MEDINFO 2007
K. Kuhn et al. (Eds)
IOS Press, 2007

The *open*EHR Java Reference Implementation Project

Rong Chen[a], Gunnar Klein[b]

[a] *Department of Biomedical Engineering, Linköping University, Sweden*
[b] *Department of Medicine, Karolinska Institutet, Sweden*

Abstract

The openEHR foundation has developed an innovative design for interoperable and future-proof Electronic Health Record (EHR) systems based on a dual model approach with a stable reference information model complemented by archetypes for specific clinical purposes.

A team from Sweden has implemented all the stable specifications in the Java programming language and donated the source code to the openEHR foundation. It was adopted as the openEHR Java Reference Implementation in March 2005 and released under open source licenses. This encourages early EHR implementation projects around the world and a number of groups have already started to use this code.

The early Java implementation experience has also led to the publication of the openEHR Java Implementation Technology Specification. A number of design changes to the specifications and important minor corrections have been directly initiated by the implementation project over the last two years. The Java Implementation has been important for the validation and improvement of the openEHR design specifications and provides building blocks for future EHR systems.

Keywords:

Electronic Health Records, Health Information Systems, *open*EHR, Java, open source.

Introduction

The *open*EHR foundation, a not-for-profit organization has since 2004 published a series of design specifications [1] for semantically interoperable and future-proof Electronic Health Records (EHR) systems based on more than 15 years of research and development projects in the EU and Australia. The most distinctive feature of the *open*EHR design is that it separates the clinical concerns and the technical design of EHR systems by an innovative design approach called two-level modelling [2]. The first level model takes care of the technical concerns and deals with the information structure and data types with a small set of information model classes, upon which the core of EHR systems can be built. The second level of model handles the concerns of the clinical domains, which are about how to represent and communicate the semantics of the clinical content. Because of this separation of concerns, stable EHR systems can be built without knowledge about specific clinical content necessary in different fields. The specification of clinical content can be authored and amended later, shared between health organizations since it can be loaded into EHR systems at runtime. The result is highly adaptive EHR systems and vastly improved semantic interoperability [3]. Such systems are sustainable since they can evolve as the clinical requirements changes.

A group of developers from Sweden specialized in EHR systems became inspired by the design ideas behind *open*EHR and started initial exploration in 2004. This led to the initial Java implementation of the core *open*EHR specifications. The group was invited to release the software under open source license so the work could be reused and possible enhanced by other developers. The group favoured the idea and announced the availability of the Java code initially released under GPL license. Later the same year, the software was donated to the *open*EHR Foundation and the license changed to the less restrictive MPL/GPL/LPGL triple-license to be compatible with the *open*EHR Foundation's licensing model. The action was welcomed by the *open*EHR community and the software was formally adopted as the Reference Implementation in the Java programming language in 2005.

Materials and methods

The Java implementation is implemented with the Java programming language with the Java 5.0 platform to take advantage of the "generics" and other new language features. Several well-established open source libraries have been used, e.g. log4j and commons-lang from the Apache Software Foundation. The implementation does not make any assumption of where and how the components would be used, e.g. client or server side application. A principle of the Java implementation is to be as faithful as possible with the specification while keeping the Java look-and-feel. Being faithful with the original design specifications means that the complete semantics of the design should always be correctly implemented in the Java language to ensure interoperability with *open*EHR systems implemented in another programming language. It involves mapping between *open*EHR assumed data types and native Java types.

One of the difficulties was how to properly handle invariants checking in the class specifications. It is well-known

that Design by Contract is not natively supported by the Java platform although there are 3rd party libraries aiming to support it in Java. The same effect is achieved through the use of immutable Data Value object pattern with incoming parameters checking by the object constructors. Keeping the Java look-and-feel makes the software familiar to Java developers and involves use of the standard naming convention and common Java idioms. The details of the work have been documented and explained in the *open*EHR Java Implementation Technology Specification (ITS) [4] and the design document of the Java Implementation.

