The Clinical Document Editor – a Powerful Tool for CDA Implementation

Philip Scott[1], **Robert Worden**[2], **David Bowen**[3]

[1]University of Portsmouth, Portsmouth, UK

[2]Open Mapping Software Ltd, Cambridge, UK

[3]Great Ormond Street Hospital for Children, London, UK

Abstract

Background: Documents constructed according to the HL7 Clinical Document Architecture (CDA) need to be reviewed, edited and approved by an author, who is usually a clinician. The CDA specification does not require any particular editing tool to be used in creating CDAs. The tools used to edit CDAs have typically been specific to a particular profile of CDA, and even to the application using that profile. There are widely used style sheets to render and view CDA documents, but there are no comparable widely deployed tools for creating the CDA itself. **Objectives:** This paper describes an open source toolset to create production-quality, clinician-oriented, web-based editors and viewers for any profile of CDA. **Methods:** We illustrate the process of developing and deploying a clinical document editor using the example of the Consent Directive CDA. We introduce the Clinical Document Constructor Architecture. **Results:** The Clinical Document Editor toolset allows fully-functioning, clinician-ready editors to be rapidly developed and deployed for any CDA profile. **Conclusions:** The emphasis hitherto has been on the structure and computability of CDA documents, but the lack of tools to produce and edit the CDA format is now a significant constraint on its adoption. Tools to efficiently create and edit CDA documents are an essential step in making CDA applications acceptable to practicing clinicians, and thus in realizing the potential of CDA for healthcare interoperability.

Keywords

CDA, HL7, greenCDA, composition, construction, creation, authorship.

Correspondence to:

Dr Philip Scott
University of Portsmouth
Address: Lion Terrace, Portsmouth PO1 3HE, UK
E–mail: Philip.scott@port.ac.uk

EJBI 2012; 8(3):22–30

1 Introduction

Healthcare interoperability is the business of making healthcare IT systems effectively share information. Interoperability is becoming increasingly important, at both national and local levels, to deliver better patient care and value for money, in a period of over-stretched healthcare budgets and ageing populations. It depends almost entirely on having agreed and workable standards for healthcare information. The HL7 Clinical Document Architecture (CDA) is such a standard, and is a central component of interoperability programmes across the world [2, 4, 5, 6]. However, there are a number of issues in the practical deployment of CDA.

1.1 CDA Complexities

The first issue is that the HL7 CDA standard is technically highly complex. This complexity imposes a steep learning curve on developers using CDA to interface their systems, and significantly increases the costs and timescales of building CDA interfaces.

A second problem is that, as the name implies, CDAs are documents. They are therefore not just to be created automatically from data in healthcare IT systems; they need to be reviewed, edited and signed off by authors,

who are typically qualified clinicians. There is a dearth of effective tools to allow clinicians to do this job. The task of creating CDA documents is often supported with only a low level of automation, leading to inefficient use of clinicians' time, and to errors and omissions in the documents.

These two problems are related. The high technical complexity means that the serialized XML format of CDA is very different from anything a clinician would want to review or edit; therefore the editing tools have a lot of work to do, to bridge the gap between the complex XML representation and a clinician-friendly editor view. So editing tools are hard to build, and there are not many good ones available. Those that do exist tend to be specialized to a particular domain or application, such as for creation of discharge summaries.

1.2 The role of greenCDA

This paper suggests that not only are the two problems closely related; but also their solutions are closely related, through the development of 'greenCDA' [3]. The term greenCDA defines a much simplified XML format, which contains all the variable data in a CDA for a specific use case, but which can be transformed to the full standard-conformant CDA by an automatic, reliable transform. The idea of greenCDA was proposed to simplify the task for developers needing to interface their systems via standard CDA; but it can equally simplify the task of presenting that information to a clinician for editing and approval.

The 'green' version of any CDA contains precisely that variable information which differs from one CDA instance to another, and which needs to be populated (for instance by extracting data from existing applications) to make a full CDA instance. It contains this information in the simplest possible form, with none of the HL7 fixed information (such as classCodes and coding systems) which adds to the developer's workload. So a greenCDA is typically three times smaller, and much simpler to understand, than a full CDA. By the same token, the greenCDA contains precisely the variable information which needs to be presented to the clinician and edited by him or her. Therefore the greenCDA is a very good starting point for presenting information in a clinician-friendly form, for editing and approval.

1.3 Semantic mapping

We have previously reported the use of semantic mapping to simplify the use of CDA by generating reliable two-way transforms between canonical and 'green' CDA. The essence of the method is to map disparate data structures (defined as XML schemas) to a common UML class model and from this to generate two-way transformations [7, 8]. The result is a flattened XML structure that preserves the semantic precision of canonical CDA while replacing machine-orientated fixed attribute codes in deeply nested XML structures with humanly meaningful business names. This paper describes an Open Source toolset which extends that approach to the construction of CDA documents.

