An Electronic Healthcare Record Server Implemented in PostgreSQL

Tony Austin[1*], Shanghua Sun[2], Yin Su Lim[1], David Nguyen[2], Nathan Lea[1], Archana Tapuria[1] and Dipak Kalra[1]
[1]Centre for Health Informatics and Multiprofessional Education (CHIME), University College London, London, UK
[2]Helicon Health Ltd., London, UK

Submitted February 2015. Accepted for publication June 2015.

ABSTRACT
This paper describes the implementation of an Electronic Healthcare Record server inside a PostgreSQL relational database without dependency on any further middleware infrastructure. The five-part international standard for communicating healthcare records (ISO EN 13606) is used as the information basis for the design of the server. We describe some of the features that this standard demands that are provided by the server, and other areas where assumptions about the durability of communications or the presence of middleware lead to a poor fit. Finally, we discuss the use of the server in two real-world scenarios including a commercial application.

1. INTRODUCTION

Electronic Healthcare Records (EHRs) are increasingly perceived as the only mechanism to ensure capture of detailed clinical information over the lifetime of every citizen, and the availability of this information at any moment [1]. Paper records are liable to be misunderstood or unavailable at the moment of need and do not support sufficient care quality [2]. However, an EHR repository is more than just a database. It must exhibit properties of security and auditability that reassure patients and users that their data are not abused [3]. It must of course also be accessible so that the clinical benefits may be realised.

1.1. ISO EN 13606

A suitable representation for the EHR has required significant international research. Starting with European framework programme projects such as the Good European Health Record [1] and Synapses [4], the requirements for EHR architectures are then documented in national projects (e.g., [5]), academic work (e.g., [6]), and international standards including the most complete internationally agreed set of requirements so far [7].

[*]Corresponding author: Tony Austin, Centre for Health Informatics and Multiprofessional Education (CHIME), University College London, 222 Euston Road, London NW1 2DA, United Kingdom. Phone: +44 (0)20 3549 5300. Fax: +44 (0)20 7679 8002. E-mail: tonyaustin@heliconhealth.co.uk . Other authors: postmaster.electronsea@gmail.com; y.lim@ucl.ac.uk; d.nguyen@ucl.ac.uk; n.lea@ucl.ac.uk; a.tapuria@ucl.ac.uk; d.kalra@ucl.ac.uk; d.kalra@ucl.ac.uk

The earliest European interoperability standard, ENV 12265 [8] was published in 1995 and defined the core information properties of an EHR. ENV 13606 [9] (published in 1999) updated this and was the first complete information model standard. The latest evolution, ISO EN 13606, is the current approved standard at European and ISO levels for the meaningful exchange of clinical information between systems. It is designed as an interoperability standard to which existing EHR systems will transform legacy data. However the requirements met and architectural approach also make it a good information basis for the architecture of an EHR server. The authors used this standard as the basis for two previous generations of EHR server [3,10].

The standard has five parts, published between 2007 and 2010. It uses what is called the "twin model" approach. Recognising that it is impossible to describe the whole of clinical medicine but that it is possible to definitively establish certain features of sensible server design, two models are offered to implementers that inform different aspects of the process. Part 1 [11] establishes the set of "global truths" that are not dependent on the clinical domain being modelled. For example, it does not matter if the application is for the management of asthma or diabetes, all entries in either domain must be dated and recorded with the specifics of the clinician who observed them. In contrast, the knowledge that a thing called a "Blood Pressure" always includes further things called "Systolic" and "Diastolic" values is unique to clinical domains related to the circulatory system. Each of these individual clinical domain knowledge aggregates is what Part 2 [12] of ISO EN 13606 refers to as an "Archetype".

Further parts describe a set of vocabularies and reference archetypes to ensure a consistent mapping between EN 13606 and other key standards [13]; a framework for communicating the EHR disclosure wishes of patients and for communicating the audit history of who has accessed a particular patient's health record [14]; and interfaces between requesting and responding processes or systems to enable EHR communication [15]. The standard is maintained through a four-yearly technical revision process which is taking place now.

Nevertheless, the standard publishes an information viewpoint for the EHR, but does not provide the practical details necessary to implement a well-performing and secure EHR repository from which conformant extracts may be taken. This paper provides a validated example of the way in which EN 13606 may be used in practice to deliver a robust repository and services.

1.1.1. University College London (UCL) Server

The standard describes a document architecture. It provides parts such as a "Composition", a medico-legally whole contribution to the record, a "Section" (heading) and an "Entry" (which groups real data values as would a paper form). This aggregational data representation is somewhat at odds with the real-world penetration of relational (tabular) format databases into the healthcare enterprise.

The authors have produced two previous generations of server based on the evolving European and International standards described above. Both adopted the "twin model" approach. The first generation [3] predated many if not all of the modern turnkey Object-Relational Mapping (ORM) middleware tools and needed significant amounts

of hand-crafted code to marshal and de-marshal the hierarchical data from the tabular storage. Internally, the structure was a wide table populated as necessary. This could be searched quickly but was not an obvious architecture when used for queries outside the controlled environment, necessitating further bespoke code.

Object-Relational Model (ORM) and Web toolkits were much more advanced by the time of the second server [10] and it was felt that an implementation using Hibernate™, now a part of JBoss™, itself implementing Enterprise Javabeans (EJB) was appropriate. Hibernate provides more than one candidate for internal storage including the wide table previously chosen. However, rather than having identical fields with an "archetype identifier" slot populated with the desired clinical concept as in the first server, on this occasion, there were real classes for each concept that Hibernate then instantiated in the database. This meant that queries had a more natural feel, being based on tables with names and columns that made sense to domain experts. Sadly, there is an upper limit on the number of tables that may be used in an exhaustive query plan in PostgreSQL, and consequently, we used a hybrid arrangement where the reference model data was represented predominantly by a wide table, but with non-Elements (data containers) also having a table each to store their inter-object indexes. This allowed the number of tables in the query to be reduced.

Considerable effort was expended for the second server ensuring that the accessor Application Programming Interface (API) could not be subverted by (for example) requesting data for an inaccessible subject of care as part of a request for legitimate data, or maliciously rewriting identifiers prior to a database update to change the wrong record. This meant that the syntax made available for queries was much less rich than EJB Query Language (EJB-QL) would normally have permitted. This ultimately proved a problem in two ways: firstly, because we had too little control over the order in which queries were optimised and could not resort to native queries to overcome this; and secondly, because the syntax used was subsequently deprecated by Hibernate and we could not upgrade without major rewriting.

