ITEC808 Project Report

Applying Social Networking to the Enterprise – Making an Intelligent Status Tracking Application

Sidney Shek (41419979)

Supervisor: Dr Rolf Schwitter

5th June 2009

Abstract

A common problem in the workplace is determining how best to communicate with one another especially between mobile workers. This report presents the results of a feasibility study and selection of technologies for a prototype intelligent status tracking system called 'Can We Talk Now?' (CWTN). CWTN is inspired by principles from Social Networking Services (SNS) and makes use of various features from mobile devices (Apple iPhone initially) such as calendars and GPS to automatically populate status information. A rules engine will be used to implement the logic to determine the "best" means of communication between users. CWTN's automated nature should encourage use of the system by busy individuals who would otherwise be unable to maintain status information in standard SNSs. In addition, unlike other systems CWTN aims to be independent of telecommunications providers thereby allowing any enterprise to use it. The JBoss Drools rule engine was selected to implement the status querying logic. Server-side infrastructure would be deployed on Google App Engine, making use of free publically accessible infrastructure and authentication to address security concerns. Client-server communications will be via REST services using JSON format. The next phase of the project will be the development of the prototype using the selected technologies.

Acknowledgements

I would like to express my appreciation and gratitude to the following people and groups:

Dr R. Schwitter for his encouragement during the project and valuable advice in the various write ups and presentations throughout the unit.

CSC Australia for sponsoring my enrolment in the course and encouraging development of mobile device applications.

Table of Contents

List of Figures

List of Tables

1 Introduction

Team communications are essential in an enterprise environment. However, team members are often busy with meetings or delivery for deadlines and may be mobile. As a result, a common problem faced by many individuals is determining the most effective means of communication with another person at a point in time. For example, an urgent question is probably best answered in-person or via a telephone conversation. However if the target of the question is in an important meeting with a client, he/she may not appreciate an interruption.

As an added dimension to this problem, many users now have smartphones (e.g. Apple iPhone or Blackberry). While these users may not always have access to a computer such as while travelling or between meetings, they do have access to the Internet or the cellular network and hence can be contacted, albeit with potentially limited capabilities. By providing accurate status information about these mobile users, it is possible to facilitate phone conversations or other limited forms of communication, thereby allowing productive use of their time.

There are tools such as emails, calendars and instant messaging that partially address the problem of providing up-to-date status information. However they have several issues such as:

1. Calendars are not often kept up to date, and do not always accurately reflect the duration of the meetings or a person's availability (e.g. a person may not be in a meeting, but may be busy and not interruptible).
2. Instant messaging tools typically only provide basic status information to identify whether someone is busy or idle. Also, it is a common occurrence for users to forget to update their status. In addition, instant messaging tools are typically used only on PCs and not mobile devices due to constant network activity draining limited battery life.
3. Emails and other 'word by mouth' status updates may be easily lost or not distributed.

Social networking services (SNS) such as Facebook[1] and Twitter[2] provide a central location for status information that is manually maintained, yet they have demonstrated that near real-time updates to status information can be achieved. This information could then be used by others to determine the best means of communication. It is the application of this principle to mobile workers in the enterprise that is the focus of this particular project.

1.1 Project aims

The main aim of this project is to develop a prototype system named CanWeTalkNow? (CWTN) for recording and querying a person's current status. The key aspects of the system are:

- Automatic population of status information based on various sources available from mobile devices. This includes location information based on Global Positioning System (GPS), calendar entries and network availability.

[1] See www.facebook.com
[2] See www.twitter.com

- Provision of inbuilt logic to determine the ‘best’ means of communication between two users of the system.

- Low development, deployment and support costs so as to encourage uptake of the prototype in pilot and potentially full deployment scenarios.

- Support integration into an enterprise environment for providing status information to other systems, or potentially obtaining necessary user data from other systems.

- Only make use of end user functionality in mobile devices, thereby removing dependencies on telecommunications providers.

The prototype could serve as the basis of a pilot in an organisation where the effectiveness of this particular application can be measured, possibly resulting in future research work or development into a larger scale solution.

1.2 Significance

CWTN serves as a platform for a number of potential uses:

- It is a tool designed to improve communications within mobile teams. This could lead to productivity improvements in organisations.

- Successful uptake of the CWTN prototype with resulting productivity benefits could promote SNS deployments in enterprises, which are typically conservative and require quantitative evidence before deployment state-of-the-art technologies such as SNS. These enterprises could benefit from resulting productivity improvements from SNS usage to share knowledge.

- CWTN can also be a research vehicle or test bed into SNSs, use of mobile devices in the enterprise, integration with various enterprise systems, and application of reasoning to SNS knowledge bases.

- CWTN could also be turned into a niche product for use by many enterprises.

1.3 Project Approach and Expected Outcomes

The project is primarily software development. However significant effort is required to scope and analyse the system to minimise project risk.

The project will run over two semesters covering units ITEC808 and ITEC809. In order to fit into ITEC808/809 unit split constraints, the project will run in two phases: (1) feasibility study, technology selection and scoping, and (2) software development.

- Phase 1: Feasibility study, technology selection and scoping. This is the main focus during ITEC808, and the results are presented in this report. The expected outcomes of this phase are:
    - An understanding of similar SNSs and systems, allowing targeting of CWTN features not currently available in other tools.
    - Defined requirements for the system, and defined scope and plans for the next project phase.
    - Selection of technologies to be used in the system.
    - An understanding of potential issues for deployment of the prototype.

- Phase 2: Software development. This will be the focus during ITEC809 and potentially beyond. It will consist of system design, development and test. It will be based heavily on analysis and scoping work done as part of phase 1. The expected outcomes of this phase are a developed prototype system, and accompanying design documentation to allow further development post ITEC809.

1.4 Task Plan for Phase 1

The following tasks were conducted for phase 1:

1. Literature review of existing public and private SNSs, and other similar systems. Investigation into the capabilities of mobile devices which would be the primary input into the system was also performed. Outcomes of this task are presented in Chapter 2.

2. Requirements and systems analysis producing a high level understanding of application logic, key data entities and requirements. This is presented in Chapter 3.

3. Review, evaluation and selection of a suitable rule engine. This is presented in Chapter 4.

4. Investigation into client-server communications technologies. This is presented in Chapter 5.

5. Investigation into server platforms with the resulting selection of a target platform. This is presented in Chapter 6.

6. Scoping and planning for the development phase, including discussion of potential deployment issues and future research areas. This is presented in Chapter 7.

2 Literature Review – Social Networking Services and Mobile Devices

CWTN is inspired by social networking principles so a number of SNSs were reviewed to identify key learnings. Particular interest is also given to the 'Intuitive Network Application' framework from Bell Labs which targets a similar problem to CWTN but from a telecommunications service provider's perspective. This chapter also presents results of review into mobile device capabilities and development considerations, which are key inputs into requirements for the system.

2.1 Social Networking Services

The term Social Network Service (SNS) refers to tools that provide means to:

1. Share information between members and build up knowledge repositories. Wikis and forums are simple examples of this [DiMicco et al., 2008][Newman and Thomas, 2009: Ch 5].

2. Find other people based on common interests, and also through relationships. Network searching capabilities are a key aspect to this [DiMicco et al., 2008][Newman and Thomas, 2009: Ch 5].

3. Maintain relationships, especially weak or long distance relationships [Wikipedia Social Network Service, 2009].

Well known publically available SNSs such as Facebook [3] and Twitter [4], as well as enterprise specific SNSs such as SocialEngine [5], Lotus Connections [6], ThoughtFarmer [7] and HiveLive [8] were briefly investigated. Common features and characteristics that these systems share include:

- Entry and display of user profile and status information. These are typically user entered free-text information with minimal structure so as to be flexible. More recently, sites such as Twitter have provided near real-time 'pushes' of information to interested parties including SMS messages to mobile phone users [Alleven, 2007].

- Provision of mini-applications to users. These range from simple games through to information sharing tools such as photo galleries [3].

- Support for network searches. SNSs provide sophisticated searching capabilities to find users with similar interests [Newman and Thomas, 2009: Ch 12].

- Message boards for posting public and private notices. Wikis form the basis of this functionality in some enterprise SNSs [7].

- Provision of privacy mechanisms. The SNSs provide means of hiding profile and other posted information from the general public [3]. The impact of this is discussed further in Section 2.1.2.

[3] See www.facebook.com
[4] See www.twitter.com
[5] See http://www.socialengine.net by Webligo
[6] See http://www-01.ibm.com/software/lotus/products/connections/homepage.html by IBM Corporation
[7] See http://www.thoughtfarmer.com by OpenRoad Communications, 2008
[8] See http://www.hivelive.com by HiveLive, 2008

2.1.1 Key outcomes affecting requirements of CWTN

Two key outcomes were identified from the review of SNSs that should affect the requirements and targeting of CWTN. These are as follows:

1. SNSs are flexible in terms of the structure of information. Operation and use of SNSs is user driven, as opposed to being defined by the system or developers. This is because how SNSs are used typically evolves over time [DiMicco et al., 2008]. As such flexible data structures need to be considered for CWTN.

2. There are a number of existing enterprise SNSs that provide functions that may be useful for the proposed system such as building a network of users interested in a person's status, as well as message boards for low priority notifications. However, there are no specific tools supporting intelligent status tracking with integration with mobile devices. As such CWTN is a niche application, and its architecture should support integration with other SNSs so as to avoid re-development of common SNS functionality. In the long term, it may be considered as a mini-application to be deployed onto existing SNSs.

2.1.2 Potential issues in deploying SNSs to the enterprise

Deploying SNSs into the enterprise has specific issues quite dissimilar to a public deployment, and must be addressed by CWTN. These include scale, culture, privacy and security. These are described in more detail below.

1. Scale: The number of users of enterprise SNS is significantly less than for a publically accessible service, and hence the potential benefit due to the scale of the deployment is less [Newman and Thomas, 2009: Ch 15].

2. Culture: Newman and Thomas [2009: Ch 3 and 14], and Casarez et al. [2009: Ch 9] describe several issues related to the business culture that can hinder the uptake of SNS. These include:

    a. SNS may be seen as tools used for personal benefit and hence detract from productivity.

    b. Benefits to users may not be immediately obvious, which then reduces the motivation to use SNSs.

    c. Different levels of SNS expertise. While some enterprise users are familiar with publically available SNSs, other users would not be. The latter set may not receive the full benefits of an enterprise SNS without assistance. One such mechanism is to automatically populate information where possible. This is an important consideration for this project, and is indeed one of the driving factors for creating an intelligent system.

3. Privacy: Additional information about users is typically exposed in an SNS (e.g. for profiles), which has privacy implications. However, research by DiMicco et al. [2008] has indicated that privacy issues are not as significant as for publically accessible services. This is probably due to the formal relationship between employees and an organisation as well as legal requirements for privacy in the workplace. Privacy mechanisms in the system should be considered, but are less critical compared to operation of the system.

4. Security: Any deployed system needs to prevent unauthorised access to stored data and the enterprise network. In particular since CWTN requires access to the server via the Internet as mobile devices are external to the internal network, appropriate technologies need to be used such as HTTPS encryption and enterprise approved programming languages [Newman and Thomas, 2009: Ch 15].

2.1.3 Future trends in enterprise SNSs

With the increase in collaboration and knowledge repositories from SNSs, four trends are anticipated: Use of data mining, need for advanced searches, use of Semantic Web technology, and integration between SNS and enterprise systems. While these may not have a direct impact on CWTN, they can be considered as future requirements for the system. The trends are described in detail below:

1. Use of data mining on information stored in SNSs, thereby enhancing the value of the shared information [Casarez et al., 2009: Ch 9]. For example, by analysing the historical recording of status changes of users, it may be possible to estimate the amount of time spent travelling, or in meetings. This could then be used to identify productivity improvement mechanisms.

2. Need for advanced searches for information – With large volumes of information available through SNSs, identifying relevant information becomes important [Casarez et al., 2009: Ch 9][Newman and Thomas, 2009: Ch 13]. This particular trend does not directly impact on the project apart from the need to expose status tracking information for use by other systems.

3. Use of Semantic Web technologies to store data such as RDF and OWL ontologies – This allows machines to easily process user entered information which greatly assists with the above two trends [Newman and Thomas, 2009: Ch 13]. Semantic Wikis such as SweetWiki [Buffa et al., 2008] are an example of combining Semantic Web technologies with Wikis (which are becoming more common in enterprises [Newman and Thomas, 2009: Ch 5]). A specific ontology could be developed to represent status tracking information as the information is structured and has well defined relationships. The benefit of using Semantic Web technology to store data from this application would need to be evaluated further before use in development. It may be an extension to this project in the future. There are tools available for converting standard ‘object’ based data structures into RDF which may be used [Diaconescu and Wagner, 2007][Kalyanpur et al., 2004].

