2011

Development of an Internet-Based Chronic Disease Self-Management System

Ali Sunyaev
University of Cologne, Germany, sunyaev@wiso.uni-koeln.de

Dmitry Chornyi
Technische Universität München, Germany, chornyi@cs.tum.edu

24th Bled eConference
eFuture:
Creating Solutions for the Individual, Organisations and Society
June 12 - 15, 2011; Bled, Slovenia

Development of an Internet-Based Chronic Disease Self-Management System

Ali Sunyaev

University of Cologne, Germany

sunyaev@wiso.uni-koeln.de

Dmitry Chornyi

Technische Universität München, Germany

chornyi@cs.tum.edu

Abstract

Patient self-management programs and information systems that support them can improve the quality of healthcare. Flaws in user experience reduce the willingness of patients to adopt such systems. To explore how emerging technology such as rich Internet applications can be used to address the usability issues of personal health information systems, we developed a health self-management application that is based on an open-source framework. In this work we present the architecture of the system, discuss the issues we faced and lessons we learned while developing it. This work can help researchers and practitioners in evaluating approaches towards developing new generation of personal health solutions. Furthermore, this work serves as a basis for implementing a feature-rich system that can improve chronic disease self-management.

Keywords: distributed information systems, medical services, programming, user-centered design

1 Introduction

Chronic medical conditions take a huge toll on the lives (Undem, 2009) of a growing number of people (Heisler, 2006) and are a major contributor to the rising costs of health care (Hoffman, Rice, & Sung, 1996; Kanaan, 2008). As public attitudes towards roles in healthcare change, the evidence is growing that chronic patients need more comprehensive treatment than they can receive at their doctor’s office (Holman & Lorig, 2000). Instead of being passive recipients of care, patients recognize their responsibility in managing their condition through day-to-day decisions about diet, self-measurement, medications, and exercise. A new trend is emerging: people with chronic conditions become their own principal caregivers, with health care professionals acting as consultants supporting them in this role (Holman & Lorig, 2000). By encouraging patients to participate actively in determining the course of their diseases and ensuring they have the skills, knowledge, and confidence to manage their health, patient self-management programs can improve health outcomes (Bodenheimer, Lorig, & Holman, 2002), increase treatment satisfaction (Sawicki, 1999) and safety (Koutkias & Malousi, 2010), as well as reduce healthcare costs (Lahdensuo et al., 1998).

Some patient self-management programs rely on recording patient data over extended periods of time, analyzing it, and encouraging patients to make daily healthcare decisions based on this data (Von Korff et al., 1997). Information technology represents a key tool in supporting such scenarios (Bu et al., 2007). Designers of information systems for patient self-management face a new set of challenges.

Maintaining motivation to support long-term user commitment was shown to be a problem that can be solved through increased personalization, interactivity, and social support through data sharing (Mattila et al., 2010). In this regard, seamless integration with the existing healthcare IT ecosystem is beneficial, as a system that can collaborate with existing Personal Health Record (PHR) providers can take advantage of their infrastructure and data to deliver additional value to the user. Finally, usability is important, as it was shown that flaws in user experience reduce the willingness of patients to adopt personal health information systems (Peters et al., 2009).

To explore how these issues can be addressed in practice, in this article we described the development of a prototype of a distributed health self-management service that can support patients with diabetes at tracking their blood glucose levels. Our design efforts were guided by the goals of usability, security, extensibility, and interoperability with third-party healthcare information systems.

The rest of this article is organized as follows. Section 2 discusses different types of web clients in the context of their suitability for our project. Section 3 looks at the system we developed. Section 4 presents the evaluation of the system, while Section 5 discusses the lessons learned in the course of the project. Section 6 summarizes our work.