The software from the Java Reference Implementation project is released under open source licenses, namely Mozilla Public License (MPL), Gnu Public License (GPL) and Less Gnu Public License (LGPL). The users can chose any one of these three that suits them best. It is worth to mention that among these three, MPL is the least restrictive and GPL is the most prohibitive for commercial use. This means that the even commercial suppliers can use this software as part of their offerings without any commitment to *open*EHR. The software is freely accessible through Internet and all changes are public and accessible through the source repository. Software bug fixes and patches contributed by developers are verified and committed to the source repository.

A set of rules have been applied in the early stage of the project to guarantee the quality of the software. Unit testing is used extensively to make sure the implementation meets the intention of the design. It provides the extra value as further documentation beside the specification since it exercises the code in real use scenarios. Unit testing is usually integrated into the build script so that the test cases can be executed by anyone at any time. Since it can be automated as part of the build cycle and run often, unexpected errors can be caught when they occur. The development guidelines require that each bug fix in the production code should be accompanied with a minimum test case that demonstrates the bug before the fix and gets passed when the bug is fixed. This so called regression test provides a safety net to the code base to make sure the old bugs will not re-appear in the subsequent versions.

The software build is managed by Maven from the Apache Software Foundation. It provides comprehensive support of software projects and handles software library dependency particularly gracefully. Unit testing of the software is seamlessly integrated with build. A Continuous Integration tool called Continuum also from Apache is used to execute the Maven build script directly for monitoring the source code changes in the source repository and make automated software builds.

The Java implementation projection software is currently change controlled by a Subversion repository hosted on the *open*EHR site. Like CVS, Subversion is an open source product for managing software changes and favoured by many open source software projects around the world. The URL to reach the root directory of the Java project in Subversion is http://svn.openehr.og/ref_impl_java.

Results

Table 1 - Software artifacts provided by the Java Reference Implementation Project

Component Name	Implemented Specifications	Brief Description
openehr-rm	*open*EHR Reference Information Model – Common, Support, Data Types, Data Structures, Demographics, EHR	Base component that provides the Java implementation of all *open*EHR Reference Information Models
openehr-aom	*open*EHR Archetype Object Model	Object validation and construction
openehr-ap	*open*EHR Archetype Profile	Implementation of the domain data types
adl-parser	*open*EHR Archetype Definition Language	Translates ADL to AOM
adl-serializer	*open*EHR Archetype Definition Language	Serializes AOM to ADL
rm-builder	*open*EHR semantics	Reference Model objects builder

The software from the Java implementation is divided into several software components each with its own clearly defined responsibility. The openehr-rm, which stands for *open*EHR Reference Model (RM), is the base component that implements all Reference Information Model specifications, namely Data Types, Data Structures, Common, Support, EHR Extract, and Demographics. It maps *open*EHR assumed data types into Java native data types and implements other higher level data types. It also implements path based queries for finding leaf nodes in a large object tree with single crafted paths. All constructors of the RM classes have annotations for the parameters so that the invocation of object construction can be done automatically. This feature is used by the rm-builder for RM objects construction with Archetypes. This component is the fundamental building block for all subsequent components. According to the *open*EHR two-level modelling approach, the core of EHR systems can be built entirely based on the RM components, e.g. EHR data query and storage mechanism can be built on top of the RM component.

We also implemented the openehr-aom and openehr-ap, which stands for *open*EHR Archetype Object Model (AOM) and *open*EHR Archetype Profile, respectively. These components provides in-memory representation of Archetypes, which are usually authored and transmitted in Archetype Definition Language (ADL) format but parsed into AOM in-memory format to be used in runtime. They also provide support for object validation and creation on single archetype constraint level.

The rm-builder component is based on the openehr-rm, openehr-aom and openehr-ap components and provides Archetype based object validation and creation support. The rm-builder guarantees that the semantics in the archetypes are faithfully reflected in the process of RM objects validation and construction. The implementation of RM, AOM and the support for Archetypes based validation and construction is often referred to collectively as the *open*EHR Kernel.