4. Integration between SNS and other enterprise systems – Enterprises typically have vast amounts of information already available (e.g. Human Resource systems or directories for profile information [DiMicco et al., 2008]). Integrating these systems with SNS increases the volume of data available to users and allows features such as auto-population of information (e.g. a user’s profile can be partially populated [DiMicco et al., 2008]). Integration is typically via Service Oriented Architecture (SOA) technologies and approach [Newman and Thomas, 2009: Ch 5]. As such, CWTN should consider the use of SOA technologies such as XML and Web Services and exposing proper ‘services’ rather than plain database views.

2.2 Intuitive Network Application Framework

Dinoff et al. [2007] reports a similar system to CWTN using a service-provider or carrier-centric solution based on Bell Lab's Intuitive Network Application (INA) framework. It uses Alcatel-Lucent's DataGrid data mediation and Vortex rule engine products to extract necessary status information from devices and then reason on the data in order to provide customised services such as advertisements. Their system aims to support high data volumes for status information and queries, with flexible rules or policies for individual users down to configuring what statuses should be seen by other specific users. This level of granularity is quite important for a public-based system, but is likely to be too detailed for busy enterprise users.

Being a carrier-centric solution their system has more information available without user intervention. However, it relies on a carrier providing the service which is not always feasible for an individual enterprise. CWTN makes use of end-user mobile device capabilities only, and hence is carrier-independent. Mobile workers may not all use the same carrier, as some choose to use their personal mobile device (e.g. their personal Apple iPhone).

Dinoff et al. [2007] also present results from surveys of users their system which highlights two considerations that should be taken into account for CWTN:

1. There is a need for a simple user interface with as much automated data entry or discovery as possible. Survey results indicated that users found it difficult to maintain their status information solely with manual data entry.

2. Users would be willing to share some personal information for convenience in services. This confirms that privacy is not the primary concern that CWTN would need to address.

The INA-based system also suggested some data entities which may be useful for CWTN, and were independently identified by us (see Chapter 3 for details).

2.3 Mobile device capabilities and issues

Many mobile device platforms now support similar device features, and encourage development of applications through provision of free software development kits and emulator packages. The capabilities and design considerations for Apple iPhone [Apple AppProgGuide, 2009][Apple HIG, 2009] and Google Android [Google Seamlessness, 2009] platforms were investigated further as representative of advanced mobile devices available at this point in time. The results of this investigation are presented below.

2.3.1 Available inputs for determining status information

Mobile devices have a number of capabilities that may be used to provide inputs into determining a user's current status including location detection, calendar information, network availability monitoring, time information and local storage [Pajunen and Chande, 2007]. These inputs along with their design considerations are listed below:

- Location detection – mobile devices can determine a user's location using network-based triangulation (WiFi or cellular network) as well as GPS. Network-based triangulation is quicker and hence uses less power but is less accurate [Apple AppProgGuide, 2009]. A design decision will need to be made as to the accuracy required for this application. In addition, geocoding is available which provides

translations between physical coordinates and addresses [Google Geocoder, 2009]. This could be useful in mapping a user's location, in addition to determining whether users are 'nearby'.

- Calendar and address book – Mobile devices store calendar information, however access typically requires communication to a server (e.g. via Google Calendar API [Google Calendar, 2009]). APIs are however readily available to local address books, which could be used to provide location information in conjunction with geocoding. For example, GPS and geocoding may identify the current location as 123 First Street, which may map to Client X in the address book.

- Network Availability – Mobile device APIs provide information about the device's capabilities such as whether WiFi or cellular network access is available [Apple AppProgGuide, 2009].

- Time of day and timezone – Mobile device APIs provide easy access to the device's clock and time zone information [Apple AppProgGuide, 2009].

- Local storage – User preferences can be entered and saved onto the device using HTML 5 standard storage (SQLite), or other device specific user preference storage [Apple AppProgGuide, 2009]. This would assist with the automatic population of data entry fields.

There are continual improvements in APIs, and so this will need to be monitored during the project. In addition, the overall design of the proposed system should not limit the number of inputs into the system. This is also in line with the need for flexible 'user driven' data structures.

2.3.2 Design considerations

There are a number of issues that would affect the development of the application client including battery life, limited performance, device specificity of native applications, minimal availability of libraries, '30 second use case' and limited screen real estate:

1. Battery life – There is limited battery life in mobile devices. As such, use of power intensive features such as network access, location determination, and CPU usage should be minimised on the client [Apple AppProgGuide, 2009][Google Seamlessness, 2009]. Complex logic should be performed on the server.

2. Limited performance – CPU and available memory on mobile devices is also limited [Google Seamlessness, 2009]. As such, processor and memory intensive operations should be done on the server.

3. Device specificity of native applications - Mobile enabled web pages can be developed to run on different devices, however only native applications can make full use of mobile device inputs [Apple AppProgGuide, 2009]. As such, for CWTN, a specific platform should be targeted, with the architecture to support expansion to different devices.

4. Minimal availability of libraries – Although operating systems are now available on many mobile devices, there are typically less libraries available than for standard operating systems [Apple AppProgGuide, 2009]. This need to be taken into account

when developing the interface specification between the mobile application and the server (e.g. simplified XML structures or JSON may be required to minimise complex processing that needs to be developed for the device).

5. '30 second' use case – Mobile devices are designed for quick tasks (i.e. to be used in 30 seconds or less). This is especially the case for the target users of CWTN that have minimal time to enter their current status information [Apple HIG, 2009]. As such, the user interface needs to have minimal data entry and support quick navigation through application features.

6. Limited screen real estate and usability consideration – Mobile devices have small screens and as such vendors have developed specific navigation standards to suit [Apple HIG, 2009]. After selection of the target mobile device platform, there is a need to review these standards, and ensure that the proposed client application design meets the standards. This assists with above design consideration for '30 second' use case.

2.4 Summary

In this chapter, key common features of a number of public and private SNSs were discussed, along with potential issues for deployment. Relevant features of the Intuitive Network Application framework-based system similar to CWTN were also presented. A significant outcome of the review presented in this chapter is that CWTN does target a niche area not reached by other systems as yet. Learnings from similar systems will have an impact on the requirements for CWTN, which are presented in the next chapter.

3 Requirements and System Analysis

This chapter presents the requirements analysis for CWTN, and the resulting system analysis to identify the high-level system architecture, application data entities and logic. This analysis is used in the selection of suitable technologies for the project.

3.1 Guiding principles and functional requirements

Based on the literature review presented in the previous chapter, a number of guiding principles for the design of the system were identified. These can be broken up into 'system usage' and 'enterprise-related' principles. These principles build on the fundamental aims of CWTN presented in Section 1.1 Project Aims.

For the resulting detailed list of functional and non-functional requirements that are used for formal scoping of the development phase, please refer to Appendix A.

3.1.1 System usage principles

These principles relate to the functionality and usability of the system from the end-user's perspective.

1. CWTN must provide a user-friendly interface and require minimal data entry to update a person's status. This would be a typical aim for any system; however it has been identified as critical to success of an SNS. One feature CWTN should support is entering current and future status information at a point in time. This allows sensible 'default' values to be used for future status information in case the user does not have time to update CWTN.

2. Where possible, CWTN should support automatic data entry by making use of previous usage history and available information from the user's mobile device. Smartphones, which are a major target for CWTN, support many useful features for the system such as GPS location detection, reverse geocoding, address books, and Internet access. This also works in conjunction with the first principle.

3. The CWTN system architecture should support multiple mobile device platforms. There are a number of popular platforms which have similar features, but implementation would need to be device specific. As such, one platform should be targeted initially with the system architecture supporting extension to other platforms in the future. The prototype system would be developed to support the Apple iPhone platform primarily due to the availability of physical devices to the development team.

4. System functionality must be as flexible as possible, supporting user configuration in order to provide a personalised experience. This allows CWTN to adapt to changing or unforeseen user requirements. In particular this affects the CWTN data structures, and the logic to determine the 'best' means of communication.

   While the prototype may not initially be completely configurable due to development time constraints, the design should support refactoring of data structures and service interfaces in the future to improve flexibility of CWTN. This can be done by identifying and documenting where flexibility may be added as part of initial design.

In terms of the application logic, one possible solution is to use rules or reasoning engines which are designed to support development and deployment of logic independent of the main application deployment. This is discussed further in the next chapter.

3.1.2 Enterprise-related principles

The investigation into private SNSs identified a number of principles that CWTN must follow in order to be considered for deployment in an enterprise environment:

1. The CWTN system architecture should support integration into an enterprise environment. This includes being able to make use of existing infrastructure in the future, such as Enterprise Resource Planning (ERP) systems to retrieve organisational structure data. In addition, the system should expose services that can be used by other systems through SOA-compatible interfaces. This allows other systems to make use of data stored in CWTN, and also allows CWTN to be plugged into existing portal or SNS software in the future.

2. CWTN should operate on an enterprise-grade software platform. This is important for maintainability of the system as personnel with appropriate skill sets are readily available. It also addresses some security concerns as these platforms typically have best practices that cover access and data security. Java Enterprise Edition would be used given the development team's familiarity with the platform.

3. CWTN data and user authentication details must be protected from access by unauthorised parties so as to address any enterprise security concerns. Secure channels and authentication would need to be used for communications.

3.2 High-level system architecture

Based on the guiding principles presented above, a high-level system architecture was devised in order to identify components of the system that need to be developed. This is presented below:

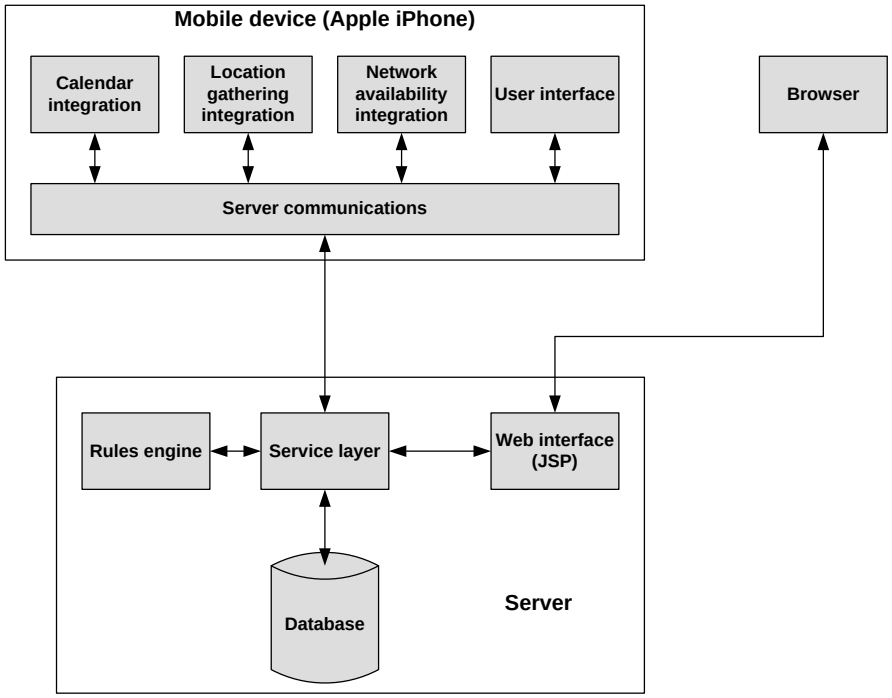


Figure 1 Diagram showing the major components of the CWTN system

CWTN is broken up into a client component (mobile device and web interface/browser), and a server component which would run on the Java Enterprise Edition platform.

On the mobile device the 'User interface', 'Network availability integration', 'Location gathering integration' and 'Calendar integration' modules provide inputs into status information data entry. Additional modules can be created as more inputs become available over time. The 'User interface' module supports manual data entry and triggering of other modules as required. The 'Server communications' module is responsible for sending and retrieving status information to the server.

The 'Browser' and 'Web interface' modules represent a standard desktop PC interface into CWTN.

The primary interface into CWTN server component is through the 'Service layer' module. This is responsible for storing status information into the database, and delegating status queries to the 'Rules engine' which contains the application logic to process status information. The 'Service layer' module would need to be SOA-enabled to support integration into an enterprise environment. Also, the separation between the 'Rules engine' and 'Service layer' allows changes to application logic to be isolated and hence implemented with minimal impact on the rest of the system.

With the initial mobile platform selected, suitable technologies for the rules engine, service layer/client-server communications and the overall server platform need to be identified. These are presented in subsequent chapters of this report.

3.3 Key application data entities

An initial version of the application data model was developed to assist with application logic development and rules engine evaluation. A class diagram showing this model is presented in Figure 2. The key data elements were identified as *Users*, *Activities,* and *Questions*.

