A platform for the development of patient applications in the domain of personalized health

Dario Salvi , Joe Gorman, Maria Teresa Arredondo, Cecilia Vera-Muñoz, Manuel Ottaviano, Sergio Salvi

A B S T R A C T

Personalized health (p-health) systems can contribute significantly to the sustainability of healthcare systems, though their feasibility is yet to be proven. One of the problems related to their development is the lack of well-established development tools for this domain. As the p-health paradigm is focused on patient self-management, big challenges arise around the design and implementation of patient systems. This paper presents a reference platform created for the development of these applications, and shows the advantages of its adoption in a complex project dealing with cardio-vascular diseases.

1. Introduction

Western societies are currently facing severe socio-economical challenges that are threatening current health care provision models. Specifically, the ageing population and the global financial crisis increase the demand for more efficient and sustainable healthcare, i.e. better healthcare for lower costs. Current systems have been created and optimized to handle acute illnesses, but most healthcare expenditure is due to chronic diseases, which are likely to increase in the near future. It is recognized that the incidence of these kinds of diseases can be reduced with proper early prevention [1], by means, for instance, of health promotion and empowering citizens in the management of their own health. This requires a shift in the current health provision from an illness-centric to a patient-centric model.

Personalized health arose as the paradigm that embraces this concept and its implementation in practice is now strongly supported [2,3]. The main principles of p-health can be summarized as follows:

- Treatments must be personalized, based on the physiology of the patient, including clinical history, environmental factors, and genetics.
- Continuous management of patients' health through their whole life, which includes continuous monitoring of the patient in all possible environments (at home, at work, doing sport, etc.), allowing early detection of risky situations.
- Adherence to treatment is seen as a key aspect that must be improved [4], for instance, by motivating patients towards a more active management of their health and adopting a healthy lifestyle.

- Lifestyle is as important as genetics when dealing with diseases [5], thus prevention is a key factor that can be implemented through education.

The implementation of these strategies requires an important effort on the political, societal, economic and technological levels. It is recognized that ICT has a major role to play in this scenario [6–8], and this is leading to an increasing interest by industry. In support of this, the European Commission has funded around 40 research projects since 2007 within a specific objective of Seventh Framework Programme named Personal Health Systems (ICT Challenge 5: Objective ICT-2011.5.1) [9] and has included e-health in the "Lead Market Initiative For Europe" [10] as a major player in the future industrial development of Europe.

The main challenges introduced by the p-health concept are on the patient side, as the patient is put in the centre of the care process. The development of patient applications involves many actors (e.g. interaction designers, software designers, programmers, algorithm specialists, instructional designers), each with different methodologies, languages and points of view. Facilitating the work of such complex teams is a big issue. While the development of clinical applications for health professionals can exploit experience from more established contexts like e-health and clinical engineering, the development of patient applications still lacks well-established methodologies and tools.

In p-health applications user experience is of vital importance in engaging patients to use the system, and consequently improve their compliance to prescribed treatments. For this reason user centred design (UCD) methodologies have been successfully adopted in different initiatives in this area [11–14]. At the same time, Agile methodologies [15] are becoming very popular in the software industry as they emphasize the involvement of stakeholders in a context where requirements change continuously. As p-health is a complex and multi-disciplinary domain, Agile methodologies have shown to be suitable for this domain [16]. More recently, the combination of both methodologies (UCD and Agile) has shown great promise [17], nonetheless the integration between user interaction specialists and programmers is recognized as still being an issue [18,19].

An emerging area in software development that addresses these issues is model driven engineering [20] (MDE). MDE promotes the use of visual, technology-independent design tools which are able to automatically generate part of the application. This method can be especially useful when coupled with UCD, as it offers tools for representing user interaction paths. More specifically, interaction paths can be represented by machine-readable workflows, which can then be directly converted into code, automating the step between UI design and implementation [21,22].

In this spectrum of multiple methodologies, SW architectures play a fundamental role as they are the centre of the development process. Based on the project architecture, teams are selected, the work is organized among developers, and documentation and code repositories are established [23]. Thus, a proper software architecture must be coupled with the development methodology (UCD [24], Agile or model driven), and must support separation of roles within the development team.