A third architecture was thus investigated that dispensed with the middleware sitting between the database (considered only as a dumb repository) and the client application. In this new version, the client and database would be expected to share the functions of the former middleware layer in the hope of reducing infrastructure and development complexity, and increasing performance through a reduction in data exchange.

1.2. PostgreSQL

We chose Hibernate at least in part because it freed us from having to require the use of a specific database. Oracle™ and SQL Server™ are commercial databases deployed widely in this locale. However, on no occasion have we been asked to migrate to another database from our default of PostgreSQL. This is because of cost issues of course but also because partners could then reasonably expect to benefit from our continued developments. It was therefore easy to conclude that giving up support for multiple databases was acceptable if greater control over queries could be obtained.

PostgreSQL™ [16] is a cross-platform open-source object-relational database management system originally developed at the University of California, Berkeley,

USA, as a successor to the Ingres database. It complies fully with the principles of Atomicity, Consistency, Isolation and Durability (ACID), the transactional principles that guarantee database modifications are processed reliably. The original Postgres was commercialised but in 1995 what was to become PostgreSQL began again as an open-source project. The database continues under active development adding many new features with each annual iteration. It is of course only one such database but the local team has considerable experience with it and it is a natural choice.

PostgreSQL has been used previously in healthcare, especially to provide type systems suitable for use with HL7 [17] and with ISO EN 13606 [18]. Khushi found that PostgreSQL exhibits favourable performance compared to other open-source databases in the related field of genomics [19]. PostgreSQL and indeed the relational database management system approach itself is of course only one possibility and object-oriented and XML databases are equally reasonable [3]. A comparison with NoSQL databases is offered in [20].

Since middleware by definition does not include a database, it must follow that storing the output from a middleware computation requires one. If the results of the middleware computation are ISO EN 13606 extracts, it must follow that it is possible for a database to store ISO EN 13606 extracts because that is what the middleware asks of it. By asking a client to emit data in an identical fashion, the same result will be achieved without middleware.

Of course, asking a client to perform the storage is potentially insecure and in any case will result in a data schema that is highly atypical. The more interesting questions, and the ones that the remainder of this paper applies itself to, are whether this can be done securely and whether it will result in a schema that would be recognisable to a database manager grounded in relational theory but not necessarily in EHRs.

2. METHOD

In order to create and deploy the database, a number of database "creation" scripts have been written that group modifications logically. Four of these scripts are publicly accessible [20] and represent directly the types, termlists, demographic entities and record model implied by the standard. However these are by no means enough and further scripts are needed for security and audit, and to support external clinical data modelling and versioned record storage. Key aspects of these are described below.

2.1. Patterns

The standard defines a methodology for modelling the structure of a record known as the "Archetype". Archetypes can inform a variety of formalisms including XML [22] but do not address quality issues directly [23]. The development of servers and clients can require the modelling of more than simply the structure of the record [24,25], and in the previous version of the server, all modelling was provided in the Hibernate Java classes using Java class-file annotations. This has been formalised in the current version as the "Pattern", literally a model for collections of Java annotation-like [26] statements from which an application can be derived. Healthcare applications lend themselves to Model-Driven Development and for example, OntoCRF [27] is also model-driven; this time grounded in the Web Ontology Language and targeted initially at research.

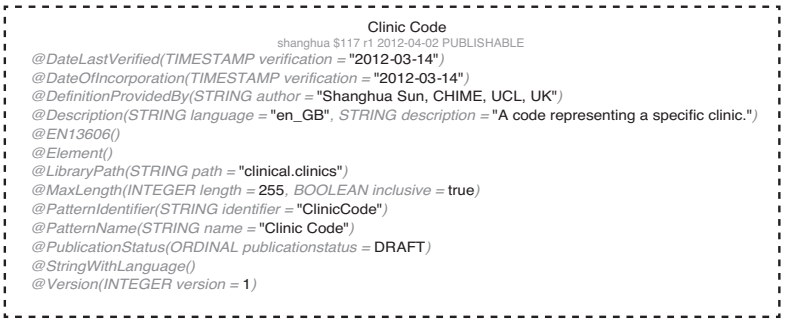

Figure 1. An example Pattern for "Clinic Code".

Patterns are defined in tools separately from a Record Server and grouped in "Distributions". An example of a Pattern from a public repository is shown in Figure 1. It comprises a collection of statements that must be true of a semantic concept. A Distribution comprises a collection of Patterns in specific versions and in theory, inclusion of a Distribution identifier should be enough to permit them to be dereferenced from a knowledge environment. The database schema should then embody that associated information model.

Because detailed Pattern information appears in the knowledge environment, and in particular children of a Composition appear there, it is not necessary to repeat this information in the database. The need is simply to:

1. tell a client what Patterns are supported in the database (via the Distribution identifier and version);
2. permit a view to elide inappropriate values (based on a Pattern identifier and sensitivity);
3. enable an audit trail to include which Pattern definition an inclusion was derived from; and
4. enable schema evolution to occur semi-automatically based on knowledge of how data for a Pattern is stored.

Versioning and sensitivity information are centred on the Composition and so only information relating to that content type are replicated. In order to determine that there is more than one variant of a Pattern in the database, and where the data for each may be located, a join table is necessary. It ensures that results removed from a query based on inclusion in a specific distribution are not removed if they appear in another one for which response is permitted.

2.2. Security and Login

Front-line anti-virus and firewall protection falls to the host operating system. PostgreSQL also has a perimeter defence of its own, which is configured with the file "pg_hba.conf" inside the cluster data directory. Configuration options include those to prevent access outright to individual databases from certain users or IP-address ranges or to require SSL to encrypt connections. While the latter is less important when an application server and the database are executing on the same device, it is more warranted if a network conversation is needed.

For a record server, it is necessary not only to provide a username and password at login, but also the care setting that restricts the scope of a patient search and a role that limits the sensitivity of records that may be accessed. All four must be known; a user is not the superset of all their roles, even if they qualify for more than one. Consider that a consultant may also be involved in research and may have legitimate access to a record for the purposes of clinical care but may be denied it explicitly if a patient does not consent to the research. The nature of the access to a record will be indelibly marked with the purpose for which it was accessed.

