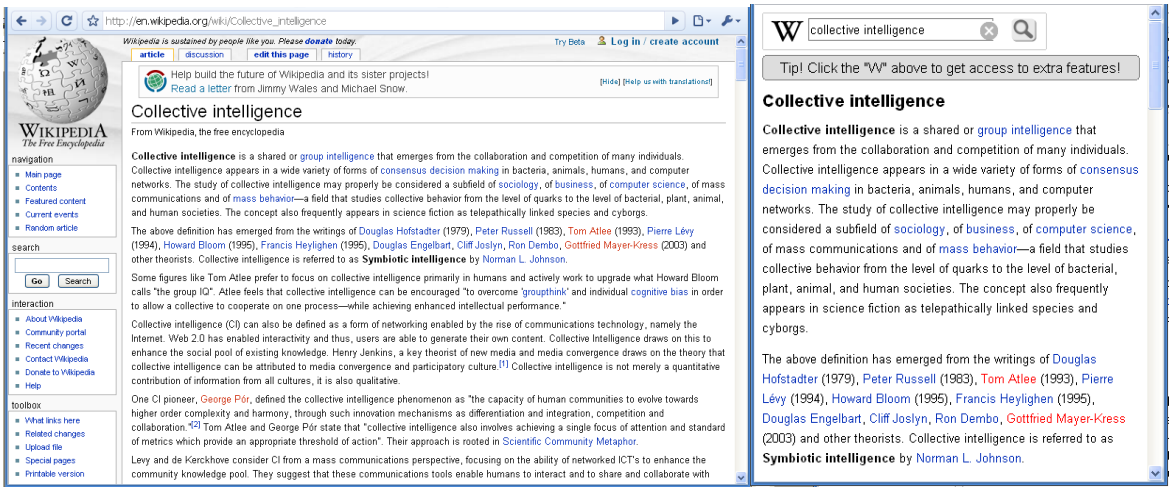

Figure 6: Examples of the Full Wikipedia Page (Left) and the Mobile Page (Right)

So it can be seen that although the mobile interface is not as rich as its Desktop counterpart – the essential actions that are required to use the application are clearly available to the user and this allows a mobile user to have relatively full access to the application at a lower bandwidth.

This approach has been followed in WeKnowIt to support access across different modalities. Two interfaces have been designed to support the management of Personal Intelligence – one for a mobile modality and one for a desktop modality.

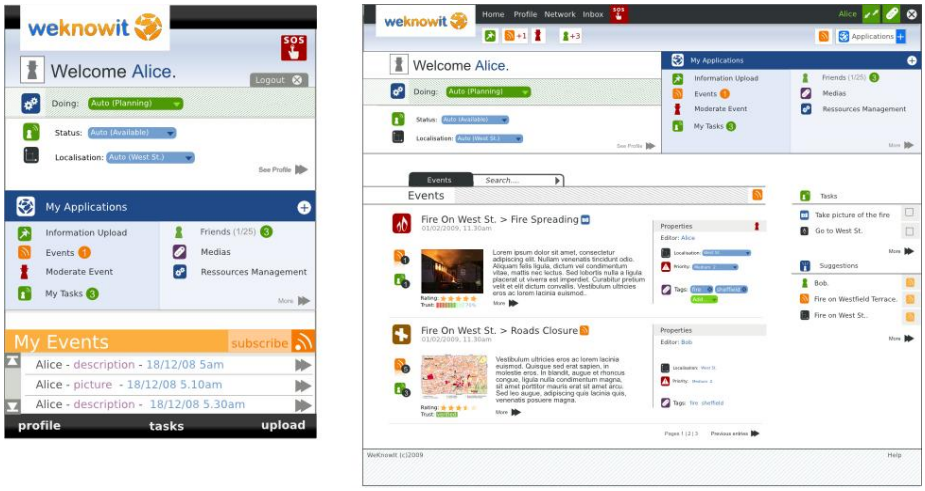

Figure 7: Different Interface Modalities in WKI.

So whilst the two interfaces perform the same function, they offer different views of information and highlight different tasks. In addition to this, in cases where the mobile interface contains sufficient detail to perform a relevant task (for example in the case of uploading information) the information light mobile interfaces are re-used in the desktop interface to support this task. As described above, this approach is further supported by the use of the interaction model to keep a record of user transactions with WeKnowIt applications.

In addition to ensuring that WeKnowIt applications are able to retain their context across different modalities it is also important that applications are able to adapt to the requirements of those modalities. This is especially critical at points where the user has limited resources to access the system – for example, during an emergency situation the user may access WeKnowIt using a mobile device under a degree of stress. Thus WeKnowIt applications should adapt to these conditions. To support this problem, WeKnowIt has developed a simplified login procedure to allow access to WeKnowIt applications that is tailored towards mobile devices as well as also being usable by desktop devices.