2 Web Clients

2.1 Traditional Web Applications

In the past few years, the World Wide Web has become the de-facto deployment environment for new software systems (Mikkonen & Taivalsaari, 2008). Since the 90's, web-based applications have been used to accomplish an increasing range of tasks including purchasing goods, bidding on auctions, booking tickets, trading stocks, and since recently, managing personal health records (Sunyaev et al., 2010). Compared to desktop applications, web applications are characterized by reduced deployment and maintenance costs, simple architectures, intrinsic multiplatform availability, and broader user appeal. As network and hardware capabilities expanded, web applications evolved from simple web sites to robust multitier systems that are aimed at replacing complex desktop applications. These developments amplified two serious shortcomings of traditional web applications in the areas of software engineering and usability, as illustrated in the following paragraphs.

The principle technologies that browsers use to display web pages—HTML markup language, CSS style sheet language and JavaScript scripting language—do not introduce a coherent foundation for real applications on the web. They rather reflect the historical evolution of the web, where new features have been added on top of existing features in a mostly ad hoc fashion (Mikkonen & Taivalsaari, 2008). Moreover, the methods and tools that are currently used for web application development often overlook the principles of sound software engineering, such as modularity, consistency, simplicity, reusability, and portability (Mikkonen & Taivalsaari, 2008). As Mikkonen and Taivalsaari put it, "The use of the web as an application platform undermines the work that has been done in the software engineering area in the past thirty years or so." (Mikkonen & Taivalsaari, 2008)

Web applications have also undone decades in usability advances. The proliferation of the Internet has created a void in terms of acceptable user experiences for desktop and Internet applications (Farrell & Nezlek, 2007). For instance, the page-based display update model of the web browser that requires a complete page refresh for every user action is outright antiquated and is reminiscent of the I/O model of the IBM 3270 series terminals from the 1970s (Mikkonen & Taivalsaari, 2008). So far, many users and developers were willing to give up the user interface improvements brought by desktop computers in return for immediate access to new data and applications (O'Rourke, 2004). However, as web applications are expanded to the new areas of use, with online competition and user expectations rising, developers are increasingly pushed to bring web experience closer to that of a desktop. To do this, several usability problems of web applications need to be addressed (Preciado et al., 2005):

- *Process problems*: complex web applications often force a user to navigate through a series of pages to complete a single task.
- *Data Problems*: web applications do not support interactive explorations of the data. Usually, user has to search data through the use of input forms and then to navigate the hypertext in order to be able to see the desired data.
- *Configuration problems*: many web applications require the configuration of a product/system from multi-criteria choices, but are, in general, unable to present a customized product/system to users in an intuitive way and in a single step.
- *Feedback Problems*: web applications do not allow a continued and ordered interaction without page refreshments, so the user interaction with traditional web pages is limited.

2.2 Rich Internet Applications

Rich Internet Applications (RIAs) can solve both the usability- and software engineering-related problems of traditional web clients. In particular, RIAs can offer sophisticated user interfaces with rich interaction possibilities that are close to those of desktop applications. RIAs also rely on different development methods than traditional web applications. In this regard, RIAs can be organized into three distinct types (Farrell & Nezlek, 2007):

- *Plugin-based*: involve creating the application for a dedicated platform and then deploying this application as an embedded solution or a standalone application launched from the browser. Examples are Adobe Flash/Adobe Flex/AIR, Java/JavaFX and Microsoft Silverlight.

- *Script-based*: employ a combination of technologies to achieve their results, typically including XHTML/HTML, CSS, DOM, and JavaScript. They are characterized by revised use of JavaScript to load data asynchronously, modify static content and interact with page elements. To simplify development, a variety of frameworks exist, including Prototype, Dojo, Google Web Toolkit, and Eclipse RAP.
- *Browser-based*: use browser facilities and a user interface language to define an application. An example is XUL developed by the Mozilla Project.

We consider script-based RIAs to be particularly promising for shaping the future web landscape, because unlike the other two categories, RIAs are not confined to any single plugin or browser, have small footprint, and are fast to download and to launch (Noda & Helwig, 2005). Furthermore, the functionality gap between script-based and other types of RIAs can be expected to narrow as the HTML 5 standard is adopted and implemented. Although script-based RIAs internally still use HTML, CSS, and JavaScript to control the browser, manually writing markup and scripts is no longer the main method of developing RIAs. Instead, developers use frameworks that allow writing RIAs in conventional programming languages, thus bringing the web software engineering process closer to that of the desktop.

