MyCare Card Development: Portable GUI Framework for the Personal Electronic Health Record Device

V. O. Rybynok, P. A. Kyriacou, *Senior Member, IEEE*, J. Binnersley, and A. Woodcock

***Abstract*—In most emergency situations, health professionals rely on patients to provide information about their medical history. However, in some cases patients might not be able to communicate this information, and in most countries an online integrated patient record system has not been adopted yet. Therefore, in order to address this issue the ongoing project *MyCare Card* (MyC^2, www.myc2.org) has been established. The aim of this project is to design, implement, and evaluate a prototype patient held electronic health record device. Due to the wide range of user requirements, the device, its communication interface, and its software have to be compatible with many common platforms and operating systems. Thus, this paper is addressing one of the software compatibility matters—the cross-platform GUI implementation. It introduces a portable object-oriented GUI framework, suitable for a declarative layout definition, components customization, and fine model-view code separation. It also rationalizes the hardware and software solutions selected for this project implementation.**

***Index Terms*—Graphical user interface (GUI), health care, patient care, personal health record, Python.**

I. Introduction

IN MANY areas of health care, particularly emergency care, health professionals rely on patients to provide information about their medical history. However, reliable information may be difficult to acquire from patients who are unwell, confused, or have communication difficulties. It is suggested that a personal electronic health record device might empower patients to be aware and have more control of their health status. Such a technology will also enable them to have the most updated health and personal information at their disposal which will provide a means of security and safety [1], [2].

Over the years, there have been many attempts to develop personal electronic medical record devices such as chip-and-pin cards, USB memory sticks, etc. [3]. Such technologies have been embraced by people and governments with various levels of success (e.g., [1], [4], [5]). In the U.K., there has been a slow uptake of such technologies despite the efforts of the health sector to convert all medical records from paper to electronic format. However, the first attempt to test such technologies in the British public was successfully trialed in the U.K., between 1989 and 1992 [5]. During this trial, over 13 000 patients were provided with smart cards containing health information that only they and health professionals treating them were able to access. The results showed that the majority of participants were in favor of having the cards. However, their use was not continued, as the technology was not sufficiently mature at this time.

In an effort to re-engage such an approach, new social attitude surveys [6] toward electronic personalized records have been conducted in this study with a focus on the identification of the design requirements toward the developments of a new electronic personalized medical record device. The results of these surveys were used in this project to produce the prototype system (record media and access software) which is being continuously evaluated and refactored. The system code name utilized for this project is *MyCare Card*, abbreviated as MyC^2.

This project aims to design and implement a system, which will be intuitive and transparent for users of almost any computer literacy. Therefore, from the beginning, the project development had to be focused on the end-user interface. Thus, the software GUI had to be developed first.

To control the project's complexity, MyCare health record system development was split into two subprojects: *MyCare Card*—medical records storage media device; and *MyCare Card Browser*—GUI and database software, which allows card owners and health professionals to view and edit, where appropriate, information stored on the MyCare Card. The focus of this paper is the development of MyCare Card Browser GUI. MyCare Card design and hardware are only reviewed to the extent required to justify some of the solutions proposed in software (see [7], [8] for further information).

GUI-driven and agile development styles provoked some stability and refactoring issues during MyCare Card Browser development. These were related to the lack of explicit separation in the source code between the following software parts: GUI layout; composite components (e.g., frames, panels); common components view and behavior customizations (e.g., buttons, text boxes); and the binding code between data model and GUI interfaces. Such separation was mandatory due to the source code changes induced by the end-user evaluations conducted

simultaneously with the software development. Different approaches were reviewed concerning how such code could be organized. Considerable effort was made to follow good development practices and to keep the data model and view codes separate. Despite these measures, the card browser software had to undergo at least three major evaluation-refactoring cycles aimed to redefine the source code internal structure.

Eventually, when the source code stabilized, it became possible to recognize certain patterns in its components organization and to form a general object-oriented framework. This framework is suitable for a simple declarative layout definition, chosen components customization, and fine model-view code separation. The concepts comprising the developed GUI framework are discussed in this paper.

The programming language and cross-platform GUI toolkit used in this project are Python and wxPython, respectively. Although all examples shown in this paper are written in Python, the concepts of the presented framework can be used with almost any object-oriented programming language and GUI toolkit.

