D R A F T

Towards Automatically Generating Graphical User Interfaces from *open*EHR Archetypes

Thilo Schuler[a,b,1], Sebastian Garde[b], Sam Heard[b], and Thomas Beale[c]
[a] *University of Freiburg, Department of Medical Informatics, Freiburg, Germany*
[b] *Health Informatics Research Group, Central Queensland University, Rockhampton, Australia*
[c] *Centre for Health Informatics and Multiprofessional Education, University College London, United Kingdom*

Abstract. One of the main challenges in the field of Electronic Health Records (EHRs) is semantic interoperability. To utilise the full potential of interoperable EHR systems they have to be accepted by their users, the health care providers. Good Graphical User Interfaces (GUIs) that support customisation and data validation play a decisive role for user acceptance and data quality. This study investigates the use of *open*EHR archetypes to automatically generate coherent, customizable, data-validating GUIs. Using the Mozilla XML User Interface Language (XUL) a series of prototypes has been developed. The results show that the automatic generation of GUIs from *open*EHR archetypes is feasible in principle. Although XUL revealed some problems, the advantages of XML-based GUI languages are evident.
Keywords: Computerized Medical Record, User-Computer Interface, Archetype, openEHR, Electronic Health Record.

1 Introduction

'Semantic interoperability' is the goal when it comes to Electronic Health Records (EHRs) and the perceived means by which medical informatics can improve today's shared healthcare environment. Standards development organisations, open source communities, research groups, and early implementers in different countries (usually funded by their governments) are putting in considerable effort to make this vision a reality.

To achieve semantically interoperable and future-proof EHR systems, the *open*EHR initiative (http://www.openehr.org/) introduced a two-level methodology ([1]). Formally expressed, distinct, reusable domain-level content models, called archetypes, are used to constrain data instances that conform to a generic reference model

[1] Corresponding Author: thilo.schuler@gmail.com

([2]). The basic constraining notion of this approach has been adopted or considered by most major standardisation efforts in the EHR realm. The revised CEN EN 13606 (EHRcom) will incorporate this archetype methodology directly, while the HL7 is working on a templating methodology for this purpose - "HL7 CDA Templates and DICOM Structured Reporting Templates are conceptually very similar to archetypes" ([4]). However, due to features of the HL7 version 3 RIM, these templates are not able to support the clear separation between information and domain models expressed in *open*EHR's two-level methodology.

*Open*EHR archetypes are formally defined using the Archetype Definition Language (ADL). An example for an ADL modelled medical concept would be a blood sugar level archetype. The modelling process is supported for example by the Ocean Informatics Archetype Editor available under the Mozilla Tri-license (http://oceaninformatics.biz/). Using this editor, clinicians without informatics skills can define archetypes themselves and therefore guarantee flexibility and coverage of clinical documentation needs ([6]). According to the *open*EHR methodology, several archetypes are grouped to form a particular data structure e.g. to document a diabetes monitoring examination. For a user to fill this data structure it has to be presented as a GUI for data entry.

The GUI is arguably the most important part of an EHR system with which a health professional interacts and is therefore of utmost importance for its success ([14], [5], [7], [8]). In this paper, we present a series of prototypes that aim to automatically generate a GUI from *open*EHR archetypes expressed in ADL syntax.