2.3 Eclipse RAP

Eclipse Rich Ajax Platform (RAP)[1] is a script-based RIA framework that brings the Eclipse Rich Client Platform (RCP)[2] to the web. It allows a developer to build a rich web client almost just as if it were a regular desktop Java application. This includes access to all Java APIs, UI design using the RWT widget toolkit, and access to the standard services that Eclipse workbench offers (e.g., JFace viewers and data binding, job scheduling). In fact, RCP and RAP frameworks are so similar that applications can be developed from the same code base in the strategy of single-sourcing (Lange, 2008). Also, Eclipse RAP can make use of OSGi inside the web container, enabling the full usage of the Eclipse plugin model (Eclipse Foundation, 2006).

3 Design and Implementation

3.1 Architecture

In the course of this project we developed a health self-management system (*Health Management System* or *HMS*) that is oriented towards patients with diabetes and is based on the Eclipse RAP platform. The goals of the project were to test how RIAs can be used to improve usability and acceptance of the system and to create a robust architecture that can serve as a base for more complex and feature-rich systems. We identified the following key usage scenarios that drove the discovery and the design of the architecture:

[1] http://www.eclipse.org/rap/
[2] http://www.eclipse.org/home/categories/rcp.php

- *Measurement management*: the system can be used to keep track of blood glucose test results that record the blood sugar levels. Measurements can be created, viewed, grouped by various criteria.
- *Interoperability*: user should be able to import measurements from Microsoft HealthVault and Google Health—the two major PHR platforms.
- *Data visualization*: user can view a graph of blood sugar levels over a period of time or compare several periods.

All of the aforementioned functions are only to be available after the proper authentication and authorization. Due to sensitivity of the medical data, it should be stored and accessed on the per-user basis.

The derived design resulted in a multilayer architecture, as presented in the Figure 1. The decision was made to use Java-based products in the overall architecture, as these mapped to our existing experience and skills best.

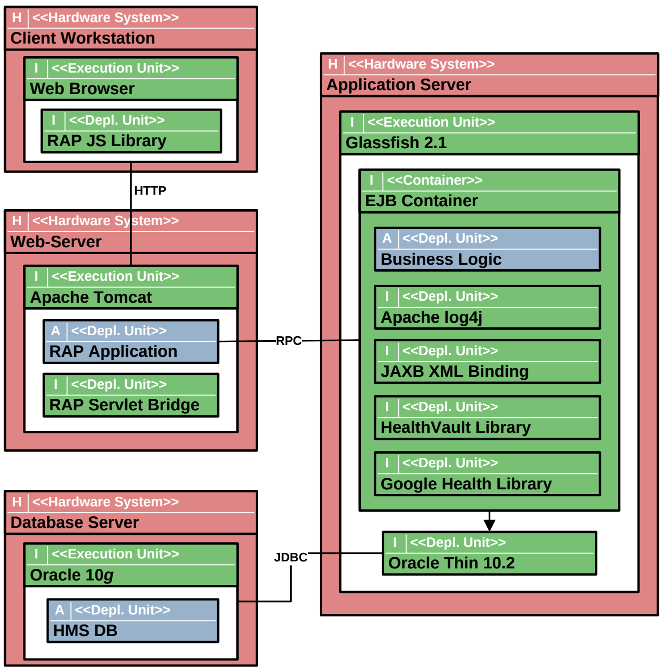

Figure 1: Execution view of the architecture

On the back-end side the system is implemented using Enterprise JavaBeans 3 and Java Persistence API ORM framework. These components are deployed to a Sun GlassFish Application Server v2.1 that provides a number of services through the EJB container. Particularly, the *HMS* takes advantage of the EJB 3 declarative security to restrict access to certain functions only for registered users, as well as declarative transaction management. User data is stored in the Oracle Database 10*g*. The database also serves as an Authentication Provider through the Glassfish *JDBCRealm* (Chan, 2006). This ensures end-to-end industry-strength system security, as no security code was required to be written manually. Additionally, the security of the *HMS* can be strengthened by enabling the transparent database encryption that is offered by the selected database (Nanda, 2005). The system uses JAXB (Sun Microsystems, 2008) to process data imported from external systems and Apache log4j (Gulcu, 2002) for the technical and audit logging.

