Architecture and Design of a Patient-Friendly eHealth Web Application: Patient Information Leaflets and Supplementary Services

Tobias Dehling
Department of Information Systems, University of Cologne, Cologne, Germany., dehling@wiso.uni-koeln.de

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

Architecture and Design of a Patient-Friendly eHealth Web Application: Patient Information Leaflets and Supplementary Services

Tobias Dehling
University of Cologne
dehling@wiso.uni-koeln.de

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

ABSTRACT

Patients benefit from information on pharmaceuticals and most patients are willing to read patient information leaflets for their pharmaceuticals. However, the quality of written information on pharmaceuticals leaves room for improvement. To mend insufficiencies of patient information leaflets, an alternative approach for provision of information on pharmaceuticals is illustrated. We present the design and architecture of a web application that provides information in patient information leaflets as well as supplementary services. With a web application supplementary services that cannot as easily be realised with patient information leaflets can be offered. An open-source framework with a robust architecture for rapid application development serves as a foundation of the web application. In combination with the proposed design and architecture, this leads to an extensible, reliable, scalable, customisable and patient-friendly web application with high availability.

Keywords

web application, usability, patient information leaflet, written information, pharmaceutical, architecture, design

INTRODUCTION

Providing patients with information leads to positive effects (Johansson, Katajisto and Salanterä, 2010; Sheard and Garrud, 2006) and most patients are inclined to read patient information leaflets for their pharmaceuticals (Fuchs, Banow, Görbert and Hippius, 2007; Rajasundaram, Phillips and Clay, 2006). In the European Union supply of patient information leaflets is even mandatory with every supply of a pharmaceutical (Council of the European Communities, 1992).

Nevertheless, written information on pharmaceuticals is often in need of improvement: The provided font size is too small (Chubaty, Sadowski and Carrie, 2009; Luk, Tasker, Raynor and Aslani, 2010; Winterstein, Linden, Lee, Fernandez and Kimberlin, 2010). Further formatting aspects like line spacing or margins are not appropriately chosen (Chubaty et al., 2009, Winterstein et al., 2010). Readability is impaired by printing on unsuitable paper with text showing through or other distractions (Chubaty et al., 2009). The required reading level is too high (Wallace, Roskos and Weiss, 2006; Winterstein et al., 2010) and many difficult/technical terms are used (Rajasundaram et al., 2006). Headings and bullet points could be used more effectively for text structuring (Luk et al., 2010). Moreover, an excessive amount of information as well as irrelevant information is provided (Rajasundaram et al., 2006; Wallace et al., 2006; Winterstein et al., 2010).

The aforementioned issues of written information on pharmaceuticals were determined in various studies in English-speaking countries. Distribution systems and quality of written information on pharmaceuticals vary from country to country (Luk et al., 2010; Raynor, Svarstad, Knapp, Aslani, Rogers, Koo, Krass and Silcock, 2007). Provision of written information can be voluntary or be mandatory with every or only the first supply of a pharmaceutical. Written information can be offered as package insert or leaflet/printout at the source of supply. In the U.S. some pharmaceuticals are even distributed without any written information besides the short instructions on the container and information for patients is not standardised (Winterstein et al., 2010).

However, written information on pharmaceuticals should not be considered lacking in general. Sundry pharmaceutical manufactures and third parties produce good patient information leaflets. Written information from Australia and New Zealand performs well in international comparisons (Luk et al., 2010; Raynor et al., 2007). Nevertheless, written information on pharmaceuticals has potential for improvement. In this paper, we will focus on information in German patient information leaflets for which similar problems regarding readability, comprehensibility, and content have been reported (Fuchs et al., 2007; Nink and Schröder, 2005).

A web application for the presentation of information in patient information leaflets can mend insufficiencies of patient information leaflets. In contrast to patient information leaflets, a web application is not limited to a sheet of paper, can be

easily accessed prior to purchase of the pharmaceutical, and can ease locating of desired information. Moreover, a web application can more comfortably point users to further information, be updated easily, and offer a customisable structure as well as interactive content. The web application has access to information on many pharmaceuticals at once. Thus, supplementary services that aggregate information on pharmaceuticals can be offered. Appearance and range of offered features can be tailored by/for the individual user. More and more users have not only internet access at home but also on their mobile devices so that the web application can be accessed and consulted at any time.