1.4 Functional objectives

The editing toolset described in this paper allows developers rapidly to create clinical document editors with the following properties:

a) Pre-population of the CDA with coded data from any data source, which can then be edited;

b) Full text editing of any variable field in the CDA, including extended descriptive text;

c) Easy parameterized configuration of the layout and appearance of the editor;

d) Drop-down menu selection for fields with coded values;

e) Automatic populating of fields which depend on the values of other fields;

f) Validation of all user-entered data;

g) Role-based access control, giving different authors the ability to edit or view different parts of the document, in a pipelined workflow;

h) Several differently-configured editors may be developed for the same CDA profile;

i) The editor generates a fully conformant CDA instance at the end of the process.

2 Methods

We illustrate the process of developing and deploying a clinical document editor using the example of the Consent Directive CDA, a draft standard for trial use (DSTU) issued by HL7 in May 2010. This is a deliberately simple use case that does not take content from any back-end data store. We also introduce the Clinical Document Constructor Architecture.

2.1 Consent Directive Use Case

The Consent Directive addresses the problem of patients or their authorised representatives giving permission for certain people to see their medical records. A consent directive is in effect a statement by the patient along the following lines: "I wish the following set of people (A, B, C...) to be able to see my healthcare records (of types X, Y, Z...) for the following purpose (...), over the time period (...)".

Patient consent is becoming increasingly important in territories such as the UK, where patient access to and


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ClinicalDocument xmlns="urn:hl7-org:v3">
        <id docId="1234"/>
        <patient birthDate="19461028" gender="2" patientId="11223344">
                <address>
                        <city>Leicester</city>
                        <streetAddressLine>3 Oadby Road</streetAddressLine>
                        <state>Leics</state>
                </address>
                <email address="aperson@me.com"/>
                <name>
                        <prefix>Mrs.</prefix>
                        <given>Alice</given>
                        <family>Person</family>
                </name>
        </patient>

        <author dateTime="20120531">
                <id authorId="11223344"/>
        </author>

        <body>
                <consentSection purpose="Treatment">
                        <title>Consent Directive Details</title>
                        <receivers>
                                <receiver>
                                        <address>
                                                <city>Leicester</city>
                                                <streetAddressLine>3 Memory Lane</streetAddressLine>
                                        </address>
                                        <email address="bob@gppractice.com"/>
                                        <name>
                                                <prefix>Dr.</prefix>
                                                <given>Bob</given>
                                                <family>Hippocrates</family>
                                        </name>
                                </receiver>
                        </receivers>