2 Material and Method

Mozilla XML User Interface Language (XUL; [15]) and Microsoft's C# were selected as prototyping technologies. Web-based applications have the advantage that they can be run with minimal requirements on the user side and are easily distributable to many users. Their downside is a restrictive user interface due to the intrinsic limitations of HTML and the traditional web paradigm with complete page refreshes. Compared to desktop applications, web applications are less dynamic (e.g. real-time input validation or context-sensitive interface alterations are generally not supported) and do not offer rich controls such as tabbed panels or tree components. The declarative language XUL tries to combine the strengths of both approaches. XUL uses XML markup to define the GUI in a declarative way. A number of such languages exist ([16]). XUL is one of the oldest and used in all Mozilla software projects, such as the Mozilla Firefox browser, to create the GUI. A browser based on the Mozilla Gecko layout engine (e.g. Mozilla Firefox) has to be used to render XUL markup into rich web user interfaces. An example for a XUL web application is the Mozilla Amazon Browser (MAB; http://www.-faser.net/mab/index.cfm). The XUL framework reuses and supports a broad range of existing W3C standards. In order to implement a web application, XUL uses Cascading Style Sheets (CSS) to change the appearance. Functionality is realised through ECMAscript, the standardised version of JavaScript (http://www.ecma-international.org/publications/standards/Ecma-262.htm). ECMAscript has implemented part of the W3C Document Object Model (DOM), a language neutral interface to access and manipulate XML documents, and provides specialised objects such as SOAPCall, which can be used for the communication with a web service. Custom controls for XUL interfaces can be developed with the XML Binding Language (XBL). Currently XBL has

the status of a W3C Note (http://www.w3.org/TR/xbl/).

An XML based format appears to be of advantage for generating a GUI description as there are many tools for XML editing. The reasons we selected Mozilla XUL were the comparatively long usage as part of the Mozilla projects and available documentation. Other advantages include the built-in "skinning" ability via CSS and the multi-language feature.

The proposed XUL interface communicates with a middleware implemented as a web service. The main reasons for choosing C# to implement the middleware and the GUI generating software were the good support of web service-based communication and availability of an advanced development environment.

3 Results

Before and during the work on the prototypes we thought about possible purposes of a GUI generator that produces declarative markup. We came up with a number of different use cases:

- To give archetype authors a feel for what could result from their work in a real system (preview function).
- To provide a basic interface pattern for EHR system implementers that helps to improve consistency between archetypes and the actual user interface, and also helps to maximise efficiency during the creation and future adaptations of EHR systems (pattern function).
- To enable customisation of the interface in accordance with the health care professional's individual preferences and workflows (customisation function).
- To create efficient, data-driven GUIs which are restricted to the necessary information and function (data-fitting function).
- To assure documentation quality through archetype-based validation (quality assurance function).

At the beginning of our research we produced a model of how the archetype-based generation of GUIs could function. The generation mechanism has been designed to answer the question: "To what extent can GUIs be automatically generated from *open*EHR archetypes?" Real world implementations would require many additional components like security features (encryption, user authentication, etc.). The following diagram describes the system from a high level view:

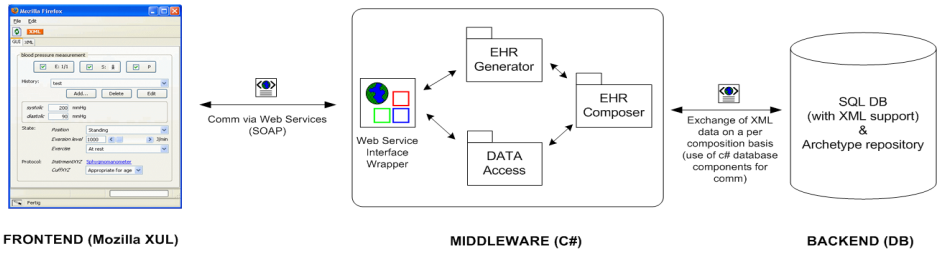

Figure 1: Archetype based GUI generator system

Using this approach as a starting point we implemented a series of prototypes to ana-

lyse different aspects of this concept. The diagram shows a backend, which has been included to get a complete view of a possible system, but the backend is not subject of our research.

3.1 Prototype 1: Handcoded GUI of blood pressure archetype

The intention of this prototype was to test the possibilities of XUL. It helped to gain experience in using the framework to create rich GUIs and to implement communication with the middleware via web service messages. The prototype is one example of what the generator could produce and therefore it helped to assess the applicability of the system design in general.