In this paper, we present the design and architecture of a prototype for a web application that provides information in patient information leaflets in combination with supplementary services. Objectives for the design of the web application were availability, extensibility, reliability, scalability, security, and usability. The remainder of the paper is structured as follows: In the chapter 'Web Applications' we briefly introduce web applications. Afterwards, we describe the chosen design and architecture in the chapter 'Architecture and Design'. In the chapter 'Achievement of Design Objectives' we succinctly correlate the chosen design and architecture with the design objectives. Subsequently, we discuss the lessons learned during the project in chapter 'Lessons Learned'. Finally, we provide the conclusion and an outlook in the chapter 'Conclusion'.

WEB APPLICATIONS

Information on pharmaceuticals could be obtained from a database, formatted with HTML, and presented to the user as static web pages. However, using a web application is a more sophisticated approach that leads to a flexible and adaptable service. We chose the general-purpose programming language (GPL) Java for the realisation of the web application. A Java "web application is a collection of servlets, html pages, classes, and other resources that make up a complete application on a web server" (Sun Microsystems, 2001). "A servlet is a Java technology based web component, managed by a container, that generates dynamic content. Like other Java-based components, servlets are platform independent Java classes that are compiled to platform neutral bytecode that can be loaded dynamically into and run by a Java enabled web server" (Sun Microsystems, 2001).

Hence, a web application is a fusion of conventional desktop applications with web pages. Since it is programmed in a GPL, the application logic can perform the same operations as a desktop application and established algorithms as well as third party libraries can be utilised. Yet, clients can access the application everywhere with any browser that supports JavaScript. Additionally, a web application can utilise/integrate further offerings of the internet. This ranges from links to other information sources, like Wikipedia (http://www.wikipedia.org) or specialised medical information, to the integration of services like personal health records. Integration with personal health records could be utilised to automatically provide personalised information based on a user's medical information. Integration with personal health records was demonstrated with a similar web application that offers an internet-based chronic disease self-management system (Sunyaev and Chornyi, 2011; Sunyaev and Chornyi, 2012). Yet, integration with personal health records would introduce privacy and trust challenges that need to be assessed and addressed (Kaletsch and Sunyaev, 2011).

In order to offer information on pharmaceuticals and supplementary services, the web application has to be able to perform three basic tasks.

- Search for pharmaceuticals: Enable users to specify some parameters and search corresponding pharmaceuticals in the underlying database.
- Display information on pharmaceuticals: Enable users to view patient information leaflet information.
- Supplementary services: Offer supplementary services like refining the displayed information, linking to similar information on other pharmaceuticals, or aggregating information on pharmaceuticals.

ARCHITECTURE AND DESIGN

The architecture is divided in three main application modules that handle the Graphical User Interface (GUI), the application logic, and access to the database. We chose a modular architecture because it can be easily extended and adapted. For example, if we were to offer a dedicated user interface for mobile devices in the future, we could implement additional modules implementing the user interface for various platforms like Android or iOS and still use the same modules providing the application logic and access to the database. Similarly, we could easily use another database by adopting the module for data access accordingly. The GUI is the part of the application visible to the user. Consequently, the GUI constitutes an important part of the application because users interact with the GUI and remaining parts of the application are transparent to users. In the following, we present the main characteristics and aspects of the GUI. Subsequently, we illustrate the physical architecture of the application.

