A WoT Approach to eHealth: Case Study of a Hospital Laboratory Alert Escalation System

Andreas Ruppen
Software Engineering Group
University of Fribourg
1700 Switzerland
andreas.ruppen@unifr.ch

Jacques Pasquier
Software Engineering Group
University of Fribourg
1700 Switzerland
jacques.pasquier@unifr.ch

Jean-Frédéric Wagen
University of Applied Science
Fribourg
1700 Switzerland
jean-frederic.wagen@hefr.ch

Beat Wolf
University of Applied Science
Fribourg
1700 Switzerland
beat.wolf@hefr.ch

Raphael Guye
University of Applied Science
Fribourg
1700 Switzerland
raphael.guye@gmail.com

ABSTRACT

With the generalization of network-enabled devices such as smart phones, slate computers and tablets, new challenges await the eHealth research community. Indeed, these devices should not only integrate seamlessly into the daily workflow, but their usage must appear as ordinary as possible to the different caregivers. Using RESTful architectures, it is possible to model custom objects in the health domain as resources, interact with them and combine them to mashup applications, enhancing and facilitating in a natural way the work of caregivers.

We claim that embedding eHealth workflows into the Web of Things is not only possible, but even enhances the whole process. An alert is no more an isolated event, but becomes connected to other resources providing additional information about its general context. In this paper, we illustrate some of the challenges of bringing REST to the eHealth domain by studying an existing hospital laboratory alerts system and by proposing to generalize it in order to encompass the whole escalating process and exchanges of information among caregivers, patients and their medical records.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures—*Service-oriented architecture (SOA)*; C.2.4 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS—*Distributed applications*; J.3 [**Computer Applications**]: LIFE AND MEDICAL SCIENCES—*Health*

General Terms

Design, eHealth

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WoT 2012, June 2012; Newcastle, UK
Copyright 2012 ACM 978-1-4503-0624-9/11/06 ...$10.00.

Keywords

Web of Things, REST, ROA, eHealth, Smart Alert System

1. INTRODUCTION

Nowadays, the widespread acceptance of smart phones and other network-enabled devices such as slate computers and "post-PC" tablets has reached a level where developers can seriously start thinking about how to integrate them into our daily lives in order to make our surrounding professional environments smarter. One of the main challenges of this undertaking is to make the manipulation of these new tools as ordinary as possible for the professionals using them. Using RESTful architectures, it is possible to model custom objects as resources, interact with them and combine them to mashup applications, enhancing and facilitating in a natural way the work of professionals. Thus, we strongly believe that the key architecture for a seamless integration of these new resources is REST and RESTful services [10]. Several research papers show that they have many advantages over fully fledges WS-* web-services [9, 5, 3, 6, 8, 4].

eHealth existed before the upcoming of smart devices. These tools, however have opened new possibilities and perspectives in this domain. eHealth is a well established research and application domain. It is becoming the standard for health-care. Already today one can see the emergence of computers and other connected devices into health-care. A prominent example is the acquisition of images by computer tomography. Unlike X-ray radiographs, the images obtained from a computer tomography only exist in the digital world. However, there are still many open challenges regarding eHealth, among them privacy and security and the seamless integration of computer aided tasks in the daily business. Other examples can be found in [2].

For a funded applied research project, we were confronted with these two domains. We had to design and implement a smart alert escalation system in hospital environments. The system should seamlessly integrate into the daily workflow, giving caregivers the needed information at the right time without disturbing them with non relevant information. Conducting this research allowed us to draw two very interesting conclusions: 1. With the right choices for the

architecture and for the escalation framework, it becomes possible to generalize the presented case to many other situations. This paper will focus on an eHealth use-case, but we will give a few hints about other situations and use-cases; 2. By considering the different participants in a hospital as resources within a WoT and with the help of a RESTful architecture, is it possible to greatly enhance the alert system toward a global eHealth one.

In order to best illustrate our findings, this paper focuses first on a concrete use-case for a hospital alerts escalation system. In its second part, the paper describes an extended use-case, where almost everything becomes a resource and can be accessed in a RESTful manner. This illustrates the advantages of RESTful architectures in the eHealth domain.