The server's security model was described in a previous paper [10] and is reproduced in Figure 2. An "Account" defines a clinical care setting and groups Users

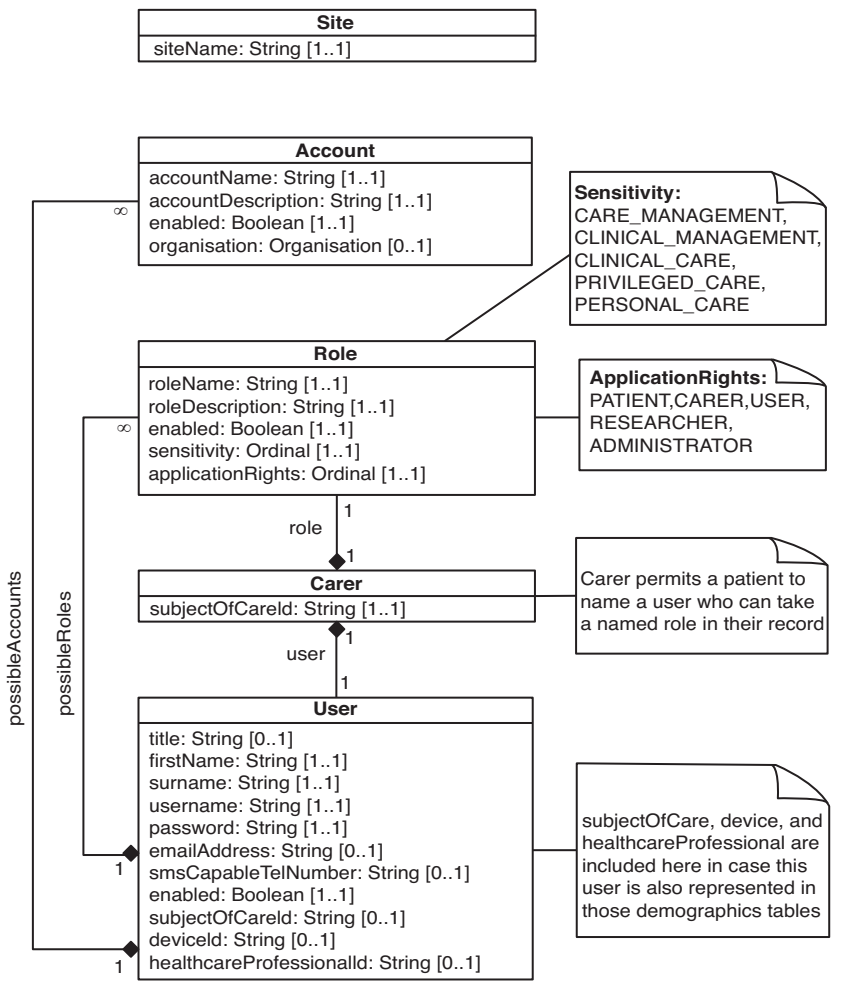

Figure 2. The model for Users [10]. (Reproduced with permission).

and Patients. Patients are visible to Users that share the logged-in Account. A "Role" describes the reason that a record is accessed and, in practical terms, sets the sensitivity level (from five) of records that may be retrieved.

All logged-in Users have "Application Rights" that define at a coarse granularity what they are permitted to do. These rights are the following:

Administrator may create users, and has access to demographic data (only) for patients in order to bulk-assign accounts;

User is a regular clinical user with access to record and demographic data for patients in their logged-in account;

Researcher is the inverse of an administrator, having access to global record data for population queries but not to demographic data with which he/she might identify individuals;

Carer has similar access privileges to a user, but only with respect to those patients with whom the carer shares an account and has been nominated as carer by a patient;

Patient has universal access to his/her own data (only).

The record server does not provide dynamic roles. Such roles are associated with sets of permissions at run time by an administrator and govern what tables (and rows) may be accessed. Unfortunately for a record server, a database does not record what permissions were available to a role at any time, so that it is not possible to recover the set of available data to a user at a later date. Nonetheless, it is still necessary to be able to vary the rights of a record accessor from the default values in certain circumstances. The server permits those variations to be recorded in a version-historied format, and for them to inform the views from which the data release occurs. The server:

1. provides a space where documentation describing reasons for variance from default behaviour may optionally be provided.
2. provides a versioned trail of global sensitivity variances.
3. provides a versioned trail of patient consent preferences
    a. which can vary the default create, amend, access rights on a per-Role, per-Account and per-User basis,
    b. where in addition, specific Compositions, Patterns or Distributions can be always included or excluded where they are found,
    c. ensuring users and patients cannot view or change each other's provisions.

An administrator may vary sensitivities for patients in the entire database (2 in the list immediately above). This is important because if the database holds data for Distributions of significantly varying sensitivity then disclosure will occur. Consider a hypothetical database containing sensitive drug rehabilitation data and less sensitive influenza data. For users of the addiction database, a default sensitivity of 3 might be appropriate with respect to their own data cohort. Likewise, a default sensitivity of 3 would be fine for the users of the influenza data. However, it would not be appropriate for users of the application for influenza to treat the addiction data as of sensitivity 3. The global_sensitivity_variances provide a means by which an administrator can declare overriding default sensitivities for roles with respect to Distributions (in other words, a way of attaching an optional role default sensitivity to cohorts of data in addition to the global default one).

As well as overriding the sensitivity evaluation, a result may be blurred so that a recipient either:

- **must** be given this value if it appears in a result set irrespective of its sensitivity;
- **must not** be given this value even if it appears in a result set irrespective of its sensitivity;
- **can** be given this value if it appears in a result set and is insensitive enough but it should be disguised in some way.

Of course, in the absence of an identifier, variance results will be presented unchanged as their sensitivities suggest.

2.3. Audit

One of the key features that sets a record server apart from a "normal" database is its ability to record what actions are performed using it. This is desirable because research shows if users know that their actions are being recorded, then they are less likely to abuse the trust they have been given and use the data inappropriately [28].

For those connections that pass perimeter defences, PostgreSQL provides a logging facility that produces a durable record of the queries actioned on the server. This is usually forwarded to a different machine from that being logged, to prevent tampering, and discharges a widespread legal requirement to record actions performed on a database holding sensitive personal information. The log can be re-imported into the database for analysis.

Unfortunately, this fine grained logging cannot explain why a select was performed. In a middleware-oriented record server, it is easy enough to provide functions on the server-side which write a line to an audit table whenever they are called. This constrains what users are able to do (i.e., they can only accomplish what there exists a function for) but also provides a context that helps explain what action is being taken. For example, "getSubjectOfCare(String identifier)" is a function that limits accessibility of a subject to that which matches a provided identifier, but also enables an internal request "select * from subject_of_care where id = identifier" to be explained in terms of the call that preceded it. Providing direct access to the database removes restrictions on user requests, but in the process loses the context that enabled the auditing to occur. Any actions requiring a subject could cause the same select to be executed, but without an API call, there is nothing to label them with.