A *User* is a user of CWTN, and has relationships to other users (*UserRelationship*) such as User A being a manager of User B. A number of *UserRelationshipTypes* would be configured with a priority defined so that the importance of users can be identified.

Each user also has a list of other users that they are interested in, represented by *UsersOfInterest*. The status information for users of interest could be displayed on the initial screen of CWTN, similar to a 'friends' list of SNSs.

A user also has a number of *UserCommunicationsMethods* which contain information about how a user can be contacted. For example, many users may have a mobile phone and some also have a pager. Each *UserCommunicationsMethod* has a *CommunicationsType* which allows different means of communication to be ranked in terms of preference when evaluated by the application logic.

At a given point in time, a user is involved in an *Activity*, such as a meeting. An *Activity* has a *Location*, other participants (*ActivityParticipants)*, a type (*ActivityType*) and a level of importance (*Urgency*). The user's *Status* is defined by the *Activity* that they are involved in. The *Status* may also have a list of *AvailableCommunicationsTypes* which restrict how a user can be contacted. For example, while overseas a user may not have mobile phone access but could have wireless Internet access.

A 'questioner' would query CWTN to determine the best means of communication to a 'target' user. Both the questioner and target are *Users* of CWTN. The query (represented by a *Question*) would need to include the *Urgency* of the question and expected length of the question in order to match and compare to the target's activities. The answer to the *Question* would be a list of *CommunicationTypes* that could be used in order of preference (e.g. phone first, followed by email).

The data model described above is designed to be minimal and flexible. For example, values for *ActivityType*, *Urgency*, and *UserRelationshipType* are configurable. There is further scope to increase flexibility, such as providing per-user configuration for these variables.

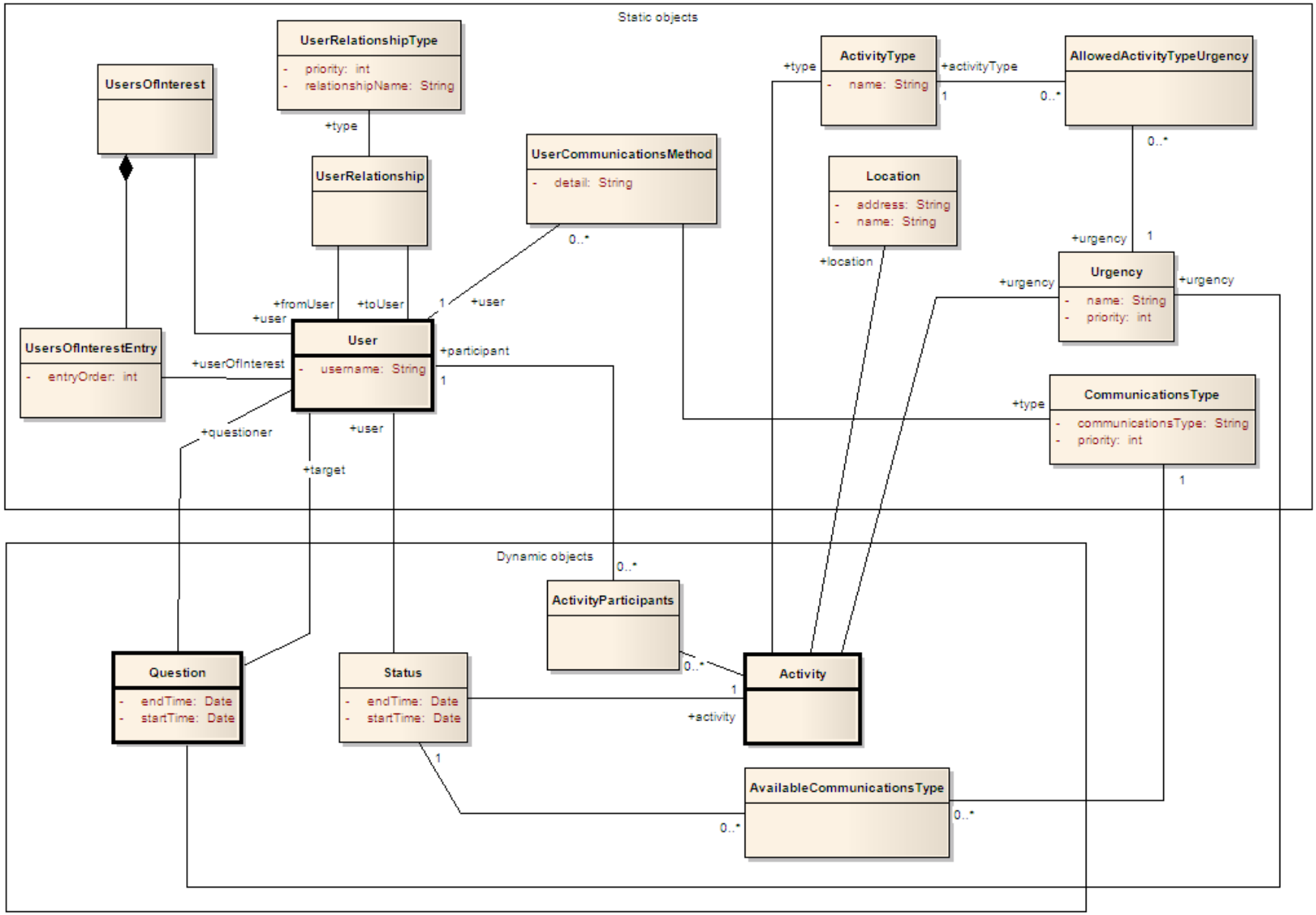


Figure 2 Initial application data model. Key data entities are shown in bold.

3.4 Application logic

The application logic to store and query status information can be divided into three groups: (1) population of *Activity* data or 'facts', (2) determining available means of communications between two users, and (3) ordering of the means of communications.

3.4.1 Populating Activity facts

The logic for populating *Activity* facts in CWTN would be specific to the source of information. Some initial examples of this type of logic are as follows:

- Pop-1: *Status* times and timezones can be identified from the mobile device settings.
- Pop-2: Populate a 'meeting' *Activity* based on calendar entry. Calendar entries can be used to create *Activities* of type 'meeting', with meeting participants defined in the entry set as *ActivityParticipants*. Note that this logic can be used to populate current and future *Activity* entries so that the user does not need to constantly update their status information.
- Pop-3: Create an 'idle' *Activity* when there are no calendar entries.
- Pop-4: Populate the *Location* of an *Activity* based on a calendar entry if one exists. This avoids the need to use battery-consuming GPS functionality on devices, and allows location information to be populated for future activities.
- Pop-5: Populate the *Location* of the current *Activity* based on the mobile device's location obtained through GPS and then performing reverse geocoding to find an address. The address can be mapped to a known location which is stored in the user's address book.
- Pop-6: Populate *AvailableCommunicationsType* based on network availability recorded by the device. If the mobile device detects the lack of phone network access, then 'phone' communications would not be available. If the mobile device detects slow Internet network access, then 'email', 'instant messaging', and other network dependent means of communications would be unavailable.

Additional logic can be developed as more device capabilities are used with CWTN.

3.4.2 Determining available means of communication

The initial rules for this group are as follows:

- Comms-1 – Two users have a 'matching location' at a point in time if both are in the same *Location*.
- Comms-2 – Two users can communicate 'in person' if they have a 'matching location' and the target is 'free to talk'. It is assumed that the questioner is 'free to talk'.
- Comms-3 – Two users can communicate using the 'phone' if both have 'phone' communications available and the target is 'free to talk'.

- Comms-4 – Two users can communicate using ‘instant messaging’ if both have ‘network’ communications available and the target is ‘free to talk’.
- Comms-5 – Two users can communicate using ‘email’ if the questioner has ‘network’ communications available.
- Comms-6 – Two users can communicate using a ‘noticeboard’ if the questioner has ‘network’ communications available.
- Comms-7 – A user is ‘free to talk’ if he/she is not involved in an *Activity* (i.e. no *Activity* recorded) or the *Activity* is of type ‘idle’.
- Comms-8 – A user is also ‘free to talk’ if he/she is involved in an *Activity* but the *Question* is of higher urgency. This is determined by the priority values of the corresponding *Urgency* objects.
- Comms-9 – A user is also ‘free to talk’ if the user is involved in an *Activity* but the questioner is more important than all of the other *ActivityParticipants*. Importance is determined by the priority assigned to *UserRelationshipType.*

It can be seen that there is a hierarchy of rules. The result of Comms-1 is used by rule Comms-2. Also, the combined results of Comms-7, Comms-8 and Comms-9 (i.e. determining whether a user is ‘free to talk’) are used by rules Comms-3 and Comms-4. This hierarchy of rules may allow code reuse or use of subgoals.

3.4.3 Ordering means of communications

Means of communications are prioritised (e.g. phone conversations are preferred over email). After running the rules from the previous section, ordering of the results would need to be performed to identify ‘best’ means of communication. This can be done by sorting the results on *priority* value configured in *CommunicationsType*.

3.5 Summary

In this chapter, the fundamental aims of the project were extended into principles and requirements for CWTN based on review of similar systems. The resulting high level system architecture was presented identifying the various system components for which technology evaluation and selection are described in the following chapters. An initial data model and application logic were also presented, which were used during the technology evaluation process presented in the following chapters.

4 Rules Engine Evaluation and Selection

One of the main aims of CWTN is to provide logic for determining the best means of communications between two users. This complex logic needs to be as flexible as possible in the longer term to allow the system to be finetuned and adapt to changing needs. One possible solution is to use a rules or reasoning engine to implement the logic, as they are renowned for systems where logic needs to be flexible [Olivieri, 2008]. An evaluation of rules engines with respect to CWTN's application logic is presented in this chapter.

4.1 Overview of rules/reasoning engines

Rules or reasoning engines store rules that apply to a knowledge base containing facts (e.g. *Activity* and *Status* objects in the CWTN data model). When the engine receives a query with a goal to achieve, it would apply rules as necessary to the knowledge base to determine an outcome (i.e. whether goal has been achieved).

Rules/reasoning engine are best used when:

- There is complex logic (e.g. many nested if-then-else statements in code). This logic may be more easily described in rule language or description logic and hence this allows better maintenance [Olivieri, 2008][Wiseman, 2006].
- There is a need to externalise logic from the application code. This allows rules to be developed and maintained by non-programmers. In addition, it allows rules to change and be deployed without recompiling the base application. This is suitable for systems where business rules need to be flexible [Olivieri, 2008][Wiseman, 2006].

Both of the above cases apply to CWTN, especially for the logic required to determine available means of communication based on status information. In addition, the application logic needs to be flexible in line with the 'user driven' requirement for SNS software. As such, a rules engine would be suitable for this application.

For CWTN, facts would be built from mobile device inputs and rules would be defined the best means of communication for a person based on various inputs, as well as to output human readable status information.

4.1.1 Types of rules/reasoning engines

Grosof and Dean [2006], and Bry and Marchiori [2005] classify rule engines into four general types: (1) Database views, (2) Inference engines, (3) Production rules, and (4) Event-Condition-Action rules.

Database views incorporate logic into SQL SELECT statements which is suitable for simple logic. The logic for this application is quite complex, and would involve combining information from multiple tables. Developing and maintaining the rules in SQL would be a major challenge, and hence this approach was not considered further.

Inference engines make use of logic programming with rules written in clausal logic form (i.e. (head of rule) 'applies if' (condition is true)), and typically support backward chaining [9].

[9] Forward and backward chaining are discussed in the next section.

Production rules make use of rules written in 'If (condition)-Then (action)' form. These typically support forward chaining, although some engines support both forward and backward chaining.

Event-Condition-Action rule engines support 'On (event) – if (condition) – then (action)' logic. These are a special type of engine used to update/modify data or perform actions upon certain conditions (e.g. for business activity monitoring or workflow systems) and so are not suitable for this application [Alferes et al., 2006].

4.1.2 Backward chaining versus forward chaining

Rule/reasoning engines can use forward chaining, backward chaining or a combination of both to determine the outcome. The best method to use is dependent on the rules required, and would determine the possible rule/reasoning engines that can be used (i.e. needed to be evaluated).

With forward chaining, rules are applied to facts to generate more facts until the goal that is specified in the query is achieved. Forward chaining performs better where there is high fan in (i.e. many conditions and fewer facts) [Etzion, 2008][Vincent, 2008][Cook, 2008]. Also, if caching of generated facts is performed, this can speed up future queries. A pattern matching algorithm such as Rete or a derivative is used to search for suitable rules to apply to facts. Cycles can also be built into the rules to cause re-running (or re-firing) of other rules over modified data, thereby supporting recursive logic. This however could increase CPU and memory consumption. Conflict resolution is also important to avoid multiple output data sets based on a single input, and is typically based on priorities specified for rules [Wikipedia: Production System, 2009].

