Can We Talk Now?
Technology Selection for an Intelligent Status Tracking System

Sidney Shek
Department of Computing
Macquarie University
Sydney, Australia
sidney.shek@students.mq.edu.au

Abstract

A common problem in the workplace is determining how best to communicate with one another especially between mobile workers. This paper presents the results of a feasibility study and technology selection for a prototype intelligent status tracking system ('CanWeTalkNow?' or CWTN). CWTN is inspired by principles from Social Networking Services and makes use of various features from mobile devices (Apple iPhone initially) such as calendars and GPS to automatically populate status information so as to encourage uptake of CWTN. The Drools rule engine will be used to implement rules to determine the "best" means of communication between users. 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.

1 Introduction

Team communications is 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 person being questioned is in an important meeting with a client, he/she may not appreciate an interruption.

Public Social Networking Services (SNS) such as Facebook[1] and Twitter[2] have demonstrated that by providing a manually updated central store of status data, users can track availability of each other which is similar to what is required in this context.

The aim of the project is to develop a low-cost prototype client-server system ('CanWeTalkNow?' or CWTN) to apply status tracking features available in public SNSs to the enterprise. Status information can be *automatically* populated based on various information sources from mobile devices (Apple iPhone initially) such as calendars and the user's location, thereby making it easier for busy workers to keep their status up-to-date compared to manually updated status tracking in standard SNSs. In addition by applying logic rules to status information, CWTN can determine the most effective means of communication between two users at a point in time. CWTN is intended to make use of mobile device capabilities of smartphones, as the benefits are greatest for mobile workers.

The results of an initial feasibility study and technology selection for this project are presented in this paper. This information will be used to scope the development of the prototype system, and feed directly into its design.

The present paper is structured as follows. Section 2 presents related work on the project topic. Section 3 presents a discussion on required rules and the data model for CWTN, and an evaluation of different rules engines is presented in Section 3, representing a major component of the feasibility study. Section 4 discusses results of investigation into server-side platform and client-server communications. Section 5 presents current and ongoing work on this project. Section 6 concludes the paper.

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

2 Related Work

There are a number of public SNSs such as Facebook and Twitter, as well as enterprise or private SNSs such as SocialEngine[3] and Lotus Connections[4]. Many of these tools provide a free text field that users can manually update to inform others of their status. Some tools such as mobile Twitter also store location information[5]. However, these systems do not provide automated population of status information based on multiple sources including location, network available, calendar data, etc. In addition, existing SNSs do not apply logic to status information to answer questions as would be done in the proposed system. The CWTN architecture would allow integration with SNSs through Service Oriented Architecture, thereby making use of existing SNS functionality.

Dinoff et al. (2007) reported a system similar to CWTN but targeted at service providers or carriers. In contrast CWTN focuses on individual enterprises and only makes use of end-user functionality of mobile devices; it does not rely on service providers. CWTN is intended to be a low cost 'add-on' system that can integrate into an enterprise's existing environment.

The capabilities of the above-mentioned SNSs highlight a number of key requirements or issues to be resolved for the proposed system.

Firstly, operation and use of SNSs are user driven, as opposed to being defined by the system or developers (DiMicco et al., 2008). This allows SNSs use to evolve over time to meet user needs and hence encourage uptake. As a result, CWTN needs to have flexible data structures and the logic may need to be tuned over time.

Secondly, reported deployments of private enterprise SNSs highlight the need to address security concerns in relation to exposing potentially sensitive data, and the need to provide incentives to use CWTN to overcome cultural hindrances. Privacy, while a major issue for public SNSs, is less of a concern for enterprise SNSs (Newman and Thomas, 2009; Farzan et. al, 2008).

3 Rules Analysis and Engine Evaluation

A key aspect of CWTN is to provide sophisticated logic to populate and process status information. It is possible to place the required logic into programming code; however this makes it difficult to modify the logic as would be needed to provide a flexible system. One solution is to use a rules engine, which can dynamically import and run the logic in the form of rules without need for recompilation of source code. This approach was considered appropriate for this application at least for the querying component, so a number of rule engines were evaluated for this system.