2. THE ORIGINAL ALERT SYSTEM

Several alerting systems have been proposed over the past years [1, 7, 13]. They all have in common that they try to solve the global problematic in a smart way. Indeed, creating an alert system involves the resolution of several challenges. Among them, it has to be ensured that alerts reach their destinations or that the system can handle delivery failures. Furthermore, the receiving devices might not all be the same and use the same technology. The system must handle this heterogeneity. Besides these technical aspects, a good alert system also has to fulfill some psychological guidelines. Sending too many unrelated alerts, for example, would be counterproductive. People will get bored and stop checking or acknowledging them. User studies have been made [12] in order to determine success criteria for such systems. Furthermore, systems like [2] show the emergence of eHealth application using embedded and connected devices. Using such equipments to monitor patients and sending out alerts in case of critical conditions is just one step further in the development of eHealth.

2.1 Requirements

Today's healthcare remains largely paper-based. Patient's medical files are on paper, medical analysis results are on paper, some other documents exist only on paper and are added manually to the patients file. This approach is cumbersome. The focus of this project is to provide a seamless user experience when working with information about a patient especially when dealing with automated medical laboratory results, such as a blood analysis. Today most hospitals already have in use software systems to handle such analysis and the associated results in electronic form. A caregiver will for example take blood samples and fill out a sheet to select which analyses have to be done. Upon arriving at the laboratory, this sheet is scanned, processed automatically and inserted into the system. By scanning the sheet, a bar-code is generated which has to be sticked on each blood sample. They will then wait for being analyzed. Immediately after the laboratory has done the required tests, the results are inserted into the system and added to the electronic form created earlier. As soon as the results are introduced into the system, a caregiver can check them. Besides, the system does some automatic checking for abnormal values. This process of checking for critical values can be seen as a black box with some business logic doing well what it was designed for. If critical values are found, the accountable caregiver has to be informed promptly about them. Today, this information is passed through phone calls. A secretary is monitoring the database for new critical results and if one happens, he informs the accountable caregiver of it. The caregiver takes then the responsibility to handle this case in a suitable manner. Since this system is for a hospital environment it is essential to have a record of who did what, e.g to decide upon responsibilities. The above process has many drawbacks. It requires manual intervention and monitoring of the alerts database. Additionally, the accountable caregiver may not be reachable at the time of the alert. Thus, the secretary has to decide which is the next caregiver to alert. Such an approach is not only cumbersome, but also time-consuming and error prone.

We collaborated with a medical software company specialized in the development of IT solutions in the eHealth domain. Actually, it offers a complete solution for an automated medical laboratory system, which we will call AMLS. Through this application, doctors and laboratories can stay in touch. Besides holding the results of each analysis, The AMLS system does some verification in order to search for critical results in each analysis. However, it is up to a person to check for new critical results and inform the accountable caregiver. Based on the scenario presented above and the installation already in place, we propose a solution seamlessly integrated into the Web of Things. The system conducting the analysis and doing the verification of the results should be integrated into the solution as much as possible. The process of alerting a caregiver should feel natural and should not overwhelm him with alerts he is not accountable for. In this chapter, we will focus on how to enhance this system with a smart escalating alert framework to overcome these problems.

Handling alerts in a smart and autonomous manner implies several technical decisions. At some point, an alert will be sent to a device. However, some caregivers use smartphones with the Android or the iPhone operating system, others prefer receiving a SMS and when abroad many would prefer a phone call. Thus, an alert system has to handle these different technologies without requiring human intervention to deliver the alert. Additionally, a caregiver might want to register more than one device, using either of them depending on the task he is actually doing. He might for example use a tablet pc during medical visits, but is more likely to only have his smartphone in his office. Another important point is the handling of failures. Since alerts are about critical results, it has to be ensured that at least one caregiver will take care of the alert and handle it accordingly. Thus, an alert needs some sort of *escalation system* where the alert is propagated to someone else if the first caregiver either rejects it or does not answer it. Since the use-case is in a hospital environment, security and privacy need to be ensured. Security is related to the escalation of alerts. The system needs to know at any time in which state an alert is (open or accepted) and who is actually in charge of it. To ensure the successful delivery of such an alert, the final option in the escalation list will always be a phone call. As for the privacy, very strict confidentiality policies apply for most medical documents.

