Mobile Platform for Executing Medical Business Processes and Data Collecting

Jerzy Brzeziński, Anna Kobusińska, Jacek Kobusiński, Andrzej Stroiński, and Konrad Szałkowski

Institute of Computing Science, Poznań University of Technology
Piotrowo 2, 60-965 Poznań, Poland
{jkobusinski,akobusinska,astroinski}@cs.put.poznan.pl
jbrzezinski@put.poznan.pl, kszalkowski@gmail.com

Abstract. Medicine becomes more and more complex domain. The process from patient registration through to the provision of the right treatment becomes complex and sophisticated, and any mistakes or inaccuracies can have a significant consequences for both the patient and the care professionals. This paper presents a MMDCP — Medical Mobile Data Collecting Platform, which aims is to reduce the error rate and speed up the process of data collection. Since medical staff should have the access to medical data from any place of the healthcare facility, the MMDCP provides it's users the mobility. The proposed platform also distinguishes itself with flexibility, simplicity of maintenance and integration.

Keywords: data collecting, business process, electronic health record, integration, SOA, REST.

1 Introduction

Medicine becomes more and more complex domain. The process of patient treatment, the amount of various drugs treatments available, and other aspects of modern healthcare become increasingly sophisticated and difficult. On the other hand, the expectation of a greater efficiency means that the time spent on the execution of medical tasks is shrinking.

These aspects can lead to errors and mistakes unacceptable in any medical environment. In the process from patient registration through to the provision of the right treatment, any mistakes or inaccuracies can have significant consequences for both the patient and the care professionals.

While the training, procedural checks, double checks and clear processes can reduce error rate, the humans are still the weakest link. Care professionals are required to record information on the performed medical procedures and the medications given to patients. Unfortunately, when humans read or transcribe information, there is a possibility that this process will lead to an error. Since the process of recording medical information is relatively time consuming, it is often carried out post-factum, which only increases probability of mistake.

G. Huang et al. (Eds.): HIS 2013, LNCS 7798, pp. 149–159, 2013.

Therefore it is important to provide tools for medical staff, which will support collecting data in a way that requires less attention and time to do so. As care professionals have administrative tasks that are frequently very time-consuming, there is only a limited time available for the patient. By maximizing the efficiency of such tasks with the use of just mentioned tools, more time can be spent providing quality patient care. Moreover, the acquired data can be analyzed and utilized to automatically fill in patient electronic health record (EHR).

Another crucial feature of considered a tool is mobility. Medical staff should have an access to the medical data from any place of the healthcare facility, whether they are in the hospital pharmacy or close to the patient's bed on the ward.

According to HIMSS [17] the significant benefits can be achieved, e.g., by using barcode technology in patient registration and admission process, patient safety, clinical care delivery, tracking and accounting, product logistics management coordination. The problem with practical implementation is the real workflow and business processes that occur in healthcare units e.g. hospitals or other long term care facilities are very individual and unique. Thus the information system that supports them should be very flexible, easily configurable and transparent for the end user.

To address above mention problems, in this paper we propose a MMDCP — a mobile platform that supports automatic medical data collection. The proposed platform provides mobility, simplifies and speeds up the medical data entry process.

The paper is organized as follows. In Section 2 we describe the design and implementation of the platform. Next, in Section 3 the approach to integration of our platform with HIS systems is presented. Section 4 shows the practical application of the platform. Finally, in the last Section we conclude the paper.

2 Mobile Medical Business Process Architecture and Execution Environment

The implementation of MMDCP medical platform proposed in this paper is based on the Service Oriented Architecture (SOA) [16]. According to the SOA, the system functionality is distributed among independent applications, called web services. Such web services can be composed into so called business process when more complex and sophisticated functionality is required. In the result of SOA-based architecture of MMDCP, the functionality of the proposed platform can be easily expanded without changing already offered functions, which is important in case of all medical systems. Additionally, due to the service-oriented architecture of MMDCP, an easy and flexible integration of the proposed platform with various HIS systems is possible.

