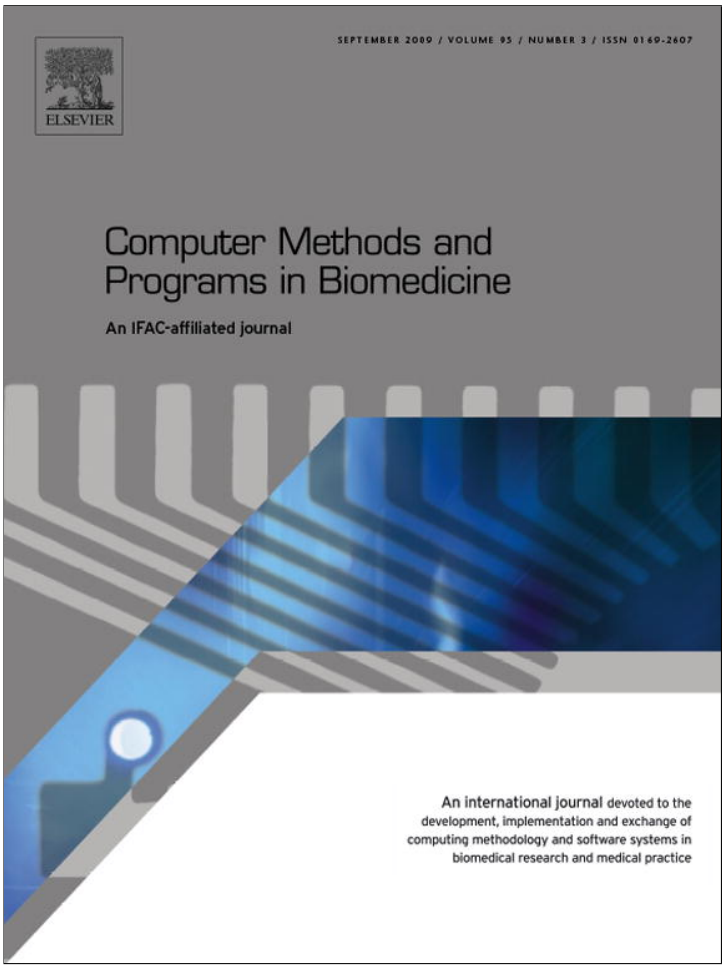
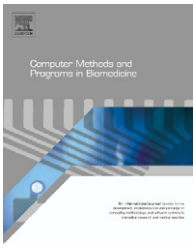


COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE 95 (2009) 213–226

journal homepage: www.intl.elsevierhealth.com/journals/cmpb



Generic screen representations for future-proof systems, is it possible? There is more to a GUI than meets the eye

Helma van der Linden[a,*], *Tony Austin*[b], *Jan Talmon*[a]

[a] *School for Public Health and Primary Care: Caphri, Maastricht University, Maastricht, The Netherlands*
[b] *CHIME, University College London, United Kingdom*

ARTICLE INFO

Article history:
Received 18 July 2008
Received in revised form
2 February 2009
Accepted 15 March 2009

Keywords:
Computerized Medical Record Systems
User–computer interface
*open*EHR
Archetypes
HL7

ABSTRACT

Background: Future-proof EHR systems must be capable of interpreting information structures for medical concepts that were not available at the build-time of the system. The two-model approach of CEN 13606/*open*EHR using archetypes achieves this by separating generic clinical knowledge from domain-related knowledge. The presentation of this information can either itself be generic, or require design time awareness of the domain knowledge being employed.
Objective: To develop a Graphical User Interface (GUI) that would be capable of displaying previously unencountered clinical data structures in a meaningful way.
Methods: Through "reasoning by analogy" we defined an approach for the representation and implementation of "presentational knowledge". A proof-of-concept implementation was built to validate its implementability and to test for unanticipated issues.
Results: A two-model approach to specifying and generating a screen representation for archetype-based information, inspired by the two-model approach of archetypes, was developed. There is a separation between software-related display knowledge and domain-related display knowledge and the toolkit is designed with the reuse of components in mind.
Conclusions: The approach leads to a flexible GUI that can adapt not only to information structures that had not been predefined within the receiving system, but also to novel ways of displaying the information. We also found that, ideally, the *open*EHR Archetype Definition Language should receive minor adjustments to allow for generic binding.

1. Introduction

Interoperability is considered a key property of the generation of electronic health records (EHR) systems to come. It will allow EHRs to communicate and to interpret the data received.

The two major standardisation approaches in this respect are HL7 v3[1] [1], where the architecture of the messages between systems are standardised, and CEN/TC251 13606 combined with archetypes[2] [2,3] which standardises the record itself.

* *Corresponding author at*: Medical Informatics, Maastricht University, POBOX 616, 6200 MD Maastricht, The Netherlands. Tel.: +31 433882235.
E-mail address: h.vanderlinden@mi.unimaas.nl (H. van der Linden).

[1] In this article, HL7 will refer to the new v3 standard in its latest form.

[2] In this article, we refer to the CEN 13606/archetypes as archetypes for brevity.

0169-2607/$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.cmpb.2009.03.003

As advocated by *open*EHR [4], true future-proof electronic health record systems will be able to accommodate new medical concepts without the need for large adjustments to the system [5]. This is possible because a Reference Model is implemented in the information system, which need not change, and "archetypes" then express structured medical concepts as constraints to and combinations of these classes. A consequence of this approach is that patient information in structures that have not been previously encountered can still be satisfactorily interpreted by the system.

Although future-proof systems are capable of comprehending previously unknown (*ad hoc*) information structures since they still adhere to the underlying Reference Model, there is currently no method for the optimal display of such data. A clinical application receiving data for which it has no predefined presentation format will have to resort to a low level generic representation of the received data that may be difficult to understand by the user; for example, a family tree might be presented in the form of a nested XML structure because the application has not been configured for a graphical display of this information. In order to present this kind of information in the future the application will need to be modified, and continually so for each newly encountered data structure. This is clearly not a scalable approach, given that clinical data structures do evolve and EHR systems in the future will receive patient data from other widely distributed and heterogeneous EHR systems using different data structures. Throughout this article we will use the non-trivial yet easy to understand example of a blood pressure concept. In a local system, it may suffice to define a blood pressure by the systolic and diastolic values. However, when the blood pressure is communicated to others, contextual information such as the cuff size and the position of the patient are required to make a proper interpretation. Over time, definitions of concepts may change and become more complex and new concepts may be defined in particular areas of medicine. Systems that are part of a larger network need to be able to deal with these evolving definitions, preferably without a (major) effort in the redesign of the software.

This article discusses an approach to solve these problems by developing a systematic way of representing and implementing *presentational knowledge*.

1.1. Background

Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [6]. Both HL7 and CEN 13606 aim to achieve interoperability both at the data structure level and at the domain model level. At the lowest level, medical concepts are described using predefined data structures. This ensures that the information exchanged is complete (it contains all relevant and required data and metadata) and can be parsed, stored and subsequently retrieved. At the higher domain level, additional metadata are used to avoid ambiguity in understanding. For example, in textual values these might include the code and a reference to the coding scheme.

The PropeR project studied the architecture of a generic electronic health record system. During this study, a prototype implementation was built and tested [7,8]. The system was configured for use as multidisciplinary EHR by several therapists in primary care in the context of rehabilitation of stroke patients in their home environment.

PropeR showed that the successful use of received information is not simply a matter of communications between systems, but also between the receiving system and its users. For example, even if the GP's system is capable of storing and subsequently retrieving a family tree, it is very difficult for the GP to correctly interpret the information if there is no suitable screen representation.