1.4. Mobile Access Interfaces

In typical mobile situations the user will access WeKnowIt using a simple device, such as a PDA or a smartphone. These devices are characterised by having limited bandwidth and capabilities as compared to desktop machines. In addition, the interaction methodology is significantly different as such devices rarely support a full physical keyboard – instead they make use of numeric keypads or a touch-screen keyboard to enter textual information.

An issue therefore arises that users of these devices are likely to be distracted due to their environment[7] and additionally the size of the keyboard leads to increased text-entry errors[9]. Thus the text entry process is especially error-prone when logging in to mobile systems. Here the user enters their username and password in order to access mobile systems. The requirement of the user to recall their credentials, coupled with the lack of feedback that arises when entering a complex password means that alternate interfaces can significantly improve the login process in these conditions.

Graphical passwords have been introduced as a means of overcoming the inherent usability problems of traditional alphanumeric passwords. Users typically have to recall a complex alphanumeric string which is specific to the application they are logging in to. Such work has shown that using graphical passwords not only reduces the cognitive load of users when logging into applications but can additionally overcome the issue of users reusing or choosing simplistic passwords when choosing their credentials[11].

In addition to this, however, since graphical passwords negate the need for keyboard entry of passwords they can also overcome the keyboard limitations present in mobile devices. A graphical authentication interface that is designed to address these limitations is described below.

1.5. GraphPad

GraphPad is a graphical authentication interface which has been designed for mobile devices with either a touch-screen interface or a numeric keypad. The interface is shown below:

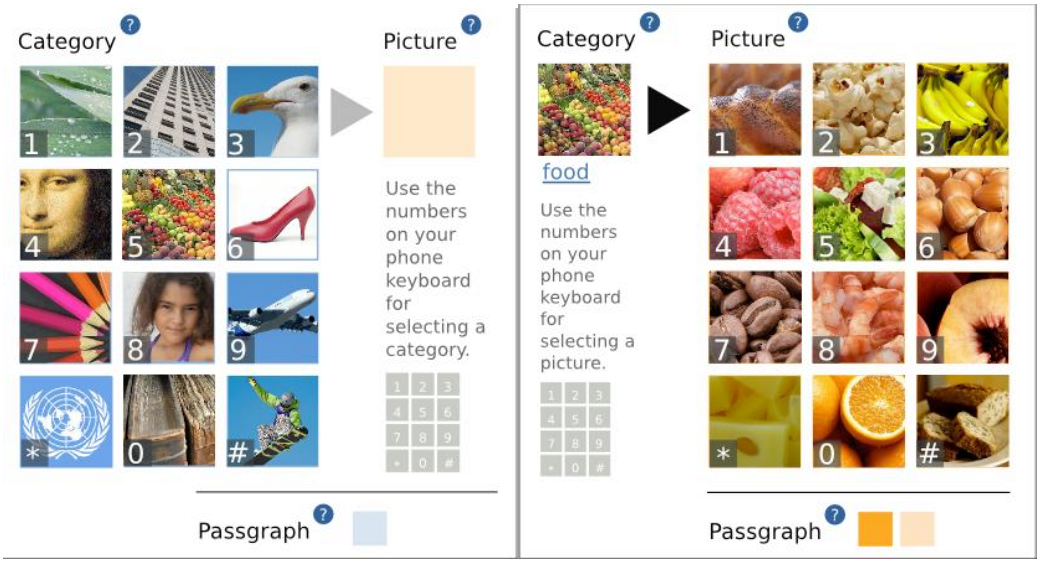

Figure 8: Example of GraphPad Interface

In GraphPad, the password consists of concepts rather than alphanumeric characters. The first character entered defines the high-level category (e.g. fruit, colours, animals etc.). The second character entered selects a

specific image from within that category (e.g. pineapple, orange, banana etc.). The interface has two novel components.

1.5.1. Supporting Recognition over Recall

Since the passcode entry consists of two steps with the initial step being used to select the high-level category the passcode supports recognition rather than recall in two ways. Firstly the high-level categories act as a memory jog for the user when recalling their password. Secondly the user is able to see their passcode image directly rather than having to just recall the relevant character. Both of these factors will mean that the entry of the passcode should be more straightforward than with a standard password interface.

1.5.2. Accounts for multiple entry modalities

The buttons on the GraphPad interface are significantly larger than the keys on typical touch-screen keypads. This reduces the likelihood that the user will make entry errors. In addition, each key is mapped directly to a key on a numeric keypad meaning that if the user does not have a touch screen interface they are still able to easily enter their passcode. Furthermore the interface is also usable in a Desktop environment since the user can simply select the relevant images using a standard pointing device. Thus GraphPad is a consistent interface across multiple modalities.