                        <informationTypes>
                                <informationType code="GAIN" displayName="Global Appraisal of Individual Needs
(GAIN)"/>
                        </informationTypes>
                </consentSection>
        </body>
</ClinicalDocument>
```

Figure 1: Example of a 'green' Consent Directive

control of their own healthcare records is now an important element of the national healthcare information strategy [1]. Therefore it is important to have a standards-based way to exchange information about the access consents which a patient has defined.

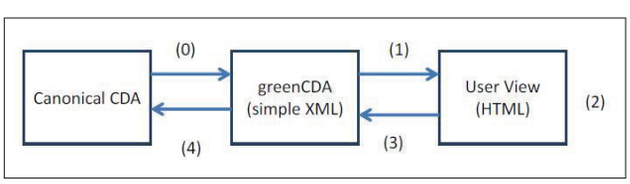


Figure 2: CDA editing steps

There needs to be a means for the patient (or their representative) to edit and alter the permissions he or she has defined, for different caregivers to have access to his or her health records. So the patient needs to be able to create, edit and modify HL7 Consent Directive CDAs. As part of the "miConsent" project, funded by the UK Government's Technology Strategy Board (TSB) and the Wellcome Trust "Sintero" project, we have created an editor for this purpose, using the tools and approach described here. This example serves to illustrate the process of defining a clinical document editor for any CDA profile, by the same process.

2.2 greenCDA Mapping Definition

The first step in defining the editor is to define the greenCDA and the green to canonical CDA transforms. This process is mostly automated, using the open source tools and methods described in our previous papers [7, 8]. The developer's manual task is to pick out the leaf nodes of the full CDA tree which need to be retained in the greenCDA, to decide which internal nodes of the full CDA may be collapsed to simplify the structure and to give meaningful 'business' names to all the nodes. The tools do the rest. This includes generating a schema and example of the simplified XML and generating the precise two-way CDA transforms.

An example of a 'green' Consent Directive is shown at Figure 1.

Manifestly this is a piece of vanilla XML, with none of

the technical complexity usually associated with HL7 V3 messages, such as CDA. The equivalent full CDA, which can be produced by applying the automatically generated transform, is about four times larger and is much more deeply nested.

2.3 CDA Editor

Having created the greenCDA mapping definition we now have a functioning editor for the CDA, which works as shown in Figure 2.

Step 0 occurs before the other steps in the case where there is an existing full CDA to be updated – for instance, when a patient wishes to change a consent directive he has previously made. Then in step 0 the editor may use another generated transform to convert the full CDA into a greenCDA, to be used in step 1. Alternatively, step 0 may be used with step 1 to view an existing CDA, with no ability to change it. The other steps are as follows.

1. In the case where there is no existing CDA document, the editor (which is resident on a web server) starts with either an empty 'skeleton' instance of the green CDA, or a skeleton instance in which pieces of coded data from other systems have already been added, and transforms it into HTML, which is sent to be shown on a browser.

2. The HTML displays whatever content is already in the Green CDA, and has input fields where the user may enter text, and control buttons allowing the user to add or delete optional and repeated parts of the green XML. The user fills in some of the text fields, and may add optional or repeating elements.

3. JavaScript in the browser passes back information about the user actions as they occur. This allows the editor on the server to update its green XML instance, and pass back updated information in HTML to the browser (see later for what this information may be). The cycle (1) – (2) – (3) may repeat an unlimited number of times.