1.2. The CEN 13606 archetype approach

The two-model approach of CEN 13606 consists of a Reference Model with predefined classes that can be implemented in software and an Archetype Model that defines a UML model that constrains and combines the classes from the Reference Model to express medical concepts in standardised structures. This approach separates the software development from the medical knowledge implementation and permits clinical users to define structures describing their clinical specialities without needing to understand how those structures will be exchanged or committed to persistent storage.

An *archetype* or *archetype definition* is a description of the structure of a medical concept, as it would be documented in an EHR system. For example, a blood pressure archetype describes two physical qualities, one called systolic, the other called diastolic, each having a unit of measurement, along with optional metadata for the position of the patient, the cuff size used and the time of the observation. These values are represented in a containing class labelled "Blood Pressure". Since archetypes support object-oriented principles, generic archetypes can be further sub-specialized. This allows the definition of a generic concept such as a "laboratory test" to be specialized into a "blood sugar test" or a "complete blood count" (CBC).

Actual archetypes derived from the UML model are also referred to as *archetype instances*. An actual observation, for example a specific blood pressure measurement in a particular patient, is referred to as an *EHR instance*.

1.3. Motivation for the project

In the context of the PropeR project [7,8] a web-based EHR system was built using a simplified version of CEN 13606 archetypes. We focused on the implementation of a domain-agnostic system based on these archetypes, with generic screen representations. The feasibility of generating a GUI based on archetypes [9] was studied in a second phase project. Both projects revealed that a generic GUI would result in a sub-optimal display where the structure of the information is used as the only basis to derive the presentation, rather than displaying the information in a form familiar to the user. It was found not to be possible to support the definition of an optimal display format while keeping the GUI generic.

The two-model approach, which is the basis for the archetypes, has proven to be useful in separating software development from knowledge implementation. This inspired the authors to apply the same approach to the GUI domain, in an attempt to enable the generation of good quality and use-

ful presentations of EHR data without requiring that each data structure (archetype) be known in advance during the system design.

2. Methods

The objective of the project reported in this paper was to develop a new framework for presentation-level interoperability. However, since it was expected that the proof-of-concept implementation would elicit further requirements and require iterative development cycles, we focused on reusing already available tools for implementation. Two existing candidate presentation frameworks were tested for the proof-of-concept implementation:

1. The Cocoon Forms Framework, based on XForms and suitable for data entry as well as data display [10].
2. The XML User Interface Language (XUL), the Mozilla Foundation framework for developing GUIs [11].

These frameworks were chosen to take advantage of existing experience from earlier projects [9,12] and to allow for rapid implementation. It was also hoped that the choice of industry-standard tools would ensure that the focus remained on the feasibility of the new presentation layer and not be diverted towards other implementation issues relating to the development of a new technology. However, it was recognised that neither framework had been tested for this purpose previously and that it might later become necessary to develop a proprietary (albeit open) alternative. This might need either to be an amalgam of the two above, or entirely new from scratch.

In all cases, the implementations were integrated in the Apache Cocoon Web application to actually deliver the application pages. A set of various EHR instances was presented via both implementations.

The authors also tested the generic nature of the approach by mapping the framework to an HL7 structure representing a blood pressure, to assess the work that would be required for this.

2.1. *Presentation-level interoperability*

Irrespective of the purpose of an element of displayed information, it is likely that the presentation of information via an application will have used three types of knowledge:

- Knowledge of the best way to display the information (content-related).
- Knowledge of the way a user is accustomed to view information (localization).
- Knowledge of the device that is used to display the information (device-related).

These different types of knowledge are often mixed and hard-coded into the GUI of the client application. This not only makes it difficult to display information from a domain different to the one for which it was designed, but it also necessitates duplication of presentation knowledge within the application to accommodate different display devices.

2.1.1. *Content-related presentation knowledge*

Content-related presentation knowledge relates presentation behaviour to the constituents of the information structure; for example, numbers might be displayed differently from text. Such simple knowledge, however, is not sufficient. In medicine, more complex information structures exist. For example, a common visual form for blood pressures is of two numbers separated by a slash (Systolic/Diastolic). An upside-down tree might be the best visual form of a genealogical history.

2.1.2. *Localized presentation knowledge*

Visual information may be subject to local customisation, from the local language and date format to the preferred units (e.g. mg/dl vs. μmol/l) and for the use of different coding schemes. These may be established countrywide, or within an individual institution. There are also personal preference differences such as learned behaviour or different cognition preferences (e.g. pictorial, textual).

2.1.3. *Device-related presentation knowledge*

There is now a large range of devices capable of sending and retrieving information on behalf of a user. They include not only desktop computers and laptops, but tablet PCs, PDAs and smartphones. The aspiration for smaller devices today is towards achieving the same browsing experience as that provided by the more full-featured devices like desktops. However, this is not universally realised yet and the presentation toolkits must ideally retain the flexibility to modify content based on the rendering device.

2.2. *A two-model approach to generic GUI generation*

From the PropeRWeb application the authors identified that including screen presentation knowledge in the archetype definitions introduces dependencies that make content modification difficult [12]. Not only does it introduce two different kinds of knowledge (medical domain knowledge and presentation knowledge) into a single model, it also adds complexity to the display of the same information in different ways according to context.

Moreover, adding specific display information to the archetype would duplicate the effort of definition. For example, if numerical information from different medical concepts could be displayed in a similar way, the definition for that display would still have to be added to each archetype that represents numeric data. This approach would also increase the effort to maintain archetype definitions if each one had to be updated whenever a display definition was added or modified.

The authors distinguish two models: a display-oriented model (the GUI model) that defines Widgets as screen presentation units; and a domain-oriented model (the Content model) that defines Content Units, which create meaningful presentations using Widgets. The first model is the realm of the GUI designer, while domain experts use the second model. If this is compared to the archetype approach, the GUI model resembles the Reference Model, while the Content Model resembles the Archetype Model. During development, a strict separation of domains was maintained echoing those archetype characteristics.