5. Personal Intelligence Management Services

In addition to the technologies described above for supporting the different access requirements there is also a need for a low level support for the management of personal intelligence. These are implemented as services which fit into the framework defined in D6.2.1. These services were created to meet the requirements for both Personal Intelligence Upload and Access that were outlined in D1.1. The services are described in more detail in the appendix.

1.6. Core Personal Intelligence Services

The core of personal intelligence management is the process of provision of information to WeKnowIt. This upload process is handled by a single service, described below.

1.6.1. ManageItems

This service is concerned with uploading and deleting items like photo photographs, videos, and documents in the WKI knowledge base. As such it forms the basis of the management of Personal Intelligence in terms of information provision.

The service exposes one method for each of these actions. Uploading an item inserts a selected document or media file into the data store along with a use policy or set of permissions to access it. A successful request to this service also modifies the user interaction model to reflect the ownership of the item and changes in an event.

The other method deals with the deletion of items from the knowledge base, should the user want to do this. This simply eliminates the document and its references from the knowledge base.

1.7. Information Enrichment and Editing

Information Enrichment in this case means the processes of tagging, commenting and rating information. Each of these processes is handled by a separate service.

1.7.1. Tag

The tag service allows users to add, change and delete tags that have been assigned to the uploaded items. Tags are attached to the items through references in the interaction model. A tag is a single descriptive word which is attached to the item for descriptive purposes.

More specifically, this service provides functionality to add a tag or set of tags to an existing document in the knowledge base, to retrieve the set of tags associated with an uploaded item, to modify a tag's label for a given

item, and finally, to delete these tags from the set of tags associated with an item.

1.7.2. Comment

Adding comments is another major data entry point in E-WKI. To support this user action, this service was designed to allow to add a textual description of an uploaded item, and also to allow other users to add more comments in order to generate a discussion about said items. A comment is a detailed explanation about an item in the Knowledge Base, it generally has a more interpretative role in information processing since it is intended to be read by end users rather than, as with tags, for internal processing.

The methods exposed by this service permit a user to attach a free text comment to an uploaded item, and also to modify or delete it.

1.7.3. Rate

The rate service is designed to allow users to give a numeric rating from a predefined set of values to an item they will upload or have previously uploaded. The rating themselves can be used for multiple functions and need not be explicitly set by the end user. For example, in the CSG use case the ratings service can be used to ascribe a measure of quality to points of interest. In the ER use case the rating function can be used to assign a level of severity or trust to an event or a piece of knowledge.

This service provides its functionality through one method for attaching a rating to an item in the knowledge base and another method to modify a previously assigned value.

1.8. Information Access

For information access there are three services – two for handling the process of determining who the user is and authenticating them and one for dealing with the process of searching the knowledge base.

1.8.1. Account Manager

This service allows a new user to register with WeKnowIt. Once the user has an account, he will be able to access the other services provided by WKI. This is centred on the OpenID standard and the concept of a federated identity. This service allows any interaction with the system to be identified with a user and thereby ensures that each piece of information within the Knowledge Base can be linked in some way to a registered user of the system.

The exposed functionality of this service allows a previously authenticated user to create an account, update it or entirely delete it from the system.

1.8.2. Log In

The login service is the main entry point to the rest of the services in WKI, provided a user has already created an account.

The log in service deals with two scenarios. The first one considers the case of a user that has an OpenId and that has already been authenticated by an application and wishes to access the site. In the case, this service merely registers the fact that a new session of activity for the user who owns that OpenId has begun. The second case addresses the case when the user does not have an OpenId and wishes to provide a username and password to log in.

The functionality in this service directly reflects these cases and allows the user to log in or out of the system by providing his OpenId or a pair of username and password to validate.

1.8.3. Search Knowledge Base

This service is concerned with querying and accessing the information stored in the knowledge base. The search service will be used by end users

To achieve this, it exposes only one method that can obtain a list of results associated with that keyword query in the knowledge base. This leverages the content added in the information upload services and the content which has been further enriched by other intelligence layers

6. Non-Functional Requirements

The requirements described above have been concerned with functional aspects of Personal Intelligence Management. The previous deliverable also highlighted some non-functional requirements that define more abstract qualities of the system.

The majority of these non-functional requirements focused on elements of the system which can only be addressed by specific implementations of WeKnowIt applications. For example, requirements like the system being reliable and easy to use cannot be addressed here.

This document has, however, shown how the technologies implemented for the management of Personal Intelligence have been designed to meet some of these non-functional requirements. How this has been done is described below.

Trust
To a certain extent the implementation of the system will determine how trustworthy it is considered by users. As highlighted previously this can be done through the design of the interface and ensuring that user confirmation is sought before taking any actions.