User Interface

We based the application on the Vaadin Framework (https://vaadin.com). Vaadin is a Java framework for the development of web-based user interfaces (Grönroos, 2011). Vaadin hides web technologies like HTML, JavaScript or AJAX from the developer so that the user interface can be implemented by writing Java classes. Thus, the resulting software is a web application. Developing with Vaadin leads to a thin-client architecture, where everything except the display of the user interface and the capture of user interaction is handled on the server side. Since the application logic is hidden from the client side, security is increased. Additionally, executing the application logic mainly on the server side eases integration of further components, like the database providing information on pharmaceutical or third party libraries, because the integration affects only the server side. Rendering of the user interface on the client side is handled by an abstraction tier that ensures compatibility with common browsers. Developers are only required to interact with the abstraction tier so that they are relieved of effort for managing the client side like handling browser compatibility issues. Therefore, developers can compose the user interface by adding, adopting, and expanding widgets provided by the framework. Widgets are GUI modules like buttons, text boxes or lists that can be combined to create the user interface.

The main feature of the user interface is its flexibility. Different users prefer different GUI designs. Inexperienced users are likely to prefer a simple GUI, which provides only the most important controls and settings. On the contrary, more experienced users might prefer fast access to the available functionality and do not mind a GUI that displays multiple settings and controls at once. To satisfy both preferences and avoid confusion of inexperienced users, initially, we display only the header and a single tab that displays introductory information. Users can choose between a simple view, an advanced view, or individually configure the displayed GUI components. Consequently, users can configure the application according to their preferences with minimal effort.

Multiple components can be used for the same application configurations. Some users prefer, for example, to select the visible sections of information on pharmaceuticals in a sidebar while other users prefer a sub-window that they can close after selection. Satisfying both preferences, requires two GUI components which manage the same data. Redundant management of the data is superfluous and bothersome. Keeping the data consistent consumes precious computing power and becomes more complex with a rising number of components that work with the same data. Hence, we adopted the Model View Controller (MVC) pattern to manage the data required by the GUI components.

MVC basically consists of three entities for data management, which are called model, view, and controller. The model is used to store the data in a single location. Views visualise the data stored in the model in similar or completely different ways for the users. User interaction with a view is captured and expressed in the model by the controller, conversely, the controller translates and expresses changes of the model in the views. Tasks of the controller are mainly handled by Vaadin.

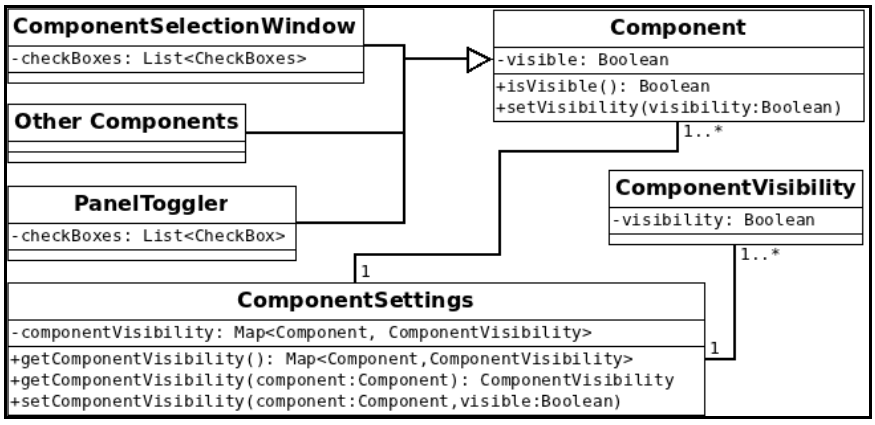

Figure 1. MVC Pattern Example

As an example for our MVC implementation the visibility management of the individual GUI components is illustrated with the class diagram in figure 1. The classes ComponentSettings and ComponentVisibility represent the model. In Component-Settings every GUI component whose visibility can be configured by users is associated with a ComponentVisibilty object which stores the visibility of the component. PanelToggler is a view for the model and offers check boxes that users can utilise to toggle the visibility for a selection of GUI components. Another view is the ComponentSelectionWindow, which, in contrast to the PanelToggler, offers check boxes for all GUI components that can be hidden and opens as a sub-window.