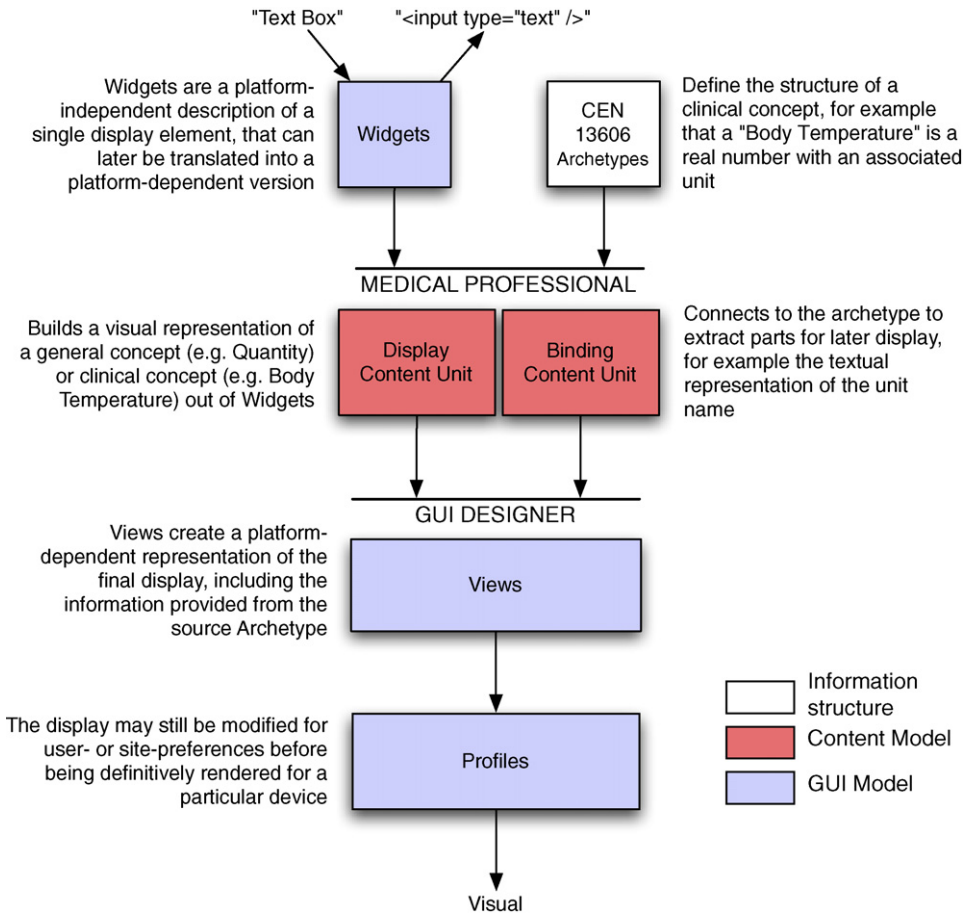


Fig. 1 – Diagram showing the various model components and their relation.

An overview of how a visual is generated using the two models is given in Fig. 1. The process is based on the concept of pipelines. Various units handle binding and widget selection. Localized presentation knowledge is defined in the final profile.

Given the premise that future-proof systems are also capable of receiving information from other domains, it is necessary that these systems contain domain-agnostic presentational functionality. All units follow object-oriented design principles in which a specific unit inherits characteristics from a more generic unit. This improves consistency and flexibility, and implies that multiple units can be defined for any archetype to broaden the range of possible displays available.

2.2.1. *GUI model*

The GUI model consists of:

- Widgets, the building blocks of a GUI.
- Views, the definition of a screen.
- Profiles, which tailor the presentation to the local environment.

Widgets are commonly known in software development as the elements that make up the windows of an application, such as text boxes and labels. A Widget in this approach is a platform-independent display unit that contains presentation knowledge for a single data type. These widgets can be mapped to classes in the underlying Reference Model. Two types of widgets exist: data-oriented widgets such as “text”, “image” and “number”, and group-oriented widgets such as “list”, “table” and “graph”. The definition of these widgets is typically a one-time investment since both the Reference Model (and therefore the widget set), is stable over time.

Widgets are converted to specific (device-dependent and/or platform-dependent) versions in the underlying system by using Views. A View is focused on presentation of the content and therefore part of the GUI designers’ remit. Views provide the transformation from platform-independence to platform-dependency.

In a View, the screen/window is divided in parts, each with a different purpose. Each part is bound to a data source. This can be as simple as a static image for a logo or a predefined tree for navigation bars, up to more complex sources such as the demographics of the current patient for the header part and the content units for the content part. An application is therefore not expected to create the “screen chrome” itself, but rather to devolve the responsibility for that to the View.

Profiles implement localized presentation knowledge. Profiles manage the conversion of information to match user expectations to avoid interpretation errors.

A profile contains preferences at various levels that modify the presentation of the information. There are three levels:

- *System level*. This level contains generic preferences that should always be applied, e.g. language, date format, and metric vs. imperial units.
- *Local level*. This level contains generic preferences that are organization or location specific and are more domain-related. These preferences include preferred units and preferred terminologies. This level updates preferences on a per-role basis.
- *User level*. This level contains user-specified preferences that can modify the preferred view for a certain type of content unit e.g. if the user prefers graphs to tables for the same content.

By combining these tiers, the GUI can be tailored to the user and organizational preferences.

2.2.2. *Content model*

The Content model describes the content-related presentation knowledge in Content Units. A Content Unit consists of two parts: a Display Content Unit and a Binding Content Unit.

A Display Content Unit is a composition of one or more Widgets, combined with other display-oriented information to provide a platform- and device-independent description of the layout of a medical concept, which will have been described in an archetype. The authors suggest using a Content Unit Definition Language (CUDL), which is still to be defined formally, for this description. Display Content Units can be regarded as the display counterpart of archetypes. Like archetypes, they can include other Display Content Units.

Display Content Units can also include (references to) normal ranges for semantic interpretation of the value. For example, the value of a body mass index (BMI) can be colour-coded based on the semantic interpretation (e.g. "normal" is green, "obese" is red).

The Binding Content Unit describes how the parts of the Display Content Units are connected to the associated parts in the archetyped EHR instance, through the Content Unit Binding Language (CUBL), which also needs to be defined formally.

Separation of the binding from the archetype allows for multiple content units that refer to a single archetype and provides the flexibility to display the same information in different ways. It should also be possible to map content units not only to archetypes, but also to other data structures.

Like archetypes, Content Units are stored in a repository in a format that is ready for use. The exact implementation could be anything from simple source code files to high performing databases. Since they are a platform-independent representation of the presentation of EHR data conforming to an archetype, they can be shared in the same way archetypes are sharable.

Any profile that specifies how a specific widget has to be displayed will be applied to the components of the content model that refer to such a widget. This mechanism makes it possible to apply local/system/user preferences also to data that was not seen before.

3. Results

A proof of concept Web application was built using the Apache Cocoon Web application framework [13]. This Web application can display EHR instances conforming to various archetypes based on the approach described above. The EHR instances are offered to the system as an XML structure.

The Apache Cocoon Web application framework is a generic open source framework that is fundamentally based on the concept of separation of concerns. This is reflected in the component-based implementation and supports the corresponding pursuit of separation of knowledge types. It implements the pipeline concept as described above. Since Cocoon excels in processing XML, it is a good candidate to build a generic Web-based GUI that can be generated.

3.1. *Cocoon Forms Framework*

The Cocoon Forms Framework, or CForms block [10], is a standard part of the Cocoon framework and provides widgets that can be used for both data display and for data entry. In its simplest form, a CForms widget is defined by three XML structures: the definition, the binding and the template. The *definition* is a description of the label and the data type. Optionally, validation rules can be added, such as checks on date ranges or lists of predefined values, but also more elaborate validations that can be added as functions in JavaScript or Java, because an API is available to access the widgets in these languages. The validation can be done, not only at the widget level, but also at the form level, i.e. across widgets.

The *binding* is based on XPath and connects the widget to its underlying information structure. The latter can be anything from XML to JavaBeans to SQL-results or even custom-built structures. In its simplest form, the binding merely provides the mapping between the CForms widget and the underlying data structure, but in more complicated structures it also holds the definition of how to add or delete substructures (e.g. in a table where rows can be added and/or deleted).

Archetypes are hierarchical structures and support an XPath-like definition [14] to access substructures, which matches the binding used in CForms. The CForms binding file would be an implementation of the Binding Content Unit.

The *template* provides a platform-dependent description of the actual display of the widget, for example an HTML page where the label and the value are displayed as rows in a table or a text area used for the data entry of a string. Note that a (HTML) form is typically implemented in Cocoon as three different XML files, one for the definition of all the widgets, one for the binding of all the widgets and one for the template of the form, and its widgets. These files can be handcrafted or partially or fully automatically generated. The definition defines the aggregation of the CForms widgets and provides a simple ID to the template for further layout specification.

The template combined with the definition would be the implementation of the Display Content Unit.