The client is implemented as an Eclipse RAP application. It declares a single *perspective* with two *views* (Clayberg & Rubel, 2008): *History and Overview* (Figure 2). The *History* view displays blood glucose measurements, grouped by years and months as a tree. The tree nodes are loaded lazily; they can be expanded, collapsed, and selected arbitrarily. The tree is implemented as a JFace TreeViewer and has a context menu accessible through a right mouse click with menu items depending on the current selection. Using this menu and the toolbar menu that is defined for the view, measurements can be created, updated, viewed, and deleted. The *Overview* view uses the Annotated Time Line component from the Google Visualization API to draw graphs of blood sugar levels according to the user selection. It can also be used to compare these levels over several months or years. The global application menu provides access to additional functions like account management and data import.

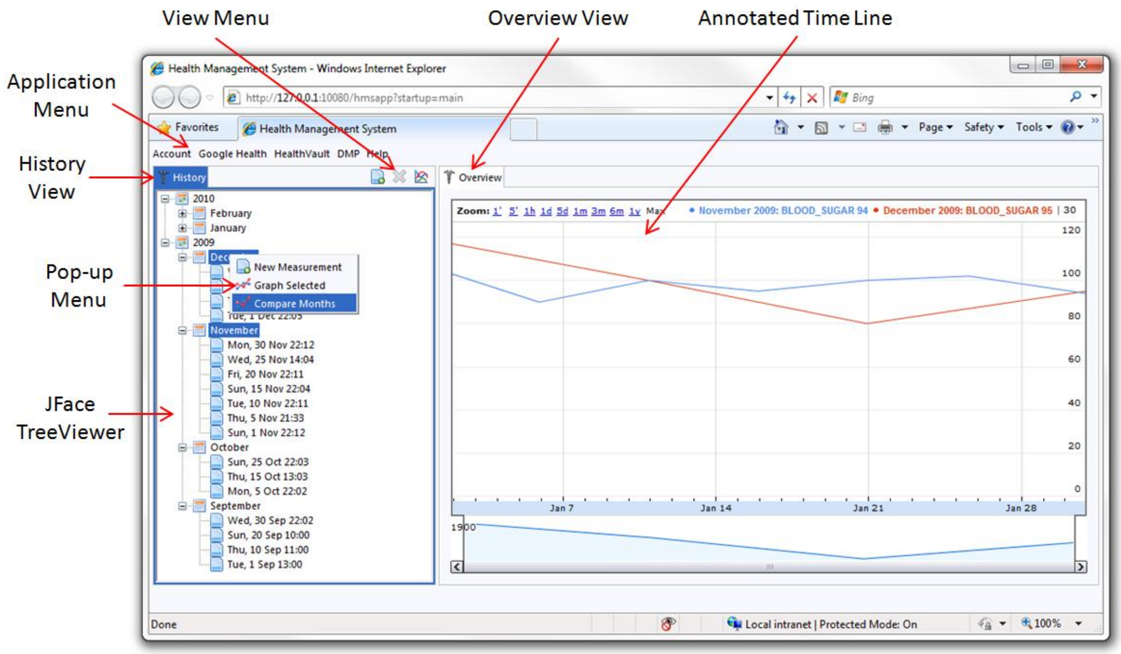

Figure 2: User interface

3.2 PHR Interoperability

Integration with third-party PHR providers is crucial to making a personal health information system useful to a wide circle of people. It enables users to take advantage of the existing PHR ecosystem instead of being locked-in to a single provider. To this end, the *HMS* supports importing blood glucose measurements directly from a linked Google Health or Microsoft HealthVault user account.

Both Microsoft and Google offer extensive support for developers, who are willing to integrate with their PHR platforms. This includes protocol and development documentation, tutorials, support forums, libraries for several programming languages and even a full development SDK, a Device Development Kit, and an application configuration program in the case of Microsoft (Sunyaev, Kaletsch, & Krcmar, 2010). These resources were used to implement seamless integration with each of the presented PHR platforms. At current project stage, only a one-way import function was implemented; however, data export can also be added with little incremental effort as soon as a data reconciliation strategy is established.