Vaadin manages the state of the check boxes and sets the flag in the ComponentVisibilty objects accordingly. Another controller task, which the application needs to handle, is the actual enabling/disabling of the GUI components. This task is handled by the ComponentSettings object as well so that it serves as model and controller simultaneously. The Component-Settings object already has the references to the components and toggles the visibility of these based on events that are fired by the ComponentVisibility objects and indicate a changed visibility value.

Adoption of MVC kept configuration of the application manageable. Otherwise, it would have been rather complex to enable users to customise the application according to their preferences and maintain consistency. Hence, we developed a flexible GUI that users can personalise according to their preferences. Nevertheless, the GUI can be easily modified or extended due to its modularity and utilisation of the MVC pattern. Offering various GUI components does not impede performance because components are only instantiated when they are needed.

Physical Infrastructure

The physical infrastructure of the web application is depicted in figure 2. When a client starts the application a session is initiated on a Java application server. In order to service many clients simultaneously and to match the available computing power to a changing number of clients, the application is deployed to a cluster of Java application servers. Each node in the cluster hosts the application and a load balancing node redirects client requests based on a load balancing strategy like, for example, redirecting clients to the node with the lowest load. By adding further nodes to or removing nodes from the cluster the available computing power can be matched to the demand of clients. Additionally, the load balancer avoids situations in which some nodes are overburdened while others are idling. Availability is increased through session replication so that sessions and tasks of a failed node can be handled by other nodes. The nodes access the database over a private network.

Similar to the application server cluster, the database cluster uses replication to meet scalability and availability needs. Database services are provided by multiple physical nodes hosting a replicated version of the database. One node serves as a master and processes write requests which are replicated to the other nodes. The other nodes process read requests, which, in our case, represent most accesses to the database. Write accesses are only necessary to update the provided information and not required for the handling of client sessions. Hence, one node is sufficient to handle database write operations. If a future extension leads to more write operations initiated by users, like storage of user settings in the database, than one node can handle, it will be possible to adopt the replication strategy accordingly. The database nodes are also managed by a load balancer. Except for redirecting write accesses to the master of the database nodes, it works like the load balancer of the application server cluster. Scalability is facilitated through adding or removing physical nodes. Availability is ensured to a high degree because SQL statements assigned to crashed servers can be handled by available instances of the database.

The physical infrastructure can be easily adopted to variable demands and a changing environment. Since the application runs in the Java Virtual Machine (Sun Microsystems, 2001), the web application requires no specific hardware as long as a fitting Java Runtime Environment is available.

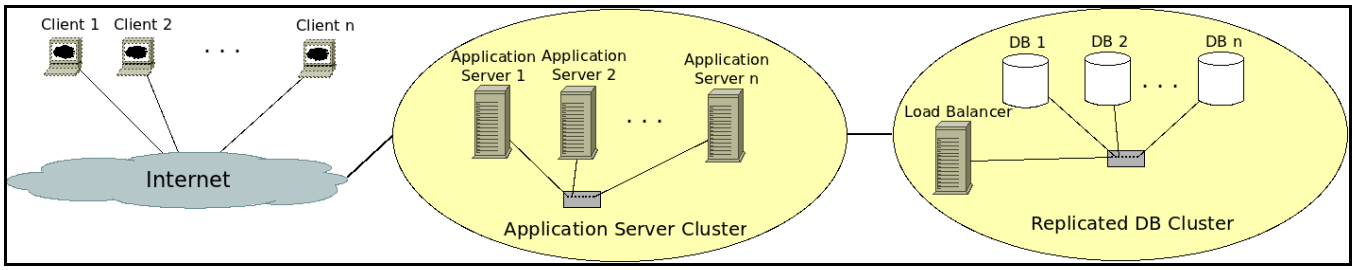

Figure 2. Physical Infrastructure

Obtaining the Required Data

When thinking about acquisition of the required data three alternatives come to mind: digitising the information on one's own, data input by third parties or utilisation of existing data. Developing an individual data model and digitising information in patient information leaflets on one's own is probably the most effective way to ensure that the provided data fits the needs of the application and achieves desired quality levels. However, this approach is inefficient since much time would be wasted on data input. Data input by third parties seems quite promising; the workload could be distributed between multiple entities, data quality could be ensured by input interfaces, and an own data model that soundly supports the web application could be