Figs. 2 and 3 show the same list of blood pressures. The first is generated by a generic template resulting in a long list, while the second is optimized by a Blood Pressure Content Unit.

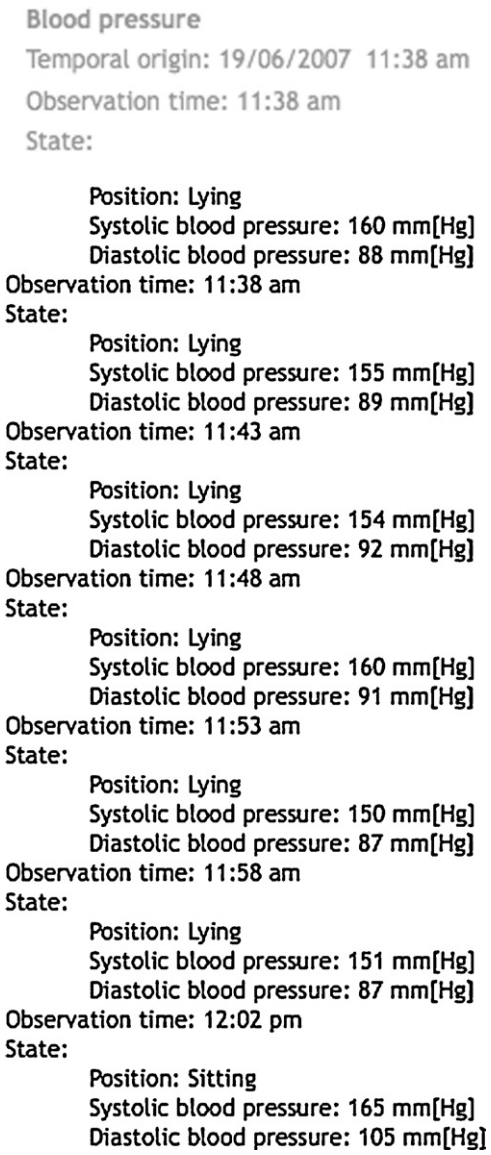


Fig. 2 – Default display of a list of blood pressures.

CForms work well with other Cocoon blocks such as the internationalization block (i18n). Together they provide out-of-the box localization. The use of the i18n block in Cocoon implies that text that would require localization is tagged with i18n tags. Based on the provided locale, the i18n block replaces the tagged text with its localized counterpart. Other conversions such as a localized date format are also supported. The i18n block would be part of the implementation of the Profiles, which would reduce the amount of implementation work necessary.

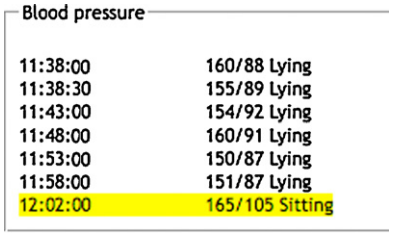


Fig. 3 – Same list of blood pressures, but using an optimized presentation, the last entry is highlighted to signal the change in position.

The first version of the CForms implementation revealed that it is relatively easy to define content units based on CForms widgets for simple archetypes such as a body temperature. Using JavaScript it was possible to define validation rules across widgets and to write simple conversion rules for units. This allowed for on the fly calculation of the body mass index for example, and for quick conversions into a different unit.

The drawbacks of the CForms became evident when moving on to more complex data structures such as the blood pressure. CForms have no definition of a generic layout (e.g. horizontal orientation or grid), but rely on a set of available Extensible Stylesheet Language (XSL) stylesheets that provide a standard rendering in HTML. These stylesheets can be overruled, but are generally considered an integral part of the CForms block.

Therefore, in the template, it was not possible to define a generic display of the blood pressure in the previously mentioned Systolic/Diastolic format without selecting a specific approach such as an HTML DIV or TABLE structure. This approach should be based on the overall design and device constraints and is therefore part of the view, not the content unit.

In the definition, it was very difficult to build a composite content unit based on a composition of widgets. To define a generic Quantity widget in CForms a composite CForms widget would be required. Standard CForms widgets consist of a label and a value for a specific data type. The label content is defined in the definition, rather than bound to an external information source. There is no provision for displaying a unit. The implementation of a Quantity widget in CForms would be a so-called class widget consisting of a common string widget for the label, a common widget with a numerical type and another common string widget for the units. The challenge lies in the mapping of multiple instances of similar values, e.g. if a list of blood pressure measurements is provided, they all map onto the same definition, but still need a mechanism that distinguishes them from each other.

The rudimentary implementation of validation rules using JavaScript showed that it is necessary to have a way to reference other archetype values as in the case of the BMI calculation where the values of the weight and the height archetypes are necessary. CForms widgets have IDs and, as stated before, validations can be defined on a form level, across widgets. The reference to the individual values in the BMI calculation would be easy to achieve in the CForms validation. However, a BMI calculation is part of the domain knowledge, not of the domain display knowledge. Therefore, it should not be part of the Content Unit, but of the archetype. This also raised the question where in general the validation, calculation and conversion methods should be stored.

Not all platforms support JavaScript. A platform-independent specification language is required for widespread adoption.

3.2. XML User Language (XUL)

XUL is a definition language created by the Mozilla Foundation and used for their products such as FireFox and Thunderbird. Apart from the usual data type oriented widgets it also provides widgets that are more generic such as grids and horizontally and vertically oriented boxes. This provides the necessary abstraction for the display and therefore solves the drawback of CForms' reliance on HTML for these types of widgets. XUL templates [15] provide a means to connect the XUL widgets to data sources. However, its implementation revealed that it lacks the flexible and easy binding of CForms and requires complicated definitions of templates, queries and data sources. Although this complexity provides a lot of flexibility, it makes it difficult for a non-programmer to define the necessary structures.

Other disadvantages are:

- XUL template data sources are subject to a 'same-domain' restriction. This effectively means that the XUL template and the referenced data sources should be part of the same URL domain. This not only prevents separation of the GUI model, the Content model and the data, but also prevents the generation of a screen based on multiple sources from different URL domains.
- The implementation of a XUL template and the data sources are part of the definition of the XUL widgets (e.g. 'datasource' is an attribute of a vbox, a vertically oriented box). This makes a separation of view and data binding impossible and prevents multiple binding definitions to a single view.

Schuler et al. [9] also noted the immaturity of the language. Bowers also describes the immaturity of the XUL templates [16] in a web article, in which he points out that although the concept looks promising, the current implementation prevents development of an application beyond the simple examples provided.

Since XUL is only supported by Mozilla products, the final display requires a conversion to standard HTML or, if possible, the use of a Mozilla browser.

3.3. Experimental proprietary framework

Drawing from the experiences of both the CForms and the XUL implementation efforts described above, we defined a simple Display Content Unit Language in XML based on the XUL vocabulary, but added the XPATH binding of CForms. The necessary conversions were done using various Extensible Stylesheet Language stylesheets. We focused on data display.

For each archetype, a matching Display Content Unit was implemented as a small XSL stylesheet. This XSL stylesheet puts the appropriate information in the label, value, and unit triple.

Without the functionality of the CForms validation rules, it was necessary to implement unit conversion rules in XSL. Although it was possible to achieve this, the attempt demonstrated that the available functionality in XSL is too limited and cumbersome to use. Unit conversions ideally require different mechanisms than simple XSL transformations.

4. Evaluation

4.1. Enhanced example

The example below shows the necessary components and the process in the pipeline that provides a meaningful display of a blood pressure on a user's smartphone.