This paper presents a development platform for creating patient applications intended to facilitate team-working in complex and multidisciplinary p-health projects. The design of the architecture took into account UCD and MDE methodologies, and was based on the idea of separating modules to suit the expertise and level of abstraction p-health engineers are used to work with.

The structure of the paper is as follows: in Section 2 we explain how we designed and implemented the development platform; Section 3 provides a proof of our solution in a case study related to cardio-vascular diseases (CVD); and, finally, Section 4 provides the conclusions and some possible future refinements of this work.

2. Methods

The work here presented was conducted using an Agile methodology. The first step was to conduct a survey to gather general requirements and to identify the needs of the developers working in the p-health domain. Based on these requirements we designed a first version of the platform to specifically address the separation of concerns according to how teams are usually organized.

The first prototype was then adopted in HeartCycle [25], a project partially funded by the European Commission with the aim of providing solutions for fighting cardiovascular diseases. During the development of the project's use cases, all the artefacts, from requirements to platform implementation, were continuously refined and improved. Programming was conducted in a continuous integration [26] environment, meaning that the development of the code was accompanied by the creation of automatic tests both at unit and integration level. The complete project was periodically compiled on a server that was also in charge of running all the tests, publishing test reports and deploying the latest version of the application on a website.

At the end of the implementation of the HeartCycle applications, an evaluation of its performances as an instrument for developing p-health applications was conducted. The project's developers were interviewed about their experience with the platform, and subjective feedbacks were collected and analyzed.

The following paragraphs detail how requirements were obtained and how the platform was designed and implemented.

2.1. *Requirements*

A first version of requirements was created by analyzing relevant literature. In order to confirm and complete our assumptions we complemented this analysis with a survey we submitted to senior engineers working in the field of p-health. The objectives of the survey were:

- to generate a profile of the common expertises in the field;
- to identify the most common problems in the development of p-health patient applications;

- to determine preferences related to the adoption of architectural designs;
- to collect the needs of developers concerning development tools;
- to identify the common sources of data used in the domain.

We interviewed 11 software engineers who had been working in the domain in the last 5 years, including contacts from both research institutions and industry. In total 2 programmers, 1 designer and 8 people who covered both profiles were included; their roles spanned from project managers and developers to specialists in user interaction and educators.

The main findings of the survey were:

1. The use of a well defined architecture is considered important (7 of 11), and a relaxed model is preferred over a strict one (7 of 11). The design of the architecture is considered important, especially for structuring team working and for documenting and communicating the design among developers.
2. Both designers and programmers use visual tools (11 of 11) and consider their use especially useful for making development more intuitive, and for shortening the distance between design and development. They also aim at having improved tools (10 of 11) for their work, especially the programmers (8 of 10).
3. The use of tools is especially appreciated for designing the graphical user interface (GUI) and the data model (8 of 10). Regarding data models, both relational databases (5 of 10) and OWL ontologies (2 of 10) are used as formalisms.
4. Contextual information is considered very useful for improving the design of applications (4 of 9) or at least of some help (5 of 9).
5. Both programmers and designers identified unclear (8 of 10) or continuously changing requirements (8 of 9) as the major threats to their work. In particular, requirements regarding user interaction seem to be the most volatile (6 of 11) as their changes are frequent and small. They also considered that user interaction must be the central aspect in the development of this kind of applications (7 of 11).

The analysis of these results led to identification of the following high-level requirements for creating the platform:

1. Clear and simple architectural design.
2. Clear separation among: (a) management of data; (b) development of user interfaces; (c) design of user interaction flows; (d) business logic.
3. Provision of visual tools for programming.
4. Ability to exploit contextual data.

2.2. *Design and implementation of the development platform*

The design of the architecture of our platform was based on the separation of concerns suggested by requirement number 2. We identified four main modules whose functionalities are detailed in the following paragraphs. The synthetic representation of the architecture is shown in Fig. 1.