Nowadays, one of the popular approaches to implement SOA is the RPC-based approach, where Web Services and business process are implemented with a huge stack of standards like: WSDL [9], SOAP [7], BPEL [12] or set of WS-* standards (eg. [11], [10]). However, at present, also a new approach called REST [15] is getting more and more attention. In contrast to the conventional

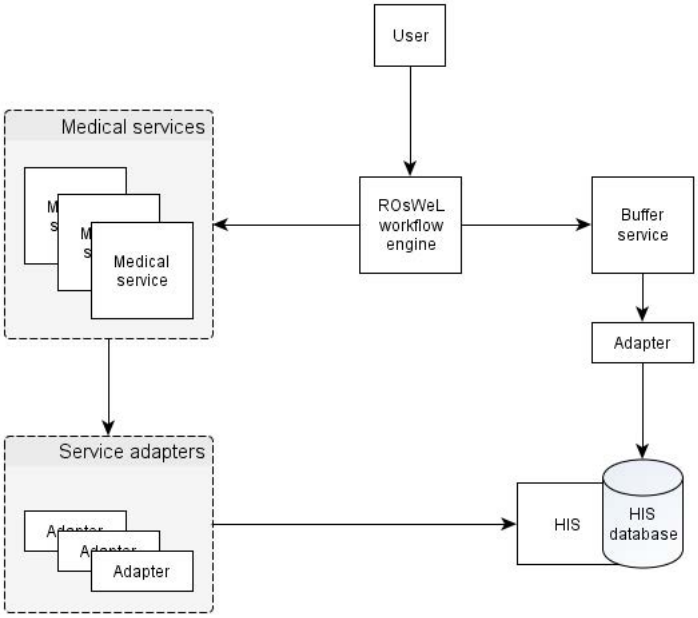

Fig. 1. Architecture of mobile medical business process platform (MMDCP)

Web Service, RESTful web services [18] are lightweight, fast and well suited to the current web architecture [8] (HTTP protocol). Additionally, RESTful web services organize functionality of the system into collection of resources available via network and provide a uniform interface to manage them. Consequently, systems implemented accordingly to REST paradigm are easier to integrate with other information systems, even those that already exist. This is a result of the use of HTTP protocol' [5] semantics which is well known and 7 used. Therefore, in MMDCP the REST paradigm was used as SOA implementation. The designed and implemented MMDCP system architecture is presented in Figure 1.

The central element of the proposed platform is ROsWeL business process execution engine [13], which is an execution environment for medical business processes. In order to create in the considered environment a medical business process the following steps are made. First the description of the business process is prepared using ROsWeL language [14] syntax. Then, the document with the defined business process is uploaded to the specified URL address of the engine service. Next, based at uploaded business process description, the new Web Service implementing the desired functionality is created and installed at ROsWeL Workflow Engine. In consequent, the business process in a form of a RESTful Web Service is provided to the user. Such a RESTful Web Service can be invoked on almost any type of device (personal computer, smart phone or tablet). In order to fully utilize the proposed platform, such a device has only to support HTTP communication protocol, and HTML format to display a user interface. In practice, it can be any device that has a web browser installed.

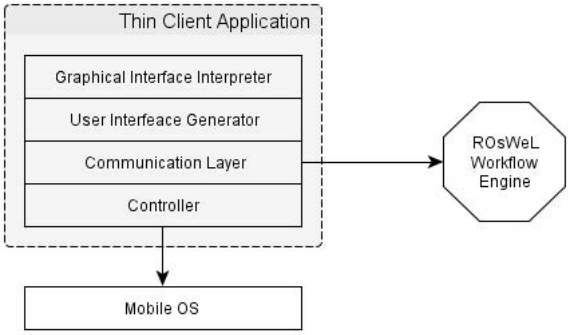

Fig. 2. "Thin client" architecture

It is important to note, that ROsWeL engine is responsible for processing both logic, and communication between the system components, and the end-user device, including the information on how to display data, what information should be enter to the system, and how. In other words, ROsWeL engine allows to compose several advanced medical services (being a collection of RESTful medical resources) providing a complex functionality, by invoking single medical services in a defined in a ROsWeL file order. Details on the description of a business process can be found in [14] and information on design and implementation of the business process engine are in [13].