When the requesting system application receives an EHR instance containing a blood pressure, it looks up the reference of the archetype (1) in the EHR instance and finds a matching content unit (2) going from specific to generic in the archetype hierarchy. It then retrieves the appropriate archetype and content unit from their respective central repositories. A similar mapping process is executed to find an appropriate view (3) based on the targeted device and the references to the content unit. This mapping process is also performed from specific to generic. The view is requested from a local repository. Based on the user credentials the appropriate profile (4) is also retrieved from a local repository. All elements are combined to display the blood pressure onto the user's smartphone with his or her preferred language and formats.

More detailed descriptions of how the various presentational units could look are presented below. Simple XML notation was used for clarity. Paths are expressed in XPath notation.

4.1.1. Archetype (1)

An archetype definition of a blood pressure contains two ELEMENTs of type Quantity each consisting of a value (the actual measurement) and a unit (usually mmHg), marked with a name ("systolic" or "diastolic") (see Fig. 4). Additional information such as cuff size, and patient position during measurement can also be described using this archetype.

As a clinician, one is typically only interested in displaying the date and time and the values of the blood pressure. The most common way of displaying this is in the familiar layout of: [time] S/D [unit], with S being the systolic blood pressure and D the diastolic blood pressure.

4.1.2. Widgets

As stated before, Content Units are based on Widgets, which in turn represent the data type classes of the Reference Model. Timestamps can be displayed by date–time widgets, numbers by number widgets and labels by text widgets.

4.1.3. Content units (2)

A specialized Blood Pressure Content Unit defines a blood pressure as a combination of a date–time widget with two number widgets for the values, separated by a slash and followed by a label to display the unit (see Fig. 5). All this should be displayed in a horizontal layout. The content units also contain mappings that bind the widgets to the appropriate substructures of the archetype definition (see Fig. 6).

4.1.4. Views (3)

The GUI designer defines a generic view that defines a part holding a content unit for the patient identification information, a part containing a navigation menu and a part that serves as a placeholder for any type of content unit (see Fig. 7,

```
archetype
        openEHR-EHR-OBSERVATION.blood_pressure.v1

concept
        [at0000]        -- Blood pressure measurement
description
        original_author = <
                ["name"] = <"Sam Heard">
                ["organisation"] = <"Ocean Informatics">
                ["date"] = <"22/03/2006">
                ["email"] = <"sam.heard@oceaninformatics.biz">
        >
        details = <
                ["en"] = <
                        language = <"en">
                        purpose = <"To record the systemic blood pressure of a person. The measurement records the systolic and the diastolic pres
                        use = <"All blood pressure measurements are recorded using this archetype. There is a rich state model for use with exerci
                        keywords = <"observations", "blood pressure", "measurement">
                        misuse = <"Not to be used for intravascular pressure.">
                >
        >
        lifecycle_state = <"AuthorDraft">

definition
        OBSERVATION[at0000] matches {   -- Blood pressure measurement
                data matches {
                        HISTORY[at0001] matches {       -- history
                                events cardinality matches {1..*; unordered} matches {
                                        EVENT[at0006] occurrences matches {0..*} matches {      -- any event
                                                data matches {
                                                        ITEM_LIST[at0003] matches {     -- blood pressure
                                                                items cardinality matches {0..*; ordered} matches {
                                                                        ELEMENT[at0004] occurrences matches {0..1} matches {    -- systolic
                                                                                value matches {
                                                                                        C_QUANTITY <
                                                                                                property = <"openehr::125">
                                                                                                list = <
                                                                                                        ["1"] = <
                                                                                                                units = <"mm[Hg]">
                                                                                                                magnitude = <|0.0..1000.0|>
                                                                                                        >
                                                                                                >
                                                                                        >
                                                                                }
                                                                        }
                                                                        ELEMENT[at0005] occurrences matches {0..1} matches {    -- diastolic
                                                                                value matches {
                                                                                        C_QUANTITY <
                                                                                                property = <"openehr::125">
                                                                                                list = <
                                                                                                        ["1"] = <
                                                                                                                units = <"mm[Hg]">
                                                                                                                magnitude = <|0.0..1000.0|>
                                                                                                        >
                                                                                                >
                                                                                        >
                                                                                }
                                                                        }
                                                                        ELEMENT[at0033] occurrences matches {0..1} matches {    -- Comment
                                                                                value matches {
                                                                                        TEXT matches {*}
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                        POINT_EVENT[at0002] occurrences matches {0..1} matches {        -- baseline reading
                                                offset matches {|PT0s|}
                                                data matches {
                                                        use_node ITEM_LIST /data[at0001]/events[at0006]/data[at0003]
                                                }
                                        }
```

Fig. 4 – Archetype definition of a blood pressure (*source*: the *open*EHR Foundation www.openehr.org).

top definition). The specialized view in this smartphone example is derived by mapping the widgets to their smartphone equivalents, in Fig. 7, bottom definition, shown as a simple HTML table. This view can also contain information on design such as font and font size choice, and colours.

4.1.5. *Profiles (4)*

A system-wide profile is created that defines the preferred language, the metric system, the date and time format and so on for the organization where it is implemented. If necessary, additional profiles could be created to accommodate the system to special users, e.g. visiting physicians from the USA.

4.2. *Mapping to HL7 structures*

HL7 focuses primarily on message exchange and therefore considers this display problem to be part of the receiving system's domain. However, this approach is equally useful in that realm.

To test the generic nature of the approach the Content Unit of a blood pressure was mapped to the HL7 structure for a

```
<contentunit id='bloodpressure' archetype='openEHR-EHR-OBSERVATION.blood_pressure.v1'>
  <hbox>
    <label id='bp_label' widgetpath='datetime/value'/>
    <value id='systolic' widgetpath='number'/>
    <text>/</text>
    <value id='diastolic' widgetpath='number'/>
    <value id='units' widgetpath='text'/>
  </hbox>
</contentunit>
```

Fig. 5 – Display Content Unit definition.

```
<contentbinding id='bloodpressure' path=
'//OBSERVATION/data/HISTORY[at0001]/events/EVENT/data/ITEM_LIST[at0003]/items'/>
  <value id='bp_label' path='/POINT_EVENT[at0002]/offset/value'/>
  <value id='systolic' path='/ELEMENT[at0005]/value/magnitude'/>
  <value id='diastolic' path='/ELEMENT[at0004]/value/magnitude'/>
  <value id='units' path='/ELEMENT[at0005]/value/units'/>
</contentbinding>
```

Fig. 6 – Binding of content unit to archetype definition. Note that the paths in the "value" definitions are relative to the path in the binding tag.

```
<view id='simplescreen' type='generic'>
  <vbox>
    <hbox>
      <part id='logo'/>
      <part id='header'/>
    </hbox>
    <hbox>
      <part id='navtree'/>
      <part id='content'/>
    </hbox>
  </vbox>
</view>
```

```
<view id='simplescreen' type='smartphone'>
  <table>
    <tr>
      <td><part id='logo'/></td>
      <td><part id='header'/></td>
    </tr>
    <tr>
      <td><part id='navtree'/></td>
      <td><part id='content'/></td>
    </tr>
  </table>
</view>
```

Fig. 7 – Generic view definition and a similar definition for a smartphone.

blood pressure. Since no HL7 tools were available to generate an XML version, the mapping was performed by hand using an example of a blood pressure measurement that was part of a CDA document example in the HL7 ballot of May 2008 [17] and the R-MIM model of a blood pressure as modelled in the Netherlands [18].

Fig. 6 shows a binding to an archetype blood pressure. Figs. 8 and 9 show it is possible to create a slightly different Binding Content Unit and reuse the Display Content Unit to display an HL7 structure.