2.2 Design Considerations

The actual deployed business solution is composed of two parts. The first one consists in a client application, which proposes a complete interface to the AMLS database containing all the analysis data. The second one is the smart

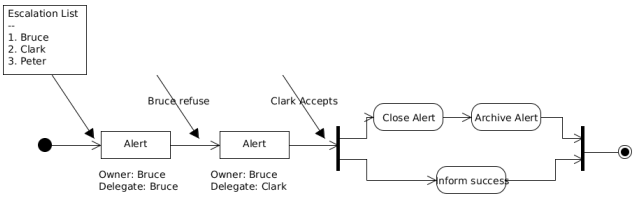

Figure 1: Alert escalation over two caregivers

alert software, which connects the caregivers' devices to the database; forwards an alert according to its escalation list; and monitors its status. Immediately after a critical result is detected, this new framework starts creating the appropriate alert. The latter can be seen as a bucket containing information about a critical result and its actual state. For each alert, a notification is sent. The framework proposes a transparent way to handle the different underlying mechanisms for successfully delivering a notification. As new technologies will emerge, it is sufficient to create a new adapter for this device and plug it into the framework.

Since these notifications are out of the hospital control, special care has to be taken regarding privacy issues. Caregivers are required to respond to the medical confidentiality. The same applies for documents concerning patients and by that also the notifications. Thus, sending information about critical results in an analysis over systems which are out of the scope of the hospital would imply a breach of secrecy. To avoid this situation, the notification does not contain any information related to the patient or to the actual content of the alert. It rather contains a link (i.e. a URI), where the doctor can check the details and see which result has created the alert and for which patient. Since this interaction is done over HTTP, standard web mechanisms such as HTTPS for protecting and securing access to resources is applied. The refusal or validation of an alert is also done directly on the web server. Thus, alerts can be seen like first-class citizen resources in a WoT ecosystem. Architecting the alerts as a RESTful resources has several advantages. Besides the privacy issues mentioned before, such an architecture is not restricted to one application for handling the alerts, but rather every web-enabled device can be used to check, refuse, or validate them. Of course, not all CRUD[1] actions are allowed on these resources. It is for example not allowed to delete an alert. However, consultation and modification are possible. Modifications include the refusal and validation of alerts. This approach allows the creation of interesting mashup applications in the eHealth domain as we will see in Chapter 3.

One of the requirements discussed in Section 2.1 is the necessity to ensure that each alert will be handled by some caregiver. As it might happen that the caregiver who initiated the analysis is not available when critical results are detected, somebody else has to take care of it. Thus, let us introduce the following two concepts: 1. Each alert has an *owner*. The owner designates the caregiver in charge of the patient and usually the caregiver who asked for the analysis. The owner is also the person who will be alerted first; 2. Besides, we introduce the concept of *alert delegate*. An alert delegate is the caregiver currently in charge of a given alert. This might be the same person as the alert owner, but it can be somebody else. These two concepts are the core of the escalation mechanism. Imagine the following situation: caregiver *A* requested for an analysis and a critical result is detected. Thus, he will get a notification and should take care of it. However, at the time of receiving the notification, he is occupied with another patient. The system has to react to such a situation and find somebody else who will take care of the situation. Thus, the alert will get a new delegate which will receive a new notification. Figure 1 shows such an escalation. Yet, the system will not escalate in a random manner. The escalation of an alert is predictable and given by the system. A hospital is divided in several sections. Thus, at least each section has a different way of escalating alerts. This is why each alert has an associated *escalation list*, dictating which will be the next alert delegate. Therefore, we propose as underlying architecture the one of Figure 2.

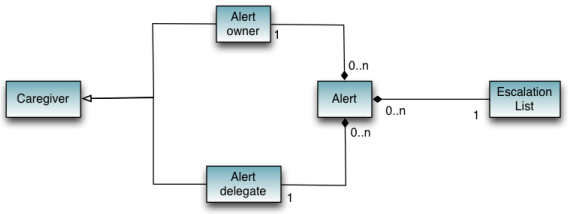

Figure 2: Simplified class model for the alert system

2.3 The Implemented Prototype and its Generalization

Based on the requirements of Section 2.1 and the design considerations of Section 2.2, we propose the following architecture: AMLS is preserved and continues to be used to consult the details of an analysis. Also, the core of AMLS, which encapsulates the business logic for the search of critical results in analysis, is kept as part of the implemented system. Figure 3 gives an overview of the involved components. On the top is the AMLS business logic and its database. Below sits the Smart alert framework, handling everything related to alerts.