This implementation provides a new explain_log table to overcome the loss of context and forces clients to explain themselves while taking retrieval actions. Provision is made for audits on role (and associated behaviours), subject of care, and records, in an effort to ensure anything that might affect data retrieval is noted. The basic procedure is as follows:

1. The client begins a new transaction and writes a row in the explain_log table with the explanation. This is committed with the transaction identifier automatically.
2. The client then executes their SELECT.
3. The database receives the request and invokes a trigger on the table which looks for the explain_log row with which it shares the transaction identifier and throws an exception if one can't be found.

4. A "proceedings" table is updated to show that the database performed an action (for example, "select CLIN_Remarks").
5. The client can continue to perform additional actions that are conceptually part of the same activity.
6. Finally, the client commits the transaction in preparation for the next activity.

The database refuses to perform an action unless there is an explanatory row in the explain_log and only one log entry is permissible per transaction. If the transaction identifier were not the primary key of the audit table, a client would have to supply one and this would have to be via some sort of session variable since both the client and the tables need to be able to independently find the same row.

Note that addition of record data and subsequent change (e.g., after noticing an error) can both be unambiguously audited even without an explain_log as they only apply to one named table at a time. But select queries may span tables and deliver arbitrary results, and it is these that must be explained. Used in conjunction with PostgreSQL's own logging facility, the explain_log can ensure compliance both with the desire to make usage information available to applications, and with the legal requirement to log usage data to an external machine.

2.4. Record Structure

Adding versioning to Record Components is arguably the most complex part of the server development. It will be instructive to consider the standards-mandated requirements for this. For the sake of argument only, the server in the single-version case could be rendered in one sparse, fixed-width "record_components" table where each row below the Composition level is provided with a foreign key reference to its immediate parent. According to the standard, any attempt to update one of these rows must necessarily invalidate not only the row updated, but every row "above" it in the aggregation hierarchy. Moreover, it must be apparent that any content that was not changed remains the same in the revised container.

To illustrate, consider a Composition with identifier C, an Entry with identifier E, and two Elements with identifiers e1 and e2, respectively. These have been established in the record server, but it is later found that Element e1 was entered in error and is to be revised with a corrected version. "Relationally", an UPDATE to e1 would be expected, giving rise to e1'. However, this is not sufficient. Since the container of e1 now contains new data, E must become E', and likewise C becomes C'. But, since e2 did not change, the new contents of E' are e1' and the original e2. Moreover, the rows must be protected from subsequent edit. Once E' has been established, it cannot be possible that another Element e3 is added later (i.e. with a foreign key to E') without there having been an "official" revision made.

There are no perfect realisations of this versioning requirement that both behave relationally and yet preserve the standard's semantics. Many options were considered, a full treatment of which is beyond the scope of this paper. Eventually, we elected to simplify the standard itself. This imposes a document-orientation on something that is not itself necessarily an aggregate structure. Most screens would naturally be represented as a single table (corresponding to the Entry) with columns for the values (Elements). The Cluster provides only reuse value if individual Elements within it can

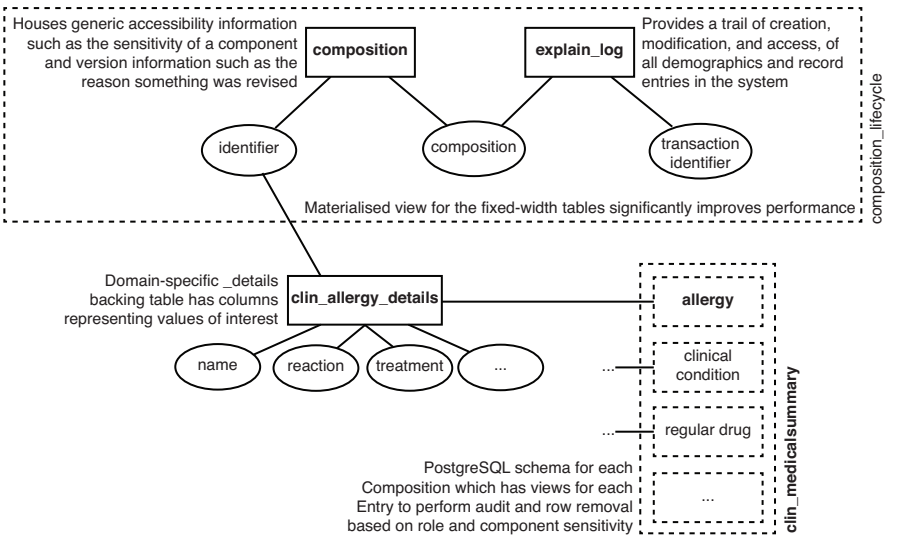

Figure 3. The Record Architecture.

be flattened and provided as Elements inside the Entry without it. Likewise, the Section provides a useful grouping at the UI level but has no persistence implication beyond the irrelevant need to store its ID. So long as an export tool can manufacture appropriate IDs repeatably when needed, there is no necessity for their storage.

A hybrid solution (shown in Figure 3) is therefore taken with a genuine Composition table that stores generic sensitivity and audit information but with other distinct tables unique to a domain-specific representation (such as an "allergy" table) each with columns that are meaningful in the domain (e.g., "reaction"). These then have appropriate table constraints that prevent data being updated after their original commit. Note that the domain-specific tables and views are derived directly from the Pattern definitions by automated code generators.

In its current version the server comprises 26 PostgreSQL scripts, in total about half a megabyte in size not including default imports for UK NHS demographic entities. The scripts include just over 11,000 lines of code including comments but not including additional test code that executes externally within the Play Framework.

3. RESULTS

The server has been used in two distinct clinical domains, underpinning one academic and one commercial Web application.

3.1. Cortext

Approximately ten years ago within the Dementia Research Centre in London, a Microsoft Access™ database was created to facilitate both clinical and research data

Figure 4. The Cortext Application.

recording. Since then, the data requirements changed significantly but the database design did not remain up to date. As a result, user communities developed their own local solutions. Recently, funds became available to decommission the Access database in favour of a new Web-based solution based on the server described in this paper.