Experience proved the integration with both PHR platforms to be fairly similar, both in development effort and in the inner mechanisms. In both cases, a third-party application needs first to be registered with the PHR provider. In case of Google, this is done through registering a domain and getting a manual approval from the Google Health team. With Microsoft, registration is automated and is done by the means of generating a certificate in the HealthVault Application Manager and uploading it to the HealthVault Application Center.

After the application is approved, it can request a custom authentication URL to direct the user to a special login page where he can authorize this application to access his data. In case a user grants their permission for data access, the *HMS* acquires and stores the authorization token for this user in the database. This token can then be used to sign SOAP requests to the PHR platform that issued it. Both Microsoft HealthVault and Google Health respond with a XML string that contains the requested data. The *HMS* uses JAXB to construct Java objects from the received XML and present them to the user. User then selects the measurements they want to be persisted to their account. Because the *HMS* uses what HealthVault and Google Health call *offline access* (MSDN, 2009b) and *access for installed applications* (Google, 2009b) respectively, user does not need to reauthorize the system during each session, but rather the token can be reused for a an arbitrary number of requests until it is deleted.

4 Evaluation

The presented software architecture and the developed prototype are the main constituents of the effort to explore the feasibility of rich Internet applications for personal health information systems described in this article. At the present stage of the project the produced artifacts lend themselves to *formative evaluation*. According to (Scriven, 1991), formative evaluation is conducted during the development of a program by the in-house staff *with the intent to improve*. The following two subsections present a brief general architectural evaluation as well as a more detailed usability evaluation of the prototype.

4.1 Architecture Evaluation

Software architecture evaluation encompasses assessing to which extent the architecture fulfills the quality criteria that are derived from system requirements. Table 1 summarizes the relevant criteria for the HMS and their embodiment in the architecture of the system:

Criterion	Concretization in the Architecture
Interoperability	Support for SOAP communication protocol, data exchange in the ASTM Continuity of Care Record format.
Extensibility	Flexibility through the Eclipse RAP plug-in model, loose coupling in the multilayer architecture, clear separation of concerns.

Security	Industry- strength authentication and authorization through the EJB security, transport layer security through SSL, database encryption.
Ubiquity	Web application available on all PC platforms. Since application logic APIs can be exposed as web services, thin clients can be developed for mobile platforms with little effort.
Scalability	Enterprise JavaBeans and the Oracle database are highly scalable. In Eclipse RAP, clustering with load-balancing can be used to scale horizontally.

Table 1: Architecture evaluation summary

4.2 Usability Evaluation

Two popular approaches to evaluating the usability of user interfaces are user testing and usability inspection (Nielsen, 1994). While user testing is the most commonly applied method, it requires recruiting a sufficient number of users to test all the versions of an evolving design, which may be problematic given the temporal and budgetary constraints. Informal methods like inspection on the other hand are more cost-effective and are also capable to find problems overlooked by user testing. Hence, the two methods can be complimentary (Nielsen, 1994) and can be used as building blocks for constructing an evaluation method that is appropriate to a particular situation (see e.g., (Sunyaev, Hansen, & Krcmar, 2009)).

Cognitive walkthrough has been suggested as a usability inspection method that can be used early in the development cycle and can be conducted by developers alone (Wharton et al., 1994). Its essence is the description and evaluation of a hypothetical process—a conjecture about the steps user takes when faced with certain problems and situations, with the focus on the interplay between user's intentions on the one hand, and cues and feedback provided by the interface on the other (Wharton, et al., 1994).

The cognitive walkthrough was selected as an appropriate evaluation method due to the early project stage and the volatility of requirements. One of the goals of this scientific project was to explore the general suitability of Eclipse RAP to data-intensive personal health information systems, particularly in comparison to competing technology.