used. Yet, third parties might not be motivated to help create a database for a web application in its infancy. Getting support from third parties is more likely once the web application is operational and third parties deem it beneficial. This might be useful to improve the provided information later on. Another problem that occurs in our case is that we need to provide reliable information in order to avoid misinforming users on pharmaceuticals which could obviously lead to serious and harmful consequences. Since checking reliability of information with software is too complex and difficult and no software is without defects, data input by third parties would require too much effort for checking the reliability and validity of the entered information. Hence, we decided to use existing data.

Downsides of using existing data are having little influence on the formatting of the data and the need to cope with used abbreviations, technical terms, or concatenations of information as well as the general data model and inconsistencies in the data. Further insufficiencies of the used database were that it contained a lot of unnecessary and redundant information, foreign key restrictions were not specified and the data model lacked documentation. To alleviate the situation we developed an own data model, sifted through the source database, and copied useful data in a new database that uses our data model. We added primary key restrictions and foreign key relationships so that connections between data are expressed and enforced on the database level. Moreover, we removed inconsistencies and transferred only the information we deemed useful.

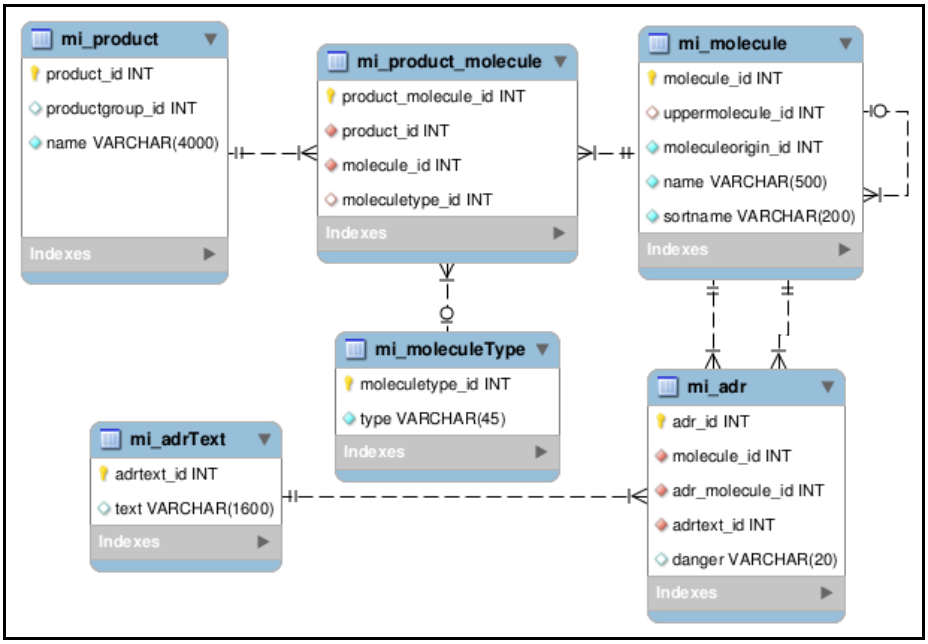

Figure 3. Extract of Database Model

The following example illustrates an extract of the resulting database model as well as a supplementary service for the detection of adverse drug reactions that utilises the data. Taking multiple pharmaceuticals in combination might lead to some undesired effects. One pharmaceutical might, for example, lessen the effectiveness of another pharmaceutical. These effects are called adverse drug reactions. Checking a set of pharmaceuticals for adverse drug reactions can be useful for users if they want to take a pharmaceutical that requires no prescription, like Aspirin, and would like to know whether the pharmaceutical can be taken in combination with other pharmaceuticals they are taking on a regular basis. Reading all relevant package information leaflets would be a time-consuming task. The web application, on the other hand, is well suited for this task because it can access and process relevant information much faster as long as the information is stored in the database. Prior to the detection of adverse drug reactions users need to add pharmaceuticals to the set of pharmaceuticals they want to check for adverse drug reactions. From now on we will refer to this set with the term 'selection set'. The actual algorithm works as follows: At first it is necessary to retrieve the required data from the database. The relevant tables are depicted in figure 3. Pharmaceuticals are stored in mi_product and ingredients of pharmaceuticals, which are hold in the table mi_molecule, are associated with pharmaceuticals in the table mi_product_molecule. Adverse drug reactions are expressed in the table mi_adr. They are stored based on ingredients of pharmaceuticals. Each recorded adverse drug reaction specifies a pair of ingredients that cause an adverse drug reaction. Additionally, a rating for the graveness of an adverse drug reaction and a text describing the adverse drug reaction are provided. The description is stored in mi_adrText. For every pharmaceutical in the selection set