II. Methods and Materials

A. Initial Requirements

In order to establish preliminary end-user requirements of the MyCare Card, two similar questionnaire surveys were designed to collect attitudes, to patient held records and requirements for an electronic patient held record device, from the public and health professionals. Over 500 participants took part in the survey. Approximately half of these were members of the public and half were health care professionals. Ethical clearance was obtained to consult health care professionals. Data were collected in different areas of the U.K. in order to include participants from different geographical locations and working environments [7].

The combined results were used to derive an initial set of development requirements for the user interface software, the data that needed to be stored and the preferred storage device. When initial development requirements became available, software tool chain, programmatic libraries, and development model were selected. Issue tracking and source code version control systems were also established [8].

To reduce the cost of the development and to minimize its dependence on the major computer systems manufacturers, open source (OSrc) [9] software tools and programmatic libraries were utilized.

Projects that gain most from the OSrc are networking projects and community-driven projects. MyCare is a community-driven project, as its success is directly related to its popularization and wide acceptance of its future communication and data storage standards. Therefore, *MyCare* project *might* achieve its maximum development performance under the OSrc development model [10].

B. Programming Language and Its Environment

The Python programming language has been utilized in this project as a major language, run-time environment, and a common algorithms infrastructure. The C++ programming language was selected for security-related and low-level access algorithms implementation. Simplified wrapper and interface generator (SWIG) is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages including Python. SWIG was chosen in this project to join low-level C++ code components with a high-level Python code. Python library py2exe and the Windows installation package builder named Inno Setup were utilized to make software distribution packages familiar to Windows users.

Python's clean syntax, excellent cross-platform capabilities, extensive libraries, its dynamic and object-oriented nature, and its integration capabilities make it a good choice for a prototyping development project such as MyCare Card. Availability of Python's source code, commercial support from a number of companies, and extensive user community make it suitable for the further development.

In the development of MyCare Card Browser, wxPython was used as a primary GUI toolkit. The core component of wxPython is wxWidgets toolkit. At base, wxWidgets is a GUI component library implemented in C++, which means it is a set of C++ classes that encapsulate a wide range of features. The goal of wxWidgets is to allow a C++ program to compile and run on all supported platforms with only minor changes to the source code from platform to platform, and with a reasonably consistent look and feel between the platforms. Both wxWidgets and wxPython are OSrc libraries.

C. Card Device Interface

In this prototyping study, conventional USB mass storage protocol was utilized to store user's medical data, security and authentication data, and the browser software itself on the MyCare Card. All data are stored on the card in files, which are available to the operational system (OS) services via standard portable disk drivers. USB mass storage devices are natively supported by nearly any desktop OS available today. In further development, for security reasons, additional USB device protocols will need to be implemented. Those protocols will protect MyCare Card user medical data and software files from unauthorized access and from accidental corruption. Therefore, files access algorithms implemented in MyCare Card Browser software are isolated into interface classes to allow future embedding of such security protocols, without affecting other software parts or the overall source code structure.

III. Results

A. Establishing Types of Data Records

The survey investigation showed that 85% of the public said they would find an electronic patient held medical record device useful, especially if they were too ill to give information to a health professional. Approximately half of this group had used some form of patient held health record and cited the major benefit of doing so as being access to health information. The concerns of this group related to inaccuracy of information (74%), loss of records by the patients (80%), and unauthorized

access (75%). Some 94% of the health professionals said they would find a patient held record useful and, in this case, the most common reason was to overcome communication problems [6], [11], [12].

Regarding the types of information that should be stored on the device, most of the members of the public surveyed thought that current medication, name, allergies, blood group, and long-term conditions should be included, and that all health professionals should be able to access these items. However, over three quarters also agreed that access to other information should depend on the role of the health care professional [6].

For health care professionals, the most important pieces of information to be held on the device are related to allergies; current medication; name; long-term conditions; age; major health problems in the past; and next of kin. The majority of these participants thought that all health professionals should be able to access the most important pieces of information. Again, the majority of participants supported the use of a restricted access system, where the viewing of certain pieces of information was restricted to particular groups of professionals [12].