3.1 Rules to be used in CWTN

In order to properly evaluate rule engines, an initial set of rules were developed so that a minimal set of functionality can be identified. The following subsections describe the data structures to be supported for facts, along with three sets of logic required for the application – (1) population of *Activity* facts, (2) determining available means of communications between two users, and (3) ordering of the means of communications.

Data structures for application logic

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.

At a given point in time, user is performing an *Activity*, such as being in 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 performing.

A 'questioner' would query CWTN to determine the best means of communication to a 'target' user. Both 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 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.

Populating Activity facts

The logic for populating *Activity* facts in CWTN would be specific to the source of information.

[3] See http://www.socialengine.net
[4] See http://www-01.ibm.com/software/lotus/products /connections /homepage.html
[5] See http://apiwiki.twitter.com/REST+API+Documentation.

For example, calendar entries can be used to create *Activities* of type 'meeting', with meeting participants set as *ActivityParticipants*. When there are no calendar entries, an activity type of 'idle' may be created.

Determining available means of communication

CWTN requires a small number of rules for determine available means of communications between the questioner and target users (in the order of 10). Some examples include:

1. If the target and questioner are in the same location, and the target is 'free to talk', then 'in person' communication is possible.
2. If the target is not free to talk, 'email' and 'noticeboard' communications should be used.
3. A target can only use instant messaging if they have network access.
4. If the question is of higher importance than the target's activity, the target is considered 'free to talk'.
5. If the questioner is superior in terms of relationship to all participants in the target's activity, the target is considered 'free to talk'.

It can be seen that there is a hierarchy of rules. Rules 1, 2 and 3 describe available communications as based on, network availability, matching location and 'free to talk' condition; the latter being dependent on rules 4 and 5. This hierarchy of rules may allow code reuse or use of subgoals.

Ordering means of communications

Means of communications are prioritised (e.g. phone is 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.

3.2 Brief introduction to rules engines

Forward versus backward chaining

A key differentiator of rule engines is whether they support forward or backward chaining (or potentially both).

Forward chaining involves running rules on data (known as 'facts') that match rule conditions. The rules may modify or create facts, which may trigger other rules to run. This approach is considered 'data driven' (Haley, 2008), and can be used to populate or enhance existing data sets. One downside is that the creation of facts can significantly increase memory consumption (Cook, 2008).

Backward chaining involves identifying rules and facts required to meet a requested goal. If suitable facts can be found, then the goal is deemed to have been met. Subgoals may be generated based on rules, which allows a form of deduction to occur. Backward chaining is considered 'goal driven' (Haley, 2008).

Rule engine types

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, 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. SWI-Prolog[6] and Prova[7] were evaluated as examples of inference engines.

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. Drools[8] and JESS[9] were evaluated as examples of production rule engines.

Event-Condition-Action rule engines apply rules upon receipt of an event, and so are not applicable for CWTN.

3.3 Evaluation method

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. Test applications were also written for SWI-Prolog and Drools as representatives of inference and production rule engines respectively.

The evaluation criteria used in order of importance were as follows:

1. Support for rules presented in section 3.1. Reusability of code between rules is also

[6] See www.swi-prolog.org
[7] See www.prova.ws
[8] See www.jboss.org/drools
[9] See www.jessrules.com

desirable.

2. Ease of integration with Java. The rule engine would need to integrate with the technology used for server-side components (Java Enterprise Edition or JEE).
3. Availability of an integrated development environment (IDE) and documentation, to support rule development and ongoing maintenance/tuning of the prototype.
4. Support for online redeployment of rules.
5. Licensing cost for enterprise deployment. The aim is to provide a lost cost system, therefore only open source or free to use rule engines at least for evaluation purposes were considered.
6. Future proofing. This includes support for standardised languages such as RuleML and Semantic Web data structures.

Only a theoretical analysis of memory and CPU performance for the rule engines was performed in terms of forward versus backward chaining. 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.

The four rule engines that were evaluated (SWI-Prolog, Prova, Drools and JESS) were selected as they were open source or free for academic use, and could be integrated with Java.

3.4 Evaluation results

No rule engine stood out as the clear winner. As a result, a modular system architecture is needed to allow the rule engine component to be easily replaced in the future.