Two issues emerged:

1. In the CDA example the only distinction between the systolic and the diastolic part of the blood pressure could be made by including a test for a specified code in the path to the value. This implies that the code system is stable or the Binding Content Unit needs to be extended with a mechanism that allows a selection of the path based on the results of various tests for the code systems used. In the R-MIM model of the blood pressure, this issue is resolved by having a specific 'SystolicPressure' and 'DiastolicPressure' part. In Archetype Definition Language (ADL), this problem is solved by using stable internal references that are independent of the coding schemes used.
2. In the R-MIM model, the values of the systolic and diastolic parts have no separate unit entry. From the definition of the type of the value, it was not clear if a unit was implied (e.g. through the coding scheme used), part of the value structure (but not stated separately) or even part of the content of the value item (i.e. '120 mmHg' rather than '120'). In the CDA sample a separate entry for the unit was available. To overcome such a situation, the Binding Content Unit needs to be extended with a mechanism to set the value to a fixed entry. Obviously, the better solution is to include a clear separate unit entry in the R-MIM model.

5. Discussion

5.1. *Applicability of the approach*

There are several advantages to the solution proposed here:

- First and foremost, it offers the flexibility of defining specific, optimized screen presentations for known information structures, while providing the means to generate usable screen presentation of previously unknown information structures.
- By separating the various types of presentation knowledge into distinct models, it is possible to separate pure GUI knowledge from medical presentational knowledge, thus adopting another two-model approach and promoting reuse.

```
<contentbinding id='bloodpressure' path='//BloodPressure'>
  <value id='bp_label' path='@effectiveTime'/>
  <value id='systolic' path='/component1/SystolicPressure/@value'/>
  <value id='diastolic' path='/component/DiastolicPressure/@value'/>
  <value id='units' path='/component1/SystolicPressure/@unit'/>
</contentbinding>
```

Fig. 8 – Binding of content unit to HL7 R-MIM-model of blood pressure.

```
<contentbinding id='bloodpressure' path='//entry/observation'>
  <value id='bp_label' path='/effectivetime/@value'/>
  <value id='systolic' path='/entryRelationship[observation/code/@code="271649006"]/value/@value'/>
  <value id='diastolic' path='/entryRelationship[observation/code/@code="271650006"]/value/@value'/>
  <value id='units' path='/entryRelationship[observation/code/@code="271649006"]/value/@unit'/>
</contentbinding>
```

Fig. 9 – Binding of content unit to HL7 CDA Document Sample of blood pressure.

- It is flexible enough to build an adaptive GUI based on roles. For example, nurses and physicians can see the same information but optimally presented for their specific needs, while the only difference in development might be a document definition.
- The evolution of medical knowledge will always create new data types and new archetypes, which would lead to new display representations. This approach ensures that the available display knowledge can be reused as much as possible.
- New and novel ways to view EHR information are a topic of ongoing research. By adhering to the proposed approach, these views can benefit from the available display knowledge that is already expressed in content units [19]. In the blood pressure example earlier, the content unit for the single blood pressure can be reused to define a list of blood pressures that can be used in a view to display an interactive timeline of measurements as well as a simple list view. More specifically, by separating the Display Content Unit from the Binding Content Unit, the same Binding Content Unit of the blood pressure could be mapped onto a Display Content Unit specifying the common Systolic/Diastolic format or an X/Y format, where X is bound to the timestamp, Y1 to the systolic and Y2 to the diastolic values. By adding the appropriate calculation, the Y value could even be mapped onto the mean blood pressure. A Chart content unit can take the list of X/Y formats as input for producing the graph.
- Both the HL7 RIM and CEN 13606 archetypes use a limited set of predefined data types, with ongoing efforts to harmonize the two sets. By describing one or more widgets for each data type, it should be possible to provide a meaningful display of the information without incorporating presentation knowledge in the information structure. This means the current archetype or message specifications need not be extended and the number of widgets remains small.
- The information can be converted to match local and user preferences such as preferred coding scheme, language, units and more.
- A fallback mechanism is used to select a more generic representation in the absence of a specific one.
- Standardised content units could be shared between systems the same way archetypes can be shared. This increases the intelligence and usability of the system.
- The pipeline approach not only allows reuse of components, but also offers flexibility in functionality by a simple addition of pipelines.

This approach complements the *open*EHR architecture where templates are used to create a higher-level composition by constraining and ordering of archetypes. However, the *open*EHR templates are used to create an information structure, while the approach proposed in this paper is used to display the structured information.

There are also disadvantages:

- Specialized views, containing special content units, can only exist for predefined information types. New information types or information types from different domains will fall back to a more basic representation. The latter, however, can benefit from shared content units.
- A repository equal to that for archetypes is necessary for the various presentation units.
- A mechanism for retrieving an appropriate screen representation for the current archetype is necessary, since the most appropriate selection is based on multiple parameters that cannot be stored in the archetype instance.

From the proof-of-concept implementation two issues surfaced:

- *Value-related display*. Uniquely marking special values (e.g. out-of-range values) enhances quick interpretation of the data. This involves the union of several types of knowledge. The GUI designer, along with users, decides on the display of the special value (e.g. large red font, flashing and/or audible signal), the content modeller decides whether the out-of-range value needs to be marked (or not), while either the archetype definition or a separate knowledge base holds the information on when the value is out of range. Complexity increases when a range has several subranges with different interpretations, for example the BMI range can be divided in underweight, normal, overweight and obese. The content definition language needs to provide a method of indicating when and how a special value needs to be marked as well as a reference to the location of the underlying knowledge. All this should be combined with a consistent, yet appropriate display (e.g. always red for alerts, unless the concept requires otherwise).
- *Conversion rules*. Users need to be able to specify their preferred units, localized date formats, etc., because even when the unit of measure is displayed, sometimes the value in itself looks 'unfamiliar' at first glance thus hindering immediate interpretation. This requires an extensive conversion library, which needs to be implemented in a generic way. It is not clear where this implementation should be located when the conversion extends beyond the universal conversions (e.g. Celsius to Fahrenheit) and enters the clinical domain (e.g. the BMI calculation). The latter would ideally be located in the archetype, but there is currently no suitable expression language defined.

More work needs to be done to define a flexible content definition language that solves these issues and is capable of describing domain-related display knowledge at a sufficient level.

The advantages of having flexible GUI interfaces outweigh the disadvantages. By incrementally defining screen presentations that can be built on top of each other, there is less duplication of work in building a GUI. A higher-level screen presentation allows the user to better interpret the presented information leading to more efficient and more reliable information exchange. Screen presentations of new information structures can be added to the system without major redevelopment of the application.

Moreover, less effort is needed to define content units compared to the effort of defining archetype definitions, since an existing archetype definition can be used as the basis for the content unit definition. Since many medical concepts can be displayed using generic content units such as a simple label, value, unit triplet, the majority of concepts can be defined using a limited set of generic content units.

5.2. *Requirements*

From the implementations of the three frameworks (CForms, XUL and proprietary), the authors draw several conclusions that elaborate the requirements of valid content definition languages. The implementation also led to proposed enhancements for the archetype structures.