4. When the user is satisfied with what he sees, he presses a 'Finish' button. This causes the server code to take its current copy of the greenCDA instance, and convert it to a full CDA instance, to be used in whatever way CDAs are used. The conversion is done using the green to full CDA transform which was automatically created when defining the greenCDA.

As soon as any greenCDA definition has been created using the toolset, an editor with these capabilities can be installed and run from any web server. It is only a basic raw editor, in the following respects:

- Every variable field in the greenCDA can be edited, but only by free text input from the user.
- The layout of all text fields is a default vertical layout, which is a set of 2*N HTML tables with two columns: label and input area.
- The labels of the text input fields are default labels, taken from the tag names of the greenCDA.
- All text input fields are single-line fields with a default size.
- There are button controls to add or remove all optional or repeating parts of the XML.
- There is no validation of entered data.

While this is only a very basic editor, it is editorially complete, in that the user can in principle use it to create any greenCDA instance which is compatible with the greenCDA schema – and thus to create any full CDA needed for the use case. It gives all the controls needed to do this from a browser.

2.4 Editor Configuration

The next step is iteratively to refine the editor to remove these basic restrictions, and to turn it into an editor fit to be put in front of end users. This is done in a largely parameter-driven fashion. For all its operations, the editor consults a tabular 'Editor Configuration File'. This file is created by the developer, using a spreadsheet tool such as Microsoft Excel®, and stored as a comma-separated-values (csv) file. It has detailed directives telling the editor what to do at any node of the green XML. Editor configuration files can be developed and tested in a rapid develop-and-test cycle with a turnaround of minutes for each iteration.

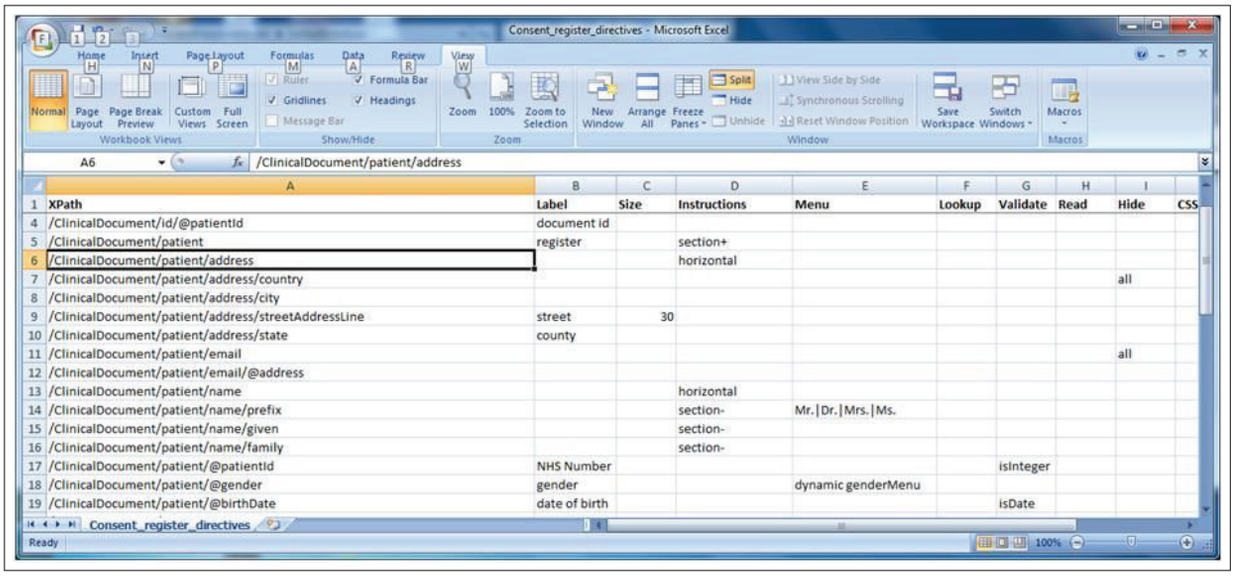
An extract of a typical editor configuration file is shown in Figure 3.

The Editor Configuration file has one row for every possible XPath in the greenCDA, as defined in the first column (which is automatically generated by the tools). The rest of each row tells the editor, through a number of different directives in the other columns, what to do at any node reached by that XPath. As the screenshot shows, there are several different types of directive (Label, Size, etc.) each with different effects – and it is not necessary to use all of them, if you are content with the default behaviour at that node.

Some editor directives (for instance, those for validation of user input) invoke Java methods in an Editor support class supplied by the editor developer. A set of useful methods for these purposes are supplied with the tools in a base support class.

The editor configuration file and a few Java support methods are sufficient to support a fully-functioning, clinician-ready editor.

The editor is deployed by installing a header web page and a simple package on a web server, such as Apache TomCat. The installation package includes the usual things required for a web service, including:



	A	B	C	D	E	F	G	H	I	
1	XPath	Label	Size	Instructions	Menu	Lookup	Validate	Read	Hide	CSS
4	/ClinicalDocument/id/@patientId	document id								
5	/ClinicalDocument/patient	register		section+						
6	/ClinicalDocument/patient/address			horizontal						
7	/ClinicalDocument/patient/address/country								all	
8	/ClinicalDocument/patient/address/city									