With backward chaining, the engine will work backwards from the goal and determine rules that can be applied, and the facts required to achieve the goal based on the rules. Backward chaining generates subgoals that may not have explicitly been intended and hence allows deduction [Haley, 2008]. This may be useful to reduce the number of rules required. Also, where there is high fan out (i.e. few conditions in rules but many facts), backward chaining is more appropriate [Cook, 2008].

4.1.3 Future trends in Rule/Reasoning Engines

As previously mentioned in Section 2.1.3, there is a trend towards storing information using Semantic Web technologies such as RDF and OWL-based ontologies. Reasoning engines are also being developed to run on RDF-stored data [Bry and Marchiori2, 2005]. These use existing engines (e.g. SweetProlog translates the information in order to run on a Prolog engine [Laera et al., 2004]), as well as newly developed engines (e.g. Pellet which is a description logic reasoner that can work directly with OWL [Sirin et al., 2007]). In addition, the definition of rules is being standardised into XML format (RuleML) [Grosof and Dean, 2006].

For the purpose of CWTN, these may be considered at a later stage as they are still in development. However, the design of CWTN should include an upgrade path to allow future research based on the prototype.

4.2 Evaluation method

Four rule engines were evaluated based on a mix of technical requirements and considerations for enterprise deployment. The evaluation process included analysing

documentation and example code. SWI-Prolog[10] and Prova[11] were evaluated as examples of inference engines, and JBoss Drools[12] and JESS[13] were evaluated as examples of production rule engines. As a test of ease of development and suitability of different types of rules engines, test applications implementing some of the logic presented in Section 3.4.2 "Determining available means of communication" were written for SWI-Prolog and JBoss Drools. For interested readers, the rule code can be found in Appendix C Sample Prolog Code implementing application logic and Appendix D Sample JBoss Drools rules implementing application logic.

4.3 Evaluation criteria

The evaluation criteria there were used were as followed (presented in order of importance):

1. Support for rules as presented in Section 3.4.2. In particular, the following features would be of use:
    a. Support deduction of outcomes where rules do not define the complete set of inputs and outcomes.
    b. Support prioritisation of rules to handle conflicts.
    c. Support hierarchy of rules so as to minimise code duplication between rules.
2. Ease of integration with Java. The rule engine would need to run and integrate with a Java Enterprise Edition-based server platform. This includes accessing the rules engine from Java, as well as supporting Java objects as data. It may also be necessary to support 'procedural attachment' (i.e. allow the rules engine to call Java methods to perform tasks) in order to support more complex logic requiring data lookups.
3. Record of stable usage in enterprise environments. While CWTN is a prototype, the intent is to deploy into an enterprise environment in the future, which requires proof of stability and robustness i.e. no known inherent faults that could cause crashing, infinite loops or excessive CPU or memory footprint. In addition, there needs to be administration documentation that allows the rules engine to be maintained and troubleshooting to be performed.
4. Availability of an integrated development environment (IDE) that can deploy rules without recompilation of the main application. This is necessary as rules may need to be adjusted or fine tuned during development, and having this feature would reduce development time. Online deployment of rules may not be initially used in the prototype; however it is important for ongoing maintenance if the prototype is transitioned into a fully-functional product.
5. Licensing cost for enterprise deployment. While enterprises are more likely to use paid products, costs for a prototype deployment may be difficult to justify. Therefore only open source or free to use rule engines at least for evaluation purposes were considered.

[10] See www.swi-prolog.org
[11] See www.prova.ws
[12] See www.jboss.org/drools
[13] See www.jessrules.com

6. Future proofing. The rule engine ideally should support features such as standardised languages (e.g. RuleML) and Semantic Web data structures. While not important at this point in time, having these features would allow evolution of CWTN for future research purposes.

Note that memory and CPU performance of the rule engines was not explicitly included in the above evaluation criteria. While performance is important for systems, in this particular case it is anticipated that there would not be a high volume of transactions so other criteria were deemed to be of higher priority. A brief theoretical comparison of CPU and memory performance between forward and backward chaining is presented in Section 4.7.2 based on the development of the test applications.

4.4 Product Summaries and Evaluation

The table below shows a comparison of the four rules engines against the evaluation criteria. Each product was ranked for each criteria as either poor, okay, good or very good with a justification provided.

More detailed descriptions of product key differentiators are discussed in the following subsection.

Criteria	SWI-Prolog	Prova	JBoss Drools	JESS
1. Support application logic	**Good** – Backward chaining is suited to the rules. Some logic such as date comparisons need extra code.	**Very Good** – As per SWI-Prolog comment, and supports inbuilt Java features such as date comparisons	**Good** – Rules can be implemented, but care is required to prioritise rules or duplicate rule conditions.	**Good** – As per Drools comment. Backward chaining rules can be written, but the same performance issues with forward chaining would remain.
2. Integration with Java	**Poor** – Requires JNI interface, and explicit conversion of facts into Prolog.	**Very Good** – Product is Java based.	**Very Good** – Product is Java based.	**Very Good** – Product is Java based.
3. Usage in enterprise environment	**Okay** – Prolog has been used in some enterprise software, however no commercial support avenue available for this particular Prolog edition.	**Poor** – Relatively poor documentation, no known commercial support avenue.	**Very Good** – Good documentation, commercial support is also available from commercial company JBoss.	**Okay** – Known usage of JESS in enterprise products, however no commercial support avenue available (it is not a true commercial product).
4. Availability of IDE	**Okay** – Plugins available for text editors, but no strong IDE.	**Poor** – No IDE available.	**Very Good** – Eclipse-based IDE and web based rules management system available.	**Good** – Eclipse-based IDE available.
5. Licensing cost	**Good** – Open source.	**Good** – Open source.	**Good** – Open source.	**Okay** – Free for academic use, but payment required for commercial use.

Criteria	SWI-Prolog	Prova	JBoss Drools	JESS
6. Future proofing	**Good** – Modules available for Semantic Web technologies, used in Semantic Web research.	**Good** – Support for Semantic Web technologies, used in Semantic Web research.	**Poor** – No explicit support for RuleML or Semantic Web.	**Good** – Used in Semantic Web reasoning engines.

Table 1 Comparison of rule engines with respect to the evaluation criteria

4.4.1 SWI-Prolog

SWI-Prolog[10] is a popular Prolog implementation, and its backward-chaining approach was well suited to the rules to be used by CWTN. In particular, a hierarchy of rules could be easily achieved.

However, its major downfalls were relatively poor integration with Java compared to other rule engines, and lack of support for enterprise users. The integration with Java requires conversion of data in Java objects into a form suitable for Prolog thus losing type safety [14]. It also uses Java Native Interface, which limits deployment to platforms with SWI-Prolog libraries available, which does not include the targeted Google App Engine platform discussed later.

4.4.2 Prova

Prova[11] is a pure Java implementation of a backward-chaining engine with a language based on a form of Prolog. One key differentiator is its ability to extract facts directly out of JDBC-capable relational databases [Kozlenkov, 2006]. However, object-relational mapping tools such as Hibernate[15] are typically used in enterprise applications rather than direct JDBC, meaning two sets of database access code may need to be maintained if this feature of Prova is used. Since communications with the database are typically slow it may be preferable to query the database once at the start and populate facts in memory rather than querying the database upon each rule execution. Finally, Prova lacks documentation, an IDE and availability of support.

4.4.3 JBoss Drools

JBoss Drools[12] is an open source native Java-based forward-chaining production rule engine. It has a sophisticated IDE and extensive documentation, along with a web-based rules management engine for online rules deployment. Support contracts are available through the owning company JBoss. While it appeared to be a front-runner, development of the test application revealed that forward-chaining rules engines may not be suitable for this application as it required either a lot of code duplication between rules, or care to be taken to ensure correct rule execution order. It also does not have explicit support for Semantic Web technologies or standardised rule languages as yet.

4.4.4 JESS

JESS[13] is production rules engine developed by Sandia Laboratories and is considered a leading Java-based rules engine. It supports both forward and backward chaining through the use of appropriate rule annotation; however Jess' backward chaining functionality actually rewrites the rules into a suitable forward chaining format [Friedman-Hill, 2008]. It also

[14] See "High-level interface", http://www.swi-prolog.org/packages/jpl/java_api/high-level_interface.html
[15] See www.hibernate.org

provides a sophisticated IDE, and has been used in Semantic Web applications (e.g. Jena) and has support for RuleML. While it is free for academic use there is a licence fee for enterprise use, making JESS less desirable for this application. However, it could be considered if further Semantic Web-based enhancements to CWTN are added.

4.5 Rule Engine Selection Result

No rule engine stood out as the clear winner. As a result, one key outcome was the need for a modular system architecture to allow the rule engine component to be easily replaced in future.

For the purposes of the prototype, Drools would be used as it was the closest match to the criteria, and some rules have already been developed as part of the evaluation.

4.6 Discussion of other findings

4.6.1 Suitability of rule engines to the three groups of application logic

Only the logic for determining available means of communication would gain significant benefits from the rules engine as it has the most flexibility, and hence should be implemented as rules. Logic to populate *Activity* facts is highly dependent on the source of the information and may be influence the user interface, so should be implemented in the client application rather than in the rule engine on the server. Ordering the means of communications can be performed via Java coding using simple comparator logic, without the need for complex rules.

4.6.2 Theoretical CPU and Memory Performance Analysis

A backward chaining approach should be more appropriate for this application as there are relatively few rules compared to a potentially large number of facts [Cook, 2008]. This was borne out during testing. The test code for JBoss Drools was written to use a hierarchy of rules. One set of rules was used to populate a 'free to talk' flag, before another set of rules identified available means of communications partially based on that flag. This requires use of cycles and re-firing of rules. Note that only a result list was created as part of the rules – there was no significant increase in knowledge base during execution of the rules.

In terms of CPU usage, the use of cycles would increase CPU usage over backward chaining.

In terms of memory consumption, forward chaining engines would need to create a Rete network to support the rules execution, with additional objects created as the rules are run and matching facts are identified. This can quite significant for complex rules and would be larger than for backward chaining engines.

The JESS manual [Friedman-Hill, 2008] describes possible memory consumption improvements to rules by ensuring most matching patterns are placed at the top of the rules. Also, by removing the need for cycles it should be possible to reduce CPU consumption. This approach is also described in by Dinoff et al. [2007] as their Vortex rules engine does not support cycles. However, this creates code duplication between rules.

One possible approach to make better use of the forward chaining engines is to run the rules at regular intervals or upon status updates to pre-populate common status query results that can be cached. A history of questions can be recorded to be rerun. When one of these

common questions is asked of CWTN, the result can be returned quickly without running the rules. This approach could be investigated if performance is deemed to be unacceptable.

4.7 Summary

One of the main aims of CWTN is to provide logic for determining the best means of communications between two users. This complex logic needs to be as flexible as possible in the longer term to allow the system to be finetuned and adapt to changing needs. Rules engines are ideal candidates for this type of logic as they support out-of-band maintenance and support of the code with respect to the rest of the application. A number of rule engines were evaluated based on technical and enterprise considerations. While none suited all criteria, Drools was selected as the closest match and the evaluation code could be reused for the prototype. However, the system architecture should allow for easy replacement of the rule engine module.

5 Selection of Client-Server Communications Technologies

CWTN is in essence a client-server application, and there is a choice of protocol (e.g. Web Services) and data exchange format (e.g. XML). Suitable selection of client-server communications technologies is important as it has a potentially great impact on the development time, reusability of server-side code and even mobile device battery life. This section describes the considerations for technology selection, comparison of technologies and the ultimate decisions.

5.1 Selection considerations

The considerations for selecting the client-server communication technologies are as follows:

1. Technologies should be standards-based or at least commonly used (i.e. 'defacto' standard). The server-side component of CWTN needs to fit into a Service Oriented Architecture environment. This would allow multiple clients to be developed over time, as well as supporting integration with other systems such as private SNSs. Standards-based protocols and data exchange formats are essential for this.

2. Development of client-side code must be simple and require minimal libraries. Mobile devices have limited memory and in-built libraries. For example, the Apple iPhone SDK only has SAX parsing libraries for XML in-built. The client application code must fit into this streamlined environment. Ease of development on the server-side is also a consideration, but is less of a concern.

3. The size of messages sent between client and server are likely to be small (less than 1KB) as these would only contain status updates or queries. As such, efficiency of data transfer by the protocol or format is not a major concern; however network bandwidth usage should be still kept to a minimum to reduce network utilisation by the mobile device.