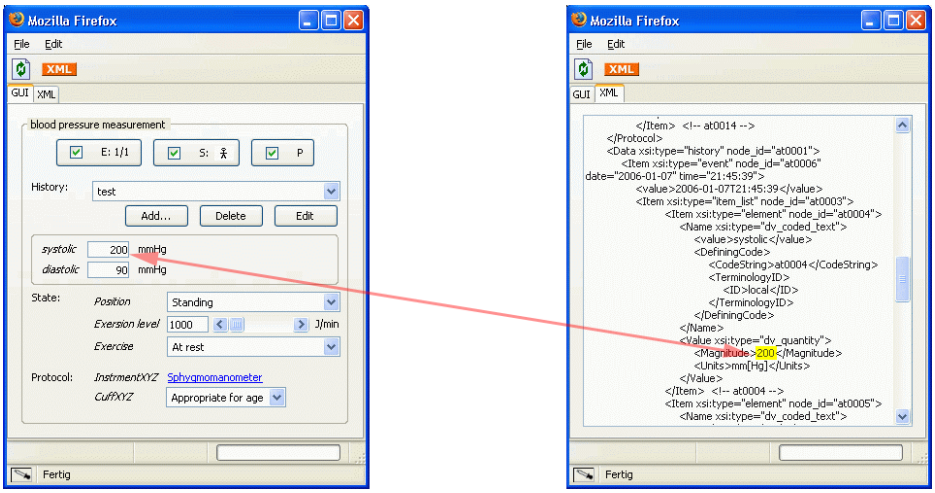

Figure 2: XUL blood pressure GUI and resulting XML data produced by middleware

Figure 2 shows a GUI of a blood pressure archetype. The archetype includes a range of information: the systolic and diastolic values; assignment of these values to a series of events (history) with recording of maximum, minimum and other aggregate values; information about the state of the patient such as position or level of exertion at the time of measurement; and protocol information about the measurement itself such as the site, or size of cuff, if a sphygmomanometer is used. In the GUI the event selector, as well as the state and protocol input fields, can be hidden while their main information is still visible as an icon. The XML tab has only been added to monitor the effects in the middleware. It shows the XML instance that the Composer produces to store in the database (persistence).

Technically the main challenge was to establish communication from XUL, via the SOAPCall object, to the middleware methods exposed as a web service. To enable the recording of a series of blood pressure measurement events, new input controls were dynamically added via the DOM. Depending on the selected event, the corresponding controls are displayed, while the others are hidden. This design decision has the advantage of swapping between events without constantly reloading content.

3.2 *Prototype 2: Custom control to validate archetype defined value range*

Good GUIs should provide as much input validation as possible. Warnings and error messages should appear as promptly as possible when syntactically or semantically inconsistent data is input. Input constraints like value ranges and cardinalities are defined in the archetype. To represent these constraints, a generic set of customized controls (one for each *open*EHR data type) has to be developed. By assembling these controls, a whole archetype can be visually represented. Later, different controls for the same data type could be developed to reflect different preferences (customisation). Specialized controls that stand for a whole archetype are imaginable as well.

This second prototype aims to show how to build a masked numerical input control that only accepts values within a certain range.

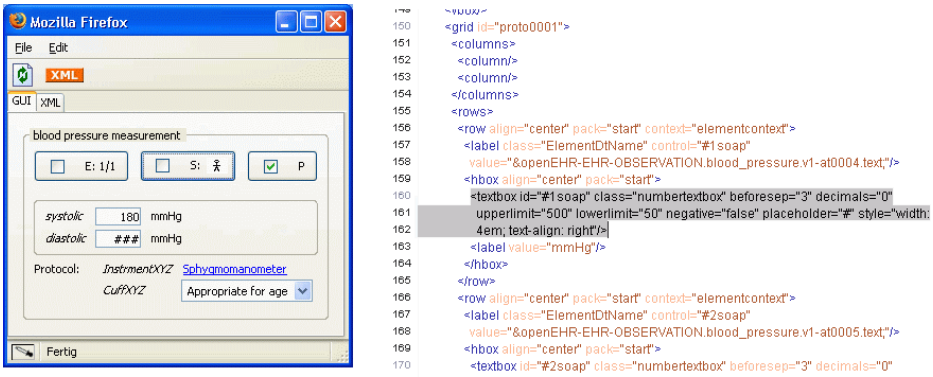

Figure 3: Custom input control and its integration into XUL

The masked fields (Figure 3) require 3 digits and accept only numerical values between 50 and 500. The XUL code example shows how a customized control is used. Different settings can be made via attributes and can therefore easily be manipulated by the generator. XBL was used to define this custom control.