a set with all associated molecules that are an ingredient of the pharmaceutical is constructed. Moreover, additional sets are constructed for every molecule of a chosen pharmaceutical. In the following we will refer to these molecules as 'primary molecules'. The sets contain the adverse drug reaction (ADR) molecules associated with the primary molecules. With the term 'ADR molecules' we refer to a molecule that has an adverse drug reaction with a primary molecule. After the initialisation we iterate over all pharmaceuticals in the selection set and check whether any set of ADR molecules of its ingredients intersects with the ingredients of another pharmaceutical in the selection set. Intersections other than the empty set constitute detected adverse drug reactions because an ingredient of another pharmaceutical has been found.

ACHIEVEMENT OF DESIGN OBJECTIVES

Relevant design objectives for the architecture and application are availability, extensibility, reliability, scalability, security and usability. Characteristics contributing to the achievements of the objectives are illustrated in table 1. The section 'Benefits for Patients' provides more detailed usability elaborations.

Objective	Contributing Characteristics
Availability	Application and database run on multiple physical computing nodes. Failed nodes can be seamlessly replaced by other nodes. Repercussions of external influences like a fire or blackout can be reduced by using nodes in multiple geographical locations.
Extensibility	Modular architecture and object orientation facilitate extension of application and offered features. Encapsulation of functionality reduces inter-module dependencies. Server-side architecture allows for easy integration of third party libraries. No redundant storage of data.
Reliability & Security	Database connection via private network. Web application has only read access on database. Utilisation of trustworthy sources of information. No storage of user data. Hiding of application logic from client side.
Scalability	The available processing power can be adapted to changing performance needs by adding/removing computing nodes.

Table 1. Characteristics Contributing to Achievement of Design Objectives

Benefits for Patients

As mentioned before, patients are troubled by the content, readability, and comprehensibility of patient information leaflets. The web application mends insufficiencies related to content by allowing users to select which information should be displayed so that only the information relevant for the individual user is provided. In contrast to leaflets that can vary depending on the manufacturer or pharmacy, the application maintains a consistent presentation of the information. While users can customise the amount of displayed information, the appearance and structure remains unchanged for all pharmaceuticals. Readability is improved with an adjustable font size, a well-arranged layout, which bundles related information, and a navigation section that allows easy location of desired information. However, a web application is not the right approach to solve comprehensibility issues since these are rooted in the data itself and need to be addressed with enhancement of the information. Comprehensibility can be improved a little through linking to explanations and more detailed, specialised information.

Moreover, the web application offers supplementary services that cannot be provided by patient information leaflets in such a comfortable and easy way. The web application can link to information on related pharmaceuticals as well as more detailed information or explanations. Information on multiple pharmaceuticals can be displayed at once. Extensive analyses like checking for adverse drug reactions in a user-defined set of pharmaceuticals can be computed. Since the web application runs mainly on the server side and is implemented in a GPL, various additional libraries can be easily integrated to provide more services for patients. We utilised, for example, a Java library for PDF creation to enable users to print the currently displayed information without any navigational elements.