AMLS scans each medical analysis to find critical results. If such a result is found, a new entry is created in the *Alerts database*. This database is the contact point between the two systems. The smart alert framework reads from this database, creates the necessary alerts and notifications and ensures that somebody will take care of if. In order to achieve this task, the framework is divided into three subsystems: one for sending the notifications, one for handling the escalation and one for giving access to alerts. The subsystem responsible for sending the notification relies on several adapters as discussed in Section 2.2. The escalation system is built in a similar manner. There are several lists depending on the needs of a hospital. In our use-case these are just simple linked list.

Finally, the *Web* component is responsible for serving incoming requests for consulting or modifying alerts. It exposes several resources which can be accessed, browsed and linked to in a RESTful manner. Figure 4 gives an overview

[1]Create, Read, Update, Delete

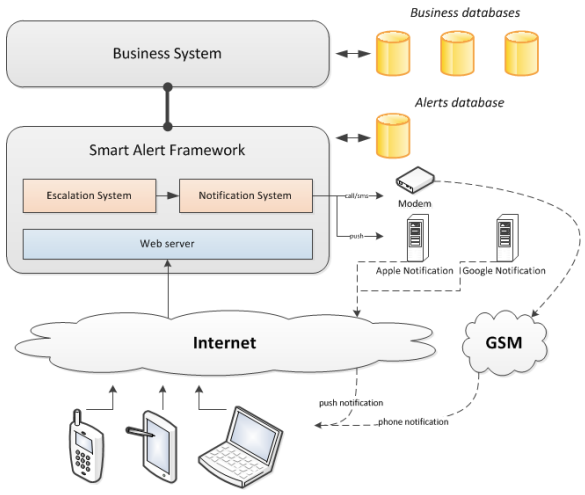

Figure 3: An Automatic Medical Laboratory System (AMLS) enhanced with the Smart Alert framework

of the available resources. As already discussed the *Alert* resource is the central one and sits at the heart of the system. Furthermore, the relations between the resources are translated into links. By that, each alert will, for example, contain a link to a *Caregiver* resource. The *Device* resource contains information about a device (e.g. its type and its priority). Using priorities makes it possible to send the first notification to one device and if there is no answer after a given time, to forward it to a second one, without changing the alert delegate. Since an analysis is always initiated by a caregiver and alerts are treated by a caregiver it makes sense to model them as resource too.

The last resource in Figure 4 is the *Escalation list*. Each alert has an associated escalation list. In our case this is just a simple list containing caregivers. Nevertheless, other systems are possible. One could imagine a system taking into account the current schedule of caregivers, and choosing the next alert delegate based on an availability criterion. In a broader case, the presented architecture allows two interesting generalizations:

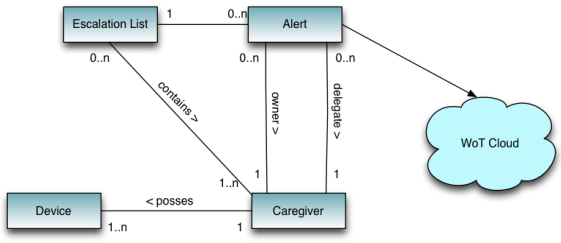

Figure 4: Identified Resources

First, it is possible to reuse the same architecture for alerting systems in other domains. In [11] we presented a use-case of an alert system for firefighters. The use-case was based on voluntary firefighters and how they organize in case of an alert. While current organization involves several phone-calls, this paper proposed a way to approach this problem involving smart alerts. Immediately after a new incident is reported to the fire central, some firefighters have to be dispatched. British firefighters for example, are organized in areas, groups, and stations. The alert has to collect enough firefighters willing to accept the mission. As for the system presented in this chapter, the same requirements apply. It has to be ensured that somebody takes care of the event. Furthermore, the fact that the notification does not contain any data, but rather a link to a given alert seems to be a smart choice for this scenario too. Depending on the alert, only a reduced number of firefighters may be needed. Thus, the link in the notification would lead each firefighter to a WebSocket enabled resource, where he cannot only refuse or validate the alert, but also see how many colleagues have already committed and how many are still needed. In order to adapt the current system to this use-case, only few changes would be needed. The escalation list, for example, would not be a simple list but would rather have a tree structure. Again, this change would not require any major architectural modifications. The attentive reader will notice that changing the escalation list business logic does not change its RESTful API.

Second, Figure 4 reveals another interesting detail: the alert resource points to some other resources, the *WoT Cloud*. This is not a simple resource as the other ones. It is more a link to other possibly interesting elements of information related to alerts. The important point is that the alert resource is the entry point to a whole eco-system. From an alert, it is possible to go to the associated caregiver but it would also be possible to go to other resources living in the WoT. We shall elaborate on this idea in the next chapter.

3. THE EXTENDED VISION

When looking at the use-case presented in Chapter 2 and the proposed architecture, it appears that the system, although working, lacks some functions which would allow working with it in a more natural manner. The end of Section 2.3 and Figure 4 already opened the discussion about additional resources. To identify interesting resources lacking in Figure 4, one has first to look a the daily business at some hospital or medical practice. A hospital is not only populated by caregivers, but also by other people (like visitors, administrative staff, cleaning staff, or patients). A patient has some general information like a name, or a phone number, similar to a caregiver. Still, patients are different since each one has its own medical record. These records are kept in a drawer-shelf organized alphabetically. Such a record contains, among others, information about past consultations, X-ray, analysis results and so on. In the morning, the doctor's assistant will fetch the medical records for the first patients from this drawer-shelf. During the consultation, the doctor may add some more information. Afterward, the medical record is put back in the drawer-shelf. Since we are trying to build a system which reflects as much as possible today's processes, we claim that, besides the alerts of Chapter 2, caregivers, patients, and medical records should also be modeled. Figure 5 gives an overview of the currently discussed resources. Since this vision is an extension from the use-case presented in Chapter 2, they have some of them in common. The interested reader will notice that other staff has no dedicated resource. This comes from the fact, that they are not directly concerned by a patients medical treatment.

Figure 5 depicts the relationship between these resources.

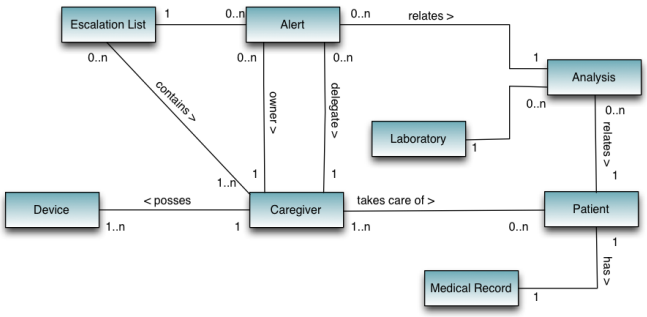

Figure 5: Adding more Resources to the system

In Chapter 2 we saw that an alert always has an associated escalation list. Moreover, the alert owner and delegate are both of type caregiver. One new part with respect to the design presented in the previous chapter, is the *Patient* resource. This will be the primary place to fetch any further information related to a patient. As shown in Figure 5, a patient is connected with almost every other resource. He can have many caregivers in charge of him. This comes from the fact that a patient's caregiver may change between different stays at the hospital. This is especially true, if the patient is treated for different diseases. Furthermore, several alerts can exist for a given patient. This can have different reasons. A patient may have multiple analysis done during one stay, or he may stay several times at a hospital, each time needing a new analysis. Whereas the patients' resource only contains general information about the patient, it does not contain anything about his health condition. Notes of medical consultations, medications, or X-ray radiographs are stored in the *Medical Record* resource. Ideally, each patient has exactly one medical record containing all his medical history in a given hospital. This resource can merely be seen as list of blobs. Each blob can contain any structured or unstructured data. Besides, a blob can also contain or link to more complex data like an X-ray radiograph or a ECG [2] recording. Each relation in Figure 5 translates into a link to other resources. As such, the representation of a patient contains information about himself but also a link to his medical record. This design allows browsing, and discovering resources related to a patient.