3.3 *Prototype 3: GUI Generator*

The goal of the third prototype was to build an archetype-based generator that creates GUIs similar to those seen in the previous prototypes. For this prototype a standalone C# application was built, but it would be possible to integrate the generator in a web service as shown in Figure 1.

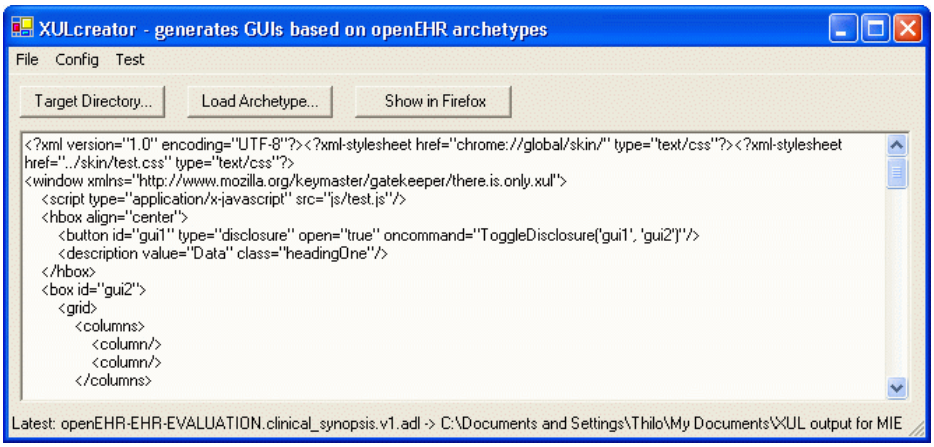

Figure 4: GUI generator application

Figure 4 shows a screen shot of the generator application. The interface is very simple. A target directory and a source archetype have to be determined and the generated XUL markup will be displayed. The "Show in Firefox" button allows the user to view the rendered GUI result.

To generate the markup, the selected archetype is passed to the kernel component. The kernel then forms an in-memory tree representation of the archetype and the GUI is generated by recursively walking through this tree. Depending on the underlying reference model class at each tree node, a suitable generator class handles the kernel information and produces XUL output.

4 Discussion

Our prototypes show that *open*EHR archetypes contain sufficient information to automatically derive functional GUIs. Thus rich GUIs, including validation rules according to the specific domain, are feasible. Our research also proves that using an XML-based GUI description language is a viable approach. The common XML format made creation and handling of the generated output straightforward, due to well-understood and supported mechanisms, such as the DOM or XPath.

We also encountered several problems, including the immaturity of the Mozilla XUL language and the lack of documentation of the early *open*EHR kernel component. Although XUL is used in major software projects within the Mozilla foundation, it is still in its first version and some inconsistencies and implementation bugs (e.g. problem with references to cloned objects) still prevail. In a "once-only" programmed application it is possible to circumnavigate these problems, but to dynamically generate GUIs a more stable development framework is necessary. Another practical limitation was the lack of modern development environments for XUL. Still, we believe XML-based GUI languages are the future. The fact that the presentation layer of Microsoft's new Windows version is expected to use a XML format called XAML is one strong indicator for this ([17], [18]). There is a multitude of such XML dialects available ([16]), but a thorough analysis in terms of functionality, maturity, and documentation should precede any usage. Apart from XUL and XAML, other promising approaches are Backbase's BXML (cross-browser AJAX applications; http://www.backbase.com/) and Xo-

etrope's XUI (Java applications; http://www.xoetrope.com/zone/index.php?zone=XUI), which are both being developed by commercial companies. On the standards front, W3C is working to consolidate the different approaches in its web application working group (http://www.w3.org/2006/appformats/).