LESSONS LEARNED

Gaining proficiency in employing the Vaadin framework required some practice. However, after a few tutorials and study of the manual we were able to efficiently implement the desired functionality and user interface components. The Vaadin framework proved to be a useful technology for the realisation of a web application and relieved us from a lot of work and

complexity. Client-side aspects of the application are hidden from developers and we could focus on implementation of the application logic and the realisation of the customisable user interface. Yet, it had to be kept in mind that we were developing a web application and not a conventional desktop application. In a web application, JavaScript code that represents the user interface components needs to be rendered on the client side. Hence, it is important to keep the user interface simple and, thereby, minimise the amount of JavaScript that needs to be rendered on the client side. It was enjoyable to implement the web application in Java and be relieved of common web page issues like browser compatibility. Only a negligible amount of HTML and CSS was necessary for adopting appearance and markup of textual information. This constitutes another feature of the Vaadin framework: It is not necessary to concern oneself with the client side, yet, it is possible to modify the client side. Possibilities for modifications of the client side range from simple CSS scripts to the development of completely new GUI components (Grönroos, 2011).

On the other hand, obtaining the required data is laborious. The relevant data is not available in a standardised, easy to process format and lacks semantic information or atomicity. In our case, most information was represented in strings. Therefore, presentation of the information could be improved even more if the available data (individual strings) were not too ambiguous to identify distinct information. Furthermore, this would facilitate further supplementary services. Enhancements of the data are not easily realised. Information on pharmaceuticals needs to be accurate because misinforming users could obviously lead to serious and harmful consequences. Letting users improve the provided information would not meet reliability constraints. Programmatic enhancements evoke similar concerns because software is not free of defects.

Compared to conventional web page development, we encountered some difficulties during implementation. The browser back button does not work because the web application uses a single uniform resource identifier (URI). Vaadin offers functionality to generate custom URI fragments (Grönroos, 2011; Berners-Lee, Fielding and Masinter, 2005). However, we decided to abstain from URI fragment utilisation because of the multitude of possible application states due to the high flexibility of the application. The complexity introduced by this functionality clearly outweighs is usefulness. Scrolling to a specific section of information was tricky as well. Another problem is that the application requires the common sense of users. Since we wanted to avoid restricting users, it is possible that users trigger processes that do not return almost instantly like searching for all pharmaceuticals that contain an 'a'. If future user tests show that such tasks are required, it might be a good idea to add parallel computing capabilities to the application to handle more complex tasks.

CONCLUSION

A web application is a suitable technology to provide information in patient information leaflets as well as supplementary services like aggregation and refinement of the available information. Moreover, such a web application can mend insufficiencies of patient information leaflets regarding readability and content. On the other hand, comprehensibility is only enhanced a little. Comprehensibility issues are better targeted by improving the data itself. We presented a design and architecture that satisfies software system attribute targets like reliability or scalability and is capable of providing information in patient information leaflets as well as supplementary services. The Java framework Vaadin proved to be a useful subsidiary library and relieved us of having to deal with common web development issues like browser compatibility. Nearly the whole application could be developed in the GPL Java. The illustrated web application is accessible with any browser that supports JavaScript. On devices with small resolutions/display sizes, like smart phones, usability could be improved with a dedicated user interface. Due to the modular architecture an alternative GUI could be easily added.

An important aspect for future considerations is the quality of the data in the underlying database. Enhancements of the data quality could be used to improve the quality of the offered services. In this paper we focused on German patient information leaflets, however, with a corresponding data source the application could be used for information on pharmaceuticals in any language. A further object for future research is integration with a healthcare telematics infrastructure like the one currently being established in Germany. This would facilitate personalised services like displaying information on the pharmaceuticals a user is currently taking or checking these pharmaceuticals for adverse drug reactions. However, such services require and create sensitive information for which security aspects need to be considered and users need to overcome their privacy concerns (Angst and Agarwal, 2009).

All in all, the web application constitutes a useful alternative/addition for provision of information in patient information leaflets as well as additional services. Additional services as comfortable cannot as easily be offered by patient information leaflets.

ACKNOWLEDGEMENTS

We thank Sebastian Dünnebeil at the Technische Universität München for his helpful advice and support.