9	/ClinicalDocument/patient/address/streetAddressLine	street	30							
10	/ClinicalDocument/patient/address/state	county								
11	/ClinicalDocument/patient/email								all	
12	/ClinicalDocument/patient/email/@address									
13	/ClinicalDocument/patient/name			horizontal						
14	/ClinicalDocument/patient/name/prefix			section-	Mr.|Dr.|Mrs.|Ms.					
15	/ClinicalDocument/patient/name/given			section-						
16	/ClinicalDocument/patient/name/family			section-						
17	/ClinicalDocument/patient/@patientId	NHS Number					isInteger			
18	/ClinicalDocument/patient/@gender	gender			dynamic genderMenu					
19	/ClinicalDocument/patient/@birthDate	date of birth					isDate			

Figure 3: Example editor configuration file

- A set of fixed Java archive (.jar) files for Eclipse EMF and the editor tools.
- A Java archive for the class and methods developed by the editor developer for special-purpose validation, dynamic menus, or data lookup.
- A small XML steering file defining the locations of other resources, such as fixed files or databases, used by the editor.
- The editor configuration file.
- A few files created in the process of defining the greenCDA, which tell the editor about the structure of the green CDA and how to transform it to and from full CDA.
- HTML for the editor entry page, together with a cascading stylesheet (css) and a small fixed JavaScript file.

The detailed contents of this web server editor package are described, for the Apache Tomcat web server, in the toolset documentation.

2.5 Editor Capabilities

In this section we describe the current set of directives available to control the editor at any node of the green XML, and the extended editor capabilities they define.

Label: this directive overrides the default label (taken from the greenCDA tag name) with something which may be more informative for the user.

Size: this directive allows the editor designer to set either the width of a single-line text-input field, or the width and depth of multi-line text input fields.

Instructions: this directive may contain a number of pre-defined instructions to the editor, separated by semicolons. Some important instructions are:

- **Horizontal:** this introduces a horizontal table layout for all nodes below the node – so that repeated groups of items become multiple rows of the table. This might, for instance, be used to show medications in a table, with one row per medication.
- **Section+, Section- :** these either force or prevent the editor from introducing a section divider at this level.

Menu: this allows the user to set the value of a data item by picking from a dropdown menu, rather than free text input. It is possible to have a static set of values defined directly in the editor directives file, and with different paired values for display to the user, and for storage in the greenCDA, if required. Alternatively, the allowed values may be defined dynamically by a Java method, which may for instance look up the values in some other file or database, or make the values depend on the value of some other field.

Validate: This allows the editor designer to call a Java method to validate the data input for the field, and to display any error message in a table. Java methods are provided out-of-the-box for common and simple validations, but the designer may define any validation methods and error messages he wishes.

Lookup: This allows the value of some field to be determined not directly by user input, but by a lookup from other values dependent on input elsewhere in the greenCDA. For instance, this gives a way to ensure that certain coded values in the CDA are consistent with values in tables in the rendered text. It also offers a way to retrieve data automatically and populate any part of the CDA which is provided by an external data source,