When viewed from the user's point of view, our experience with Eclipse RAP was largely positive. This framework allows creating web applications with much richer and smoother interactions than those possible with JavaServer Faces[3]. Particularly, Eclipse RAP offers a basis to solve process problems, data problems, and feedback problems in user interaction (Preciado, et al., 2005). For instance, the HMS RIA client allows completing a multi-step processes—editing account details on several screens—without refreshing the page. The developed solution also enables interactive data explorations, where user can select a number of measurements in the history view, graph them, and then pan and zoom to examine details without any interruptions. Finally, as illustrated in Fig. 4, RAP can provide immediate feedback for data validation, or even more sophisticated elements like progress bars and background tasks with notifications. However, Eclipse RAP creates usability problems of its own. As Lange points out, users associate appearance with certain behavior, and vice versa (Lange, 2008). Here, the challenges are twofold. On the one hand, Eclipse RAP looks most like a desktop

[3] http://java.sun.com/javaee/javaserverfaces/

application, so users may expect it to behave exactly like one. For instance they may press Ctrl+S on their keyboard, expecting the document to be saved, but instead will be presented with a browser "Save As… " dialog. On the other hand, while Eclipse RAP runs in a browser, it does not behave like a web page, in a way users are accustomed to. For instance, a RAP application cannot be scrolled in a browser, the "Back" button cannot be used, and text cannot always be selected, or images saved. Also, users do not usually expect a web application to provide functions through a right-click menu, as it is normally reserved to the Web browser.

There are several possible solutions to usability problems. Over time, users will recognize rich Internet applications as a distinct type of web applications and will learn to attach certain kind of expectations to them. This, along with the further evolution of RIA technologies will narrow the expectation gap. User acceptance will also depend on the ability of developers to acknowledge both the strong and weak sides of RIA frameworks and use them only where they are appropriate.

5 Lessons Learned

Compared to technologies like JavaServer Pages and JavaServer Faces, Eclipse RAP requires some additional configuration and integration work. In the case of *HMS*, it took us some time to integrate the RAP client with the rest of the JavaEE architecture. This included performing the *programmatic login* (Sun Microsystems, 2008) with the application server, as well as manually resolving the remote Enterprise JavaBeans through the Java Naming and Directory Interface. Also, the development environment, including a RAP runtime needed to be properly set up prior to development. This, however, is fairly simple in case the application is deployed into a Jetty Web server running in Equinox, as it is suggested in (Lange, 2008).

Nevertheless, after the initial learning curve was overcome, our development experience with RAP was positive and we managed to achieve high productivity. Particularly beneficial was the possibility to leverage our existing Eclipse RCP skills to build web clients with RAP. In the course of the project we did not write a single line of HTML, CSS, or JavaScript—all development was done in Java, which allowed us to focus on a sound object-oriented design. Some functions, like validations were significantly easier to implement in Eclipse RAP, than it would have been with JavaServer Faces. The platform proved to be stable which was quiet unexpected, given the relative youth of the project and the non-trivial browser interoperability problems Eclipse RAP has to solve. We tested our RIA client in major browsers and had no issues to report. Sometimes the side-effects of the aforementioned advantages start to present unique development challenges. For instance, script-based RIAs, and Eclipse RAP in particular, suffer from what Lange calls "No Web in Web" (Lange, 2008). In this case, the reliance on the framework to handle sessions, generate markup code and scripts backfires when custom behaviors need to be implemented for integration with third-party systems. RIAs often try to work around this limitation using IFrames, complex JavaScript and custom widgets. When developing the *HMS*, we experienced this deficiency as we implemented integration with external PHR platforms. Both Microsoft HealthVault and Google Health recommend an "online" approach to data access, where the application is reauthorized during each session, because it is more secure (Google, 2009a; MSDN, 2009a). This scenario requires redirecting the browser to the authorization page of the PHR provider and then back to the third-party system—a behavior that cannot be easily implemented with RAP. For this reason, as described previously, the *HMS* uses other type of access, where it stores an access token between sessions, but which is also less secure.