The application (shown in Figure 4) is designed to support ongoing dementia research studies conducted by the Centre, receiving patient data from neighbouring neurological institutions where consent for this has been obtained. As well as the security that would naturally be expected of such an application, the ability of the server to store complete version histories of all records was seen as an important generic feature. The application provides for the following:

Consent The application enables capture of consent for general contact, for research, for specific studies, and also suitability for studies.

Studies The application can define the properties of studies including a visit schedule for patients, and records reimbursement for visits made.

Clinical Record Although not aimed at clinical use, some relevant clinical data such as cognitive assessments and a medications summary may be made available. Familial relationships may be defined to indicate mutations that may be in common.

Imaging Images may be requested and results may be obtained through the application interface. In addition, the location of tissue samples may be recorded.

Neuropsychology Tests may be defined in the application and grouped into one of several "batteries" for administration to patients.

Post Mortem Reports from a post-mortem may be recorded, and any wish for brain donation can be registered.

3.2. HeliconHeart

As well as academic partnership, the server is available to license as part of a commercial venture. The first such licensee is Helicon Health Limited. They are using the server to underpin their flagship cardiovascular application HeliconHeart^{TM}. This delivers support for clinical practitioners involved in anticoagulation prescribing and atrial fibrillation with the intent to reduce the incidence of stroke or major complications from medication.

This application (shown in Figure 5) is focussed on clinical care of the patient (and to some extent on the day-to-day management of a clinic) rather than research. It presently underpins a distributed anticoagulation service [29] in North London (including the catchment areas of five Clinical Commissioning Groups). The distributed service there includes not only hospital-based clinics but also General Practices and secondary prescribers. The service has been recognised personally by the British Prime

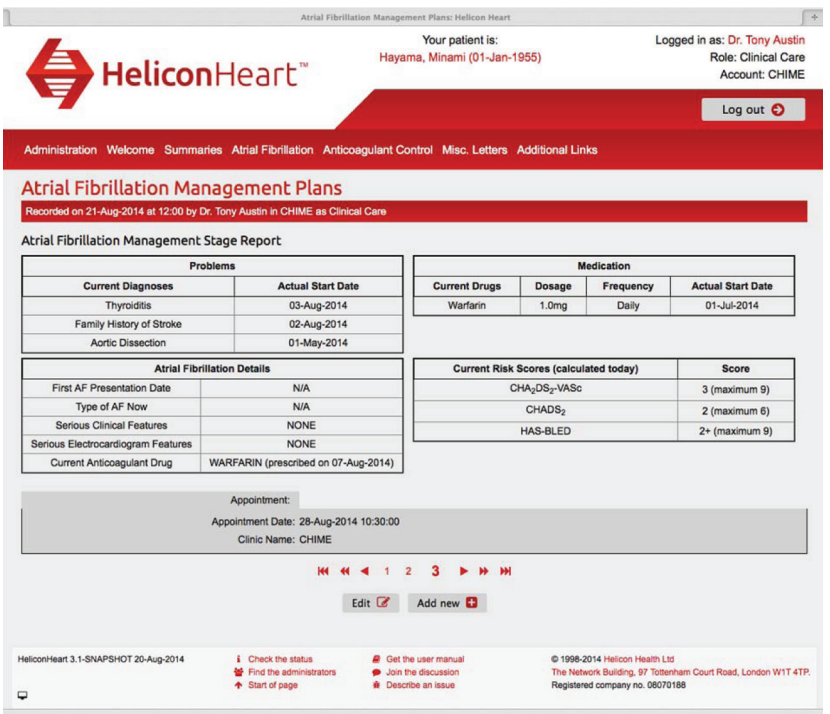

Figure 5. The HeliconHeart Application.

Minister as delivering the highest level of customer service excellence and the application has been CE accredited as an in-vitro medical device according to the relevant European directive [30]. The application includes the following:

Clinical Record Many generic data capture options are available, including those for correspondence with and about the patient, key people in the patient's wider care team, allergies, concomitant conditions, medications and educational interventions.

Service Delivery Some patient data relating to delivery of the service can be stored, such as referrals, complications that have arisen during care, and appointments not attended. In addition, Standard Operating Procedures are available for download and are stored with a date by which they must be re-validated.

Decision Support Decision Support for the administration of Vitamin-K Antagonists (specifically, warfarin) is provided on-screen. The advice has previously been shown to be effective in raising the competence of all practitioners to that of an established expert in a randomised controlled trial [31].

Scoring Well known cardiovascular risk scores are provided and populated directly from the record.

Other Advice The application features collapsible panes that appear during data capture to aid new users, as well as charts and tables that provide ready access to historic data. Alerts are given for imminent plan closure and overdue annual review.

Clinical Governance The effectiveness of each care setting and the service as a whole can be monitored in real time using an anonymised report that plots a histogram of divergence from a target International Normalised Ratio (INR).

3.3. Performance

Performance of the implementation has been tested with a mid-sized departmental record database consisting of data for approximately 15,000 patients with a combined 500,000 record entries of all types. In the sample application domain, one clinical pane is populated for all patients more frequently than any others and about 250,000 record entries are of this type. To obtain the data for a single subject of care having a large number of record entries of this type (72, over a median of 13) takes approximately 350 ms. The example result is obtained using a default installation of PostgreSQL 9.2 on a modern 16GB laptop equipped with an encrypted Solid State Disk.

4. DISCUSSION

The work described in this paper takes the ISO EN 13606 standard as its information basis in particular. The usefulness of these results is therefore somewhat limited to users of that standard, although a document-oriented record architecture-based approach of a similar type is increasingly noticeable even among other standards (for example [32,33]).

4.1. Interpretation of the Standard

It is axiomatic that the ISO EN 13606 model can represent in transit any clinical data and it therefore follows that a server capable of storage using this model could store any clinical data as well. Obviously, data represented using other models would need translation into the standard model first. However, to achieve this generality, ISO EN

13606 permits infinite nesting of Sections and Clusters and this makes for a sub-optimal relational equivalent. We reject such nesting and therefore cannot assert that all possible valid EN 13606 instances can be directly represented in the server. The usefulness of these results is therefore also potentially limited to those prepared to refactor their clinical requirement into a suitable model subset. In practice, this has always been possible so far.

Although the standard suggests that two different entities may be referred to by the same extract_identifier in different extracts, this implementation makes clear that the identifier is globally unique. It is in general not possible to update entity data reliably if the updated set does not reuse the same identifiers (because it will not be possible to locate the originals in the existing tables).