Second element of the proposed MMDCP platform is the end-user universal mobile application. Its main task is to initialize business process execution and the active participation in it. The important feature of such a mobile application is its ability to generate the user interface based on the data transmitted from the ROsWeL engine during the business process execution (called "on-the-fly"). Such an interface allows medical staff to enter the data either using a keyboard, a camera or a dedicated laser reader integrated with the device or connected as a peripheral device. In the current state of a development of MMDCP, its user can enter several types of data: strings, images and barcodes. Using the built-in camera or dedicated barcode scanner there is possibility to scan either 1D bar codes (the cheapest and widely used), or 2D bar codes (more expensive but much more reliable) like QR-code, Aztec etc. In the consequence, we will call the mobile application a "thin client", which means that even a very simple and inexpensive device can play such a role within the proposed platform.

Up to date, a "thin client" is available for the following platform: Windows Mobile (v6.1, v6.5), Windows Phone, Android and PC with a web browser. It is important to add that regardless of the device and its quality, the graphical interface design of the proposed application looks almost the same. This has been achieved by our thin client architecture depicted in Figure 2.

The architecture of mobile application is modular and consists of the following modules: Controller, Communication Layer, User Interface Generator and Graphical Interface Interpreter.

The Controller module is used to invoke native functions of different mobile devices. It is used for setting or getting the mobile device settings like language, sound, connection and uses them during a business process execution. On the top of the Controller module the Communication Layer is built. Its main goal is to send and retrieve messages between ROsWeL engine and a thin client application. Above the Communication module is the User Interface Generator placed. It retrieves messages received by Communication Layer in order to generate optimal interface for a device (screen resolution etc.). The interface's data are described with a small subset of HTML5 tags, enhanced with some missing functionality, such as a barcode and some optimizations, which consist of postponement the generation of the part of the user interface in order to accelerate their display on certain potentially weak devices. The missing part of interface is added to the already generated interface later, on the user's demand. The last module of mobile application is a Graphic Interface Interpreter, which simply draws generated interface on the mobile device screen.

3 The Integration of MMDCP with Hospital Information Systems

Nowadays there are many Hospital Informations Systems (HIS) on the market. They are required to meet demands of the fast data access, high customizability, and stringent safety. Such requirements result in complex architecture of such systems, and their custom software solutions. Therefore, the integration with HIS systems is a very difficult task. The MMDCP platform provides an internal mechanisms to integrate it with the existing HIS systems.

In order to achieve maximal interoperability, the proposed MMDCP does not communicate with a Hospital Information System (HIS) in a direct manner. In case of reading medical data, ROsWeL invokes medical services (resources), which are network interfaces for data stored in the HIS. Additionally, to allow an easy integration with various HIS systems, medical services use so called adapters to adjust the inner HIS data format (e.g. database structure) to the one understood by the MMDCP platform. The details of the way of exchanging the format are discussed in the following part of this Section. The last element of the platform is a buffer service. It is used in order enable recording some data in HIS database. Since HIS systems have various architectures and functionality, and require a strict control of what can be recorded, the proposed platform adds a buffer element to implement so called "delayed write". The main idea of a delayed write is to record temporarily data into buffer instead of HIS database directly. Afterwards, the HIS system retrieves on demand data stored in a buffer via buffer adapter and updates its medical records in internal database. This approach allows to control the data that will be stored in HIS database, and provides a high level of interoperability.

The proposed solution consists of two Java web services utilizing Jersey [6] technology and using Hibernate [3] ORM mapping framework. First of them is a Universal Identification Services (UIS), which task is to export information from

HIS database to the business process engine in order to provide data objects required by business processes. This service after a few simple configuration steps creates REST interface for accessing objects using different data types, identifying them by scanned barcodes or their natural database ids.

The second web service — Universal Buffer Service (UBS) — is burdened with task of gathering in an universal way the information along with multiple useful meta information, collected during the business processes execution.

Due to the fact that each of the above mentioned services performs different tasks, UIS and UBS are presented in the following sections.