5.2.1. *Requirements for a content model language*

The complexity of an archetype model is usually hidden behind an application (for example, the archetype editor from Ocean Informatics [20]). It would also be possible to hide the complexity of the content model language but it should nevertheless be relatively easy to understand for domain experts. It must have several functionalities:

- Description of positional relations between values and fixed information, for example in the case of a blood pressure format ("S/D") where the entire structure is horizontally oriented and the/sign is fixed and relevant.
- Description of optional parts of the content widget, for example the label is optional if a list of similar values is displayed and the list already has an appropriate header. This makes it possible to reuse the widget on a small screen such as a cell phone screen.
- A mechanism that defines the display behaviour when the value is within a specific range, for example when the BMI value is in the "obese" range it should be coloured red. This requires a mapping to a location, which holds information on the meaning of the various ranges as well as a description of the display options for each of the ranges. Note that the description of the meaning of these ranges should not be part of the content unit but of the archetype definition or a specialized knowledge server.
- Description of a binding to the underlying archetype. This binding ties the label, value and unit to the respective counterparts in the archetype. The binding language should be generic enough to accommodate other structures than archetypes.

More work needs to be done to solve these issues and to define a flexible content definition language that is capable of describing domain-related display knowledge at a sufficient level.

5.2.2. *Requirements for archetypes*

The proof-of-concept implementation has shown that correct interpretation of an archetyped EHR instance (for example, the actual description of a blood pressure) cannot be done without the associated archetype, since some semantic information is not duplicated in the instance. This is also true for localization of the information. Archetypes support translations of a concept into various languages, but the instance only contains the requested language. If the receiver wants to use several languages then the archetype itself must be available to the proposed framework. This makes the functionality to retrieve the archetype from a repository that is available to all receiving systems, a requirement, not an option.

During the proof-of-concept implementation, we observed that the same information was defined using two different archetypes. A complete blood count report was coded with a specific CBC report archetype and with a more generic laboratory report archetype. In the former, each triplet of name, value and unit was uniquely identified, thus providing a path to each of the values, while in the latter, the triplets were merely part of a list without a unique identification at item level. The former instance allowed for a specific content unit controlling the order in which the test results were displayed, while the latter could only provide a more generic list in the order available in the instance. Although both structures were valid EHR instances, the authors suggest that information should only be structured using the most specific archetype definition available. It is proposed to adjust the mechanism used in archetype selection during data entry so that the most appropriate archetype is selected more rigorously.

There is currently only a rudimentary version of an expression language defined in the *open*EHR Archetype Definition Language [14] that can be used to define calculations based on archetypes or archetype parts. The current ADL version (1.4) describes how to reference archetype parts within the archetype (e.g. the systolic and diastolic values in a blood pressure archetype) but not across archetypes, e.g. the BMI calculation which combines the weight and height archetypes.

This is a restriction that is not necessarily applicable to other formalisms. Expressing an archetype in an imperative programming language would provide access to all the computational power of that language, including validations based on cross-archetype values.

Finally, in order to implement a fall back mechanism in the selection of the content units from specific to generic, it is desirable to have inheritance information in the archetype definitions. In the above example of the CBC report, examination of the specific CBC report archetype shows it is inherited from the more general laboratory report archetype. However, there is no required indication of this inheritance in either the EHR instance or the archetype definition. The ADL as well as the Archetype Object Model (AOM) [21] do provide an entry 'parent_archetype_id', but this is optional and the accompanying text does not clarify if this should be used to indicate the

inheritance path, or to express which archetypes are allowed as parents.

This situation might be improved, if the entry were a required indication of the inheritance path, possibly adding a special value to indicate 'no-parent' or 'root-archetype'. Additionally the process of archetype definition should enforce the selection of the most appropriate parent.

The lack of this inheritance path does not invalidate the content unit approach. It merely results in a more generic display than necessary.

The requirements to the archetype model presented here are not unique for the proposed framework. Any GUI framework would benefit from them.

5.3. *Related work*

A similar approach has been developed by Ocean Informatics and implemented in their EhrView application [22]. The EhrView application modifies the information through a series of XSL stylesheets. These stylesheets are selected by matching the archetypes names in the structures. The matching process selects the most specific stylesheet available in a repository. The EhrView application does not separate content-related modelling from software-related modelling and it is only defined for one type of device: a regular screen of a desktop PC or laptop. It also offers limited options to adjust to local or user preferences.

Fiala et al. [23] have described a component-based approach for adaptive Web documents that influenced the approach reported here. They too make a distinction between content-related and display-related presentation knowledge and they used the pipeline concept to define Web document generation. However, their focus is on adapting information presentation to user preferences and devices. The information is known and defined in advance and there is no method to handle unknown information structures or describe conversions to preferred units.

University College London (UCL) uses a Java-based approach, and has developed their own rendering toolkit. Class files are annotated with Enterprise JavaBeans 3.0 (EJB 3) descriptors as well as directives to the toolkit, so that one single archetype file (including content often represented in ADL) can be used to create every tier of an application without further (archetype-related) effort being required.

Outside the healthcare domain, several studies on adaptive GUIs and UI modelling languages have been done (e.g. [24]). The majority of these studies either focus on the adaptability to different display devices (e.g. [25]) or to the disabilities of the user (e.g. [26]). They start from the premise that the information to be displayed is known, which is the problem embedded in the lower section of the GUI model in Fig. 1. However, it is possible that some existing modelling languages could be made sufficiently expressive for a complete role in archetype visualisation.

5.4. *Future work*

More research is needed to define a Content Unit Definition Language that is flexible, yet easy enough to use. This should be accompanied by a binding definition language that provides a simple mechanism to map a display definition to the associated archetyped EHR instance.

Although XUL could provide the initial version of the language to handle positions and orientations, it needs to be extended with a binding definition language that matches the ease of the CForms binding definition language.

The content definition language also needs to support descriptions of more complex content units. These require references to knowledge bases with reference information (such as the reference graphs in a paediatric growth chart) and flexible mechanisms to define highly specialized graphic structures for charts and family trees.

The Scalable Vector Graphics (SVG) specification is a modularized language for describing two-dimensional vector and mixed vector/raster graphics in XML [27]. It is a good candidate for the description of such graphic structures, but it is too complicated to manually define them therein. It could be the basis for specialized widgets that can be used in the content units. This, however, should be done carefully to avoid convolution of the separation of the domains. In the example of the family tree, the tree is composed of persons and relationships. Relationships are described as lines and persons are depicted by shapes. Since colours, line styles and shapes all have meaning; it is difficult to separate the GUI-model knowledge (colours, line style, shape, fill colour) from the Content model (meaning of relationship, gender, disease).

The authors have demonstrated that relatively little work is necessary to reuse this approach for HL7 structures. To fully evaluate the generic nature of the approach a more extensive set of archetypes and HL7 structures need to be tested.

5.4.1. *Data entry*

The approach presented here has been primarily focused on data consultation. While clearly data entry differs from data display insofar as one creates data and the data structures that contain it, and the other simply renders existing input, the layout should be shared between entry and display as much as possible. Doing this enables the technical rendering apparatus to be shared, and allows a clinician to later review data with a similar appearance to that of the original committer. However, this is non-trivial because data entry focuses on speed and data validation, not (so much) on appearance, as has been discussed by van der Meijden [28] and van Ginneken [29].

This approach can support data entry by extending the binding framework to binding events. Events can be used to perform calculations, data validation, entry support (e.g. context-related selection lists) and the creation of the data structure.

The generic workflow would be to retrieve existing data (in case of data modification) or an empty data structure (in case of new data) from the system kernel, which contacts the archetype server for the appropriate archetype references. This is passed to the GUI in the same pipeline process described earlier. Events that create data structures (e.g. a row in a table), can communicate with the system kernel on valid data structures. Finally, the resulting data structure is passed onto the system kernel for final validation and data storage.