If a mandatory value is not provided, it should either have a "DRAFT" version status or a reason for revision. This caters for two scenarios: An emergency scenario, where a doctor rushes out of his/her office to attend to a patient but needs to save (or have saved) his/her work up to that point (where DRAFT status enables mandatory checking to be bypassed) and an inappropriateness scenario, for example, where an ankle check is mandatory on a screen but the patient is an amputee and can't be checked (here, the reason can be used to explain the absence of the value but the screen as a whole can be FINISHED and not a DRAFT). This replaces the "null_flavour" provision in the standard whose value choices are questionable [18].

Translations of provided Compositions may be made into several languages using several possible units. The standard took the position that data types should be responsible for translations but such types are not typical in engineering environments and this makes use of the standard unnecessarily complex in code. In theory, a String gives rise to further versions and also further translations, which can themselves be versioned. To simplify this, translations exist within the version history and a version status of "TRANSLATION" indicates which modifications did not change the source data. All translations are assumed to apply to the most recent version and are (probably) invalidated by subsequent versions in the same language. Advanced clients can select which Compositions to view based on the territory of use and whether a compatible translation exists in the revision history.

The first part of the standard describes a significant number of attributes that may be applied to classes in the record hierarchy. For example, the "Subject of Information Category" attribute may be provided to an Entry class. However, the first part of the standard provides a generic model and so the attributes are assumed to be available for every instance of their class. There is no attention given as to whether the recording makes sense in a given context. If the context is a "Cardiovascular Check", then the subject will be the "PATIENT" in all cases, and it is redundant to record this in the database every time. In this implementation, the Pattern defining the clinical model calls for the Subject of Information attribute if desired. This simplifies the resulting relational model which only needs to record values and not associated attributes that mostly remain empty.

4.2. Login

Part 4 of the standard [14] provides security-related constructs including audit. That work devotes much effort to describing a shared policy model intended to inform

downstream servers of the relevant information governance for data that have been shared. In the meantime, very little effort is spent describing how to log in to a remote server to retrieve the policy model in the first place. Nevertheless, work has been done to realise an access control model based on the standard [34].

For our database-centric approach, there were several implementation strategies to choose from. In one, just a single global username is provided and all potential users of the record first connect to the database with that global name. The "real" contextual account, role and username are then established after login using a separate table. This approach is very advantageous for an application server, which would like to open multiple connections and recycle them for efficiency. However, it does mean that the database cannot participate in perimeter security at all. All access must be allowed and the "pg_hba.conf" file is redundant. The database would also not be able to automatically audit activity as it would have no useful "session_user". Nevertheless, this approach does work and is very common among high-throughput Web applications.

A slightly more database-centric approach is to login with a database username and password and then "set role" in order to get access to any data. A disadvantage of the approach is that using real database users limits scalability to that of the database itself. We have tested the system at the departmental scale of the order of hundreds of users, not so far at institutional or regional scale where user numbers might be in the thousands or higher. It is not possible in PostgreSQL to limit database roles to "precisely two", for the account and the role, but it is possible to limit the database roles to "precisely one" via the noinherit attribute. Therefore, it is necessary to establish combination database roles involving both aspects (for example, "set role MYHOSPITAL_NURSE"). These can be attached to groups representing the application rights so that database grants work in a natural way. Unfortunately, the User-Account-Role join table that attaches users might diverge from the corresponding database roles. An administrator with "createrole" privilege could delete database roles, and although a delete trigger on such a join table could remove database roles if a row was removed, the inverse is not possible. The solution is to deny all roles (even administrator) the ability to modify database roles and have this happen on their behalf using triggers on the accounts and roles tables.

Further scoping of accounts could be possible by associating them with "domains". These account groupings would allow one server to host networks of linked local clinic settings. A SUPER_ADMIN application right would be needed to create domains and admins for them. For the time being, the same effect is obtained by having multiple servers, at the cost of it being harder to transfer records between domains.

4.3. Exceptions

In more recent versions of PostgreSQL, it is possible to supply a SQLSTATE code among the results of an exception. However, this will be the single response from a validation and does not allow for the possibility that more than one field was in error simultaneously. Since our validity checking is "looser" than that of the database's own (for example, we allow a "non-null" field to be null if the contribution as a whole is a DRAFT), we must implement validations as separate functions. This has the additional

advantage of being available even prior to an actual attempt to commit. The function can supply the complete cohort of failures as a single set that can be highlighted on-screen. A client needs to know what values were in error, along with codes that support localisation of the response message. If the exception is with the contribution as a whole, we adopt a convention where we nominate the identifier as the field in error (since problems with identifiers would be a client bug and not a failure to validate user input).

4.4. The "explain_log" Entry

The explain_log has proven particularly tricky to write. Recall that an explain_log row is provided by a caller and then rows in a separate proceedings table are provided by the database as evidence that it really performed actions amounting to the stated objective. The two-table arrangement is needed because in PostgreSQL 9.2, it is not possible to defer the check for column not null to close of transaction, only the non-fulfilment of a foreign key constraint. The separate table enables the proceedings to be completed in subsequent steps but still exception if the transaction completes with apparently no actions taken.

The explain_log requires some fields (such as the transaction identifier and the committing user) to be computed rather than provided. For this reason, the table is updated via a function "initialise_explain_row". Because this function writes to a table with no public grant, it must be declared "security definer". That means the "current_user" will be that of the superuser that created the function, not that of the user that called it subsequently. That in turn means that the function requires the current_user to be passed to it on execution. Table triggers that could call this function would also have to be declared security definer to use it and would therefore also not have the current_user. A PostgreSQL rule does allow parameters to be provided, but this cannot be attached to a table as strongly as a trigger for the purposes of permissions checking and will always remain available to execute independently. This is a problem because we would rather reserve "created" and "updated" audits to the database as a side effect of a real modification (for example, "SUBJECTOFCARE_CREATED") and not permit end-users to add them manually without an appropriate corresponding change.

There is another important limitation of explain use in a transaction where rolling back will also roll back the explain_log entry. This is less important for INSERT, UPDATE and DELETE where the relevant activity will not then have happened either. But for SELECT, it is important because the data will still have been revealed. To prevent this happening, commercial databases use "Autonomous Transactions" inside transactions with a larger scope that can commit irrespective of the commit status of their container.

4.5. Demographics

Two table representations are applicable to demographic entities corresponding to those provided by ISO EN-13606 part 1. A normalised layout has the greatest congruence with the published standard and is the obvious choice. However, a means by which entities are updated is not described by the standard. More than one piece of

information with the same type can be received for the same entity, and normally this would cause previous instances to be overwritten. This is not the standard's intent but rather, such an occurrence implies the existence of a new version of the information. A normalised layout makes versioning quite complicated.