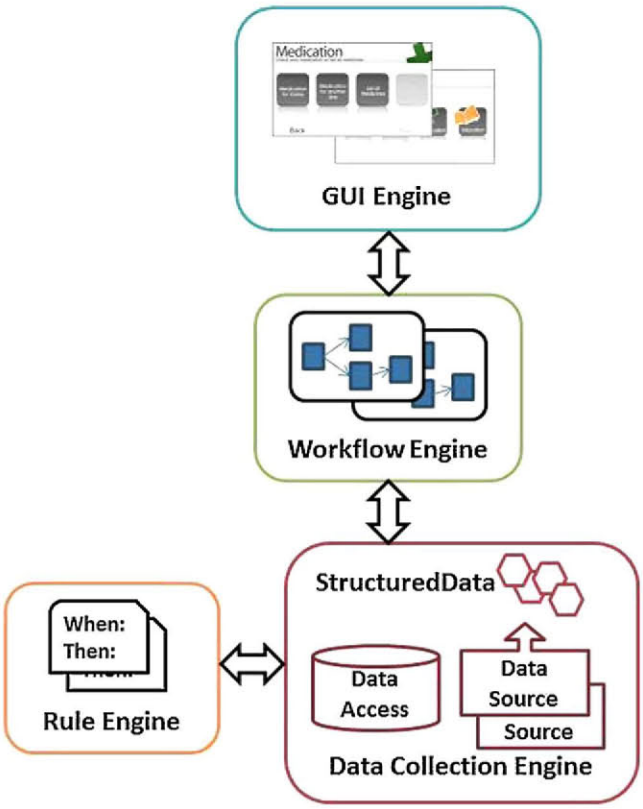

Fig. 1 – The architecture of the platform.

The architecture specifically supports UCD methodologies by placing interaction flows at the centre of development. Concretely, flows are implemented by means of workflows which communicate with two other modules via simplified interfaces.

The final implementation of the platform comprises:

1. A set of interfaces and abstract classes that reflect the architectural design.
2. A set of development tools, including MDE visual tools.
3. A set of unit tests and examples.
4. A tutorial and documentation of the API.

While high-level interfaces were abstracted and simplified, the inner implementation of the modules was obtained by reusing state-of-the-art libraries whenever possible. Most of the libraries already included visual tools while some others were developed ad-hoc. As programming language we chose Java because of the wide availability of libraries, tools, IDEs and because of an explicit preference given in the results of the initial survey (6 developers out of 10).

2.2.1. *Data collection engine*

Many authors recognize that future p-health applications will integrate many sources of information, like bio-sensors, personal profiling, and clinical servers, as well as contextual information [27,28]. The data collection engine (DCE) is the module responsible for acquiring and storing data from these multiple sources, and for making the data available to external

modules. The design of this module has been inspired by typical architectures for context aware systems [29]. It provides a complete set of interfaces for gathering, processing and analyzing data, using both event based and synchronous calls as interaction mechanisms. The module comprises the following constructs:

a) An abstraction named structured data that represents any piece of information. It implements an entity-relationship model that can be easily mapped to tables in a relational database, to classes in an ontology, or to objects in object oriented languages.
b) Data sources: abstractions that generate information as instances of structured data. They can be of two types:
   (i) Synchronous data sources, which generate data on demand. An example can be a temperature sensor that is activated by the system to gather a measurement.
   (ii) Asynchronous data sources, which launch events whenever new data is available. An example can be a weight sensor that is activated when a person steps on it.
c) Data aggregators, data sources that create new semantics from other connected data sources. An example can be a module that retrieves data from a localization system and a weight scale and identifies the exact user whom the weight corresponds to.
d) The Data Access, a module that implements persistence and provides common access to stored data. It offers a method to retrieve and filter data by means of a query language and methods to insert, update and delete single pieces of information.

These modules are meant to be used by developers who have more technical tasks, such as defining the data model of the application, or developing specific data sources to integrate sensors, servers or processing algorithms. The provided abstractions help developers in separating concerns, storing and retrieving data and being alerted when events are generated. Queries are the main mechanism used for external communication with the data collection engine, and for internal communication amongst the modules. Queries are used to explore the stored data, to create filters on events and to identify the data sources to be invoked for performing a measurement.