For developers, it is important to recognize that RIAs are not always the best choice for web clients and not to resolve to "design-by-buzzword." The main consideration in choosing between an ordinary web application and a rich Internet application is whether the system being developed is actually an application (Lange, 2008). Despite their advantages with regards to user interaction, RIA frameworks are not an optimal choice for creating regular web sites that primarily display static or dynamic content and need to adhere to a very specific design (Lange, 2008). Because the *HMS* is clearly an application, our choice to implement it as a RIA was justified.

6 Conclusion

This article established that patient self-management programs and the information systems that support them can provide a number of benefits to the patients. Recognizing that flaws in user experience can reduce the willingness of patients to adopt such systems, and that traditional web applications often fall short of user expectations in usability, rich Internet applications were explored as a possible alternative. It was further verified that Eclipse RAP can serve as a basis for building a user-friendly health self-management system that integrates with leading PHR platforms. While employing Eclipse RAP in a real-world development project, we found out that it is a mature platform that can be productively used by developers to build a RIA front-end for a Java EE system. Moreover, applications that are based on Eclipse RAP do not suffer from some of the architectural and usability drawbacks that traditional web applications have. This article provided a technology-oriented view on the conducted project. In the next project phase further input from medical professionals and patients will be gathered with an aim to expand the functions of the system and subsequently to evaluate its performance with regards to various stakeholders (e.g., as proposed by (Carson et al., 1998)). As the next step in our research, we are conducting a real-world evaluation with ten type 1 diabetes patients. This will result in detailed usage profiles for our disease self-management application which can lead to refined patient workflow support and new functions.

References

Bodenheimer, T., Lorig, K., & Holman, H. (2002). Patient Self-management of Chronic Disease in Primary Care. *Journal of the American Medical Association (JAMA), 288*(19), 2469-2475.

Bu, D., Pan, E., Walker, J., Adler-Milstein, J., Kendrick, D., Hook, J. M., . . . Middleton, B. (2007). Benefits of Information Technology–Enabled Diabetes Management. *Diabetes Care, 30*(5), 1137-1142.

Carson, E. R., Cramp, D. G., Morgan, A., & Roudsari, A. V. (1998). Clinical Decision Support, Systems Methodology, and Telemedicine: Their Role in the Management of Chronic Disease. *IEEE Transactions on Information Technology in Biomedicine, 2*(2), 80-88.

Chan, S. W. (2006). JDBCRealm in GlassFish Retrieved 01.11.2009, from http://blogs.sun.com/swchan/entry/jdbcrealm_in_glassfish

Clayberg, E., & Rubel, D. (2008). *Eclipse Plug-ins*: Addison-Wesley Professional.

Eclipse Foundation. (2006). Proposal for Rich AJAX Platform (RAP) Retrieved 01.11.2009, from http://www.eclipse.org/proposals/rap/

Farrell, J., & Nezlek, G. S. (2007). *Rich Internet Applications The Next Stage of Application Development*. Paper presented at the International Conference on Information Technology Interfaces, Cavtat.

Google. (2009a). Getting Started with Account Authorization, from http://code.google.com/apis/accounts/docs/GettingStarted.html

Google. (2009b). OAuth for Installed Applications Retrieved 01.12.2009, from http://code.google.com/apis/accounts/docs/OAuthForInstalledApps.html

Gulcu, C. (2002). *The Complete Log4j Manual: The Reliable, Fast and Flexible Logging Framework for Java* (1 ed.): QOS.ch.

Heisler, M. (2006). Building Peer Support Programs to Manage Chronic Disease: Seven Models for Success. Oakland, CA: California HealthCare Foundation.

Hoffman, C., Rice, D., & Sung, H. Y. (1996). Persons with chronic conditions. Their prevalence and costs. *Journal of the American Medical Association (JAMA), 276*(18), 1473-1479.

Holman, H., & Lorig, K. (2000). Patients as partners in managing chronic disease. Partnership is a prerequisite for effective and efficient health care. *BMJ, 320*(7234), 526-527.

Kanaan, S. B. (2008). Promoting Effective Self-Management Approaches to Improve Chronic Disease Care: Lessons Learned. Oakland, CA: California HealthCare Foundation.

Koutkias, V. G., & Malousi, A. (2010). A Personalized Framework for Medication Treatment Management in Chronic Care. *Information Technology in Biomedicine, IEEE Transactions on 14*(2), 464-472.