Events to support data entry and data validation events need a closer connection to the validation parts of the archetype, to retrieve and provide valid subsets or verify if

the entry is within the provided range. Connection to external knowledge sets (e.g. coding sets) is also necessary.

Event binding crosses the separation between the GUI model and the Content model separation because the events are widget-related, while the action is content-related. For example, pressing an "add new row" button is a simple GUI event. The related action needs to communicate with the system kernel on how to extend the data structure to add a new row. This falls into the Content model domain.

6. Conclusion

Interoperability does not stop when information from one system can be successfully understood and/or incorporated in another system. It is also necessary to provide a screen representation that gives the user of the receiving system a clear understanding of the new information.

This paper describes a two-model GUI approach that extends that currently used by EN13606 for knowledge modelling. The authors argue that this approach leads to a flexible GUI that can adapt to information structures not predefined in a receiving system and display them in an interpretable way. This flexibility will also accommodate data entry since future-proof systems are in equal need of data entry forms for newly created concept structures. The work has demonstrated its generic nature by requiring only a minimal effort to map the framework to equivalent HL7 structures. It should be noted that a future-proof generic GUI methodology also places constraints on the archetype model.

Ongoing biomedical research in areas such as biomarkers and proteomics will eventually add new medical concepts to an EHR that need specific representations, not yet found in the current EHR systems. These will be able to take advantage of any user preferences for physical and other quantities already in being in the system automatically. Nevertheless, this work also cautions that promising frameworks might not hold out in actual implementation. The main reason for this is the underlying common assumption that (part of) the domain knowledge is hard-coded into the GUI.

Author's contributions

HvdL is guarantor of the study. She initiated the idea and developed the proof of concept. T.A. was heavily involved in the discussion on the best approach. J.T. was also involved in the discussion.

HvdL wrote most of the manuscript with the help of T.A. J.T. and T.A. critically reviewed the various versions of the manuscript. All authors approved the final version.

Conflict of interest statement

All authors declare that there are no competing interests, financially or otherwise.

Acknowledgments

We would like to thank the team from Ocean Informatics who kindly provided the information we needed. We would also like to thank Patrick Ahles, Rong Chen and Arie Hasman and Dipak Kalra for their valuable insights.

We would especially like to thank Thilo Schuler for his work in the earlier versions of this article.

The authors recognise the established trademarks of technology providers in this paper.

REFERENCES

[1] Health Level 7 Inc. (last accessed 18 January 2008). Available from: http://www.hl7.org.
[2] Health Informatics—Electronic health record communication. Part 1. Reference Model, CEN/TC251 prEN13606-1:2005:E, March 2005.
[3] Health Informatics—Electronic health record communication. Part 2. Archetypes, CEN/TC251 prEN13606-2:2005:E, December 2005.
[4] openEHR (last accessed 18 January 2008). Available from: http://www.openehr.org.
[5] T. Beale, Archetypes—an interoperable knowledge methodology for future-proof information systems, Published on the Internet (last accessed 18 January 2008). Available from: http://www.openehr.org/shared-resources/publications/archetypes.html.
[6] Standards Coordinating Committee, IEEE standard glossary of software engineering terminology, IEEE Comput. Soc. (1990).
[7] H. van der Linden, G. Boers, H. Tange, J. Talmon, A. Hasman, PropeR: a multidisciplinary EPR system, Int. J. Med. Inform. 70 (2003) 149–160.
[8] H. van der Linden, J. Talmon, H. Tange, J. Grimson, A. Hasman, PropeR revisited, Int. J. Med. Inform. 74 (2005) 235–244.
[9] T. Schuler, S. Garde, S. Heard, T. Beale, Towards automatically generating graphical user interfaces from openEHR archetypes, Stud. Health Technol. Inform. 124 (2006) 221–226.
[10] Cocoon Forms Block Implementation (last accessed 2 June 2008). Available from: http://cocoon.apache.org/2.2/blocks/forms/1.0/489_1_1.html.
[11] XML User Interface Language (last accessed 18 January 2008). Available from: http://www.mozilla.org/projects/xul/.
[12] H. van der Linden, J. Grimson, H. Tange, J. Talmon, A. Hasman, Archetypes: the PropeR way, Medinfo 11 (2004) 1110–1114.
[13] Apache Cocoon Web Application Framework (last accessed 18 January 2008). Available from: http://cocoon.apache.org.
[14] T. Beale, S. Heard, Archetype Definition Language (last accessed 2008/06/04). Available from: http://www.openehr.org/svn/specification/TRUNK/publishing/architecture/am/adl.pdf.
[15] XUL: Template Guide (last accessed 3 July 2008). Available from: http://developer.mozilla.org/en/docs/XUL:Template_Guide.
[16] J. Bowers, XUL templates are a waste of time (last accessed 3 July 2008). Available from: http://www.jerf.org/resources/xblinjs/whyNotMozilla/notXulTemplates.html.
[17] HL7 Version 3 Standard Ballot Package Download (last accessed 3 July 2008). Available from: http://www.hl7.org/v3ballot/html/welcome/downloads/downloads.htm.
[18] A.M. Fleurke, W.T.F. Goossen, E.J. Hoijtink, J. van der Kooij, Swen, M. Vlastuin, Weinstein, ALGEMEEN LICHAMELIJK ONDERZOEK: BLOEDDRUK (last accessed 3 July 2008), Version 1.0. Available from: http://www.zorginformatiemodel.nl/1_documentatie/Doc_Obs_Bloeddruk_R01_V1.0.pdf.

[19] E. Sundvall, M. Nystrom, M. Forss, R. Chen, H. Petersson, H. Ahlfeldt, Graphical overview and navigation of electronic health records in a prototyping environment using Google Earth and openEHR archetypes, Medinfo 12 (2007) 1043–1047.
[20] Archetype Editor (last accessed 2 June 2008). Available from: https://wiki.oceaninformatics.com/confluence/display/TTL/Archetype+Editor.
[21] T. Beale, Archetype Object Model (last accessed 4 June 2008). Available from: http://www.openehr.org/svn/specification/TRUNK/publishing/architecture/am/aom.pdf.
[22] EhrView (last accessed 21 January 2008). Available from: http://oceaninformatics.biz/Products/ProductDescription.pdf.
[23] Z. Fiala, M. Hinz, K. Meißner, F. Wehner, A component-based approach for adaptive, dynamic Web documents, J. Web Eng. 2 (2003) 058–073.
[24] N. Souchon, J. Vanderdonckt, A Review of XML-Compliant User Interface Description Languages, DSV-IS 2003, 2003, pp. 377–391.
[25] N. Mitrovic, J.A. Royo, E. Mena, M.D. Luna, ADUS: indirect generation of user interfaces on wireless devices, in: Seventh International Workshop Mobility in Databases and Distributed Systems (MDDS 2004), 2004.
[26] K. Gajos, J. Wobbrock, D. Weld, Automatically generating user interfaces adapted to users' motor and vision capabilities. Proceedings of the 20th annual ACM symposium on User interface software and technology (2007) 231–240.
[27] D. Jackson, Scalable Vector Graphics (SVG) 1.1 Specification (last accessed 4 July 2008). Available from: http://www.w3.org/TR/SVG11/.
[28] M.J. van der Meijden, H.J. Tange, J. Boiten, J. Troost, A. Hasman, An experimental electronic patient record for stroke patients. Part 2. System description, Int. J. Med. Inform. 58–59 (2000) 127–140.
[29] A.M. van Ginneken, Considerations for the representation of meta-data for the support of structured data entry, Methods Inf. Med. 42 (2003) 226–235.