The work on the interaction model above, however, has shown that the user is able to update and augment any information they upload to WeKnowIt, meaning that the user can be confident that their associated knowledge is correct. Trust is also maintained by keeping a record of all the interactions with each piece of knowledge – whether these interactions have been system or user initiated. Thus the user is able to see and track what changes have been made to any data they have provided or annotated.

Privacy
Again, the privacy requirements of the user are accounted for by the interaction model. This model defines policies and permissions for each group of users and for each piece of knowledge uploaded to WeKnowIt. Thus this defines a framework for ensuring that the appropriate levels of privacy are maintained when interacting with WeKnowIt applications.

Personalization
Again the personalization requirement has been addressed by explicitly recording each interaction that the user has with the system. Because of this any WeKnowIt applications will be able to make use of this data in order to support personalization. In addition the Sparks framework described above will allow WeKnowIt applications to dynamically construct views of data on the basis of both the data and any knowledge about the user. Thus the framework implemented for the management of personal intelligence assists application designers in supporting personalization.

7. Conclusion

Deliverable D1.1 developed a set of requirements for the management of Personal Intelligence. It described how WeKnowIt applications will have interfaces accessible from different devices and will be used by different types of users in a variety of contexts. Thus it described requirements for WeKnowIt applications to support multiple users and to enable those users to access information using multiple visualisations. In addition to this users will access such applications in different contexts and using different modalities.

This deliverable has described how the technologies and methodologies implemented for the management of Personal Intelligence have addressed these requirements. To address the requirement for WeKnowIt applications to support multiple users an Interaction Model was developed. This model described the connection between a user of WeKnowIt, the events that the user is able to annotate and provide information for, and the specific annotations and information that the user makes. It also showed how the system is able to keep a record of any annotations or modifications made to that knowledge.

To address the requirement for multiple visualisations the Sparks framework was developed. Sparks enables the dynamic construction of visualisations on the basis of inferences made over the underlying data and connections to external sources.

A further requirement is that users should be able to access WeKnowIt applications using different modalities. This requirement is supported by two different technologies. Firstly as described above, the system keeps a record of each interaction the user makes with the application and the views are dynamically generated on the basis of the properties of the data and the user viewing the data. Thus the resulting interface to WeKnowIt can be dynamically generated according to the modality of the user. Secondly, a graphical login system, GraphPad, has been developed which supports the login process across mobile and desktop modalities with a simple to use interface.

The core requirements of the upload and access of personal intelligence have been addressed through the implementation of services within the WeKnowIt system architecture to allow users to upload and annotate resources and then search for information from the WeKnowIt knowledge store.

Finally, although a significant proportion of the non-functional requirements can only realistically be addressed by specific WeKnowIt applications this document has outlined how the technologies implemented to support personal intelligence management have met some of these non-functional requirements.

The work on the Interaction Model and on technology to support the definition of a shared user profile will provide a foundation for further research on user profiling and modelling to be described in Deliverables D1.3 and D1.4.

In addition to this, the evaluations described in D7.6.1 will be used to validate that the approaches described above adequately meet the requirements specified in D1.1 The demonstrator applications should both show how the implementation specific requirements described in D1.1 have been met and generate further requirements for the management of Personal Intelligence. This is to be expected as part of the user-centred design process.

This deliverable has shown how the generic technologies implemented by WeKnowIt enable the core requirements of information provision and access for the management of Personal Intelligence. It has also described how these technologies enable different end users to access WeKnowIt applications under different contexts and modalities and with different informational needs.

8. References

[1] Bandelloni, R., Paternò, F. Platform Awareness in Dynamic Web User Interfaces Migration. In proceedings of Mobile HCI 2003 (MHCI03), Udine, 8-11 September, 2003. Springer-Verlag, LNCS 2795, pp. 440-445.

[2] G. Burel, A. E. Cano and V. Lanfranchi *Ozone Browser: Augmenting the Web with Semantic Overlays*. Scripting and Development for the Semantic Web Workshop ESWC 2009

[3] CommonTag: http://www.commontag.org/Home

[4] Dublin Core Metadata Initiative Terms: http://dublincore.org/documents/dcmi-terms/

[5] FOAF Project: http://www.foaf-project.org/

[6] A. Harth, J.G. Breslin, I. O'Murchu, S. Decker, *Linking Semantically-Enabled Online Community Sites*, Proceedings of the 1st Workshop on Friend of a Friend, Social Networking and the (Semantic) Web (FOAF Galway), Galway, Ireland, pp. 19-29, September 2004.

[7] Kristoffersen, S., Ljungberg, F.: "Making place to make" IT work: empirical explorations of HCI for mobile CSCW. In: Proceedings of SIGGROUP '99, pp. 276–285. ACM Press, New York, 1999.