The DCE module has been implemented on top of both a relational database (using the HSQL library[1]), and OWL ontologies (using Jena[2]). Both relational databases and OWL ontologies have their own graphical editors for modeling data structures.

In order to simplify the development of data sources, a tool for converting structured data to Java objects and vice-versa was also created. This allows developers of the data sources to work with standard Java objects, while others can use SQL or ontologies as their formalism for querying or defining the data model.

[1] http://hsqldb.org/
[2] http://jena.sourceforge.net/

```
rule "patient decompensation"
when:
    Patient weight is increasing
    Bioimpedance is decreasing
then:
    Ask for liquid intake
    Send alarm to physician
```

Fig. 2 – An example of a rule defined with a domain-specific language.

2.2.2. *Rule engine*

The Rule engine provides a means to program the application's logic through the definition of human-readable rules, in the form of "if-then" clauses. It is intended to be used by high-level designers like interaction designers or instructional designers, who are willing to implement logic without having to program low level code. It can be used, for instance, to generate a higher level of knowledge about patients and their environment, for generating messages, or for inferring simple logic that can be used by the interaction flows.

The module is interfaced with the data collection engine in the following way: the conditions in the rules, namely the facts, are expressed by means of queries. The rule engine registers each query as an event filter to the DCE. Thus, each time a piece of data that is filtered by one of these queries changes, an event is raised and the collection of facts in the rule engine is updated. This mechanism minimizes the number of times the rules are evaluated by only launching them when facts are actually detected.

As output, the rule engine can insert or update data behaving as an asynchronous data source of the DCE; this way external modules can access these results by registering to events with other appropriate queries.

The rule engine was developed on top of the JBoss Drools[3] library. Drools was chosen for the availability of visual editors and also for the possibility of creating domain-specific languages that make rules very easy to be read by people with limited technical skills.

Fig. 2 shows an example of a rule using a domain-specific language (DSL). With DSL it is possible to express even quite complex logic in a form easily understood by humans.

2.2.3. *GUI engine*

The GUI engine is intended to help interaction and graphical designers develop the building blocks of the user interface, the GUI components. A GUI component, similar to the concept of widget, represents a piece of graphical information that groups basic parts of a GUI, like buttons or labels, here called GUI Elements (examples are shown in Fig. 3).

The interface to the external modules includes methods to load a GUI component, to set its GUI elements' content (text or image) and to register to user interaction events, e.g. mouse clicks.

[3] http://www.jboss.org/drools/drools-expert

Fig. 3 – Examples of GUI components.

The GUI engine was developed with the Java Swing graphical library. The library was chosen because of its long established tradition and for being cross platform; however, when the graphic was highly customized (with many images and colors), performance started to be an issue. Interaction designers and developers can use the visual tools provided with the NetBeans and Eclipse IDEs, which make development simple and completely graphical. Moreover, a tool for automatically mapping standard JPanels to our GUI component interface was created in order to develop the interface entirely with visual tools without any explicit programming.

2.2.4. *Workflow engine*

This module is for interaction designers. It provides central coordination of the system; its role is to execute interaction workflows. A workflow is composed of three elements: nodes, which are calls to specific methods or to sub-workflows; arcs, which define the direction of the flow, and decision points, which examine a workflow variable to decide which arc to choose next (an example workflow is shown in Fig. 4). In this architecture a node can call the GUI engine to show messages and interact with the user and the data collection engine to gather old and new data.

Interaction flows have fixed parts that deal with the execution of a task, like performing a measurement, guiding an exercise, filling a questionnaire or showing some content, while decision points make it possible to have context-aware decision branches, like switching among different options on the basis of the result of a query (e.g. deciding between an indoor or outdoor activity depending on the weather), or responding to contextual events (e.g. a bed sensor detects the user is waking up and the application launches a reminder for a morning drug intake).