Lahdensuo, A., Haahtela, T., Herrala, J., Kava, T., Kiviranta, K., Kuusisto, P., . . . Liljas, B. (1998). Randomised comparison of cost effectiveness of guided self management and traditional treatment of asthma in Finland. *BMJ, 316*, 1138-1139.

Lange, F. (2008). *Eclipse Rich Ajax Platform: Bringing Rich Client to the Web*: Apress.

Mattila, E., Korhonen, I., Salminen, J., Ahtinen, A., Koskinen, E., Sarela, A., . . . Lappalainen, R. (2010). Empowering Citizens for Wellbeing and Chronic Disease Management with Wellness Diary. *Information Technology in Biomedicine, IEEE Transactions on 14*(2), 456-463.

Mikkonen, T., & Taivalsaari, A. (2008). *Web Applications - Spaghetti Code for the 21st Century*. Paper presented at the Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Application.

MSDN. (2009a). Choosing a HealthVault Application Architecture Retrieved 01.11.2009, from http://msdn.microsoft.com/en-us/healthvault/cc296309.aspx

MSDN. (2009b). Offline Access Retrieved 01.11.2009, from http://msdn.microsoft.com/en-us/healthvault/bb871490.aspx

Nanda, A. (2005). Transparent Data Encryption. *Oracle Magazine* (September-October).

Nielsen, J. (1994). *Usability Inspection Methods*. Paper presented at the Conference on Human Factors in Computing Systems, Boston, Massachusetts, United States

Noda, T., & Helwig, S. (2005). Rich Internet Applications. Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA *UW E-Business-Consortium Opinion Papers*.

O'Rourke, C. (2004). A Look at Rich Internet Applications. *Oracle Magazine*(July/August).

Peters, K., Niebling, M., Slimmer, C., Green, T., Webb, J. M., & Schumacher, R. (2009). Usability Guidance for Improving the User Interface and Adoption of Online Personal Health Records. Oakbrook Terrace, IL: User Centric, Inc.

Preciado, J. C., Linaje, M., Sanchez, F., & Comai, S. (2005). *Necessity of methodologies to model Rich Internet Applications*. Paper presented at the Seventh IEEE International Symposium on Web Site Evolution, 2005. (WSE 2005).

Sawicki, P. T. (1999). A Structured Teaching and Self-management Program for Patients Receiving Oral Anticoagulation. A Randomized Controlled Trial. *Journal of the American Medical Association (JAMA), 281*, 145-150.

Scriven, M. (1991). *Evaluation Thesaurus* (4 ed.): Sage Publications.

Sun Microsystems. (2008). The Java EE 5 Tutorial. For Sun Java System Application Server 9.1. Santa Clara, CA SunMicrosystems, Inc.

Sunyaev, A., Chornyi, D., Mauro, C., & Krcmar, H. (2010). *Evaluation Framework for Personal Health Records: Microsoft HealthVault vs. Google Health*. Paper presented at the Proceedings of the Hawaii International Conference on System Sciences (HICSS 43), Kauai, Hawaii.

Sunyaev, A., Hansen, M., & Krcmar, H. (2009). Method Engineering: A Formal Description *Information Systems Development - Towards a Service Provision Society* (pp. 645-654): Springer US.

Sunyaev, A., Kaletsch, A., & Krcmar, H. (2010). *Comparative Evaluation of Google Health API vs. Microsoft Healthvault API*. Paper presented at the Proceedings of the Third International Conference on Health Informatics, Valencia, Spain.

Undem, T. (2009). Gaps in the System: Californians Struggle with Caring for Their Chronic Conditions. Washington, D.C: Lake Research Partners.

Von Korff, M., Gruman, J., Schaefer, J., Curry, S. J., & Wagner, E. H. (1997). Collaborative Management of Chronic Illness. *Annals of Internal Medicine, 127*(12), 1097-1102.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). The cognitive walkthrough method: a practitioner's guide *Usability inspection methods* (pp. 105-140). New York, NY, USA: John Wiley & Sons, Inc.