3.1 Universal Identification Service

Primary task of the UIS (architecture in Figure 3) is to fetch information from the HIS system and present it to the business process engine, which can therefore use the obtained data to complete its tasks. The proposed service neither interferes with existing HIS infrastructure, nor requires to create redundant data structures. Its requirements are very strict and simple, and are the following. Database system must cooperate with Hibernate (either by standard terms or by specific dialect and driver). Additionally, each object in that database should be supplied with barcode persisted in a string format. Moreover, the type of data stored in the table's columns must be Hibernate's standard basic types [4] (this last requirement is planned to be uplifted in the future work). As the result, the UIS will support also other types of columns: references to another tables, complex types, collection types.

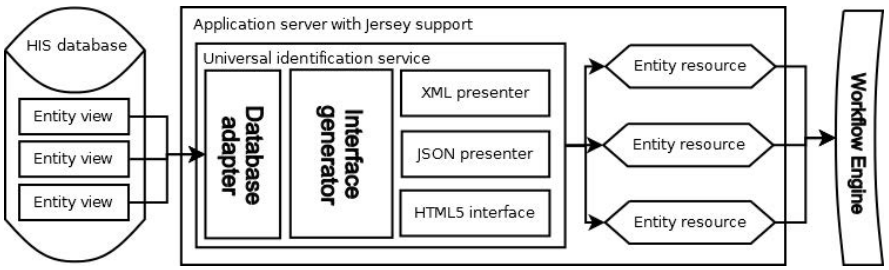

Fig. 3. Universal Identification Service architecture

Configuration process requires from the platform administrator to take a few simple steps. The Administrator should first consider what is required in the business process. Then he/she should prepare database's tables and views, and the configuration storage place. Then should be filled out the database configuration file and created Hibernate mappings for the views.

Since to complete a business process users need some data, this data can be presented to business process engine by the UIS. Choosing the right data is crucial, as too much information may be confusing, and too few will prevent

business process user from completing his task successfully. When the administrator knows what data is required, a database presentation should be prepared, because in the HIS systems data is stored in different tables or views. The UIS requires that each object class should be stored in one table or view with using basic types such as string or integer. Often this step is negligible, because data is already presented in the right way. Therefore, the administrator should prepare storage place for configuration files, a separate folder with adequate security rights if it should not be read by applications other than application server. The UIS searches for configuration files in the preconfigured location set in Java's `System.getProperties()` facility. In Tomcat this values can be found in `catalina.conf`facility. This folder should contain Hibernate database connection configuration stored in `hibernate.cfg.xml` file - a XML document. Also the application server on which the UIS is installed should be equipped with the proper JDBC connection drivers mentioned in the file. The final step for platform administrator is to create the hibernate mappings for desired objects. Those mappings must be supplemented by only one mandatory Hibernate meta-argument "barcode". Those meta-argument should mark mentioned, mandatory textual field containing the barcode.

Universal Identification Service on the basis of configuration files creates the REST interface. This interface is available instantly after the service's start under the service's URL address configured by the application server administrator. Each of the configured object classes can be accessed as separate resource. Such a resource informs also the user about the object's properties. Objects of given class can be accessed by identifying them with their natural database id or by the barcode. Each of the mentioned resources can be presented by the UIS in three different formats, namely: XML, JSON, HTML5. Whether the first two are common standard of web application communication formats, the last one is meant for humans.

3.2 Business Process Buffer Service

Business processes are often created with a purpose to perform some process along with the user, and during that process data is collected. This data can contain useful information for the HIS system, for example a number of syringes that remain in warehouse or amount of drug injected. Such a data should be stored somehow. This problem may be solved with the usage of the Universal Buffer Service (architecture at Figure 4) — a service which delays the storage of data to the moment when the HIS system will be ready to accept it ("delayed write"). The role of the service is to collect useful meta-information. The collected information can be utilized to optimize business processes, provide additional security or perform monitoring services. Such information comprises among the others: user, who executed process, device, on which process was executed, process which was executed, host name address, execution time and application name.