B. MyCare Card Browser Framework

The MyCare Card Browser source code framework has been created in this project. This framework allows the source code development and maintenance while adding the new software features and modifying existent ones, under the end-user evaluations feedback. MyCare Card Browser software built on this framework should have looked and felt like an end product to allow the end-user group evaluations from the early development stages. The implemented framework can be used to perform the following steps in a systematic way, without breaking the source code execution stability and maintainability:

1) adding new data records, new record types, and formats or removing and modifying existent ones;
2) associating GUI components, designed to display and edit user data, and corresponding data records, available via the data model interface;
3) changing validation and formatting rules for the entered data;
4) moving GUI components around and between container windows, while preserving correct data associations;
5) adding new GUI components, which may (or may not) be associated with the data model;
6) modifying existent GUI components views and behaviors (components styles);
7) centralized controlling of the common GUI parameters, such as font types, proportions, or sizes;
8) combining dynamic layout algorithms (intuitive for end user) with visual GUI designer-generated code (quicker development).

The major design pattern forming the core of the MyCare Card Browser GUI framework is `DataAware` GUI components. Those components and their place in the overall software structure are described in the following sections.

Based on the preliminary data types, and storage media specifications and end-user expectation, an initial prototype of the MyCare Card Browser software has been implemented. This paper describes the developed GUI framework, which has been used to implement MyCare Card Browser. The "view" part of the developed framework is located in `gui` package.

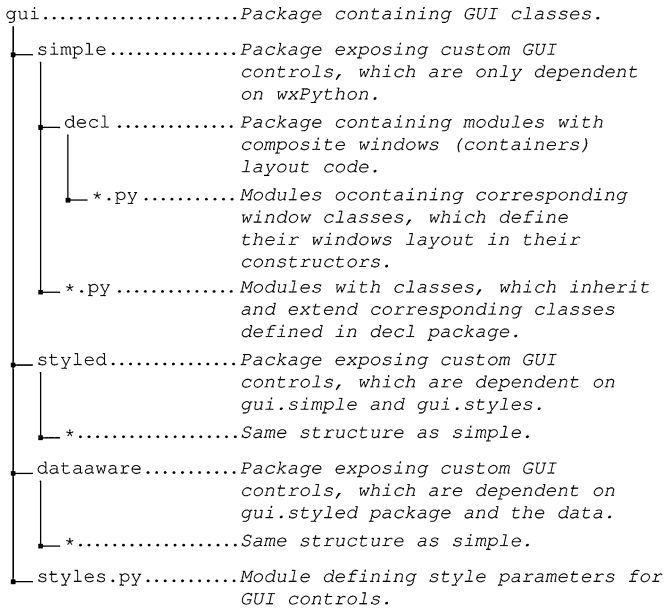

Fig. 1. GUI package namespaces structure.

C. gui Package Namespaces Structure

`gui` package namespace structure which in Python corresponds to its source code directory tree is shown in Fig. 1.

Apart from the `gui` package shown in Fig. 1, MyCare Card Browser includes `mc` module and `data` package. `mc` module ("controller" part) contains DataAware GUI components. These components are classes which by multiple inheritance join GUI components defined in `gui.styles` module and in `gui.styled` (see Fig. 1) package with the data access code contained in `data` package ("model" part).

1) GUI Components Styles: `gui.styles` module has been implemented in MyCare Card Browser to support the concepts of style and parametric control of the GUI components view and interactive behavior.[1] The concept of styles is similar to the one used in typesetting programs. For example, instead of using generic text box `TextCtrl` class directly, it can be inherited and extended by the `SingleLineTextView` class. `SingleLineTextView` can redefine fonts, colors, borders, cursors, and other properties, which control the component view and interactive behavior. `SingleLineTextView` class in its constructor can refer to the global configuration object, which holds parameters, common for the whole application, including its GUI appearance.

Such common GUI controls "style" wrapping has three main advantages:

[1]Term "interactive behavior" is used in this paper to refer to the GUI component sound, visual, tactile, and similar presentation effects. It should be distinguished from the "logic behavior," which is related to the software content and operation.

1) it allows controlling software GUI appearance properties, such as fonts, colors, etc. from a single place—the global configuration object;
2) it gives semantic meaning to GUI components and allows modifying existent GUI components view, shapes, sizes, and interactive behaviors, depending on their purpose in the software;
3) it allows replacing underling (wrapped) common GUI components with the new ones, and introducing new components without interfering with other parts of the software.