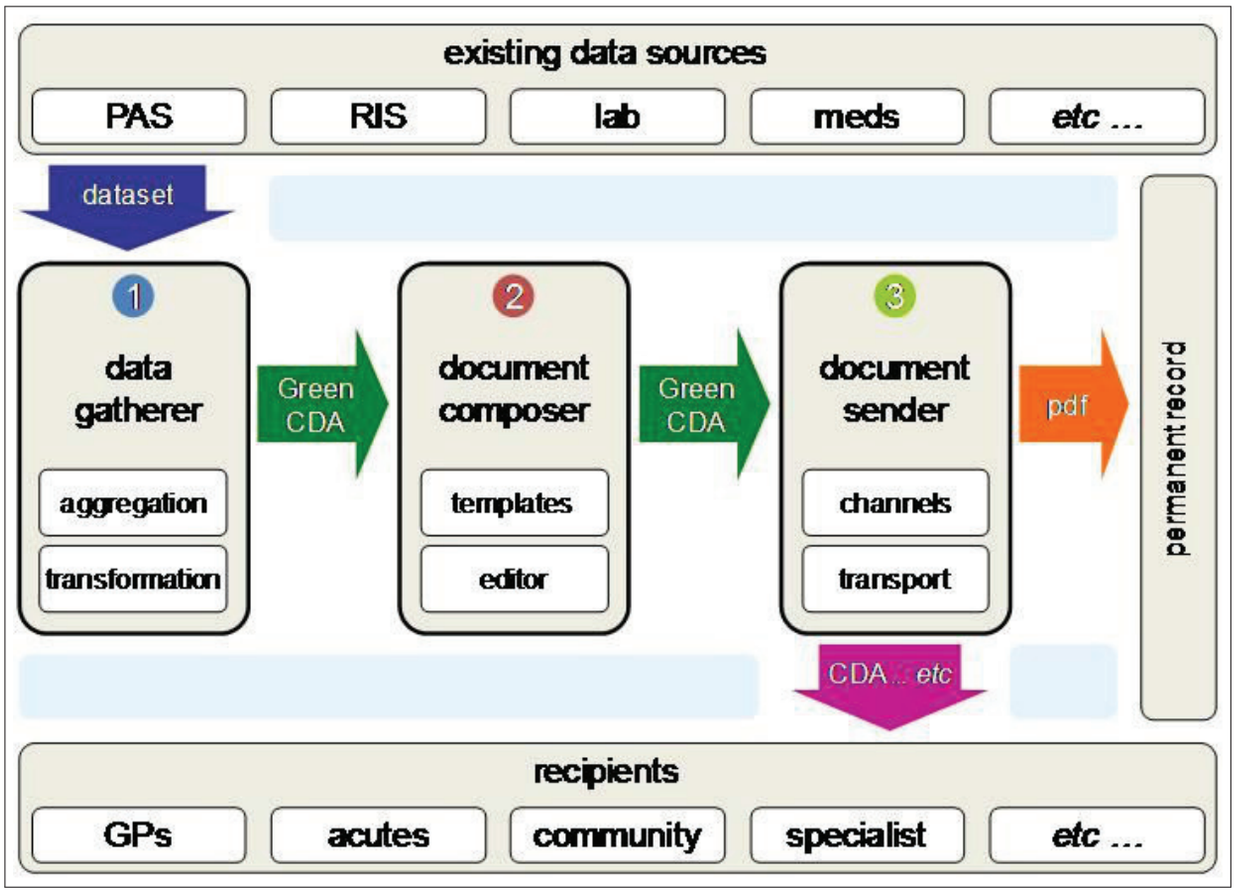


Figure 4: Clinical Document Constructor architecture

such as a patient administration system or a medications system.

Read, Hide: These directives selectively control access to parts of the document. They depend on each user of the editor being assigned a role when he signs on to the editor. Then the 'Read' directive states that all nodes below this node in the greenCDA are read-only for any of the roles defined in that column; similarly the 'Hide' directive can make the nodes in the sub-tree hidden from certain roles. The special role 'all' can be used to make parts of the document read-only or hidden for all users of the editor.

CSS_Class: This directive allows the editor designer to assign a value to the attribute 'class' in the node of the displayed HTML, so that a cascading style sheet may format the node and its descendants in some way dependent on the class.

Order: This directive allows the editor to present a set of sibling nodes of the greenCDA to the user in a different order from that defined in the greenCDA.

By using these capabilities selectively across the nodes of the greenCDA XML, and by using cascading style sheets to further control the editor appearance, it is possible to create one or more web-based editors or viewers for any CDA, which have all the capabilities needed for reliable, efficient editing and review of the necessary information.

By tuning the configuration file it is possible to do rapid iterative refinement of the appearance and behaviour of an editor, to ensure that it meets clinicians' needs.

We have described the currently available set of directives for tuning the behaviour of the editors. As the tool framework is open source and will be refined with further use, it is likely that further types of directive and capabilities will be added in response to requirements.

2.6 Clinical Document Constructor Architecture

In the discussion so far, we have dealt only lightly with two issues: the issue of gathering data together, in greenCDA format, from a number of source systems to populate the coded data parts of the CDA; and the issue of distributing the full CDA document once it has been made.

A fuller treatment of these issues is had by regarding the clinical document editor as the central component of a three-component Clinical Document Constructor architecture. This architecture is designed to have a clean simple greenCDA–based interface between its three components, so that suppliers may competitively supply any of

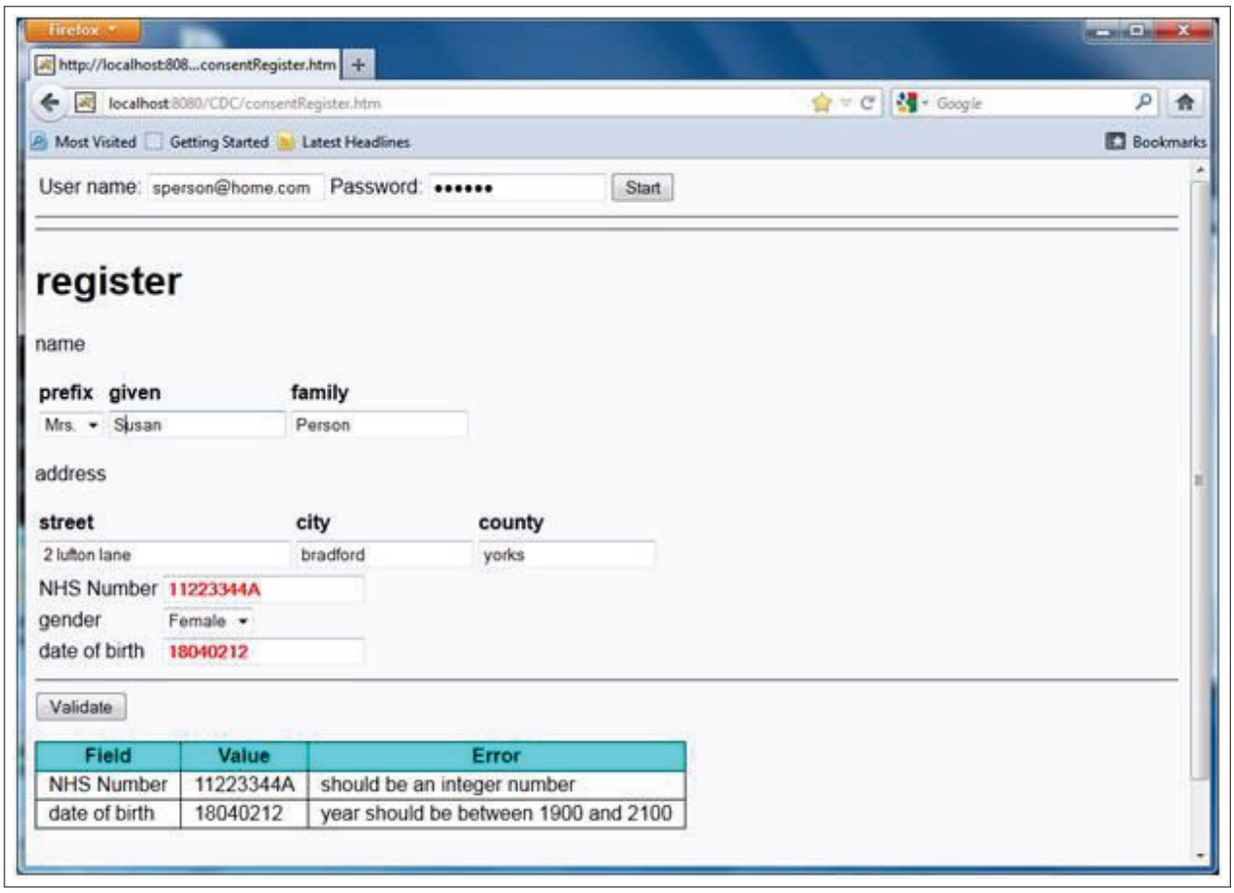


Figure 5: Register Patient Editor

the three components, and providers may independently choose best-of-breed elements for each component.

The Clinical Document Constructor architecture is illustrated in Figure 4.

Thus the task of gathering data from source systems belongs in the data gatherer component 1, which may be supplied by an integration supplier. The editor is the 'document composer' component 2. The task of distributing the resulting CDA belongs in the document sender component 3. GreenCDA XML is the interface between all three components.

Because it is usually straightforward to convert data from its native format to the simple greenCDA format, a simple version of the data gatherer component (1) can easily be built as part of the editor configuration, using the Java extension methods described above. For more complex applications involving many source systems, a specialized data gatherer component, perhaps within an integration engine, is desirable. Similar remarks apply to the document sender component.

3 Results