REFERENCES

1. Angst, C. M. and Agarwal, R. (2009) Adoption of Electronic Health Records in the Presence of Privacy Concerns: The Elaboration Likelihood Model and Individual Persuasion, MIS Quarterly, 33, 2, 339-370.
2. Berners-Lee, T., Fielding, R. and Masinter, L. (2005) RFC3986. Uniform Resource Identifier (URI): Generic Syntax, The Internet Society, http://tools.ietf.org/html/rfc3986.
3. Chubaty, A., Sadowski, C. A. and Carrie, A. G. (2009) Typeface legibility of patient information leaflets intended for community-dwelling seniors, Age and Ageing, 38, 4, 441-447.
4. Council of the European Communities (1992) Council Directive 92/27/EEC of 31 March 1992 on the labelling of medicinal products for human use and on package leaflets, Official Journal L113 of 30.04.1992, 8-12.
5. Fuchs, J., Banow, S., Görbert, N. and Hippius, M. (2007) Importance of Package Insert Information in the European Union, Pharmazeutische Industrie, 69, 2, 165-172.
6. Grönroos, M. (2011) Book of Vaadin, Vaadin Ltd, https://vaadin.com/book.
7. Johansson, K., Katajisto, J. and Salanterä, S. (2010) Pre-admission education in surgical rheumatology nursing: towards greater patient empowerment, Journal of Clinical Nursing, 19, 21/22, 2980-2988.
8. Kaletsch, A. and Sunyaev, A. (2011) Privacy Engineering: Personal Health Records in Cloud Computing Environments, in ICIS 2011 Proceedings (ICIS 2011), December 4-7, Shanghai, China, International Conference on Information Systems, paper 2.
9. Luk, A., Tasker, N., Raynor, D. K. and Aslani, P. (2010) Written Medicine Information from English-Speaking Countries – How Does It Compare?, The Annals of Pharmacotherapy, 44, 2, 285-294.
10. Nink, K. and Schröder, H. (2005) Zu Risiken und Nebenwirkungen: Lesen Sie die Packungsbeilage?, Wissenschaftliches Institut der AOK, Bonn, Germany.
11. Rajasundaram, R., Phillips, S. and Clay, N. R. (2006) Information leaflet used in out-patient clinics: A survey of attitude and understanding of the user, International Journal of Health Care Quality Assurance, 19, 7, 575-579.
12. Raynor, D. K., Svarstad B., Knapp, P., Aslani, P., Rogers, M. B., Koo, M., Krass, I. and Silcock, J. (2007) Consumer medication information in the United States, Europe, and Australia: A comparative evaluation, Journal of the American Pharmacists Association, 47, 6, 717-724.
13. Sheard, C. and Garrud, P. (2006) Evaluation of generic patient information: Effects on health outcomes, knowledge and satisfaction, Patient Education and Counseling, 61, 1, 43-47.
14. Sun Microsystems (2001) Java Servlet Specification, Version 2.3. JSR 53, Sun Microsystems, Palo Alto, CA, USA.
15. Sunyaev, A. and Chornyi, D. (2011) Development of an Internet-Based Chronic Disease Self-Management System, in Proceedings of the 24th International Bled eConference (Bled 2011), June 12-15, Bled, Slovenia, Bled eCommerce Conference, 185-196.
16. Sunyaev, A. and Chornyi, D. (2012) Supporting Chronic Disease Care Quality: Design and Implementation of a Health Service and its Integration with Electronic Health Records, ACM Journal of Data and Information Quality, 3, 2.
17. Wallace, L. S., Roskos, S. E. and Weiss, B. D. (2006) Readability Characteristics of Consumer Medication Information for Asthma Inhalation Devices, Journal of Asthma, 43, 5, 375-378.
18. Winterstein, A. G., Linden, S., Lee, A. E., Fernandez, E. M. and Kimberlin, C. L. (2010) Evaluation of Consumer Medication Information Dispensed in Retail Pharmacies, Archives of Internal Medicine, 170, 15, 1317-1324.