The entry point of Chapter 2 use-case was a patients blood analysis. It described how critical results can be escalated to caregivers. However, the analysis itself was out of focus. This is mainly because in the actual implementation, caregivers desire to continue to use AMLS to consult analysis results. With this choice, the system currently in place can continue to work. Yet, in a world completely immersed into the WoT, it should be possible to access these results in a RESTful manner. Imagine for example, a patient who has his cholesterol determined by an analysis. It is possible that the determined value does not raise an alert but that the doctor should discuss it with the patient as so to propose some possible changes in the patients' nutrition to reduce his cholesterol value. To do so, we need to introduce a new resource, the *Analyis* resource.

The introduction of the analysis resource adds some other benefits. When an alert is raised for a given result and notifications are sent out; the receiver of the notification will check the alert. In the use-case of Chapter 2 the content of an alert was limited to some basic information about the patient and the value which raised the alert. However, since analyses were out of the scope of the system, a caregiver had no seamless integrated way to consult the results of an analysis. He had either to come back to a paper version of the results or to connect to the AMLS system from his desktop. Additionally, having a look at the result sheet of an analysis upon receiving a notification of an alert, permits to put the critical values into some context. As depicted in Figure 5, the analysis resource sits between the alert and the patient one. An alert always refers to a given analysis and an analysis is always associated to a patient. By transitivity, an alert is thus always associated to a patient.

Another newly introduced resource is the *Laboratory* one. Each analysis is executed by exactly one laboratory. Yet, a hospital can work with several different laboratories, depending on the type of analysis. Furthermore, our use-case foresees that a caregiver might not only work in one hospital. Many doctors work in a hospital and have their own practice. Since analyzing blood sample is the daily business of caregivers, such a doctor might work with different laboratories. Nevertheless, he wants to receive alerts from all of them. In this scenario, it makes sense to model the laboratory as a resource. If the caregiver gets informed about a critical result which requires further analysis to be done, contacting the associated laboratory becomes easy.

Architecting the system using such a resource oriented way has several advantages over the actual system. The most obvious one is the liberation from the solution bound to personal computers. A scenario illustrating this statement would be the daily visits in a hospital. Each patient would have a NFC[3] wristlet instead of a plastic one. Upon arriving at the patients bed, the caregiver scans the wristlet and the associated resource will open on his device. Since everything is architected as resources which are, according to RESTful principles, linked together, the caregiver can now browse this information just as he would pick up the clipboard from the patients bed and flick through it.

Furthermore, adding a new resource, like a room, would be easy. Once the resource is designed and made available through a proper URI, it is very easy to integrate and link it to other ones. Thus, a patient would be linked to "his" room in which he is staying. Users of the system would then immediately discover this new resource and can start using it.

Most of the resource presented in Figure 5 are really simple and represent almost static things. While the phone number of a caregiver might change, it implies only an atomic operation on the resource to update it and reflect the new state. The same applies, for example, for the medical record. During each consultation, a new entry is added to the patients medical record. Such changes are atomic and do not require special care. Things changes when talking about the implementation of the analysis resource. An analysis is an ongoing process which has different stages during its life cycle. Moreover, some steps involved in a given analysis can be executed in parallel. This allows for a decomposition of the problem. A possible approach is to reuse the design proposed in [11]. The process of a blood analysis can be seen

[2] electrocardiogram

[3] Near Field Communication

as a long lasting decomposable process. This process can be followed starting with its root URI and continuing with its sub-URIs until it is finished.

4. CONCLUSION

We identified the challenge of designing an integrated RESTful systems in the eHealth domain. The emphasis was first set on the alerting mechanisms of the system, but we also enlarged this vision and proposes a scenario completely embedded in the WoT. The starting point of our considerations was the daily business in a hospital or a doctor's office. We then tried to model the objects implied in this daily business as resources and expressed their natural relation by using links. Replacing the current human operated system with one based on network enabled tools, will only work if the daily business will still feel natural and will be improved. With this purpose in mind, it seems that REST is a good candidate for modeling the world as it really is. Having a resource for each major physical object implied in a process is a plus. Easy access to these resources is also a plus. A standardized interface for accessing all the resources makes their usage simple. Finally, the HATEOAS[4] principles allow to start using all these resources right from any web enabled device. Besides, new resource will seamlessly integrate into the system, since they can be discovered just by browsing known ones.