One of the goals of the *open*EHR foundation is to produce a reference kernel application. This sub-project is led by researchers from the University College London and is truly open source. The software is written in Java (http://svn.openehr.org/ref_impl_java/TRUNK/project_page.htm) and it pays strict attention to both the *open*EHR specifications and abides by modern software engineering principles. The code has to be documented on an ongoing basis and current online documentation is available (http://www.acode.se/release/kernel/apidocs/). This reference kernel and the improved GUI developing frameworks provide a good foundation for further research in the area of archetype-based GUI generation.

There are other approaches that aim to provide generic solutions for collecting structured data in health, for example DOSPO from the University of Heidelberg ([9], [11]) and OpenSDE from the Erasmus MC in Rotterdam ([10]). As part of these systems, GUIs can be generated from predefined domain models. In distinction to our archetype-based approach, DOSPO's and OpenSDE's modelling mechanisms would need to be mapped to an EHR standard in order to achieve semantic interoperability or to provide other, non-generic, means of communication.

Although *open*EHR archetypes are not built to generate GUIs, they suit this purpose well. ADL defines knowledge concepts by nesting instances of types from a small, stable, generic reference model. This hierarchical structure can be further constrained by determining the cardinality relations between the instances, as well as by setting allowable values for primitive type instances at the leaf nodes ([3]). Since the number of reference model types is small, customized controls can be developed for each of them. Assembling them as described in the results section leads to a generic GUI representation for every archetype. Obviously the GUI can only enforce the constraints which have been defined in the archetype. A feature which is currently not supported by ADL is what we call "warning constraints". While the archetype might define that a systolic blood pressure *must* be between 0 and 500 mmHg (assumed physiological limits), values over 300 mmHg are very rare and constitute probably, but not always a typing error. With regard to data quality it appears to be a valuable extension to declare such additional information in the archetype (if available maybe even with a probability rating) which can be used for the generation of warning messages displayed to the user. Mludek et al. introduced this concept as part of their "integrity constraints" ([13]).

Usually each screen of a medical application would be generated from several archetypes. The *open*EHR foundation foresees an artefact called "template" to bundle archetypes. *open*EHR templates allow local adaptors to define constraints that exceed the original ones without breaking them. If the original archetype offers different terminologies, obligatory selections can be made to reflect the local conditions. Furthermore, optional sections in an archetype can be omitted, or made compulsory and default values can be set [3]. Currently no proposition concerning inter-archetypal constrains and warnings within one template has been made. This is desirable as e.g. "gender=male" and "pregnant=yes" for the same person shouldn't be allowed. Considerable theoretical work on such constraints has been done by Mludek ([12]).

Even before the advent of an EHR system with a fully "on the fly" generated GUI, the customisation function for GUIs becomes important. Only if GUIs can be adapted

to personal and organisational workflows they will be usable, accepted and efficient. In order to provide optimal adaptability, a multi-level customisation system is envisioned. The notion of "skinning" known from applications such as Winamp could be extended, and the appearance of the GUI could be changed during its usage by switching to a customising mode. Customisation would include relatively simple layout alterations such as different colours and fonts (e.g. via stylesheets) but also rearranging and defining controls. These skinning activities always have to be conformant to the rules defined in the archetype to guarantee medico-legal correctness. The described maximum level of customisation requires a presentation layer design that is completely independent from other layers of the system and that uses a ubiquitous and adaptable format such as a XML based GUI language.

5 Conclusion

Our research proves that the automatic generation of GUIs from *open*EHR archetypes is feasible in principle, based on three prototypes. Although the XML-based GUI languages like XUL are still in their infancy, they show a lot of potential despite the difficulties encountered. Further studies should be conducted to investigate automatic GUI generation for several archetypes based on an *open*EHR template.