Although `gui.styles` module and `gui.styled` package names may look similar, their purposes and contents are essentially different. All packages included in `gui` package are defining window layouts of the software composite GUI components, at various layout nesting levels.

2) GUI Layout: There are three packages contained in the `gui` package: `gui.dataaware`, `gui.simple`, and `gui.styled`. All these packages have similar namespaces structure shown in Fig. 1. In this paper, those packages are collectively referred as `layout_package`. `layout_package` can be seen by the rest of the software as a separate module which namespace includes few composite window classes such as frames or panels.

In such directory structure, dynamic layout logic can be separated from the layout optimization and interactive behavior algorithms. In this GUI framework, dynamic layout logic can be used jointly with some visual designer software (such as wxGlade). Each *layout class* that defines window layout logic is placed into `gui.layout_package.decl` package.

Layout classes defined in `gui.layout_package.decl` specify sizers-based windows layout logic in their constructors. Those layout classes are inherited by the *behavior classes*, which define windows layout optimization and interactive behavior algorithms. These algorithms are located in the class constructors and event processing member functions.

The behavior classes are defined in the modules in `gui.layout_package`. Names of the behavior classes are the same as the layout classes from which they inherit—they are distinguished by their namespaces—layout classes are in `gui.layout_package.decl` and behavior classes are in `gui.layout_package` namespaces. Following the tradition of GUI programming in MS Windows, "CamelCase" style with the first capital letter is used for the window class names.

Three separate layout packages (see Fig. 1) are used in this software due to the different meaning of their contents and also for avoiding circular dependences. `gui.simple` package is only using wxPython library. `gui.styled` package is using GUI components defined in `gui.styles` module and it can also use `gui.simple` package and wxPython library. Thus, `gui.styled` package defines composite windows, which contain styled components. `gui.dataaware` package can use the same resources as `gui.styled` package, and additionally is using DataAware GUI components, which are defined in `mc` module.

Separation between layout and behavior classes allows achieving independence from the technique by which composite window layouts are defined. Therefore, the utilized GUI visual designer software can be easily replaced with another one. The format in which the window layout is defined can also be changed easily. For example, the native Python code can be replaced by the dynamically loaded wxWidgets native xrc resources file format to achieve better customization flexibility.

3) Data-Flow Model: `mc` module represents a glue point that joins the data model defined in `data` package and the GUI components defined in `gui.styled` package. It introduces DataAware GUI components, which display and edit records available in the data model. Those connected-to-data components are used by `gui.dataaware` package to layout their instances on the containing windows, which are directly visible to the end user.

4) Data Access and Security: The class defined in `data` package, which represents data model is not accessed directly by the DataAware GUI components defined in `mc` module. Instead, the instance of the data model class is created and exported in `storage` module. In the future, after the current prototyping development stage, MyCare Card user's data should be moved into more secure location, which will not be available via the standard file system. `storage` module data model access indirection will allow us to only modify some of its functions in order to access data on another location.

Algorithms, which are accessing user's data will be located in a module abstracted by the `storage` module. The user data will be encrypted and decrypted by the data access algorithms on the fly. This approach will allow secure data storage regardless of the stored data format (e.g., `dict`).

`security` module implements authentication and authorization algorithms. Currently, those algorithms only do text strings matching to mimic correct operation during the end-user evaluations. In further development, more advanced security algorithms will need to be implemented in this module alongside with session management and data encryption keys validation (see Section IV for discussion).

D. DataAware GUI Components

Based on the list of the data record types (see Section III-A)) that are expected to be stored on MyCare Card, the model of health data has been designed. In the current version of MyCare Card Browser, this data model is represented by the native Python type `dict`. `dict` is the class used to create mapping objects, i.e., such objects that contain `key : value` pairs. Another `dict` object can be assigned to the `value`, making it possible to create nested data records or tree-like structures.

As `dict` data type is embedded in Python, it does not require additional access drivers. For this type of applications, however, the `dict` class has the major disadvantage—it has to keep the whole dataset in the main memory. At this prototyping stage of MyCare project development, the user medical data were relatively small, and `dict` was sufficient to store and process it without causing significant memory overhead or operation delays.