The workflow engine was developed using the freely available Jboss JBPM framework [4]. This implementation benefits from the good quality of the libraries and the availability of a user friendly graphical workflow editor. The basic implementation comprises only two nodes: a wrapper of the GUI engine for setting the content of the GUI elements, and a wrapper of the DCE for handling events, querying for data and starting measurements.

When the use cases to be implemented are complex, it may be necessary to extend this set of nodes in order to embed part of the logic in Java code instead of delegating all of it to the workflows. This makes workflows more readable and easy to create without loosing the possibility of quickly editing them when a requirement changes, although it requires the intervention of a programmer to implement the new nodes.

[4] http://www.jboss.org/jbpm

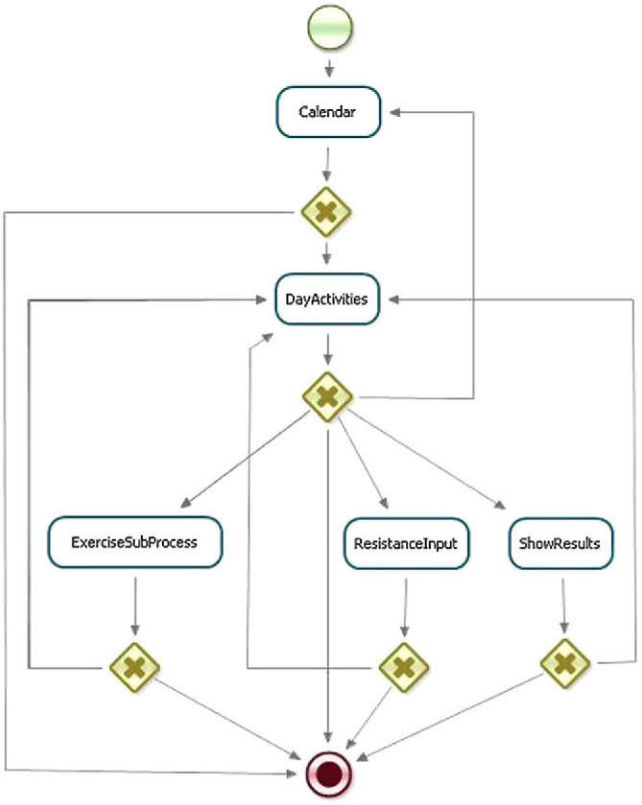

Fig. 4 – Graphical example of a workflow.

3. Results

Our development platform was used to implement part of HeartCycle, a project which aims at providing a closed-loop disease management solution able to serve cardio vascular disease (CVD) patients.

HeartCycle's use cases cover the typical problems of the p-health domain: performing measurements, visualizing contents, communicating with clinical servers, event based logic and exploitation of contextual data. More specifically, the project defines six different use cases in order to meet the goals and needs of heart failure and coronary artery disease patients. These use cases address intake of diuretics, self-management of blood pressure medicines, exercise or physical activity, improvement of the patient's educational and motivational level, continuous assessment of the disease and the management of worsening heart failure. Our platform was used in the implementation of two of them: worsening heart failure and guided exercise. These are presented in

Sections 3.1 and 3.2. The final evaluation of the suitability of the platform in relation to these implementations is shown in Section 3.3.

3.1. *Worsening heart failure*

The worsening heart failure use case is about monitoring heart failure patients, and required measurement of vital signs, dynamic provision of medication and symptom questionnaires, as well as the provision of personalized educational and motivational material depending on the patient's current health and motivational status.

The scenario can be summarized as follows: at a certain time, the user interaction device reminds the patient to measure his weight. Following instructions from the system, the patient steps on an electronic weight scale and the system stores the patient's weight. If this value is above or below a threshold defined by the doctor, the system triggers a screen warning the patient that his weight is out of range. Then the system triggers some questionnaires that aim at finding out the reasons for the weight change. Depending on the answers, the system displays some feedback to promote healthy behavior.