Configuration process of UBS is composed of three steps: finding and creating database according to the supplied schemas, providing Hibernate database configuration file, and adding eventual security restrictions on business process

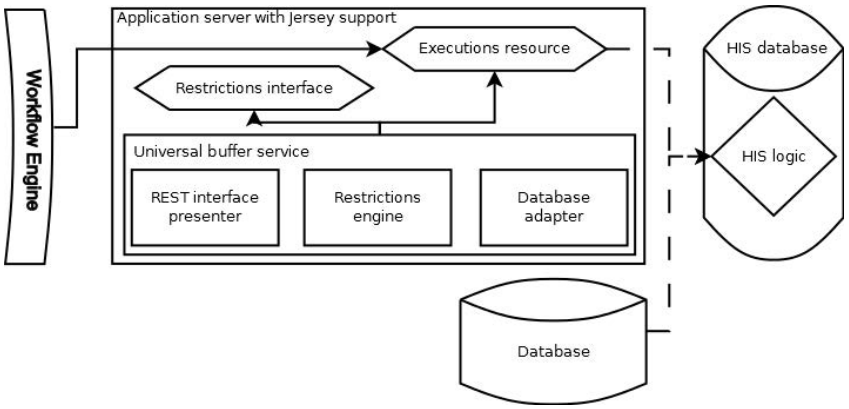

Fig. 4. Universal Buffer Service architecture

types, users or devices on which they are performed. Database, which ought to be prepared, must contain tables mentioned in the service's documentation and must cooperate with Hibernate. Database configuration file again must be put in the folder, which therefore will be read through the Java's properties facility. The last and optional step allows to restrict business process executions which will be accepted — only specific processes, users or devices can be allowed to store data inside the UBS. Also simple meta-description rules can be provided to validate the data stored by business process users in order to achieve consistency (similarly to the XSD).

Data gathered by the Universal Buffer Service is stored inside the database. It can be therefore collected by the HIS system in order to allow it to maintain its internal procedures and structures. Business process execution's data database stores object of execution, which has a mentioned meta-information. All other data is stored as its properties in a textual way, as provided by the workflow engine. This data can be accessed through database connection or through the REST interface, which provides it as a resource. REST interface also contains separate resource for configuring the mentioned restrictions. UBS presents to user additional web-based interface which allows to browse database and store additional restrictions.

Both backend services provide well defined, simple and robust interface for both the business process workflow engine (that seamlessly integrates with services), and hospital information system. Despite need of some effort to plug the Universal Buffer Service and Business Process Buffer Service, the services do not interfere with existing infrastructure. Therefore they can be integrated with almost every platform currently existing in the market.

4 MMDCP Data Collecting Process

In order to illustrate the possible usage of the proposed MMDCP platform in the healthcare facilities, an exemplary process is discussed (5). In the considered