As the project and its data requirements grow, the `dict` approach may become inadequate. Therefore, to abstract the data model from the details of its implementation as the `dict` class, the data model interface—`DataAccessor` class—has been

implemented. It effectively isolates access to the `dict` from other parts of the software that require access to the data model. Such an interface allows modular replacement of `dict` by another class or even by a full database system.

In order to make `DataAccessor` independent of the underlying data model, the concept of `DataPath` was introduced. It seemed reasonable to directly associate GUI components with the paths to records in the data model. In wxWidgets, objects of GUI component classes are related to each other via logical parent–child relationships. Components which are visually containing other components are parents to the components which they are containing. For example, the text box which is visually contained on the panel is panel's child. These visual and logical relationships can be used to compute relative paths to the records in the data model.

For example, there might be a name editor component. This name editor component may be represented by a panel class that contains four text boxes to edit title, first name, middle name, and last name. User's name in the data model may be located at `/personal/name` path. The name parts may be accessed by their relative paths, such as `title`, `first_name`, etc. Then, path to the name might be associated with the panel, and the individual name part paths might be associated with the text boxes. Thus, each text box knows its parent's path and its own path and can compute its full path to the field in the data model, e.g., `/medrec/personal/name/title`.

Such a representation of the relative paths by the logical child–parent relationship allows implementation of the generic composite GUI controls that are connected to the data model. The described name component, for example, might be used to edit card owner name, next of kin name, or any other name used in the program by changing its path to the record in the data model.

In order to avoid separate maintenance of GUI components' behavior logic and their paths to the data model, it was found convenient to edit such associations via a visual GUI editor. At the same time, components' behavior and data model access algorithms should have been kept separate to allow their modifications independently of each other. In order to meet those requirements, `DataControl` polymorphic class and its derivatives have been introduced (see [8] for the full source code of MyCare Card Browser).

All classes inherited from `DataControl` will have `DataAccessor` and `DataPath` properties. They will also be able to call `_GetData()` member function to obtain record from the data model, associated with them via `DataAccessor` and `DataPath`.

To support the concept of relative paths and to call `DataControl.UpdateControlFromData()` member functions when a control update is required, the `DataControlContainer` polymorphic mixin class has been defined.

View controls generally display their data record in some compact form, and depending on their setting may not show all record parts. Edit controls are bigger on the screen and display all record parts. `DataViewControl` and `DataEditControl` polymorphic classes were introduced to reflect this difference.

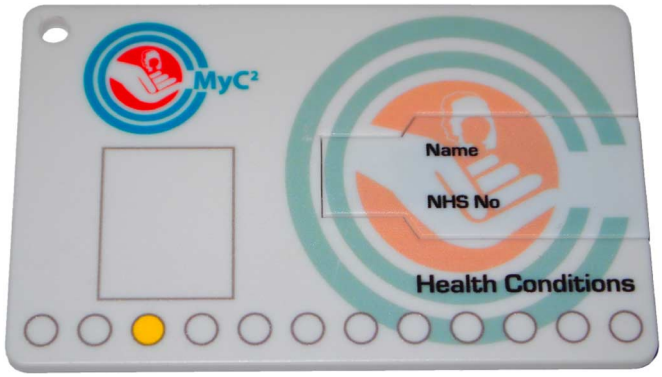

Fig. 2. *MyCare Card* working prototype with USB interface.

The implemented data-aware and styled GUI components can be used by a visual GUI designer to layout top windows and composite components. Due to the polymorphic and mixin implementation of the data-aware GUI components, the designer-generated code is isolated from the data model structure and from the components implementation.

E. MyCare Card Form-Factor

According to the conducted survey (see Section II-A), smart card media type was preferred over other proposed devices such as USB sticks, key fobs, jewelry, and devices linked to a mobile phone.

Considering the technical advantages and disadvantages of traditional smart cards and modern USB sticks, the use of USB sticks is preferable for *MyCare Card* implementation [7]. However, given the public's preference for a smart card, a compromise had to be found between the two media types. This compromise was a USB card, example of which is shown in Fig. 2. The USB card design combines the advantages of both smart card and USB stick media types.