SWI-Prolog[6] is a popular Prolog implementation, and its backward-chaining approach was well suited to the rules to be used by CWTN. However, its major downfalls were poorer integration with Java 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[10]. 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).

Prova[7] 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 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 communication with the database is 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.

Drools[8] 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 suit 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 support for Semantic Web technologies or standardised rule languages as yet.

JESS[9] is a production rules engine developed by Sandia Laboratories and is considered a leading Java-based rules engine. It also 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.

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.

3.5 Other findings for rule engine usage

During the rule engine evaluation, a number of implementation considerations and decisions were identified.

Firstly, only the logic for determining available means of communication benefit significantly from a 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, so could be easily implemented in the client application. Ordering the means of communications requires only simple comparator logic in Java.

10 See "High-level interface", http://www.swi-prolog.org/packages/jpl/java_api/high-level_interface.html

Secondly, a backward chaining approach should be more appropriate for this application in terms of CPU and memory usage as there are relatively few rules compared to a potentially large number of facts. Although rules can be written to use either method, care would need to be taken when developing for forward chaining such as ensuring most specific rules are run first to minimise working memory size (Friedman-Hill, 2008), and also minimising rule interdependencies which could result in re-running of rules and hence extra CPU loading. With a modular architecture, the rules engine can be replaced easily if performance or maintainability of the rules is deemed unacceptable.

4 Technologies for client and server components of the system

Client-server communications

As discussed in Section 2, CWTN needs to fits into a SOA environment so as to allow simple integration into an enterprise scenario in the future. Either Web Services or Representational State Transfer (REST) could be used to satisfy this (Bruno, 2007).

Web Service communication based on SOAP requires significant overhead in processing the envelope and generating requests in the required format. In addition, the Apple iPhone SDK does not provide inbuilt Web Service support[11].

REST however passes requests using standard HTTP URIs, and responses can be in either XML or Javascript Object Notation (JSON) with no parsing of envelopes required. On the client-side only a HTTP client is required, which is native in the iPhone SDK[11]. For the server-side, REST support has now been standardised for Java, with a number of libraries now available (Little, 2008). Given its simplicity of implementation, REST was selected for client-server communications. The RESTlet Java library[12] would be used as it has inbuilt support for the Google App Engine, which is discussed in the next section.

Another consideration is the use of XML versus JSON to deliver payloads between client and server. While XML has benefits over JSON of being standardised and supporting data type validation through XML Schema, JSON is considered lightweight and simple to parse (Marinescu and Tilkov, 2006), making it more suitable for this situation. In any case, selection between XML and JSON on the server side is a configuration item, and libraries exist for converting either format into native objects on the iPhone[13].

Platform for server-side application deployment

Providing secure access to the server application (i.e. rules engine and database) over the Internet is a major challenge, especially for a prototype system. Adding steps to using the application such as manually entering authentication details or setting up a Virtual Private Network connection significantly impacts system usability, and could limit uptake and success of the prototype.

The solution to be pursued is deploying the server application onto the Google App Engine cloud[14]. It provides free JEE server infrastructure for limited usage, along with integrated authentication with Google systems.

At a later stage, secure communications between the cloud and internal enterprise systems can be developed, or the application can be migrated to an Internet-facing host owned by the enterprise.

5 Future work

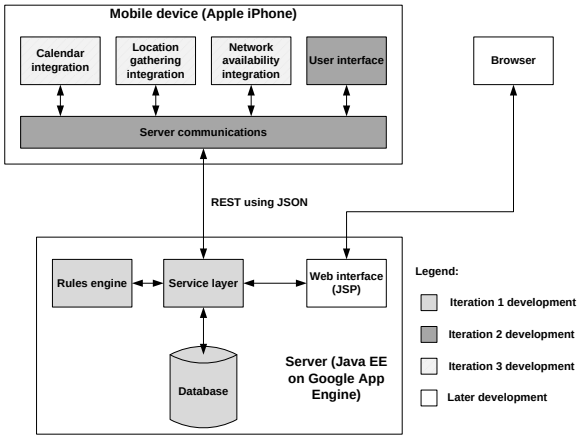

Figure 1 Proposed system architecture showing components to be developed in each iteration

The work presented in this paper represents the first part of the overall intelligent status tracking system project. The next stage is to develop a prototype system based on the selected technologies. Figure 1 shows a high level system architecture identifying the order in which components would be developed.