In this implementation, each of the demographic tables is self-contained, except for a link to a separate table storing the global extract identifier. One row in each table is sufficient to describe all the attributes of each demographics entity. Persons (Healthcare Professionals and Subjects of Care) may have a multiplicity of identifiers defined for them but this is represented using a PostgreSQL array type (with a GIN index) to avoid the need for a related table. Storage of multiple historical values is then simply a matter of adding dates of obsolescence to each table via ALTER TABLE commands.

4.6. Folders

In this implementation, sub-folders are removed. Folders hold a set of identifiers for Compositions like "tags" pointing to photos in a photo-sharing application. Sub-folders do permit a namespace to be defined for the "tags", but also make it complex to merge records for two subjects of care, since it is in general not possible to determine which was the "correct" folder containment arrangement from the merge candidates.

In practice, there's actually no need for a Folder to even be for a specific patient record. A globally unique name can be given to the Folder and a set of Compositions related to it from any subject of care. A client can retrieve the Folder contents for a specific subject, or for all subjects, knowing that the view will omit any of those not permissible to see in the same way as if the Compositions had been requested directly.

4.7. Record Architecture

The ISO EN 13606 standard assumes that middleware will be required for implementation and in particular that this will use an object-oriented, not relational, engineering approach. This sometimes makes for a sub-optimal relational model, so we limit the open-ended EN 13606 aggregation possibilities to those that result in sensible relational outputs as follows:

- a Composition containing an Entry,
- a Composition containing several Entries, each of which must be unique in the Composition scope, and
- a Composition containing one or more non-nested Sections, each of which must be unique in the Composition scope and containing one or more Entries, each of which must also be unique in the Composition (not Section) scope.

Within this containment structure, the following clinical data constructs may appear:

- a (possibly very long) list of non-repeating (that is, cardinality 0..1 or 1..1) Elements,
- a list of non-repeating Elements that includes one or more non-nested, non-repeating Clusters that themselves contain a list of non-repeating Elements,
- a list of Elements, some of which repeat, and
- a list of Elements, some of which repeat, that includes one or more non-nested Clusters that also repeat which contain a list of non-repeating Elements.

PostgreSQL provides a schema feature that naturally lends itself to compartmentalisation of data within a database. By default, PostgreSQL does not really like long names, having a 64 char total limit. With the Composition and Entry names in the title, this is easily exceeded with untidy truncation as the consequence. On the assumption that an Entry cannot appear more than once inside a Composition (even within a Section), it is now instead converted in general to the form CLIN_CompositionIdentifier.EntryIdentifier using the first part as a PostgreSQL schema identifier to double the possible name length. The EntryIdentifier refers to the view on the real table that can remove data inappropriate for the caller to see. The real table of course has a permission arrangement that prevents direct access.

Clusters that do nothing but group basic Elements, that is those that do not themselves repeat and contain no Elements that repeat, are inserted into the Entry table with the Cluster name prepended (for example, a column name of "blood_pressure_systolic"). Repeating values (that is, cardinality 0..* or 1..*) need a downstream table. Ordinarily, a relational structure based only on a foreign key from a downstream table might allow new values to be manifested inappropriately after creation, but in PostgreSQL, it is possible to use an array in the upstream table for reference instead. It then does not matter what inserts are made to the downstream table, the Composition is immutable.

All imported components should be stored with the original IDs, but this implementation cannot do so. Fortunately, we are able to store the Composition ID itself in the explain_log table, and since a standards-compliant query cannot request any data below that level, it does not matter that sub-identifiers cannot be stored.

5. CONCLUSION

The authors have created record servers based on the ISO EN 13606 standard using three different development architectures. The implementation in PostgreSQL described in this paper offers great flexibility, requiring only the support for relational databases offered by most engineering environments. It offers this flexibility while still retaining all of the auditing and non-repudiation features expected of a full-featured record server. However, delivering these features is difficult and requires some novelty in practice, for example when explaining the purpose of queries, and the server omits some less critical features of the standard such as the infinite nesting of headings.

Of course, future versions of PostgreSQL will simplify or make possible some features that are presently prohibitively difficult to achieve. For example, if and when introduced, autonomous transactions will prevent it from being possible to roll back explanations of SELECTs.

The approach detailed in this paper also simplifies provision of clinical data for research use. This is usually based on the actions of a curator who creates a physical subset of data imported and collated from primary EHR sources. The database described herein already performs data release on the basis of specific permissions. By manipulating the permissions in place, it would be possible to grant access for researchers to the actual source EHRs, knowing that only data for which consent had been obtained would be visible. This will form the basis of future work.

ACKNOWLEDGEMENT

The authors would like to thank the Dementia Research Centre for their support of the development of the Cortext application. They would also like to thank Helicon Health Ltd. for permission to use screenshots from their application in this paper. Finally, they would like to thank the current and previous PostgreSQL developers for their contributions to a product that has brought them so much *fun*.

CONFLICT OF INTEREST

The authors wish to make known their association with the following organisations and projects: The EuroRec Institute, The European EHR4CR project, The European EMIF project, The European Discipulus project, The European SemanticHealthNet project, and the UK-based RAFT project. The lead author, Tony Austin, is a co-founder and shareholder of Helicon Health. UCL CHIME received payment from the Dementia Research Centre for the development of the Cortext application.

REFERENCES

[1] Lloyd D, Kalra, D, Beale T, Maskens A, Dixon R, Ellis J, Camplin D, Grubb P, and Ingram D, (Eds.), The GEHR final architecture description. The good European health record project: deliverable 19, 250 pages. European Commission, Brussels, 1995. https://www.ucl.ac.uk/chime/research/gehr/deliverable-19.pdf. Accessed January 2015.

[2] Grimson J, Grimson W, Berry D, Stephens G, Felton E, Kalra D, Toussaint P, and Weier OW, A CORBA-based integration of distributed electronic healthcare records using the synapses approach. IEEE Trans. Inf. Technol. Biomed. 1998, 2(3):124–138.

[3] Austin T. The Development and Comparative Evaluation of Middleware and Database Architectures for the Implementation of an Electronic Healthcare Record. (Ingram D Ed.). CHIME, UCL, London, 2004.

[4] Grimson J, Grimson W, Berry D, Stephens G, Felton E, Kalra D, Toussaint P, and Weier OW. A CORBA-based integration of distributed electronic healthcare records using the synapses approach. IEEE Trans Inf Technol Biomed. 1998, 2(3):124–138.