F. MyCare Card Browser Implementation

MyCare Card Browser has been implemented utilizing the software framework, which key concepts were described in the previous sections. In this section, the resulting graphical interface is presented from user's perspective.

When the MyCare Card user inserts the card into a USB port of the computer, the "welcome" window shown in Fig. 3 appears on the screen. It only happens if the portable drive autorun option is enabled on the host operating system; otherwise, the user should run the program in a conventional way. The "welcome" window briefly outlines the purpose of this software and provides the user with the two login options. The first option is to login as the card owner and the second is to login as a health professional. Additional warning button on the welcome screen shows a security message, stating that all health professionals who have been issued with their own personal identification number (PIN) can see user's data.

When the user clicks "MyCare Card Owner" button for the first time, the program shows the card owner new PIN dialog. In this version of *MyCare Card Browser*, the concept of PIN,

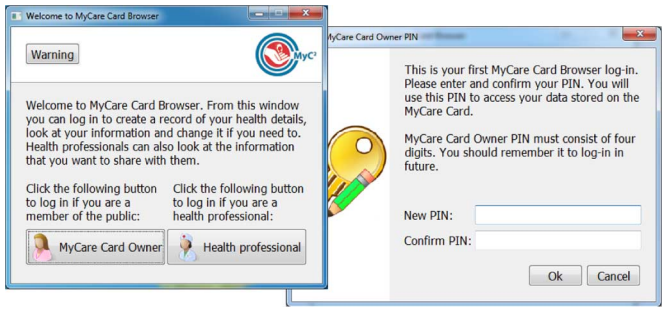

Fig. 3. *MyCare Card Browser* welcome and new PIN windows.

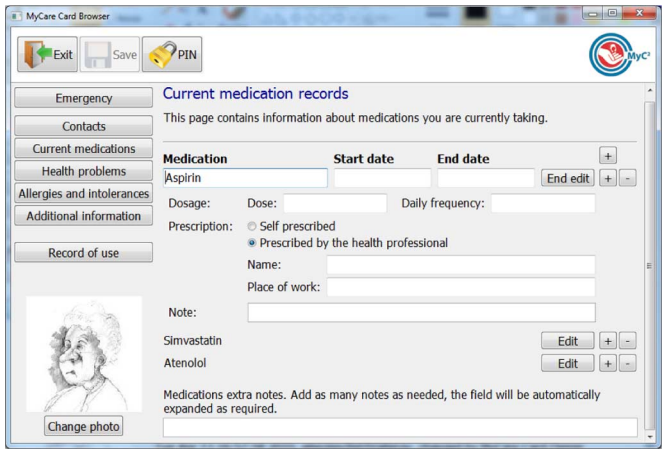

Fig. 4. *MyCare Card Browser* framework operation example, demonstrating dynamic layout in GUI designer-generated components.

similar to the one used in bank cards, was utilized. In further development, it will be replaced with a longer password for stronger data protection (see Section IV for discussion). After the user enters and confirms the new PIN, the "welcome" and PIN dialog windows are getting hidden and the main *MyCare Card Browser* window appears on the screen in a user mode. When the user clicks the "Health professional" button, the program shows the health professional login dialog. When the health professional enters a correct PIN, the "welcome" and PIN dialog windows are again hidden and *MyCare Card Browser* main window appears on the screen in a health professional mode. Similar to the card owner PIN, the health professional PIN will be replaced by the identification number and password pair in further development. In the present version of the browser, health professional credentials are stored on the card itself, which would be impractical for the real-life application. In further development, the remote health professional authentication should be implemented (see Section IV).

Example of the typical *MyCare Card Browser* screenshot is shown in Fig. 4. On this screen, the browser's main window is displayed. The tab panel opened in the main window is the current medication records editor. Although it might be difficult to demonstrate on the static media, this figure shows the dynamic nature of the GUI framework. The last two medications, identified as "Simvastatin" and "Atenolol" in Fig. 4 are displayed as a compressed list. When the user clicks the edit button, the interface expands to show the medication editing panel. All long-text fields are also automatically expanded to accommodate the entered text. When any of the GUI components expands to accommodate more data or to display more information, the window layout is automatically adjusted. Refer to [8] for more screenshot examples and for the *MyCare Card Browser* demonstration.