References

[1] Beale T. Archetypes: Constraint-based Domain Models for Future-proof Information Systems. In OOPSLA 2002 workshop on behavioural semantics. http://www.deepthought.com.au/it/archetypes/archetypes_new.pdf (last accessed Jan. 3, 2006).

[2] Beale T, Heard S. Archetype Definitions and Principles. http://svn.openehr.org/specification/TRUNK/publishing/architecture/am/archetype_principles.pdf (last accessed Jan. 3, 2006).

[3] Beale T, Heard S. Archetype Definition Language (ADL), Revision 1.3. openEHR Specification, the openEHR foundation. http://svn.openehr.org/specification/TRUNK/publishing/architecture/am/adl.pdf (last accessed Jan. 3, 2006).

[4] Eichelberg M, Aden T, Riesmeier J, Dogac A Laleci G. A Survey and Analysis of Electronic Healthcare Record Standards. ACM Computing Surveys, Vol. V, No. N 20YY pp. 1-47.

[5] Gainer A, Pancheri K, Zhang J. Improving the human computer interface design for a physician order entry system. AMIA Annu Symp Proc. 2003;:847.

[6] Hovenga E, Garde S, Heard S. Nursing constraint models for electronic health records: a vision for domain knowledge governance. Int J Med Inform. 2005 Dec;74(11-12):886-98.

[7] Johnson CM, Johnson TR, Zhang J. A user-centered framework for redesigning health care interfaces. J Biomed Inform 38 (2005) 75-87.

[8] Kim MI, Johnson KB. Patient entry of information: evaluation of user interfaces. J Med Internet Res 6 (2004) e13.

[9] Knaup P, Garde S, Merzweiler A, Graf N, Schilling F, Weber R, Haux R. Towards shared patient records: An architecture for using routine data for nationwide research. Int J Med Inform. 2005 Aug 19.

[10] Los RK, van Ginneken AM, van der Lei J. OpenSDE: a strategy for expressive and flexible structured data entry. Int J Med Inform. 2005 Jul;74(6):481-90.

[11] Merzweiler A, Weber R, Garde S, Haux R, Knaup-Gregori P. TERMTrial--terminology-based documentation systems for cooperative clinical trials. Comput Methods Programs Biomed. 2005 Apr;78(1):11-24.

[12] Mludek V. Modellierung der Integritätsbedingungen eines rechnerunterstützten Dokumentationssystems zur Studiendokumentation in einem Verbund multizentrischer Therapiestudien der Pädiatrischen Onkologie (Modelling of Integrity Constraints of a Computer-supported Documentation System for Cooperative Trials in Pediatirc Oncology). Abteilung Medizinische Informatik. Inauguraldissertation; 2002 [in German].

[13] Mludek V, Knaup P, Garde S, Merzweiler A, Weber R, and Wetter T. Formale Definition von Integritätsbedingungen für den Basisdatensatz der Pädiatrischen Onkologie (Formal Definition of Integrity Constraints for a Basic Data Set for Pediatric Oncology). Informatik, Biometrie und Epidemiologie in Medizin und Biologie 2002; 33 (2-3):338[in German].
[14] Rose AF, Schnipper JL, Park ER, Poon EG, Li Q, Middleton B. Using qualitative studies to improve the usability of an EMR. J Biomed Inform. 2005 Feb;38(1):51-60.
[15] Wusteman J. About XML: from Ghostbusters to libraries - the power of XUL. Library Hi Tech, vol.23, no.1, 2005, pp.118-29. Publisher: Emerald, UK.
[16] http://en.wikipedia.org/wiki/List_of_user_interface_markup_languages (last accessed Jan. 3, 2006).
[17] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnintlong/html/longhornch03.asp (last accessed Jan. 3, 2006).
[18] http://www.ondotnet.com/pub/a/dotnet/2004/01/19/longhorn.html (last accessed Jan. 3, 2006).