4. Processing of the protocol and format on the client should require minimal CPU power and memory consumption. Both are major limitations of mobile devices, and also high CPU usage negatively impacts on battery life.

5. The communications protocol needs to support secure transmission of data, as user login information may be exchanged.

5.2 Communications Protocols

Representational State Transfer (REST) and Web Services are now considered to be protocols of choice for use in SOA environments [Bruno, 2007].[16]

Web Services are an established standard for machine-independent communications. Request and response payloads, including error messages, are stored in SOAP envelopes in XML. The processing of the envelopes and associated protocol semantics adds significant overhead to communications. While Web Services can be easily developed and deployed on the server-side by using JAX-WS, there are no readily available client-side libraries for full Web

[16] XML-RPC was the precursor standard to Web Services, but has since fallen out of favour. Its request and response payloads are simpler than for Web Services, but still more complex than REST. As such, it offers little benefit over both REST and Web Services for this application. For similar reasons JSON-RPC (which is the JSON equivalent of XML-RPC) was also not considered.

Service communications, or parsing and producing Web Service payloads for the iPhone [17] (or other newer platforms such as Google Android).

REST was designed to be simpler with minimal overhead. Requests are passed to the server as either HTTP URI paths or query request parameters. This eliminates the need for specific request payload generation. Reponses are passed via HTTP response bodies in either XML or JSON formats. Error messages can be passed back to the client through HTTP response codes and body text (e.g. a simple text error message, or more complex XML data). One major benefit over Web Services is that it only requires a basic HTTP client (which the iPhone SDK has inbuilt [17]) and a library to parse the data exchange format; no sophisticated libraries are required.

REST implementation in Java Enterprise Edition is now standardised as JAX-RS, and there are a number of libraries available [Little, 2008].

A comparison of these two based on the considerations described in the previous section is shown in the following table:

Criteria	Web Services	REST
1. Standards-based	Yes	Yes
2. Complexity of client-side and server-side code	Complex client-side code. No libraries available for full Web Service communications or parsing Simple server-side code using JAX-WS	Simple client-side code. Only requires HTTP client and XML/JSON parser library Simple server-side code using JAX-RS
3. Relative network bandwidth usage	Higher due to envelope overhead	Lower
4. Processing and memory power required on client	Higher due to envelope overhead	Low
5. Secure transmission	Possible via WS-Security	Possible cia HTTPS authentication and encryption

Table 2 Comparison of Web Services and REST

Given the simplicity of implementation of REST on both client and server, it is clear that REST would be most suitable for this application. The RESTlet Java library[18] would be used as it has inbuilt support for the Google App Engine, which is discussed in the next chapter.

5.3 Data exchange formats

There are two main options for standard or defacto standard data exchange formats:

1. Extensible Markup Language (XML)
2. Javascript Object Notation (JSON)

XML is a W3C standard designed to be processed by any system or language, and hence is suitable for use in SOA environments. It also supports strict data types and validation through XML Schema. Also, XML is notorious for its verbose structure [Marinescu and Tilkov, 2006].

[17] See Apple iPhone Forum, "iPhone SDK: Web Services?" http://discussions.apple.com/thread.jspa?threadID=1435726&tstart=0, February 2009.
[18] See www.restlet.org

JSON is a lightweight format consisting of data stored as Javascript code. This allows JSON to be immediately translatable into Javascript objects[19]. There is currently no strict standard for JSON, and there is no equivalent to XML schema validation.

For the iPhone, small open source libraries named TouchXML[20] and TouchJSON exist to parse XML and JSON respectively. In either case, custom parsing code should not need to be written.

Given the simplicity of data structures in this application, the sophistication of XML would not be required. Hence JSON will be used as the data exchange format.

As part of JAX-RS standard, XML and JSON can be supported easily on the server through minor additions in annotations. Therefore, when integration is developed between the server component and other systems, XML can be 'switched on' as required.

5.4 Summary

In this section, the considerations for client-server communications technology selection were discussed. After evaluation, REST will be used as the communications protocol, with JSON used initially as the data exchange format between server and mobile device client.

[19] This would be a major benefit for Web applications (i.e. HTML/Javascript) that may be written to use the existing REST service in the future.
[20] See code.google.com/p/touchcode

6 Server Platform Selection

The main consideration for selecting a suitable server platform is the need to provide secure access to the server application (i.e. rules engine and database) over the Internet. Mobile users need to be able to access the system from outside the office. This is a major challenge especially for a prototype system where cost is a factor.

It is possible to deploy the server application onto an internal system and require the user to set up a Virtual Private Network (VPN) connection in order to connect to the server. However, adding steps to using the application such as setting up a VPN connection or even manually entering in authentication details significantly impacts system usability, and could limit uptake and success of the prototype

As such, the preferred option would be to deploy onto an Internet-facing host. This can be achieved in two ways – use existing freely available infrastructure (such as Google App Engine), or seek to deploy on a standalone application server owned by the enterprise. These options are discussed in forthcoming subsections.

Also, another consideration is the need to support Java Enterprise Edition (JEE). As the system is targeted at enterprises, an enterprise-grade software platform is required. JEE is considered one such platform and given the author's familiarity with JEE, it is the preferred software platform.

6.1.1 Google App Engine

Google App Engine (GAE) [21] is a publically accessible application and database server deployed on Google infrastructure ('cloud'), which now supports Java Enterprise Edition applications. Google provides free access to this, with an associated limitation on bandwidth and CPU usage. There is also support for integration with Google's authentication infrastructure.

GAE does however impose some restrictions on the application code. These are listed below:

1. GAE does not support all standard Java libraries. This could be a potential issue during development in case specific add-on libraries that are used may not work correctly. As risk mitigation, the RESTlet third party library will be used for REST implementation as it has a specific GAE module. The JBoss Drools rule engine is one specific library that remains a concern as there are no documented deployments of it to GAE, however it is compatible with Java Standard Edition and so this should not be a major issue.

2. GAE restricts database access code to using either JPA or JDO based on the JDO reference implementation by DataNucleus. Specifically, this excludes the defacto standard of Hibernate. However, given that both JPA and JDO are standards this should not be a major issue.

3. HTTPS support requires authentication using Google accounts. This means for the prototype system, there is little control over user accounts (e.g. creation of test accounts will require creation of Google accounts). Also, the authentication is through Google's single sign-on web page. Since the client application is intended to be a native application and ideally should automatically log into the system to minimise

[21] See code.google.com/appengine/

the number of user steps, this could be an issue. One possible solution is to use Google's client token API, however this will need to be investigated during development.

6.1.2 Stand-alone application server

As an alternative to GAE, a stand-alone application server such as Sun Glassfish or JBoss Application Server could be used. This has the benefit of allowing full flexibility in usage of libraries, and calls to native code using JNI (e.g. to use SWI-Prolog) could be used. However, this requires access and set up of additional infrastructure which would need to be Internet-accessible; this may be difficult to obtain for a prototype system.

6.2 Summary

For this application, the server will be targeted to be deployed on Google App Engine. This should allow deployment of the prototype for initial testing without needing to establish infrastructure.

At a later stage, secure communications between the cloud and internal enterprise systems can be developed to support integration, or the application can be migrated to an Internet-facing host owned by the server. Given the software would be Java EE compliant and that GAE enforces more restrictions on libraries that can be used compared with the standard, it should be easily portable to other application servers in the future.

It is also important to note that the "Server communications" module on the client application should have a separate authentication sub-module. This is because authentication would be tied to the Google authentication when using GAE, and it the server application is moved elsewhere, the module would need to be replaced.

7 Future Work – Development Phase Scoping and Plan

One of the main purposes of the work described in preceding sections is to feed into the scoping and planning of the development phase for the project in ITEC809, which is described in the following subsections.

7.1 Development Approach

There are two main considerations when planning the next phase of this project. Firstly, the development time possible in ITEC809 is very limited - approximately 3-4 weeks 'full-time equivalent' effort. Therefore it is important that the prototype system be demonstrable as soon as possible. Core features should be implemented first, followed by enhancements.

Secondly, the project is not officially funded and as such may cease and then restart at any time, particularly after ITEC809 is completed. To allow speedy project closedown and effective handover or pickup after project restart, design documentation should be as complete as possible by the end of ITEC809.

In order to cater for the above considerations, an iterative approach will be used for the development phase [Sommerville, 2007]. Each iteration will be run using a waterfall methodology, starting with detailed design, followed by development and then integration testing. Since the iterations are only a small part of the overall project, issues discovered during individual iterations can be fed into subsequent development work.

Using a purely iterative approach may leave some design decisions or considerations for future iterations undocumented at the end of ITEC809. To reduce the risk of this occurring, a high-level system analysis and design stage will occur before the first iteration. This also has the added benefit of ensuring the earlier iterations take into account requirements for later iterations, thereby reducing the amount of rework required.

7.2 Iteration plans

Based on priority of the functional requirements and the need to provide a demonstrable system by the end of ITEC809, four development iterations are proposed:

1. Server core: Various server components will be developed in this iteration including the data model, database access, integration with the rules engine and the REST service for integration with the client.

2. Mobile core: Core mobile application components will be developed in this iteration including integration with the server, and basic user interface for status entry and querying.

3. Mobile enhanced: Additional mobile application components for automatically populating status information from various sources would be developed in this iteration.

4. Web Interface: A basic web interface to the system would be developed to provide a desktop equivalent to the mobile user interface developed in the 'Mobile core' iteration.

The first two iterations are scheduled for completion during ITEC809. The diagram below outlines which system components will be developed in each iteration.

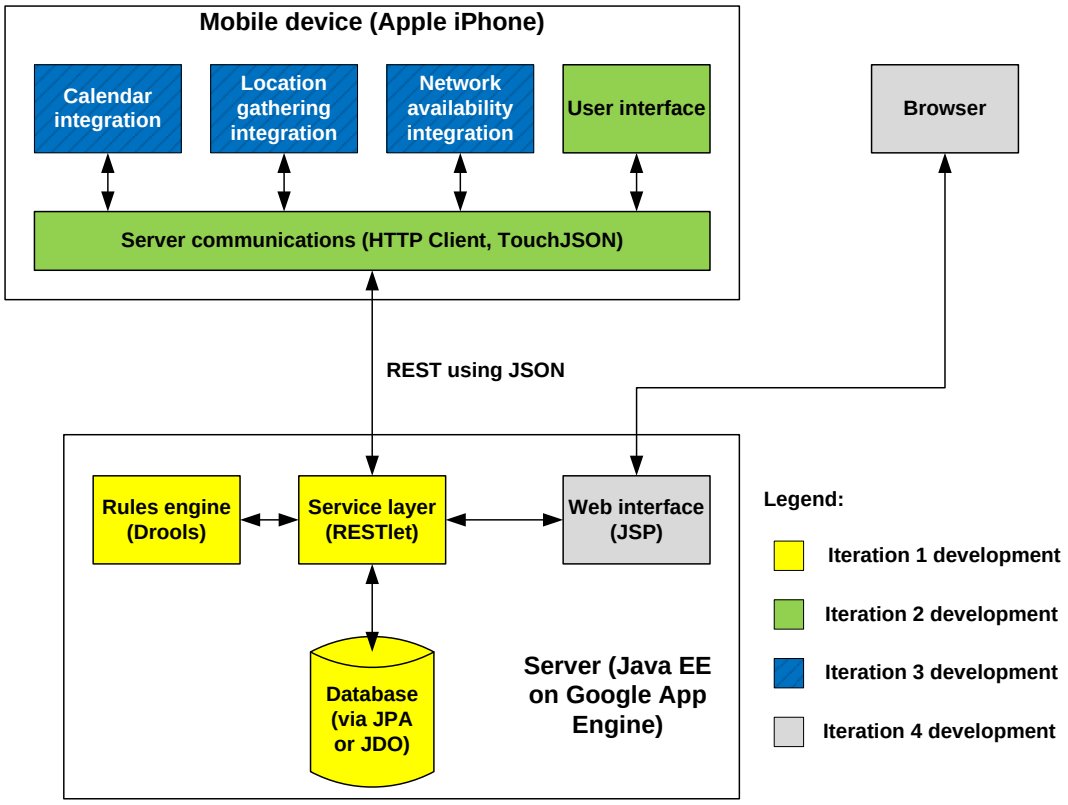


Figure 3 Diagram showing breakdown of high-level system architecture into development iterations

The mapping of iterations to functional requirements based on guidelines from Section 2 can be found in Appendix A. A draft Gantt chart showing that project schedule for the development iterations can be found in Appendix B.

7.3 Addressing deployment issues

In Section 2, four deployment issues were identified from review of SNSs – scale, culture, privacy and security. The following subsections describe how these would be addressed for the development and deployment of the prototype system.