[8] OpenID: http://www.openid.co.uk/

[9] Parhi, P., Karlson, A.K., Bederson, B.B.: "Target Size Study for One-Handed Thumb Use on Small Touchscreen Devices". In: Proceedings of MobileHCI '06, pp. 203–210. ACM Press, New York, 2006.

[10] Pftzmann, B., Waidner, M.: *Federated Identity-Management Protocols*. LNCS: 3364 (2005) 153

[11] Suo, X., Zhu, Y. and Owen, G.S, *Graphical Passwords: A Survey*, In 21st Annual Computer Security Applications Conference (ACSAC) (December 5-9), 2005.

[12] Stankovic M., *Modeling Online Presence* in Bizer C. and Anupam J., Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008) Karlsruhe, Germany, October 28, 2008

[13] W3C Geospatial Vocabulary: http://www.w3.org/2005/Incubator/geo/XGR-geo/

A. Interaction Model Overview

Below the full interaction model is detailed.

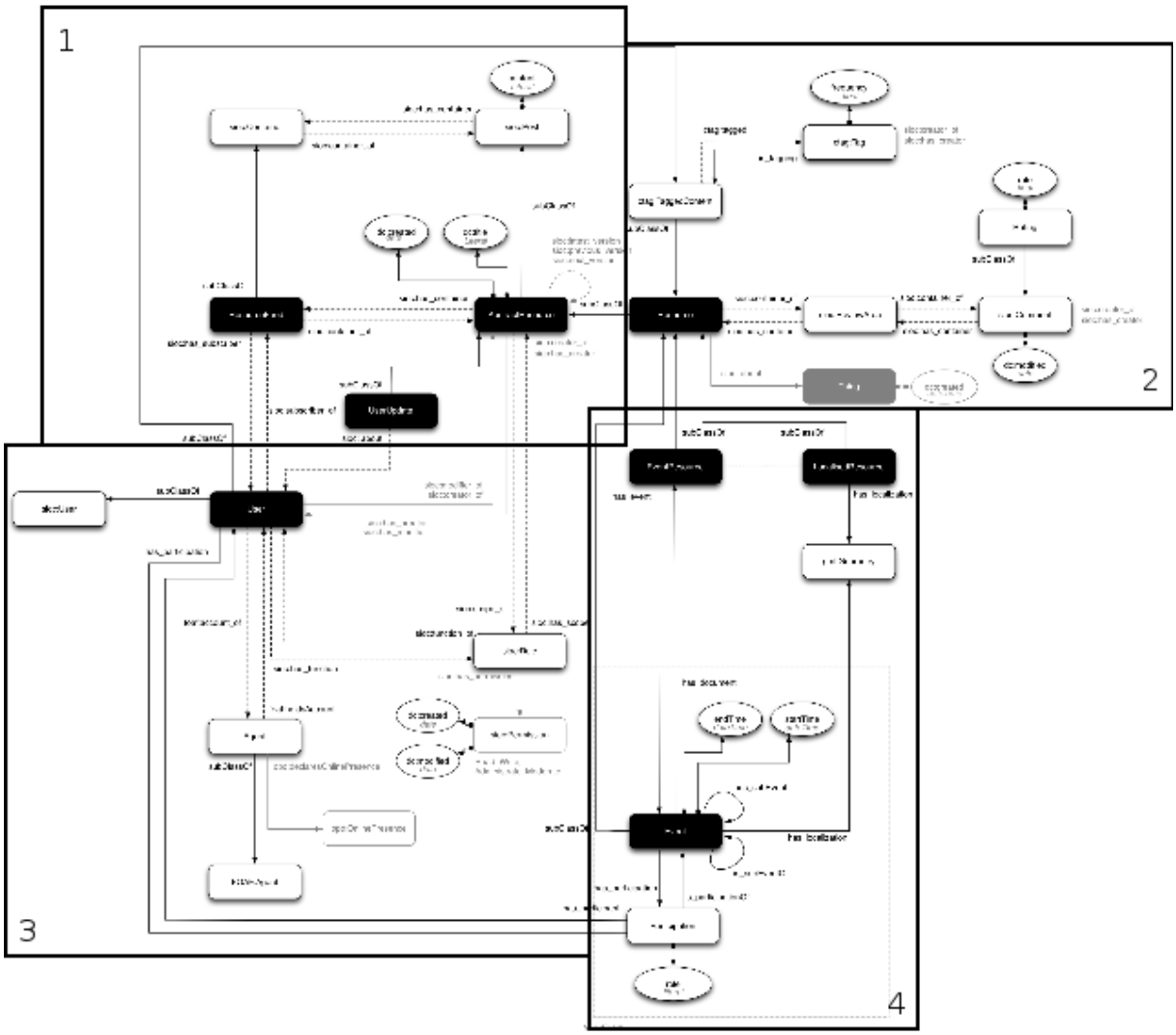

Figure 9: Overall Interaction Model

The explanation of the interaction model begins by describing how updates

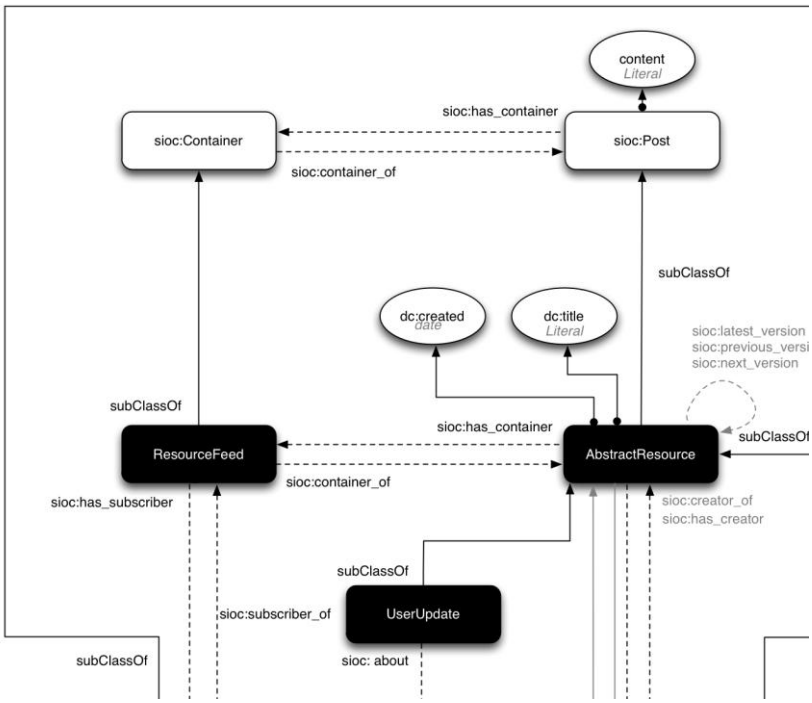

Figure 10: Part 1 of the Interaction Model