This section illustrates how these capabilities have been used in developing two editors for the HL7 Consent Directive CDA. By using two different base HTML pages and editor directive files, we have developed two independent editors for the same CDA. The first editor is used for the purpose of a patient registering himself with some consent-assigning service; this editor enables the patient to input only his or her demographic details, and fills in that part of the CDA, hiding all parts of the CDA concerned with actually giving consent. The second editor hides the demographic details, but allows the patient to add or remove the different caregivers he wishes to give consent to.

These two editors are related in a simple workflow: first define the patient demographics, then give and revoke consents for that patient. The editor framework has the necessary hooks to define this and more complex or state-dependent workflows.

The appearance of both these editors is rather basic, as the project did not have budget for extensive refinement. More could easily have been done on the appearance of the editors, with only modest extra work – mainly in refining a cascading style sheet.

Figure 5 shows the appearance of the 'register patient' editor.

Visible in this figure are:

- Drop-down menus are used for the name prefix, and for the gender. The user-visible gender values are not the values stored in the XML (which uses gen-

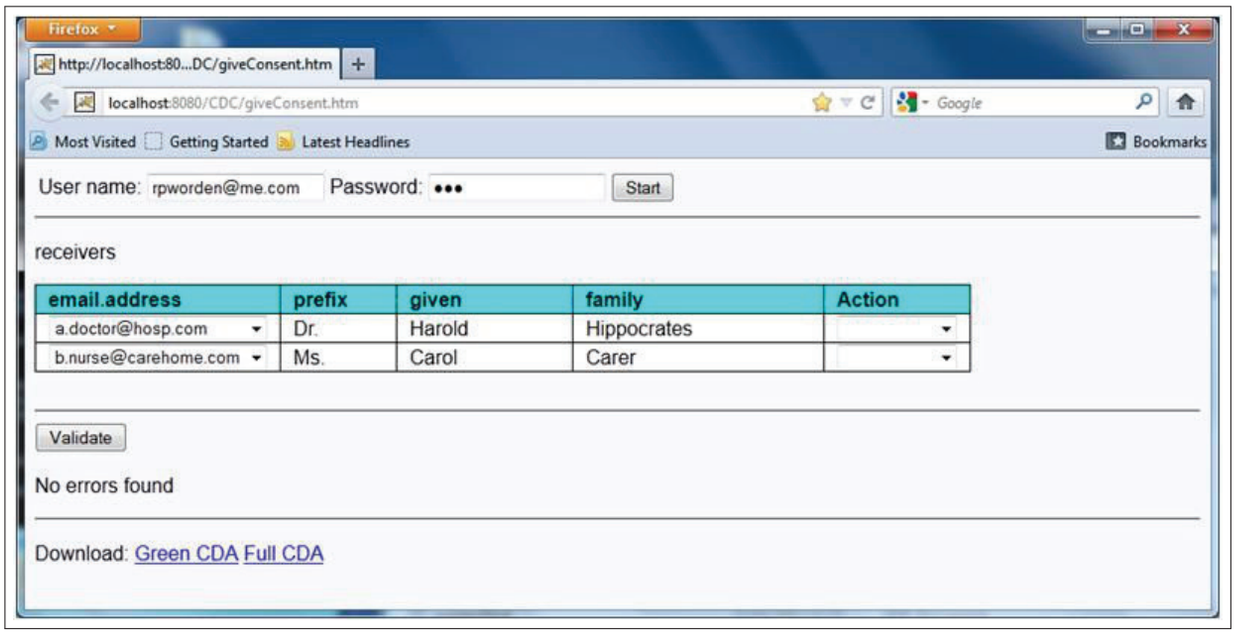


Figure 6: Consent-giving Editor

der codes 1, 2, etc.).

- Horizontal layout is used for the person's name, and address. Parts of the address have been re-ordered compared to the greenCDA.
- Validation has detected errors in the NHS number and the date of birth.

Once the patient has completed using this editor, the partially completed greenCDA is saved. At this editing session or in subsequent sessions, when the patient wishes to give or revoke consents, this partially completed CDA is retrieved, and the appearance of the second 'consent-giving' editor is shown in Figure 6.

Visible in this figure are:

- The patient's email address was used to retrieve the appropriate partially-filled consent directive.
- Patient demographic details have been hidden, as they are not relevant.
- It is assumed there is a separate process (not shown here) for registering carers, and for defining a short-list of carers whom the patient may wish to give consent to. This information is stored in a database.
- The patient sees the details of carers he has already given consent to, in a table.
- The possible actions (in the right-hand 'Action' cells of the table) are to add or remove a receiver/carer row, or to reorder the rows.
- All the patient needs to do is to choose the email address (assumed known) of a new carer he wishes to give consent to. Other details of that carer are then filled in automatically by lookup from the database. If it turns out to be the wrong carer, the row can be removed.
- The links at the bottom of the screen are for test purposes only, allowing a developer directly to view the greenCDA or full CDA

Both of these editors were developed using fairly simple tweaking of the 'raw' editor which comes out of the process of defining the green Consent Directive CDA, spending a few hours in tuning an editor configuration file and the few small Java support methods (for validation and data lookup) to which it refers.

4 Discussion

In order for the CDA standard to achieve its full potential in improving interoperability between healthcare systems, it needs to be acceptable to two key communities: to developers who need to build interfaces using CDA, and to clinicians who will author CDA documents. It can be argued that the clinicians are the more important of these two communities. If CDA-based systems are not appealing to clinicians and effectively reduce their workload, those systems will be used unwillingly or not at all in clinical practice.