7.3.1 Scale

As previously discussed, without large scale uptake of SNSs the full benefits of these systems cannot be achieved. Applying this principle to CWTN, a potential problem is that if few users enter their status information then fewer users would query the system.

For the use case that CWTN is addressing, namely finding the best means of communications with a team member, only members of the team need to update their status information for the system to be useful. Therefore the critical mass for CWTN is quite small and by demonstrating the usefulness of the system for only a small group, more users can be encouraged to join the system i.e. by addressing cultural hindrances as described in the next section.

Also, if CWTN is integrated with existing SNSs as an ‘add-on’ application it can leverage the scale available from the SNSs and vice versa.

7.3.2 Culture

Cultural hindrances are a major issue for uptake of social networking technology in the enterprise as the target users include a mix of age groups, familiarity with SNSs, and opinions on new technology. In order to address this, the following steps would need to be taken for CWTN:

- The system must be demonstrated to be useful. This can be done through deployment of the prototype with an initial set of carefully selected pilot users. These users would need to be part of a mobile team, which would receive the most benefits from CWTN. A few users should be familiar with SNSs to encourage uptake within the team, however the team itself should be of mixed demography to demonstrate to the wider community that any person can use the system.
- The benefits of the prototype system need to be measured. This includes gathering anonymous statistics on number of status updates and number of queries processed by the system. A survey of pilot users to measure perceived improvements in response time to questions by other users could also be done. By presenting quantifiable evidence of system utility, more users would be encouraged to make use of CWTN.
- The design of the system must minimise manual data entry. Dinoff et al. [2007] reports that even with only basic manual entry, users are unlikely to keep status information up to date. Note that manual data entry would be important to allay privacy concerns (see the next section), so it is important that the interface is easy to use. One possible option is to allow status information to be set initially for the entire day at once. This represents base information that can be optionally updated if the user has time. The system may need to consider that the base information has a lower level of confidence than more recently updated information.
- The broader audience in the enterprise will need to be informed about key aspects of the system, especially how privacy and security are addressed. This may be done through fact sheets, training sessions or other means of communication.
- As part of deployment of the prototype, an incentive system may be set up that rewards usage of the system [Farzan et al., 2008]. For example, it may be publically announced which user has updated their status information the most, or has made the most number of queries.

7.3.3 Privacy

As discussed previously, privacy is not considered as major an issue for private SNSs as it is for public SNSs. However, in order to address any remaining concerns of potential users it is still a requirement for users to initiate the status update process and verify the status information that will be sent to the server.

Bila et al. [2008] reported that many users would be "willing to share personal information with the telecommunications network in trade for more convenience". This same principle can apply to CWTN; by demonstrating the utility of the system through ideas presented in the previous section, privacy may be less of a concern.

7.3.4 Security

Security is a major issue for enterprise systems. This is particularly the case for this system as mobile clients needs access to the server via the Internet.

Client-server communications would be authenticated and secured through HTTPS [Newman and Thomas, 2009: Ch 15]. Proper software development practices need to be followed (e.g. use of authentication before accessing of data). Security related design decisions would need to be documented so as to facilitate approval of system deployment by network security teams in the enterprise.

7.4 Future research after prototype development

There are many possibilities for future research using a completed CWTN prototype system as a base. Some ideas include:

1. Integrating CWTN with an existing SNS such as Facebook or Twitter which have published APIs. This would allow CWTN's interfaces to be fine tuned for further integration into other enterprise environments such as data warehouse reporting systems.

2. Modify CWTN to use Semantic Web techniques to store and reason about the status information (e.g. using OWL-DL). This would allow sophisticated querying and data mining to be performed, such as to measure team member's idle or travel time. Integration with other systems using Semantic Web Services could also be investigated.

7.5 Summary

Based on the analysis work conducted during this phase of the project, the development phase will be divided into four iterations: Server Core, Mobile Core, Mobile Enhanced, and Web Interface. This will allow the high priority functionality to be implemented first in time for ITEC809 delivery. Possible resolutions for the four enterprise SNS deployment issues raised in Chapter 2 were also discussed.

8 Conclusion

The aim of the project is to develop a prototype intelligent status tracking system (Can We Talk Now? or CWTN). This system is targeted towards mobile workers in the enterprise, providing a central store of status information that is automatically populated based on various source such as calendar and location information. Automation of data entry should encourage uptake and hence reduce cultural hindrances to the use of CWTN. By querying the store, other users can identify the most effective means of communicating with the user. CWTN would be considered an add-on to existing SNSs, and so would need to support integration via Service Oriented Architecture technologies.

The project is being conducted in two phases – an initial feasibility study and technology selection phase, followed by a development phase. The outcomes of the first phase are presented in this report. A summary of these outcomes are as follows:

- A literature review was conducted on public and private Social Networking Services. These systems store status information, but none support automatic population of this data. The "Intuitive Network Applications" framework from Bell Labs was found to target a similar problem to CWTN, however from a service provider's perspective. CWTN's aim is to provide a low-cost 'add-on' system to enterprise systems, independent of service providers or carriers. The literature review identified a number of key requirements including the need for flexibility, and the need to address cultural hindrances and security concerns.
- Requirements analysis was performed for CWTN, identifying a high level system architecture comprising of a server component with rules engine integration, and a mobile device client support manual and automated data entry. An initial data model containing key application entities, as well as an initial set of rules for determining the best means of communication were also developed. Given the complexity of these rules and the need for flexibility, a rules engine was considered necessary rather than implementing the logic in code.
- A number of rules engines were evaluated (SWI-Prolog, Prova, JBoss Drools, JESS) based on suitability for the rules and other enterprise related criteria such as cost and supportability. While backward-chaining rules engines suited the rules, JBoss Drools was considered the closest match to all evaluation criteria. The system would need to have a modular architecture to allow the rules engine to be replaced at a later date if performance is deemed unacceptable, or if investigation into Semantic Web technologies is required.
- Client-server communications technologies were investigated, with REST using JSON data format selected as most approach for this application. Only the inbuilt HTTP client and small add-on libraries are required to support these technologies on the target mobile device platform (Apple iPhone).
- The server component of the prototype will be initially targeted at deployment on Google App Engine, which is a freely available and publically accessible Java Enterprise Edition application and database server. This would allow the prototype to be tested and used without implementing infrastructure in an enterprise environment.

The results of this phase will feed into the planning and execution of the development phase. Currently it is planned to have four development iterations, with a basic client-server system demonstrable after iteration 2. The results of the development phase will be presented as part of ITEC809.

9 References

[Alferes et al., 2006]	J. J. Alferes, F Banti, and A Brogi. [2006]. "An Event-Condition-Action Logic Programming Language." *10th European Conference on Logics in Artificial Intelligence.* p.29-42. Springer-Verlag. Liverpool.
[Alleven, 2007]	Monica Alleven. [15 September 2007]. "Social Nets Catch Mobile Users." *WirelessWeek.* Retrieved 15 March 2009 from http://www.wirelessweek.com/Article-Social-Nets-Mobile-Users.aspx
[Apple AppProgGuide, 2009]	Apple. [6 January 2009]. "iPhone Application Programming Guide." *iPhone Reference Library.* Retrieved 18 March 2009 from http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf.
[Apple HIG, 2009]	Apple. [4 March 2009]. "iPhone Human Interface Guidelines." *iPhone Reference Library.* Retrieved 21 March 2009 from http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/ MobileHIG/MobileHIG.pdf
[Bila et al., 2008]	Nilton Bila, Jin Cao, Robert Dinoff, Tin Kam Ho, Richard Hull, Bharat Kumar, Paulo Santos. [2008]. "Mobile User Profile Acquisition Through Network Observables and Explicit User Queries". In Proc. *The Ninth International Conference on Mobile Data Management.* p.98-107. IEEE.
[Bruno, 2007]	Eric J. Bruno. [8 June 2007]. "SOA, Web Services and RESTful systems – A framework for building RESTful systems", *Dr Dobb's Portal*. Retrived 13 April 2009 from http://www.ddj.com/web-development/ 199902676.
[Bry and Marchiori1, 2005]	Francois Bry, and Massimo Marchiori. [2005]. "Ten These on Logic Languages for the Semantic Web." In *Principles and Practice of Semantic Web Reasoning*, by F Fages and S Soliman, p. 42-49. Springer-Verlag. Dagstuhl Castle.
[Bry and Marchiori2, 2005]	Francois Bry, and Massimo Marchiori. [2005]. "Reasoning on the semantic web: beyond ontology languages and reasoners." *The 2nd European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology.* p.317-321. IEEE. London.
[Buffa et al., 2008]	Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander,and Catherine Faron. [2008]. "SweetWiki: A semantic wiki." *Web Semantics: Science, Services and Agents on the World Wide Web.* Vol. 6. p. 84-97. Elsevier
[Casarez et al., 2009: Ch 9]	Vince Casarez, Billy Cripe, Jean Sini, and Philipp Weckerle. [2009]. *Reshaping Your Business with Web 2.0: Using the New Collaborative Technologies to Lead Business Transformation*. Ch 9. McGraw-Hill/Osborne.
[Cook, 2008]	Diane J. Cook. [2008]. "More Forward Chaining vs. Backward Chaining." *Artificial Intelligence Lecture Notes.* Retreived 21 March 2009 from http://www.eecs.wsu.edu/~cook/ai/lectures/l7/node13.html.
[Diaconescu and Wagner, 2007]	Ion-Mircea Diaconescu, and Gerd Wagner. [2007]. "Towards a mapping from Java Vocabulary to RDFS." *9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2007.* p.518-521. IEEE. Timisoara, Romania.
[DiMicco et al., 2008]	Joan DiMicco, David R Millen, Werner Geyer, Casey Dugan, Beth Brownholtz, and Michael Muller. [2008]. "Motivations for Social Networking at Work.". In Proc *Computer Supported Coorperative Work'08.* p.711-720. ACM, San Diego.

[Dinoff et al., 2007]	Robert Dinoff, Tin Kam, Ho, Richard Hull, Bharat Kumar, Daniel Lieuwen, Paulo Santos. [2007]. "Intuitive Network Applications: Learning for Personalized Converged Services Involving Social Networks". *Journal of Computers.* Vol. 2 No. 6. p. 72-84. Academy Publisher.
[Etzion, 2008]	Opher Etzion. [8 March 2008]. "On Forward and Backward CEP." *Event Processing Thinking.* Retrieved 13 March 2009 from http://epthinking.blogspot.com/2008/03/on-forward-and-backward-cep.html
[Farzan et al., 2008]	Rosta Farzan, Joan DiMicco, David R Millen, Beth Brownholtz, Werner Geyer, and Casey Dugan. [2008]. "Results from Deploying a Participation Incentive Mechanism within the Enterprise." In Proc *Conference on Human Factors in Computing Systems.* p.563-572. ACM. Florence, Italy.
[Friedman-Hill, 2008]	Ernest Friedman-Hill. [2008]. *Jess The Rule Engine for the Java Platform*, Sandia National Laboratories. Retrieved 13 April 2009 from http://www.jessrules.com/jess/docs/Jess71p2.pdf.
[Grosof and Dean, 2006]	Benjamin Grosof, and Mike Dean. [11 May 2006]. *ISWC-2006 Tutorial - Semantic Web Rules with Ontologies, and their E-Service Applications.* Retreived 13 March 2009 from http://iswc2006.semanticweb.org/workshop_tutorial/iswc2006-tutorial-BGrosof+MDean-v2.pdf.
[Google Seamlessness, 2009]	Google. [20 March 2009]. "Designing for Seamlessness." *Android Developers.* Retrieved 21 March 2009 from http://developer.android.com/guide/practices/design/seamlessness.html.
[Google Geocoder, 2009]	Google. [20 March 2009]. "Reference: Geocoder." *Android Developers.* Retreived 22 March 2009 from http://developer.android.com/reference/android/location/Geocoder.html
[Google Calendar, 2009]	Google. [2009]. "Google Calendar APIs and Tools.". Retreived 21 March 2009 from http://code.google.com/apis/calendar
[Haley, 2008]	Paul Haley. [11 March 2008]. *Goals and backward chaining using the Rete Algorithm*. Retrieved 13 March 2009 from http://pvhaley.wordpress.com/2008/03/11/goals-and-backward-chaining-using-the-rete-algorithm/
[Kalyanpur et al., 2004]	Aditya Kalyanpur, Daniel Jimenez Pastor, Steve Battle, and Jullian Padget. [2004] "Automatic Mapping of OWL Ontologies into Java." *Software Engineering - Knowledge Engineering.* ACM. Banff, Canad.
[Kozlenkov, 2006]	Alex Kozlenkov. [2006] *PROVA Java Rule Language for Information Integration and Semantic Agents Version 2.0 User's Guide*. Retrieved 20 April 2009 from http://www.prova.ws/etc/provauserguide_2_0_A.pdf
[Laera et al., 2004]	Loredana Laera, Valentina Tamma, Trevor Bench-Capon, and Giovanni Semeraro. [2004]. "SweetProlog: A System to Integrate Ontologies and Rules." *Rules and Rule Markup Languages for the Semantic Web.* p.188-193. Springer-Verlag. Hiroshima.
[Little, 2008]	Mark Little. [1 October 2008] "A Comparison of JAX-RS Implementations". *InfoQ*. Retrieved 10 April 2009 from http://www.infoq.com/news/2008/10/jaxrs-comparison
[Marinescu and Tilkov, 2006]	Floyd Marinescu, Stefan Tilkov. [26 December 2006]. "Debate: JSON vs. XML as a data interchange format", *InfoQ.* Retrieved 29 April 2009 from http://www.infoq.com/news/2006/12/json-vs-xml-debate.
[Newman and Thomas, 2009: Ch 3]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation*. Ch 3. McGraw-Hill/Osborne.