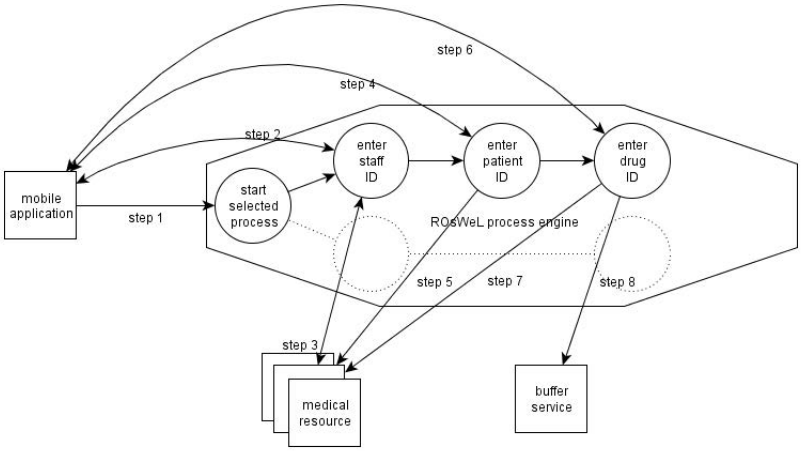

Fig. 5. Drug administration process

process the information on drugs administered to patients during medical procedures is being collected. First, medical staff using a mobile application installed at a mobile device connects to the ROsWeL engine and asks for a list of available processes. After selecting the drug administration process, ROsWeL engine executes first step of the process (step 1), which requires to enter medical staff member ID in order to verify her/his identity (step 2). ID number can be entered manually (keyboard), or scanned from the barcode placed at the ID of the employee. Next, ROsWeL engine via a Personel Identification Resource (one of the available medical resources) queries HIS database for the verification (step 3). If such a person exists in the database and is authorized to administrate drugs, the information is send back to the ROsWeL engine to proceed, otherwise authorization error is send back to the thin client. In the next step the patient's identification number same as medical staff person ID (step 4) has to be entered. The patient information is verified, and (step 5) send back by ROsWeL engine to the client in order to generate the appropriate interface at mobile device, consistent with the definition of the business process. If the above steps pass successfully, the ID number of administrated drug can be entered into the system (step 6, step 7 and step 8). It is important that usually each patient takes more then one drug. To administrate a new drug fast and easily, the MMDCP platform (ROsWeL engine) remembers medical staff and patients, and do not require to repeat first two steps of the above described process. Changing a patient, or a medical staff who takes part in the considered process, one just needs to enter a new ID of respectively patient or a medical staff.. Finally, all complete triples in form of: `<staff_id, patient_id, drug_id>` are recorded in a buffer (step 8).

5 Conclusions and Future Work

The MMDCP platform provides a flexible and easy to integrate environment that supports medical staff in the daily duties of collecting medical data. This solution

has been successfully integrated with the Eskulap Hospital Information System, which is the third most used system in polish hospitals [1]. Due to the applied service-oriented architecture and technology, the integration process was completed without any major problems, and its validity has been confirmed in practice. By providing by MMDCP a possibility of defining a new business processes, it is possible to obtain a completely new functionality that enriches the Eskulap HIS system almost out of the box.

Further development of the proposed platform will be simultaneously focused on two directions. The first one will focus on extending the possible range of MMDCP practical applications. We are going to enhance the presented platform by adding a possibility to use information on the patient's insurance and the quality of that insurance in order to propose an appropriate drug or its equivalent. This functionality also makes it possible to facilitate the easier management of hospital pharmacy in case of the lack of certain drugs. The latter option concerns with the development of the ROsWeL Workflow Engine functionality. Currently, we work on adding the semantic information on the business process steps and on resources used during the process composition. Such a semantic extension will allow, e.g. to more accurately propose equivalent drugs or even equivalent resources in case of system failures.

Finally, one might consider using the HL7 [2] standard as an internal representation of medical data structure. This standard is well recognized in the medical environment and would allow to not only add new data sources to the platform but also allow to exchange messages with other systems more easily.

References

1. ESKULAP Hospital Information System, http://www.systemeskulap.pl/
2. Health Level Seven International, http://www.hl7.org/
3. Hibernate Object-Relational Mapping Framework, http://www.hibernate.org/
4. Hibernate Reference Manual, http://docs.jboss.org/hibernate/orm/4.1/
5. Hypertext Transfer Protocol – HTTP/1.1, http://www.w3.org/Protocols/rfc2616/
6. Jersey JAX-RS (JSR 311) Reference Implementation, http://jersey.java.net/
7. Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/soap/
8. W3C: Web Architecture, http://www.w3.org/standards/webarch/
9. Web Services Description Language (WSDL) 2.0, http://www.w3.org/TR/wsdl20/
10. Web services addressing, ws-addressing (2004), http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/
11. Web services security: Soap message security 1.1, ws-security (2006), https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
12. Web services business process execution language version 2.0 (2007), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
13. Brzeziński, J., Danilecki, A., Flotyński, J., Kobusińska, A., Stroiński, A.: Workflow Engine Supporting RESTful Web Services. In: Nguyen, N.T., Kim, C.-G., Janiak, A. (eds.) ACIIDS 2011, Part I. LNCS, vol. 6591, pp. 377–385. Springer, Heidelberg (2011)

14. Brzeziński, J., Danilecki, A., Flotyński, J., Kobusińska, A., Stroiński, A.: ROsWeL Workflow Language: A Declarative, Resource-oriented Approach. New Generation Computing 30(2-3), 141–163 (2012)
15. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
16. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River (2004)
17. Menola, F., Miller, E.: Implementation Guide for the Use for Bar Code Technology in Healthcare (2003), http://www.himss.org/
18. Richardson, L., Ruby, S.: RESTful Web Services (2007)