We contend that greenCDA on its own is a major step forward in making the CDA standard acceptable and deployable by the developer community. It reduces the technical problems of interfacing systems to CDA to a standard run-of-the-mill XML interfacing problem, with none of the extensive technical complexity of HL7 RIM-based models.

The development described here of a Clinical Doc-

ument Editor framework builds on greenCDA, to make CDA-based systems more acceptable and efficient for clinicians. It is now possible rapidly to build and configure editors which automatically gather the necessary information from source systems, present it in a clear form to clinicians, allowing them to edit, review or approve it within access controls appropriate to their role.

A typical application for the Clinical Document Editor is in discharge messages from acute hospital trusts in the UK. The Royal College of Physicians of London has defined a set of headers and required content for discharge messages, to be adapted to individual local needs; and the NHS have defined a CDA-based document profile to embody this information. Providers of primary care are very keen to receive more prompt and high quality discharge messages from acute care, particularly including accurate and up-to-date information about medications. The Clinical Document Editor framework can be used to build clinician-ready editors for these CDA discharge documents, which automatically capture medication information from acute trust systems where that is available – so that clinicians do not need to re-enter medication information, in order to send it reliably in standard CDAs to primary care or other recipients. The same editing tools can also be used to view the resulting CDA discharge documents, in any clinical setting, using only a web browser. Once a generic editor for discharge summaries has been created, individual NHS trusts could then configure and extend it to meet their local needs. This illustrates the use of the Clinical Document Editor to realize the potential of CDA for healthcare interoperability.

The use of greenCDA and the associated editor tools is not restricted to the UK. The toolset is available from [9]. We look forward to working with the developer community to realize and extend the potential of the Clinical Document Editor toolset.

References

[1] Department of Health, The power of information: putting all of us in control of the health and care information we need, <http://informationstrategy.dh.gov.uk/about/the-strategy/> [Accessed 7 June 2012]

[2] R.H. Dolin, L. Alschuler, S. Boyer, C. Beebe, F.M. Behlen, P.V. Biron, A. Shabo Shvo, HL7 Clinical Document Architecture, Release 2, J Am Med Inform Assoc 13 (2006) 30-39.

[3] HL7. GreenCDA Project. <http://wiki.hl7.org/index.php?-title=GreenCDA_Project> [Accessed July 2011].

[4] D. Kaminker, HL7 Clinical Document Architecture Ambassador Briefing, 11th International HL7 Interoperability Conference, Rio de Janeiro, Brazil, 2010.

[5] NHS Connecting for Health. NHS Interoperability Toolkit Correspondence releases. <http://www.uktcregistration.-nss.cfh.nhs.uk/trud3> [Accessed 9 August 2012].

[6] Office of the National Coordinator for Health Information Technology, Health Information Technology: Initial Set of Standards, Implementation Specifications, and Certification Criteria for Electronic Health Record Technology, Federal Register 75 (144) <http://www.gpo.gov/fdsys/pkg/FR-2010-07-28/pdf/2010-17210.pdf> [Accessed January 2011]

[7] P. Scott, R. Worden, Semantic mapping to simplify deployment of HL7 v3 Clinical Document Architecture, J Biomed Inform (2012).

[8] R. Worden, P. Scott, Simplifying HL7 version 3 messages, Studies in Health Technology and Informatics 169 (2011) 709-713.

[9] http://www.openmapsw.com/downloads/downloads.htm