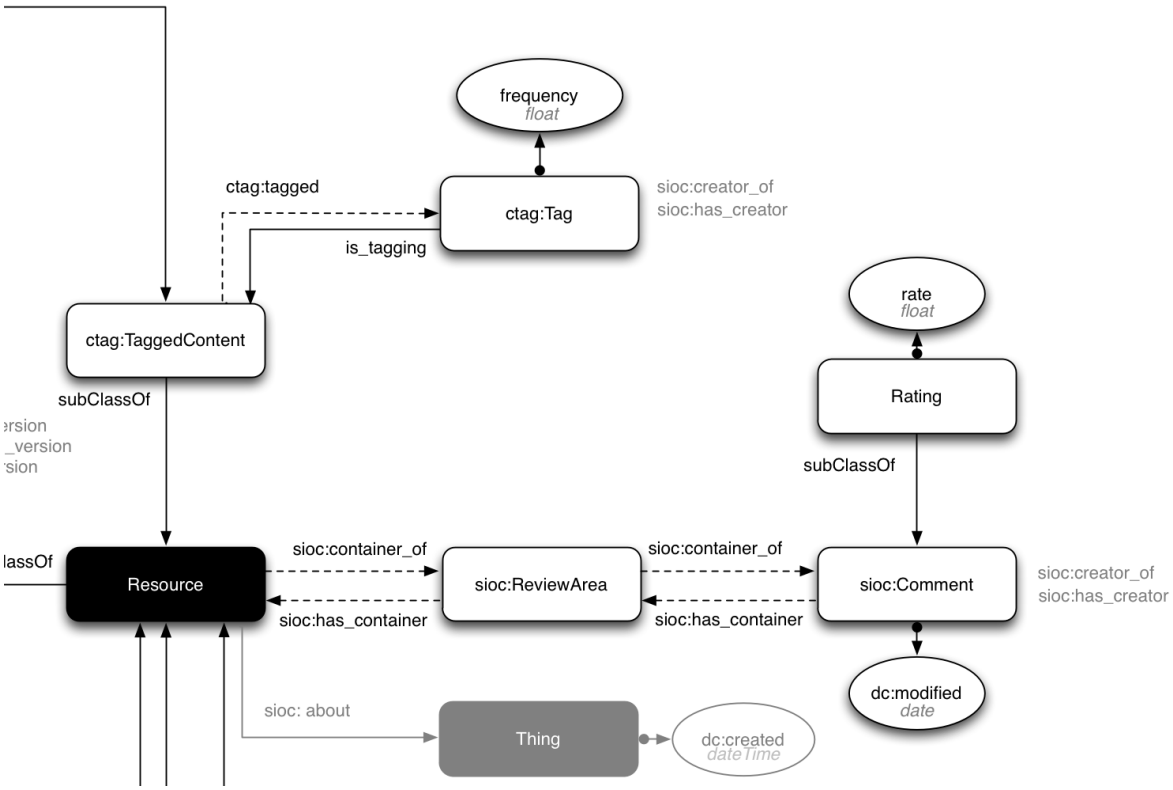

Figure 11: Part 2 of the Interaction Model

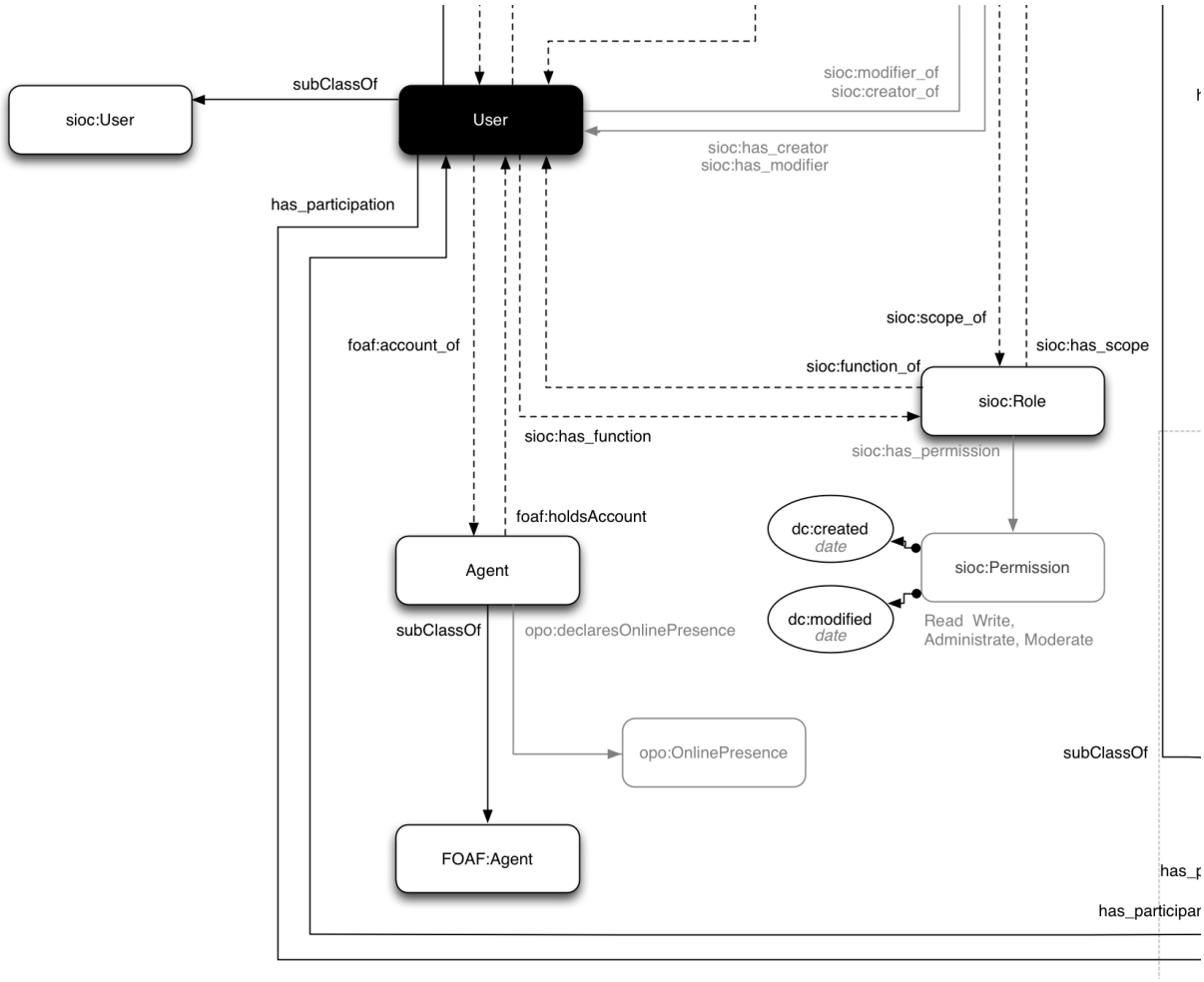

Figure 12: Part 3 of the Interaction Model

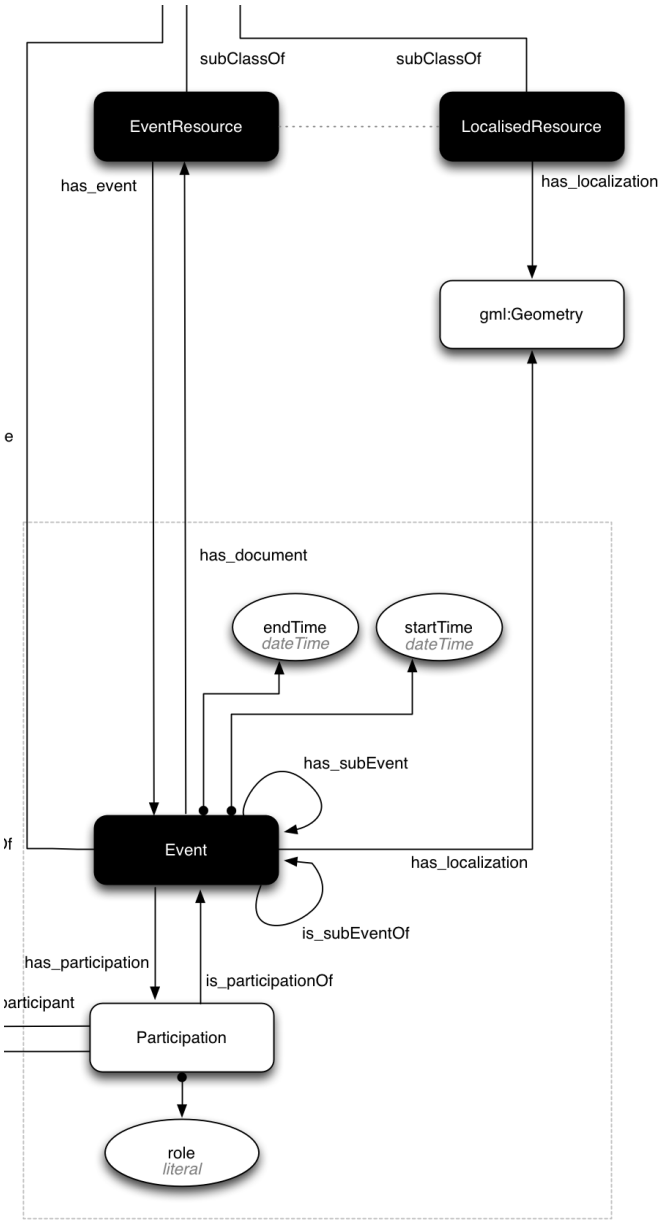

Figure 13: Part 4 of the Interaction Model

B. Personal Intelligence Management Services

This section describes the details of the specific services implemented for the management of Personal Intelligence.

B.1. ManageItems

Method	Inputs	Effect
uploadItem	The URI of the item The relevant permissions	This method stores the relevant item alongside the given permissions level in the Knowledge Base
deleteItem	The ID number of the item	Removes the item from the Knowledge Base

B.2. Tag

Method	Inputs	Effect
addTag	The ID of the item to add Tags to A list of tags to be added to the item	The tags are stored in the Knowledge Base and are linked to the given document
getItemTags	The ID of the item to retrieve tags for.	Returns the tags appropriate to this item from the Knowledge Base