IV. Conclusion and Further Work

A. Conclusion

The portable, modular, and extendable GUI software framework has been implemented in this project. The main advantage of this GUI framework is that it allows concurrent development and end-user evaluations, while preserving the program stability and code maintainability. Additionally, it lets the software developer to combine GUI designer-generated code with dynamic layout logic in a direct unobscured way. These qualities make the presented framework useful for the end-user-driven development of the personal heath record solutions.

The core design pattern forming the base of this framework is the `DataAware` GUI controls. The underlying concepts used to implement `DataAware` GUI controls are the styled GUI controls and `DataControl`-based classes, described in this paper.

Utilizing this software framework, the *MyCare Card* system has been successfully developed. This system is a prototype of a credit card-sized personal medical record USB device (see Fig. 2) and the GUI software, designed to provide access to the medical data, stored on the card to the MyCare Card owner and health professionals.

Fifty MyCare Card test units (see Fig. 2) have been manufactured. Those manufactured cards are currently being evaluated in a real-life trial by potential users in the U.K. The initial users response has been positive and indicates that such cards will be a useful acquisition for both members of public and health professionals.

The research highlights that the initial barriers to the use of electronic health record devices will be the security of information. In the proposed system, data can only be accessed using a personal identification number, and will only include information that the individual is willing to enter and share with others. A similar level of authentication will be required by health professionals to read it and, therefore, such fears appear to be exaggerated.

Additionally, the device will be independent of centrally held records, so will not provide a route in to more sensitive information. The developed *MyCare Card Browser* interface classes allow embedding of the user authentication and data encryption algorithms minimizing the potential for fraudulent use of devices that have been reported as lost or stolen.

B. Further Work

The early development versions of MyCare Card Browser featured data fields access control based on security groups and

individual health professional credentials [7], [8]. Security groups were formed by health professional roles in medical care. However, the conducted end-user evaluations have demonstrated that the concepts of security groups and access levels are difficult to understand for most people. Therefore, further work is needed to simplify the security-related parts of the GUI and abstract its internal parameters further from the end user. More specifically, by default, every field in the personal health database on the MyCare Card should have the "Private" Boolean property, represented by the check box in browser's GUI. MyCare Card Browser will have two operation modes: online and off-line. In online mode, every health professional can be authorized by his/her number and password via an authentication server. Thus, every authorized health professional will be able to access all fields on the card.

In off-line mode, the authentication server is not available, and health professional remains unauthorized. Unauthorized health professional may not access fields, which are checked as "Private" by the card owner. In off-line mode, health professionals still need to enter their numbers and passwords. MyCare Card Browser will use the entered number and password to compute a hash code and store it on the card. The unauthorized health professional should be able to edit health data fields on the card. All changes made by the unauthorized health professional are stored in a postponed cache on the card and are not applied to card owner's health records database.

When MyCare Card Browser gets online, it automatically verifies health professional number and password hash codes via authentication server. If health professional authorization passes, the postponed cache changes are applied to the health database on the card. If authorization fails, all postponed cache changes are rejected.

Nonhealth professional should be able to access the nonprivate health data fields in the read-only mode.

MyCare Card Browser online and off-line operation modes are transparent for the card user. The "Private" property is intuitive and self-explanatory for most people, and it effectively hides security-related terms such as access levels or groups from the MyCare Card user. One of the questions that is still open is whether the card user should be able to mark any field as "Private," or some of the field types should be impossible to hide from nonhealth professionals. Such open fields might include name, blood group, allergies, and other emergency-related data.

The data backup server might be implemented to keep user's medical data in cases when the card is lost or damaged. Data synchronization interfaces might be implemented to synchronize data on the card with the currently used medical data systems. Synchronization interfaces and backup server should help maintaining currency of the medical data.

Future stages of the research will entail the development of more detailed usage scenarios, testing of the MyCare Card and evaluation of the usability of the MyCare Card Browser user interface. Further, MyCare Card Browser development will include selection and implementation of the card data encryption algorithms, health professional authentication methods, corresponding authentication server, and client side algorithms.

References

[1] K. Häyrinen, K. Saranto, and P. Nykänen, "Definition, structure, content, use and impacts of electronic health records: A review of the research literature," *Int. J. Med. Informat.*, vol. 77, no. 5, pp. 291–304, May 2008.