The most important question when building an eHealth system is always to ask whether the quality of the medical support gets improved or not. From this perspective, one of the advantages of the presented system is the greatly enhanced response time to critical incidents. Reducing the time between the detection of a critical result and the successful transmission of this information to the right caregiver can be life-saving. Furthermore, the system can handle its own failures in the delivery of notifications and give guaranties about the delivery of them. This is an important feature, since it would not be acceptable that an alert remains open without anybody noticing it.

A prototype based on the scenario of Chapter 2 is currently tested by caregivers in collaboration with a medical software company and a regional hospital. This application is only a starting point to a broader usage of IT systems in the health domain. We believe that REST will be an enabling technology for eHealth together with the increasing business use of smartphones and pads. It fulfills all aspects regarding usability but also regarding security and privacy of the data. The implementation of a prototype based on the extended vision of Chapter 3 is under discussion and will depend from the results gathered from the field tests of our first prototype.

5. ACKNOWLEDGMENTS

We express our gratitude to the Fribourg cantonal fund for innovation for funding the DMAlert project.

6. REFERENCES

[1] D. K. Chiu, B. W. C. Kwok, R. L. S. Wong, S. Cheung, and E. Kafeza. Alert-driven e-service management. *Hawaii International Conference on System Sciences*, 3:30068, 2004.

[2] M. Colunas, J. Fernandes, I. Oliveira, and J. Cunha. Droidjacket: An android-based application for first responders monitoring. In *6th Iberian Conference on Information Systems and Technologies (CISTI), 2011*, pages 1 –4, june 2011.

[3] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin. pREST: a REST-based protocol for pervasive systems. In *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference*, 2004.

[4] D. Guinard, M. Mueller, and J. Pasquier. Giving RFID a REST: Building a Web-Enabled EPCIS. In *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, 2010.

[5] D. Guinard, M. Mueller, and V. Trifa. RESTifying Real-World Systems: A Practical Case Study in RFID. In E. Wilde and C. Pautasso, editors, *REST: From Research to Practice*, pages 359–379. Springer New York, 2011.

[6] D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards Physical Mashups in the Web of Things. In *Proceedings of INSS 2009 (IEEE Sixth International Conference on Networked Sensing Systems)*, Pittsburgh, USA, June 2009.

[7] E. Kafeza, D. Chiu, S. Cheung, and M. Kafeza. Alerts in mobile healthcare applications: requirements and pilot study. *Information Technology in Biomedicine, IEEE Transactions on*, 8(2):173 –181, june 2004.

[8] T. Luckenbach, P. Gober, S. Arbanowski, F. Fokus, A. Kotsopoulos, K. Kim, S. Advanced, T. Sait, and P. Box. TinyREST - a Protocol for Integrating Sensor Networks into the Internet. In *Proc. of REALWSN*, 2005.

[9] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, New York, NY, USA, 2008. ACM.

[10] L. Richardson and S. Ruby. *RESTful web services*. O'Reilly Media, Inc., 1 edition, May 2007.

[11] A. Ruppen, J. Pasquier, and T. Hurlimann. A RESTful Architecture for Integrating Decomposable Delayed Services within the Web of Things. To be published in *International Journal of Internet Protocol Technology* 2012.

[12] A. Thunberg and A.-L. Osvalder. What constitutes a well-designed alarm system? In *Human Factors and Power Plants and HPRCT 13th Annual Meeting, 2007 IEEE 8th*, pages 85 –91, aug. 2007.

[13] M. van Ettinger, J. Lipton, S. Nelwan, T. van Dam, and N. van der Putten. Multimedia paging for clinical alarms on mobile platforms. In *Computing in Cardiology, 2010*, pages 57 –60, sept. 2010.

[4]Hypermedia As The Engine Of Application State