modifyTag	The ID of the item The ID of the tag for the given item The modified Tag	Modifies the given tag associated with the relevant item in the knowledge base
deleteTag	The ID of the item The ID of the tag for the given item	Removes the specified tag from the specified item.

B.3. Comment

Method	Inputs	Effect
addComment	The ID of the item to	Attaches the provided

	comment The text of the given comment	comment to the given item within the Knowledge Base
modifyComment	The ID of the item The ID of the comment The new text of the comment	Modifies the specified comment with the new text in the Knowledge Base
deleteComment	The ID of the item The ID of the comment	Removes the item from the Knowledge Base

B.4. Rate

Method	**Inputs**	**Effect**
addRating	The ID of the item to rate The value of the rating	Stores the provided rating alongside the item in the Knowledge Base
modifyRating	The ID of the item to modify The ID of the rating to modify The new value of the rating	Updates the given rating within the Knowledge Base

B.5. Account Manager

Method	**Inputs**	**Effect**
createUser	The OpenID reference for this user	Creates this user account within the Knowledge Base
createUser	The username of this user The password for this user	Creates this user account within the Knowledge Base and attaches the given user name and password to the user
updateUser	The User to update	Updates the Knowledge Base to reflect any changes to the given User
deleteUser	The User to be deleted	Removes the provided user from the Knowledge Base

B.6. Log In

Method	Inputs	Effect
userLogin	The OpenID of the user	Logs the user in to the system
userLogout	The OpenID of the user	Logs the user out of the system
userLogin	The ID of the user The password for this user	If the credentials match the User is logged in to the system
userLogout	The ID of the user	The User is logged out of the system

B.7. Search Knowledge Base

Method	Inputs	Effects
searchQuery	The ID of the user The keywords required for searching	Builds a list of results from the information contained in the Knowledge Base