The intent is to deploy the prototype system in an enterprise environment and monitor uptake

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

[12] See www.restlet.org

[13] See code.google.com/p/touchcode

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

and utilisation to evaluate its effectiveness. The prototype may then be extended by integration with other SNSs.

One possible area for future research after development of the prototype is using Semantic Web techniques to store and reason about the status information (e.g. OWL-DL). This would allow sophisticated querying and data mining to be performed (e.g. to measure team member's idle or travel time), with potential integration via Semantic Web Services (Newman and Thomas, 2009).

6 Conclusion

In this paper, the results of a feasibility study and technology selection for a prototype intelligent status tracking system were presented. This system is targeted towards mobile workers in the enterprise, providing a central store of status information that is automatically populated based on various sources such as calendar and location information. Automation of data entry should encourage uptake and hence reduce cultural hindrances to use of CWTN. By querying the store, other users can identify the most effective means of communicating with the user.

An initial data model and rules for updating and querying the store are presented in this paper. These provide flexibility for CWTN, which is a key requirement for SNSs.

The Drools rules engine was selected for use in the prototype. Although not a perfect fit due to its forward chaining approach, with suitable modular system architecture it can be replaced easily in future.

For the prototype system, the first target client platform will be the Apple iPhone. The server component will be developed for the Google App Engine. Use of this public infrastructure negates the need for an enterprise to expose another server onto the Internet, and it also provides authentication, thus addressing security concerns. Client-server communications will use REST with JSON payloads.

This information will now be used to develop a prototype system. The first stage will be to develop the server-side component, followed by an initial mobile client application.

References

Aaron C Newman and Jeremy Thomas. 2009. *Enterprise 2.0 Implementation*. Ch 5, 13, 15. McGraw-Hill/Osborne.

Alex Kozlenkov. 2006. *PROVA Java Rule Language for Information Integration and Semantic Agents Version 2.0 User's Guide*.

Benjamin Grosof and Mike Dean. 11/5/2006. *ISWC-2006 Tutorial - Semantic Web Rules with Ontologies, and their E-Service Applications*. http://iswc2006.semanticweb.org/workshop_tutorial/iswc2006-tutorial-BGrosof+MDean-v2.pdf. Accessed 13/3/2009.

Diane J. Cook. 2008. "More Forward Chaining vs. Backward Chaining." *Artificial Intelligence Lecture Notes.* http://www.eecs.wsu.edu/~cook/ai/lectures/l7/node13.html. Accessed 21/3/2009.

Eric J. Bruno. 8/6/2007. "SOA, Web Services and RESTful systems – A framework for building RESTful systems", *Dr Dobb's Portal* http://www.ddj.com/web-development/199902676. Accessed 13/4/2009.

Ernest Friedman-Hill. 2008. *Jess The Rule Engine for the Java Platform*, Sandia National Laboratories. http://www.jessrules.com/jess/docs/Jess71p2.pdf. Accessed 13/4/2009.

Floyd Marinescu, Stefan Tilkov. 26/12/2006. "Debate: JSON vs. XML as a data interchange format", *InfoQ*. http://www.infoq.com/news/2006/12/json-vs-xml-debate, Accessed 29/4/2009

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, 42-49. Dagstuhl Castle: Springer-Verlag.

Joan DiMicco, David R Millen, Werner Geyer, Casey Dugan, Beth Brownholtz, and Michael Muller. 2008. "Motivations for Social Networking at Work.". In Proc *CSCW'08.* San Diego. ACM. p.711-720.

Mark Little. 1/10/2008. "A Comparison of JAX-RS Implementations". *InfoQ*. http://www.infoq.com/news/2008/10/jaxrs-comparison. Accessed 10/4/2009.

Paul Haley. 11/3/2008. "Goals and backward chaining using the Rete Algorithm." http://pvhaley.wordpress.com/2008/03/11/goals-and-backward-chaining-using-the-rete-algorithm/ Accessed 13/3/2009.

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.* 2(6): 72-84. Academy Publisher.

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. *CHI'08*. Florence, Italy: ACM. p.563-572.