For the realization of this use case two data sources had to be developed, one in charge of communicating with the weight scale sensor and the other taking care of synchronizing the local database with a remote clinical server. The developed GUI components were quite limited as the user interaction was simplified to just a few messages, therefore the workflows were also linear and simple. In addition, a small set of rules to launch motivational messages was developed, based on the weight values and on the answers given to the questionnaires.

3.2. *Guided exercise*

The guided exercise use case defines a more complex scenario which addresses post-infarction patients in their rehabilitation phase. It comprises an embedded sensor that measures ECG and other relevant physiological parameters, a PDA that is used to guide the user during outdoor exercises and a PC application, programmed with our platform, which is used to allow the patient to configure his exercise sessions and to receive useful feedback and motivational and educational messages.

An example scenario can be described as follows. It is a sunny day and the PC suggests that the patient go out for a running session. The PDA starts the exercise routine by prompting the user to measure his blood pressure; the user enters the blood pressure values in the PDA and then answers a questionnaire about any possible symptoms. The system analyses these inputs and decides that it is safe for the user to train today. The user then puts on the sensor and the PDA starts instructing him about performance of the exercise - whether he needs to go faster or slower, depending on the target heart rate set by the doctor. Once the exercise is finished the patient connects the PDA to the PC which receives the session measurement and shows graphs of user's performance. Based on these data, a message is shown for motivating the user to do better, and educational content is presented to explain the benefits of regular physical exercise.

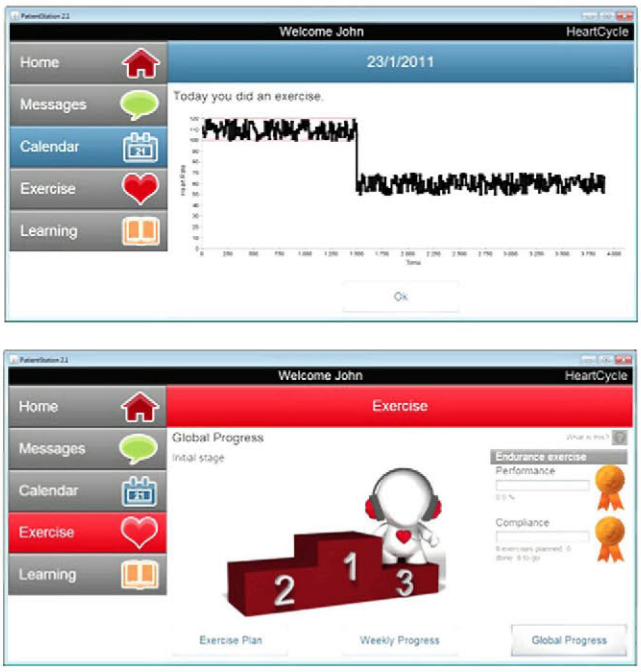

Fig. 5 – Screenshots of the guided exercise application.

The implementation of this use case was more challenging than the previous one. A richer user interaction had to be designed, which implied a wider set of GUI components (Fig. 5 shows two screen-shots) and a set of ad-hoc nodes to be used in quite complex workflows. There was a need for more data sources: one for synchronizing the PDA, created with web services; one for scheduling changes to the exercise plan; one for synchronizing with the clinical server; one for logging user activity and one for getting contextual information such as weather or public holiday dates from free on-line services. The set of rules was also richer, including actions depending on the frequency with which the user accesses the application, the educational information to send depending on exercise performance, and advising changes to the exercise plan whenever a suitable new one becomes available.

3.3. *Evaluation results*

In a standard ad-hoc implementation the described use cases would require the implementation of user interaction in a set of classes that would interact with the sensors, the database, and the GUI forms. By separating user interaction flows and high level rules from lower level implementation details, our framework allowed people with limited technical skills to easily map user requirements to workflows and rules without dealing directly with programming code.

The implementation of these use cases involved a system designer who collected the requirements, a developer who implemented the data sources for the data collection engine, a graphic designer who created the look and feel of the application and implemented the GUI components, a motivation and education specialist who programmed the rules and a business logic developer who created custom nodes for the workflow engine and implemented the workflows. The

development was conducted in three main iterations each one comprising requirements collection, implementation and partial evaluations. However, during all the iterations, requirements on specific details (mostly related to the flow of the application) were continuously coming from all stakeholders, and were dealt with as they arose.