[5] Introducing computer based patient records: prerequisites and requirements. Swedish Institute for Health Services Development (SPRI), Sweden; 1998; Report number 477; ISSN 0586–1691.

[6] Kalra D. Clinical Foundations and Information Architecture for the Implementation of a Federated Health Record Service. PhD Thesis. University of London, 2002. http://discovery.ucl.ac.uk/1584/. Accessed January 2015.

[7] ISO 18308: 2011 Health informatics Requirements for an electronic health record architecture.

[8] Hurlen P (ed). ENV 12265:1995. Electronic Healthcare Record Architecture, 1995. Brussels CEN Technical Committee/251.

[9] Kay S and Marley T (eds). EHCR Communications: Part 1 Electronic Healthcare Record Architecture. ENV 13606. CEN, Brussels, 1999.

[10] Austin T, Lim YS, Nguyen D, & Kalra D, Design of an Electronic Healthcare Record Server Based on Part 1 of ISO EN 13606. Journal of Healthcare Engineering, 2011, 2:143–160. http://multi-science.metapress.com/content/121507. Accessed January 2015.

[11] ISO. (2007). BS ISO EN 13606 part 1 (No. EN 13606). (Kalra D & Lloyd D Eds.) 2007, Vol. 1:1–103.

[12] ISO. (2009). BS ISO/IEC 19770 part 2 (No. 19770) 2009, Vol. 2: 1–112.

[13] ISO. (2008). BS ISO EN 13606 part 3 (No. EN 13606). (Kalra D & Lloyd D Eds.) 2008, Vol. 3:1–47.

[14] ISO. (2009). ISO EN 13606 part 4 (No. EN 13606). (Kalra D & Lloyd D Eds.) 2009, Vol. 4: 1–52.

[15] ISO. (2010). BS ISO EN 13606 part 5 (No. EN 13606). (Kalra D & Lloyd D Eds.) 2010, Vol. 5: 1–21.

[16] PostgreSQL Consortium. History. http://www.postgresql.org/about/history. Accessed January 2015.

[17] Havinga Y, Dijkstra W and de Keijzer A. Adding HL7 version 3 data types to PostgreSQL, 2010. http://arxiv.org/abs/1003.3370v1 17. Accessed February 2015.

[18] Sun S, Austin T, and Kalra D. A Data Types Profile Suitable for Use with ISO EN 13606. Journal of Medical Systems. 2012, 36(6):3621-3635. doi:10.1007/s10916-012-9837-z.

[19] Khushi, Matloob. Benchmarking Database Performance for Genomic Data. Journal of Cellular Biochemistry 116:877–883 (2015). doi: 10.1002/jcb.25049.

[20] Ercan MZ, Lane M. Evaluation of NoSQL databases for EHR systems. 25th Australasian Conference on Information Systems. 8th-10th Dec 2014, Auckland, New Zealand. http://aut.research gateway.ac.nz/bitstream/handle/10292/8134/acis20140_submission_117.pdf?sequence=1 Accessed February 2015.

[21] Centre for Health Informatics and Multiprofessional Education. Latest ISO EN 13606 Reference Schema. http://www.ehr.chime.ucl.ac.uk/code/schema_3.0.4/. Accessed February 2015.

[22] Kobayashi S, Bosca D, Kume N and Yoshihara H. Reforming MML (Medical Markup Language) Standard with Archetype Technology. (Basu A Ed.) Official Organ of Indian Journal of Medial Informatics. 8:2014, 8:57-60. ISSN 0973-0379.

[23] Kalra D, Tapuria A, Austin T, & de Moor G. Quality requirements for EHR Archetypes. (Mantas J, Andersen SK, Mazzoleni MC, Blobel B, Quaglini S, & Moen A Eds.), Pisa, Italy: Quality of Life through Quality of Information - Proceedings of MIE2012, 2012, 180:48–52.

[24] van der Linden H, Austin T, Talmon J, Generic screen representations for future-proof systems, is it possible? There is more to a GUI than meets the eye, Comput Methods Programs Biomed. 2009, 95: 213–226.

[25] Sánchez-de-Madariaga R, Muoz A, Caceres J, Somolinos R, Pascual M, Martínez I, et al. ccML, a new mark-up language to improve ISO/EN 13606-based electronic health record extracts practical edition. Journal of the American Medical Informatics Association. 2012, 1-7. http://jamia.oxfordjournals.org/content/20/2/298. Accessed January 2015.

[26] Rajiv Mordani (ed). Common Annotations for the Java Platform. April 19, 2006. http://download.oracle.com/otn-pub/jcp/caj-1.0-fr-eval-oth-JSpec/com_annotations-1_0-fr-spec.pdf. Accessed March 2015.

[27] Lozano-Rub R, Pastor X and Lozano E. OWLing Clinical Data Repositories With the Ontology Web Language. JMIR Med Inform. 2014, 2(2):e14. doi:10.2196/medinform.3023 [33] http://www.postgresql.org/docs/9.2/static/auth-pg-hba-conf.html. Accessed January 2015.

[28] Safran C, Rind D, Citroen M, Bakker AR, Slack WV, and Bleich HL, Protection of confidentiality in the computer-based patient record. MD Computing. May-Jun 1995, 12(3):187–192.

[29] Austin T, Kalra D, Lea N, Patterson D, & Ingram D, Analysis of Clinical Record Data for Anticoagulation Management within an EHR System. The Open Medical Informatics Journal, 2009, 3:56–64.

[30] Directive 98/79/EC In vitro diagnostic medical devices. 27 October 1998.

[31] Vadher B, Patterson DLH, Leaning MS, Evaluation of a decision support system for initiation and control of oral anticoagulation in a randomised trial. BMJ 1997; 314: 1252–1256.

[32] Beale T, Heard S (eds). openEHR Architecture - Architecture Overview revision 1.1.1. 13 November 2008. http://www.openehr.org/releases/1.0.2/architecture/overview.pdf. Accessed March 2015.

[33] Health Level Seven International. A basic overview of CDA. http://www.hl7.org.uk/repository/uploads/565/1/A%20basic%20view%20of%20CDA%20v3.doc. Accessed March 2015.

[34] Calvillo-Arbizu J, Romn-Martnez I and Roa-Romero LM. Standardized access control mechanisms for protecting ISO 13606-based electronic health record systems. IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI) 1-4 June 2014, Valencia, Spain, 539-542, IEEE DOI: 10.1109/BHI.2014.6864421.