[Newman and Thomas, 2009: Ch 5]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation*. Ch 5. McGraw-Hill/Osborne.
[Newman and Thomas, 2009: Ch 12]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation.* Ch 12. McGraw-Hill/Osborne
[Newman and Thomas, 2009: Ch 13]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation.* Ch 13. McGraw-Hill/Osborne.
[Newman and Thomas, 2009: Ch 14]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation*. Ch 14. McGraw-Hill/Osborne.
[Newman and Thomas, 2009: Ch 15]	Aaron C Newman, and Jeremy Thomas. [2009]. *Enterprise 2.0 Implementation*. Ch 15. McGraw-Hill/Osborne.
[Olivieri, 2008]	Ricardo Olivieri. [18 March 2008]. "Implement business logic with the Drools rules engine." *IBM DeveloperWorks.* Retreived 18 March 2009 from http://www.ibm.com/developerworks/java/library/j-drools/.
[Pajunen and Chande, 2007]	Lasse Pajunen, and Suresh Chande. "Developing Workflow Engine for Mobile Devices. [2007]. " *IEEE International Enterprise Distributed Object Computing Conference 2007.* p.279-286. IEEE Computer Society, Annapolis
[Sirin et al., 2007]	Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. [June 2007]. "Pellet: A practical OWL-DL reasoner." *Web Semantics: Science, Services and Agents on the World Wide Web*. Vol. 5. p.51-53. Elsevier
[Sommerville, 2007]	Ian Sommerville. [2007] *Software Engineering 8th Edition*. Ch. 4. Addison-Wesley, Harlow, England.
[Vincent, 2008]	Paul Vincent. [4 March 2008]. "Goal-directed event processing." *Complex Event Processing.* Retrieved 13 March 2009 from http://tibcoblogs.com/cep/2008/03/04/goal-directed-event-processing/
[Wikipedia: Production System, 2009]	[12 March 2009]. *Wikipedia: Production System*. Retrieved 21 March 2009 from http://en.wikipedia.org/wiki/Production_system
[Wikipedia Social Network Service, 2009]	[19 March 2009]. *Wikipedia: Social Network Service.* Retrieved 22 March 2009 from http://en.wikipedia.org/wiki/Social_Network_Service
[Wiseman, 2006]	Geoffrey Wiseman. [19 June 2006]. "Real-World Rules Engines." *InfoQ*. Retrieved 18 March 2009 from http://www.infoq.com/articles/Rule-Engines

Appendix A Mapping of Functional Requirements to Iterations

The table below shows a list of functional requirements mapped to development iterations 1 through 4, approximately based on requirement priority (1 representing the highest priority). The comments column describes any reasons for the requirement and its priority.

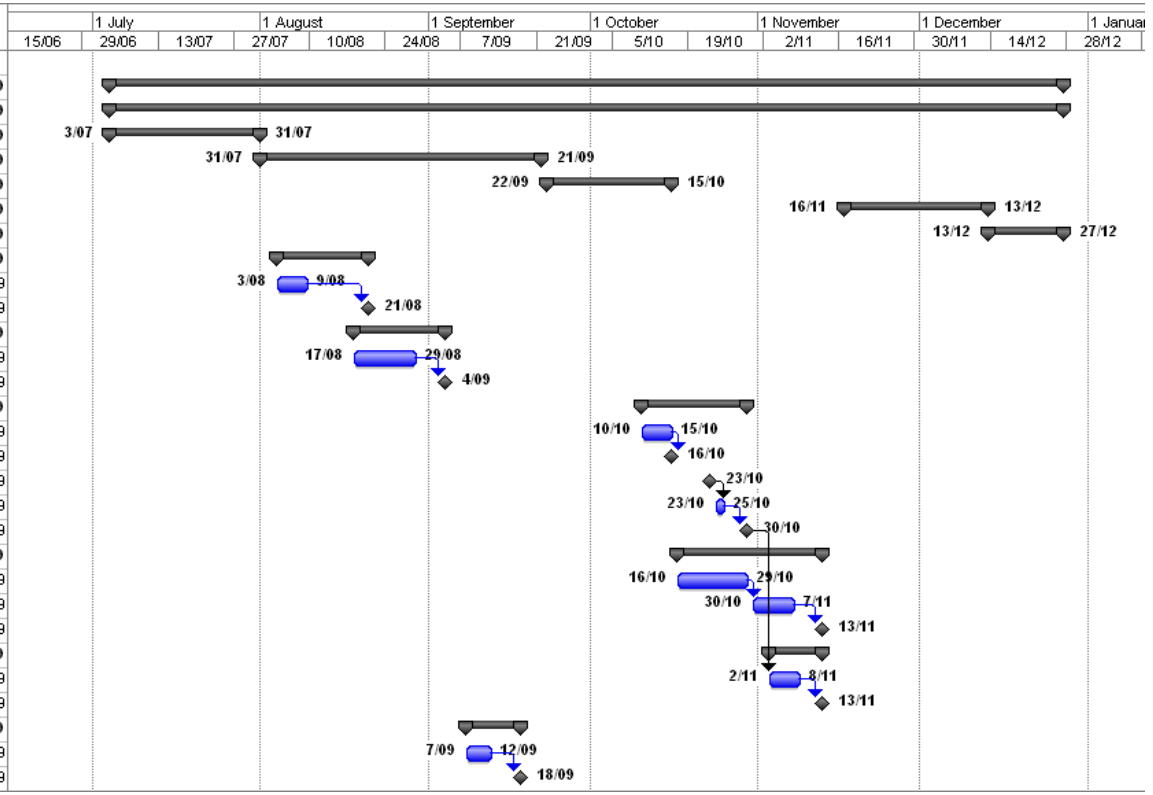
ID	Requirement	Priority	Iteration	Comment
General system requirements				
SYS1	System architecture shall support integration with other SNSs through APIs	1	1	See Section 2.1.1
SYS2	System architecture shall support integration in a Service Oriented Architecture	1	1	See Section 2.1.1
SYS3	System data model shall be extensible where possible.	2	1	See Section 3.1.1
SYS4	System data model shall support concept of users	1	1	See Section 3.3
SYS5	System data model shall support concept of relationships between users (e.g. user A is a client/peer/superior of user B)	1	1	See Section 3.3
SYS6	System shall store list of users that a particular user is interested in. These are displayed by default on the status querying screens/pages.	1	1	See Section 3.3
SYS7	System shall support authentication and sourcing of user information from an external system	3	4	See Section 2.1.1
SYS8	System shall record historical status changes to support data mining	2	1	See Section 2.1.3. Can be supported by the data model
SYS9	System shall support management of users via an administration interface	3	4	This would only be temporary until SYS4 is implemented.
SYS10	System shall integrate with Facebook SNS to update Facebook status and to retrieve user profile information.	3	4	See Section 2.1.1
SYS11	System shall integrate with Twitter SNS to update Facebook status and to retrieve user profile infromation.	3	4	See Section 2.1.1
SYS12	System shall provide business rules/logic to determine and store a user's status based on inputs.	1	1	See Section 4
SYS13	Business rules/logic shall be modifiable without requiring recompilation of system code.	1	1	See Section 4.3
SYS14	Selected rules engine shall integrate with selected server-side technology without violating non-functional requirements	1	1	See Section 4.3

ID	Requirement	Priority	Iteration	Comment
SYS15	Selected rules engine shall have a development environment so as to allow for easy tuning and update of rules.	3	1	See Section 4.3
Client application requirements				
CLI1	System shall support querying of statuses from the mobile client	1	2	
CLI2	System shall support entry of status information into the mobile client	1	2	
CLI3	System shall support integration of GPS location information into the entry of status information	2	3	
CLI4	System shall support integration with Google Calendar API for entry of status information	2	3	
CLI5	System shall support integration with mobile device sensors to detect WiFi and phone network access, and use this information in the entry of status information.	2	3	
CLI6	System shall be designed to be extensible to support additional integrations in future.	1	4	Initially considered in iteration 1, but may need refinement after prototype development.
CLI7	System shall support querying of statuses from a desktop web client	2	3	
CLI8	System shall support manual entry of status information from a desktop web client	2	3	
Non-functional requirements				
NF1	Server side component shall be deployable on Linux or Windows platforms	1	1	
NF2	System shall provide a REST based service to query and submit status information	1	1	See Section 5.2
NF3	System shall provide a Web Service based service to query and submit status information	3	4	See Section 5.2
NF4	Any software platforms/libraries used shall be open source and shall not require code developed in this project to be open source.	1	1	This is to reduce cost.
NF5	Any software platforms/libraries used shall have a support option	1	1	This is to support possible future enterprise deployment
NF6	System shall support up to 50 users, architecture shall support scaling up to 100,000 users	2	4	This will be initially considered during iteration 1, but may need refinement after prototype development

ID	Requirement	Priority	Iteration	Comment
NF7	Client-server communications shall be over a secure channel (HTTPS)	1	2	See Section 2.1.2
NF8	System shall require user to actively submit status information for privacy reasons.	1	1	See Section 2.1.2
NF9	System shall support storage of status information using Semantic Web technologies (i.e. stored in RDF store using an ontology defined in OWL)	3	4	See Section 2.1.2

Appendix B Draft ITEC809 Project Plan

ID	Task Name	Duration	Start	Finish
27				
28	**Phase 2 - Software Development**	**125 days**	**Fri 3/07/09**	**Sun 27/12/09**
29	**ITEC809 Project Development**	**125 days**	**Fri 3/07/09**	**Sun 27/12/09**
30	**High Level System Analysis and Design**	**20 days**	**Fri 3/07/09**	**Fri 31/07/09**
38	**Iteration 1 - Server Core**	**36 days**	**Fri 31/07/09**	**Mon 21/09/09**
44	**Iteration 2 - Mobile Core**	**17 days**	**Tue 22/09/09**	**Thu 15/10/09**
50	**Iteration 3 - Mobile Enhanced**	**19 days**	**Mon 16/11/09**	**Sun 13/12/09**
54	**Iteration 4 - Web Interface**	**10 days**	**Sun 13/12/09**	**Sun 27/12/09**
58	**ITEC809 Project Proposal**	**13 days**	**Mon 3/08/09**	**Fri 21/08/09**
59	Write up project proposal	12 hrs	Mon 3/08/09	Sun 9/08/09
60	ITEC809 Project Proposal Due	0 days	Fri 21/08/09	Fri 21/08/09
61	**ITEC809 Review of Sources**	**13 days**	**Mon 17/08/09**	**Fri 4/09/09**
62	Write up review of sources	24 hrs	Mon 17/08/09	Sat 29/08/09
63	ITEC809 Review of Sources Due	0 days	Fri 4/09/09	Fri 4/09/09
64	**ITEC809 Workshop Paper**	**14 days**	**Sat 10/10/09**	**Fri 30/10/09**
65	Write up draft ITEC809 workshop paper	12 hrs	Sat 10/10/09	Thu 15/10/09
66	ITEC809 Draft Workshop Paper Due	0 days	Fri 16/10/09	Fri 16/10/09
67	Receive feedback	0 days	Fri 23/10/09	Fri 23/10/09
68	Revise Workshop Paper	8 hrs	Fri 23/10/09	Sun 25/10/09
69	ITEC809 Final Workshop Paper Due	0 days	Fri 30/10/09	Fri 30/10/09
70	**ITEC809 Project Report**	**19 days**	**Fri 16/10/09**	**Fri 13/11/09**
71	Write up draft report	30 hrs	Fri 16/10/09	Thu 29/10/09
72	Review and revise project report	20 hrs	Fri 30/10/09	Sat 7/11/09
73	ITEC809 Project Report Due	0 days	Fri 13/11/09	Fri 13/11/09
74	**ITEC809 Workshop**	**8 days**	**Mon 2/11/09**	**Fri 13/11/09**
75	Prepare for Workshop Presentation	12 hrs	Mon 2/11/09	Sun 8/11/09
76	ITEC809 Workshop Presentation	0 days	Fri 13/11/09	Fri 13/11/09
77	**ITEC809 Ethics Assignment**	**8 days**	**Mon 7/09/09**	**Fri 18/09/09**
78	Write up ITEC809 Ethics Assignment	6 hrs	Mon 7/09/09	Sat 12/09/09
79	ITEC809 Ethics Assignment Due	0 days	Fri 18/09/09	Fri 18/09/09