The adl-parser component implements Archetype Definition Language and transforms Archetypes in ADL textual format into in-memory AOM form. It is built on a JavaCC, which generates the parser based on a grammar file. The grammar file is directly translated from the ADL specification with necessary changes. The adl-parser is the entry point for Archetypes into any EHR system so it is of vital importance for the correctness of the behaviour of an Archetype based system. It can also be used, e.g. in an Archetype editor, an authoring environment for Archetypes to check the correctness of the Archetypes in ADL format.

The adl-serializer provides conversion of Archetypes from in-memory AOM form to textual representation in ADL format, the reverse of what the adl-parser does. It is often used before an authored archetype can be stored and transmitted between systems.

Java implementation technology specification

The Java implementation experience has been summarized and written up as the Java Implementation Specification (ITS) of *open*EHR design, [4] which is now included as part of the official release of the *open*EHR specifications. The document is still in beta version and being updated to reflect the feedback from the implementation activities. The purpose of this ITS is to provide guidance on perspective implementation of the *open*EHR design on the Java platform and make sure the semantics of the design is kept faithful in the Java programming environment. More importantly, it should also ensure that *open*EHR systems from different computing platforms are interoperable.

During the last two years, a number of design improvements have been directly initiated from the Java implementation project and indirectly from other development projects that are based on the Java reference implementation components. A number of additional minor issues, e.g. inconsistency in the documentation have also been raised, particularly during the first stage of the implementation when the specifications were just released before version 1.0.

Table 2 - Selection of specification changes accepted by openEHR due to the Java implementation

Related Specification	Description
Data Structures	EVENT should inherit from LOCATABLE.
Data Types	Date/Time classes improvements
Archetype Object Model	Missing adl_version in ARCHETYPE
Archetype Object Model	Table for missed class ASSERTION_VARIABLE added. Assumed_value assertions corrected; standard_representation function corrected
*open*EHR Archetype Profile	Fix invariants in C_QUANTITYclasses. Correct C_QUANTITY.property to CODE_PHRASE. Correct invariants for C_CODED_TEXT; correct inheritance for C_DV_ORDERED. Corrected C_QUANTITY_ITEM class. Corrected errors in DV_STATE model by adding 2 new classes
Most of the models	Use constants instead of literals to refer to terminology in RM, mainly used in the invariants part of the class descriptions
Common, EHR	Make DIRECTORY reusable, add new directory package
Demographic	Remove details /= Void invariant from PARTY
Archetype Definition Language	Numerous clarifications of the ADL syntax and semantics

Some design issues were not obvious initially but became apparent when more implementation experiences were gained. For instance, the modelling of EHR package initially included support for a directory based way of organizing records. When the requirement was identified to support a good way of organizing demographic objects, it seemed natural to extract the generic directory support into a more generic package so it can be re-used for organizing objection other than EHR compositions.

Sometimes, implementation in the programming language other than the one used or envisioned by the original designer can lead to improvement of the design itself. One example of this is within invariants checking, present in many places of the specifications, built-in terminology names are referenced in plain text format. This is recognized as an anti-pattern among Java developers since such way of handling textual constants would lead to more brit-

tle software since any deviation, e.g. a typo of the terminology name will not be caught by the compiler during software compilation phase but only show up during system runtime. A better way of doing that is to introduce a type safe enumeration pattern. This design improvement has been well taken and incorporated into Release 1.0.1.

Building blocks for EHR applications

The original Swedish team has built a pilot Child Leukaemia Treatment management application using the Java components for Karolinska University hospital in Stockholm. It is a web-based system designed to facilitate chemotherapy management for child Lymphoblastic Leukaemia patients in maintenance stage. Unlike conventional EHR applications, the clinical content is defined by archetypes and the screens are generated dynamically.