[2] M. J. Ball, M. Y. Costin, and C. Lehmann, "The personal health record: Consumers banking on their health," *Stud. Health Technol. Informat.*, vol. 134, pp. 35–46, 2008.

[3] F. L. Maloney and A. Wright, "USB-based personal health records: An analysis of features and functionality," *Int. J. Med. Informat.*, vol. 79, no. 2, pp. 97–111, Feb. 2010.

[4] H. Phipps, "Carrying their own medical records: The perspective of pregnant women," *Australian New Zealand J. Obstetrics Gynaecol.*, vol. 41, no. 4, pp. 398–400, 2001.

[5] NHS Management Executive, *The Care Card Evaluation of the Exmouth Project*. London, HMSO, 1990.

[6] J. Binnersley, A. Woodcock, P. Kyriacou, and L. Wallace, "Establishing user requirements for a patient held electronic record system in the united kingdom," presented at the Human Factors and Ergonomics Society 53rd Ann. Meeting, San Antonio, TX, Oct. 2009.

[7] V. Rybynok, P. Kyriacou, J. Binnersley, A. Woodcock, and L. Wallace, "MyCare Card development: The patient held electronic health record device," presented at the 9th Int. Conf. on Information Technology and Applications in Biomedicine (ITAB), Larnaka, Cyprus, Nov. 4–7, 2009.

[8] MyCare Card dev. site. [Online]. Available: http://www.myc2.org/.

[9] Open Source licenses. Open Source Initiative (OSI). [Online]. Available: http://www.opensource.org/licenses.

[10] E. Raymond, *The Cathedral & The Bazaar*. Sebastopol, CA: O'Reilly Media, 2001.

[11] J. Binnersley, A. Woodcock, P. Kyriacou, and L. Wallace, "Public requirements for patient held records," in *Contemporary Ergonomics*, London, U.K., Apr. 2009, pp. 159–164.

[12] J. Binnersley, A. Woodcock, P. Kyriacou, and L. Wallace, "An investigation of health professionals' attitudes towards patient held records," presented at the Int. Ergonomics Association Conf., Aug. 2009.

Victor Rybynok received the Higher Education Dipl. in computer systems and networks engineering from Moscow Institute of Electronic Technology, Moscow, Russia, in 2001, and the M.Sc. degree in medical electronics and physics from Queen Mary, University of London, London, U.K., in 2003, and the Ph.D. degree in biomedical engineering from City University London, London, in 2009.

He is currently a Postdoctoral Research Fellow in Biomedical Research Group in City University London. His current work in biomedical technology includes the development of noninvasive methods for the blood chemical composition analysis using diffusion spectroscopy techniques.

Panayiotis Kyriacou (SM'97). He received the B.E.Sc. degree in electrical engineering from the University of Western Ontario, Canada, and the M.Sc. and Ph.D. degrees in medical electronics and physics from St. Bartholomews Medical College, the University of London, London, U.K. His PhD research was in the field of medical optics and biomedical instrumentation. His other academic achievements include CEng, CPhys, CSci, FIET, FIPEM, FInstP, FRSM.

He is currently a Professor of Biomedical Engineering at City University London, London. He is also an Associate Dean for Postgraduate studies in the School of Engineering and Mathematical Sciences and the Director of the Biomedical Engineering Research Group. His research interests include the understanding, the development, and the applications of instrumentation, sensors and physiological measurement for the facilitation of the diagnosis and treatment of disease or the rehabilitation of patients.

Jackie Binnersley is working toward the Ph.D. degree at Coventry University, Coventry, U.K.

Andree Woodcock received the Design degree in psychology and social biology and the M.Sc. degree in ergonomics from the University of California.

She is Professor of educational ergonomics and design at Coventry School of Art and Design, Coventry University, Coventry, U.K. Her Ph.D. research from Loughborough University, Loughborough, U.K. involved developing a decision support system to enable automotive designers to integrate user requirements into concept design. She is Director of the Centre of Excellence in Product and Automotive Design. She is currently engaged on a number of projects related to the Digital Economy. Her interests include computer supported co-operative working, ageing and health, sustainable transport and digital inclusion.