Appendix C Sample Prolog Code implementing application logic

```
/* Helper function to compare times */
later(date(Y, M, Day,H,Min,Sec1,_,_,_), date(Y, M, Day,H,Min,Sec2,_,_,_))
:-
      Sec1 > Sec2.
later(date(Y, M, Day,H,Min1,_,_,_,_), date(Y, M, Day,H,Min2,_,_,_,_)) :-
      Min1 > Min2.
later(date(Y, M, Day,H1,_,_,_,_,_), date(Y, M, Day,H2,_,_,_,_,_)) :-
      H1 > H2.
later(date(Y, M, Day1,_,_,_,_,_,_), date(Y, M, Day2,_,_,_,_,_,_)) :-
      Day1 > Day2.
later(date(Y, M1, _,_,_,_,_,_,_), date(Y, M2, _,_,_,_,_,_,_)) :-
      M1 > M2.
later(date(Y1, _, _,_,_,_,_,_,_), date(Y2, _, _,_,_,_,_,_,_)) :-
      Y1 > Y2.

/* Test knowledge base. For each person, needs an availableComms, location,
set of relationships, activities and activity participants */
availableComms(john, phone).
availableComms(jim, network).
availableComms(jim, phone).
availableComms(james, network).
location(john, office).
location(jim, office).
location(james, home).
relationship(john,jim,client).
relationship(jim,james,peer).


relationshipImportance(peer, client).
relationshipImportance(junior, peer).
relationshipImportance(peer, manager).
relationshipImportance(manager, ultimateManager).
relationshipImportance(manager,client).
relationshipImportance(client,ultimateManager).

/*
Activities have the following format:
activity(person, starttime, endtime, type, urgency)
activityParticipant(activity, person1, person2)
*/
activity(john,date(2008,4,8,16,0,0,0,-,-),date(2008,4,8,17,0,0,0,-,-),
meeting, now,1).
activityParticipant(1, jim).

relationship(clientA, a, client).
relationship(managerA, a, manager).
relationship(peerA, a, peer).
relationship(juniorA, a, junior).
relationship(clientB, b, client).
relationship(managerB, b, manager).
relationship(peerB, b, peer).
relationship(juniorB, b, junior).

location(clientA, office).
location(a, office).
location(b, home).
```

```
availableComms(a, phone).
availableComms(a, network).
availableComms(clientA, phone).
availableComms(clientA, network).


/*********/
/* Rules */
/*********/
/* Rules for ordering of communication. To be used as part of predicate
sort (predsort) function */
betterComms(=,X,Y):-X=Y.
betterComms(>,X,Y):- \+betterComms(<,X,Y).
betterComms(<,inPerson,_).
betterComms(<,phone,instantMessage).
betterComms(<,phone,email).
betterComms(<,phone,noticeboard).
betterComms(<,instantMessage,email).
betterComms(<,instantMessage,noticeboard).
betterComms(<,email,noticeboard).
betterComms(>,noticeboard,_).

/* Rule to find the best means of communications in order */
bestComms(X,Y,QueryStartTime, QueryEndTime, QuestionUrgency,
OrderedCommsList) :- canCommsList([inPerson, phone, instantMessage, email,
noticeboard], CommsList, [], X, Y, QueryStartTime, QueryEndTime,
QuestionUrgency), predsort(betterComms, CommsList, OrderedCommsList).

/** Rules to put together a list of available means of communications */
canCommsList([],FinalCommsList,FinalCommsList, _, _, _, _, _).
canCommsList([H|T], FinalCommsList, InitialCommsList, X, Y, QueryStartTime,
QueryEndTime, QuestionUrgency):-
      (canComms(H, X, Y, QueryStartTime, QueryEndTime, QuestionUrgency) ->
canCommsList(T, FinalCommsList, [H|InitialCommsList], X, Y, QueryStartTime,
QueryEndTime, QuestionUrgency);
            canCommsList(T, FinalCommsList, InitialCommsList, X, Y,
QueryStartTime, QueryEndTime, QuestionUrgency)).

/* Rule for matching location */
matchLocation(X, Y) :- location(X, A), location(Y, A).

/** Rules to determine available means of communication */
canComms(inPerson, X, Y, QueryStartTime, QueryEndTime, QuestionUrgency) :-
matchLocation(X,
Y),freeToTalk(X,Y,QueryStartTime,QueryEndTime,QuestionUrgency).
canComms(phone, X, Y, QueryStartTime, QueryEndTime, QuestionUrgency):-
availableComms(X,phone),availableComms(Y,phone),freeToTalk(X,Y,QueryStartTi
me,QueryEndTime,QuestionUrgency).
canComms(instantMessage, X,Y,QueryStartTime,QueryEndTime,QuestionUrgency):-
availableComms(X, network), availableComms(Y,
network),freeToTalk(X,Y,QueryStartTime,QueryEndTime,QuestionUrgency).
canComms(email,_,Y,_,_,_):-availableComms(Y,network).
canComms(noticeboard,_,Y,_,_,_):-availableComms(Y,network).

matchingActivity(X,QueryStartTime,QueryEndTime):-
      activity(X,StartTime,EndTime,_,_,_),later(QueryStartTime,StartTime),l
ater(EndTime,QueryEndTime).
```

```
/* Rule to determine if the questionee is free to talk or the activity can
be overriden */
freeToTalk(X, Y,QueryStartTime, QueryEndTime, QuestionUrgency):-
      \+matchingActivity(X,QueryStartTime,QueryEndTime);questionHigherUrgen
cy(X,Y,QueryStartTime,QueryEndTime,QuestionUrgency);
      questionerMoreImportant(X,Y,QueryStartTime,QueryEndTime).

/* Rules/knowledge base to order urgency of tasks/questions */
urgencyMatrix(now, in10Minutes).
urgencyMatrix(in10Minutes, today).
urgencyMatrix(today, tomorrow).
urgencyMatrix(tomorrow, endOfWeek).

/* Rules to find if a question is of higher urgency than the activity */
questionHigherUrgency(X,_,QueryStartTime,QueryEndTime,QuestionUrgency):-
      activity(X,StartTime,EndTime,_,ActivityUrgency,_),later(QueryStartTim
e,StartTime),later(EndTime,QueryEndTime),
      urgencyMatrix(QuestionUrgency,ActivityUrgency).

/* Rule to find if the questioner is more important than participants of
the activity */
questionerMoreImportant(X, Y, QueryStartTime, QueryEndTime):-
      activity(X,StartTime,EndTime,_,_,ActivityId),later(QueryStartTime,Sta
rtTime),later(EndTime,QueryEndTime),
      activityParticipant(ActivityId,Z),
      relationship(X,Y,QuestionerRelationship),relationship(X,Z,ActivityPar
ticipantRelationship),
      relationshipImportance(ActivityParticipantRelationship,QuestionerRela
tionship).
```

Appendix D Sample JBoss Drools rules implementing application logic

```
#created on: 15/04/2009
package TestRules

#list any import classes here.
import itec808.model.Activity;
import itec808.model.Question;
import itec808.model.Status;
import itec808.model.Location;
import itec808.model.CommunicationsType;
import itec808.model.UserRelationship;
import itec808.model.UserRelationshipType;


#declare any global variables here
global java.util.List meansOfCommunication;

rule "Match Locations"
      no-loop true
      when
         q: Question(qQuestionee:questionee, qQuestioner:questioner,
             qStartTime:startTime, qEndTime:endTime, matchLocationRun ==
false)
         statusQuestionee: Status(user == qQuestionee &&
             startTime <= qEndTime && endTime >= qStartTime,
             activityQuestionee: activity )
        statusQuestioner: Status(user == qQuestioner &&
            startTime <= qEndTime && endTime >= qStartTime,
            activityQuestioner:activity )
        Activity( this == activityQuestionee, locationQuestionee:location )
        Activity( this == activityQuestioner, location ==
locationQuestionee,
        locationQuestioner:location )
             #conditions
      then
             #actions
         System.err.println("Matched location:" +
locationQuestioner.getAddress() + " " + locationQuestioner.getAddress());
            q.setMatchLocation(true);
            q.setMatchLocationRun(true);
         update(q);
end

rule "Questionee free to talk"
    no-loop true
      #include attributes such as "salience" here...
      when
         q:
Question(qQuestionee:questionee,qStartTime:startTime,qEndTime:endTime,
         freeToTalk == false)
        statusQuestionee: Status(user == qQuestionee &&
            startTime <= qEndTime && endTime >= qStartTime,
            activityQuestionee: activity )
        Activity (this == activityQuestionee, type.name == "Free")
             #conditions
      then
             #actions
```

```
        System.err.println("Free to talk:");
            q.setFreeToTalk(true);
        update(q);
end

rule "Questionee busy but question is more important"
//    salience 100
    no-loop true
    #include attributes such as "salience" here...
    when
       q: Question( qU: questionUrgency, qQuestionee:questionee,
        qStartTime:startTime, qEndTime:endTime, freeToTalk == false )
       statusQuestionee: Status(user == qQuestionee &&
           startTime <= qEndTime && endTime >= qStartTime,
           activityQuestionee: activity )
       Activity (this == activityQuestionee,urgency.priority < qU.priority)
        #conditions
    then
        #actions
        System.err.println("Free to talk2:");
        q.setFreeToTalk(true);
        update(q);
end

rule "Questionee busy but questioner is more important that other
participants"
    no-loop true
    #include attributes such as "salience" here...
    when
       q: Question(qQuestionee:questionee, qQuestioner:questioner,
       qEndTime:endTime, qStartTime:startTime, freeToTalk == false)
       statusQuestionee: Status(user == qQuestionee &&
           startTime <= qEndTime && endTime >= qStartTime,
           activityQuestionee: activity )
       Activity (this == activityQuestionee,
activityParticipants:participants )
       UserRelationship( fromUser == qQuestioner, toUser == qQuestionee,
r1Type: type )
       UserRelationship( fromUser == qQuestioner, toUser memberOf
activityParticipants, type.priority < r1Type.priority)
        #conditions
    then
        #actions
        q.setFreeToTalk(true);
        update(q);
end

rule "Can Comms In Person"
    salience 50
    when
        q: Question(matchLocation == true, freeToTalk == true)
    then
        System.err.println("Can Comms In Person");
        meansOfCommunication.add(CommunicationsType.IN_PERSON);
end

rule "Can Comms On Phone"
    salience 45
    when
```

```
        q: Question(qQuestionee:questionee, qStartTime:startTime,
qEndTime:endTime,
        qQuestioner:questioner, freeToTalk == true)
       not (exists (Status(user == qQuestionee &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.PHONE)))
       not (exists (Status(user == qQuestioner &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.PHONE)))
    then
        meansOfCommunication.add(CommunicationsType.PHONE);
end

rule "Can Comms Instant Message"
    salience 40
    when
        q: Question(qQuestionee:questionee, qQuestioner:questioner,
        qEndTime:endTime, qStartTime:startTime, freeToTalk == true)
       not (exists (Status(user == qQuestionee &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.INSTANT_MESSAGE)))
       not (exists (Status(user == qQuestioner &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.INSTANT_MESSAGE)))
    then
        meansOfCommunication.add(CommunicationsType.INSTANT_MESSAGE);
end

rule "Can Comms Email"
    salience 35
    when
        q: Question(qQuestioner:questioner, qStartTime:startTime,
qEndTime:endTime,
            freeToTalk == true)
       not (exists (Status(user == qQuestioner &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.EMAIL)))
    then
        meansOfCommunication.add(CommunicationsType.EMAIL);
end

rule "Can Comms Noticeboard"
    salience 30
    when
        q: Question(qQuestioner:questioner, qStartTime:startTime,
qEndTime:endTime)
       not (exists (Status(user == qQuestioner &&
           startTime <= qEndTime && endTime >= qStartTime,
           availableCommunicationsTypes not contains
CommunicationsType.NOTICEBOARD)))
    then
        meansOfCommunication.add(CommunicationsType.NOTICEBOARD);
end
```