Once the implementation was finalized a focus group was run with the development team for evaluating its experience with the platform, and identifying strength and weaknesses. The assessment protocol was semi-structured. It included an open discussion session and a set of prepared questions related to how the platform fits into the development process, and how the productivity of the group was influenced.

The main results can be summarized as follows:

1. The architecture was correctly mapped to the use cases.
2. The architecture eased the separation of work among the different profiles of developers.
3. The use of visual and programming tools was particularly appreciated.
4. The centrality of workflows in the architecture was considered appropriate for the application.
5. The platform eased the work with respect to unstable user requirements.
6. The maturity of the platform was poor at the beginning, but it improved over time.

Unfortunately, no formal logging of development effort was conducted, therefore objective statistics cannot be provided. As a rough estimation it can be said that the first collection of requirements lasted two months, the first working prototype of the platform was built in one year with one man-year of effort, while the development of the HeartCycle applications, together with the refinement of the platform, took around two and a half years.

The user experience of the guided exercise application was also tested by 12 real patients who were asked to perform concrete tasks and invited to express their opinions about the usefulness and the usability of the application. The overall assessment was that the system is very easy to use and that patients feel they will be helped and motivated by it during the rehabilitation process. The application proved to be stable and is currently adopted within a clinical trial which involves 60 patients split across three different European countries.

4. Conclusions and future work

This paper shows how the implementation of complex and dynamic interaction flows is easily translated into a real implementation in terms of workflows and rules. Event-based interaction, event filtering and structured workflows have been shown to be beneficial in providing an appropriate environment for developing rich user experience.

The development of the HeartCycle use cases demonstrated how the proposed platform could be successfully integrated with the methodology adopted by a diverse, multi-disciplinary team of engineers. The main advantage introduced by the adoption of the platform was that, thanks to its clear separation of concerns and the use of visual tools, it allowed people with low technical skills to implement part of the solution, releasing the programmers from part of the burden of the development and enabling other stakeholders to directly influence the implementation in accordance with their wishes. The main drawbacks are related to the choice of the imported libraries which were under utilized with respect to their potential, and made the system very demanding in terms of computing resources and not portable, for instance, to mobile environments like J2ME or Android.

The scope of this work is limited to a particular project and medical area. Future work will aim at expanding the collection of requirements and validating the platform with more use cases from other medical fields (e.g. diabetes, Parkinson disease), or other related technological domains (e.g. Ambient Assisted Living) to prove the suitability of the platform in an extended range of scenarios.

REFERENCES

[1] F. Sassi, J. Hurst, The prevention of lifestyle-related chronic diseases: an economic framework, Technical report, OECD, 2008.

[2] The case for personalized medicine, Technical report, Personalized health coalition, 2009.

[3] Personalized health care pioneers, partnerships, progress, Technical report, United States Department of Health and Human Services, 2008.

[4] E. Sabaté, Adherence to Long-Term Therapies: Evidence for Action, World Health Organization, 2003.

[5] W.C. Willett, Balancing life-style and genomics research for disease prevention, Science 296 (2002) 695–698.

[6] K. Dean, EU: connected health essays from health innovators, Technical report, Cisco systems, 2008.

[7] J. Bowis, et al. eQuality in eHealth - stakeholders' reflections on addressing e-health challenges at the European level improved healthcare, Technical report, Health First Europe, 2011.

[8] V. Kamat, Connected health: can it save the US healthcare system ?, Technical report, Cambridge Consultants, Cambridge, England, 2010.

[9] European Commission, Ict challenge 5: objective ict-2011.5.1: personal health systems (phs), http://cordis.europa.eu/fp7/ict/programme/challenge5-objective5-1_en.html, 2012, Accessed on January 31, 2012.

[10] European Commission, Industrial innovation, ehealth, http://ec.europa.eu/enterprise/policies/innovation/policy/lead-market-initiative/ehealth/index_en.htm, 2012, Accessed on January 31, 2012.

[11] A. de Vito Dabbs, B.A. Myers, K.R.M. Curry, J. Dunbar-Jacob, R.P. Hawkins, A. Begey, M.A. Dew, User-centered design and interactive health technologies for patients, CIN: Computers, Informatics, Nursing 27 (2009).

[12] M. Rodriguez, G. Casper, P. Flatley-Brennan, Patient-centered design: the potential of user-centered design in personal health records, Journal of American Health Information Management Association (2007).

[13] E. Villalba, D. Salvi, M. Ottaviano, I. Peinado, M.T. Arredondo, A. Akay, Wearable and mobile system to manage remotely heart failure, Transactions on Information Technology in Biomedicine 13 (2009) 990–996.

[14] A. Sutcliffe, S. Thew, O. de Bruijn, I. Buchan, P. Jarvis, J. McNaught, R. Procter, User engagement by user-centred design in e-health, Philosophical Transactions of the Royal

Society A: Mathematical, Physical and Engineering Sciences 368 (2010) 4209–4224.
[15] C. Larman, Agile and Iterative Development: A Manager's Guide, Addison-Wesley Professional, 2003.
[16] A. Qumer, B. Henderson-Sellers, A framework to support the evaluation, adoption and improvement of agile methods in practice, Journal of Systems and Software 81 (2008) 1899–1919.
[17] Z. Hussain, H. Milchrahm, S. Shahzad, W. Slany, M. Tscheligi, P. Wolkerstorfer, Integration of extreme programming and user-centered design: lessons learned, Agile Processes in Software Engineering and Extreme Programming 31 (2009) 174–179.
[18] J.O. Borchers, A pattern approach to interaction design, in: Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, DIS '00, ACM, New York, NY, USA, 2000, pp. 369–378.
[19] S. Blomkvist, User-centred design and agile development of IT systems, Licentiate thesis, Department of Information Technology, Uppsala University, 2006.
[20] S. Kent, Model driven engineering, in: IFM '02 Proceedings of the Third International Conference on Integrated Formal Methods, Springer-Verlag, London, UK, 2002, pp. 286–298.
[21] S. Jeschke, H. Vieritz, O. Pfeiffer, Developing accessible applications with user-centered architecture, in: Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on, 2008, pp. 684–689.
[22] D. Salvi, I. Peinado, S. Salvi, M.T. Arredondo, An implementation framework for personalized health applications, in: MEDICON 2010, XII Mediterranean Conference on Medical and Biological Engineering and Computing, Conference Program, Chalkidiki, Greece, 2010.
[23] Z. Qin, J. Xing, X. Zheng, Software Architecture, 1st ed., Springer, 2008.
[24] M. Büscher, M. Christensen, K.M. Hansen, P. Mogensen, D. Shapiro, Bottom-up, top-down? Connecting software architecture design with use, in: A. Voß, M. Hartswood, K. Ho, R. Procter, M. Rouncefield, R. Slack, M. Büscher (Eds.), Configuring User-Designer Relations: Interdisciplinary Perspectives, Springer Verlag, 2007.
[25] H. Reiter, N. Maglaveras, HEARTCYCLE Consortium, The overall concept of heartcycle for phealth delivery to chf and cad patients through new multiparametric methods, in: Proceedings of pHealth 2010.
[26] P. Duvall, S. Matyas, A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, first edition, Addison-Wesley Professional, 2007.
[27] N. Briconsouf, C. Newman, Context awareness in health care: a review, International Journal of Medical Informatics 76 (2007) 2–12.
[28] D. Zhang, Z. Yu, C.-Y. Chin, Context-aware infrastructure for personalized healthcare, Studies in Health Technology and Informatics 117 (2005).
[29] D. Salvi, M. Ottaviano, I. Peinado, M. T. Arredondo, An architecture for data collection and processing in context-aware applications, in: Adjunct Proceedings of the 3rd European Conference on Ambient Intelligence, AmI09, Salzburg, Austria.