The Medical Informatics group from Linköping University has built a number of applications based on the Reference Java Implementation and made substantial contributions to the Java project. The Java Archetype Editor [5] has particularly received good comments and is now released under open source license. A web-based patient portal application has been built as part of a master student's thesis work. An ongoing EHR visualization project [6] uses the Java implementation as a way of representing EHR data.

A Biomedical Engineering Group from Uruguay funded by the Ministry of Education is building an Intensive Care Unit application, which is based on the existing Java components. The system covers admission to ICU, clinical variables monitoring, laboratory requests and tracking, instructions on medication, nutrition, fluids and tracking of instructions status, clinical logbook for each patient. The system will be installed for a one year field test in three ICUs in Uruguay during 2007.

In the Netherlands, a team of developers is building a commercial solution to support elderly care based on the openEHR Java implementation. The system supports a wide range of users from General Practitioners (GP), nurses, to elderly / disabled people and other involved heath professionals. The solution is particularly favoured by the customers since it is based on public specifications and open source software.

A research team from the Health Informatics Group, Central Queensland University, Australia is building a new version of the Archetype Finder that uses the Java ADL parser.

Since the software was released as open source in 2004, it has been constantly updated and maintained by the Java Reference Implementation team. Developers around the world have been giving feedback, bug reports and software patches resulting in overall increasing quality of the code base over time. The initial release was based on the specification release 0.95. During the last six months, the team has spent considerable effort to update the implementation towards the latest specification 1.0.1. At the same time, the code base has been re-organized to facilitate re-use and improve testability even further.

Discussion

Nowadays health information standards are becoming increasingly complicated. This makes it very difficult to rely on a textual document only for the description of the semantics of the design. If the intention of the standard is misinterpreted, the risk of creating non-interoperable software is very high. Unfortunately the formal standardization bodies CEN and ISO have not yet allowed publication of standards artifacts in forms other than human readable documents. Data structures intended for use by software development tools can sometimes be distributed as informative associated material but not as the core of the standard. Open source reference implementations can be very useful to ensure a common and complete understanding of a written specification facilitated by the fact that such software like ours is usually developed in close collaboration between the specification designers and the implementers. The software can be more detailed and precise than the textual specification and be easily understood and put to production by target developers.

The fact that a particular implementation is done on a specific platform means not only that it proves the specification is possible to implement but also that the implementation details have been worked out for that specific platform. There are a large number of similar examples in the IT industry. The Java specifications are released together with an open source reference implementation sometimes from Sun itself but also from open source organizations like Apache.

Innovative ideas need to be experimented with and verified before they become part of a formal specification. The IT industry has seen too many “committee-made” design specifications, which can be very hard to implement and truly annoying to work with if they unfortunately become formal standards and last for years. So it is really important to get some real implementation experience and let the feedback from that guide the refinement of the design specifications. Because open source implementations are readily accessible by large user base and perspective implementers, they are particularly suitable for validating new design ideas. The *open*EHR community, which consists of both clinical professionals from many specialities, technical developers and also medical informatics researchers, is a particularly good example of this. With experience from international standardization activities and open source software projects, the authors argue that an open source reference implementation is an ideal way of ensuring the quality of international standards for communication.

The open source licenses used by the Java Implementation Project are very relaxed, which means that they allow commercial applications to be built on top of these components. These licenses effectively remove entry barriers for small companies and academic researchers that may not have the resources to implement the design from scratch. If more companies choose to use this solution, it will undoubtedly increase the interoperability between EHR systems. Because of the availability of the open source

implementation, more medical informatics research activities could be performed in connection with EHR data e.g. Decision Support and Epidemiological statistics. The ground breaking design proposed by *open*EHR opens a lot of opportunities, which were not possible previously due to poor semantic interoperability between systems and brittle system design. One well-known problem in Decision Support Systems (DSS) is that the data query mechanism is dependent on the information model of the local EHR system and has to be adapted each time the DSS is integrated with a new EHR system. This issue is effectively resolved if *open*EHR two-level models are used since DSS can query an EHR system with a standardized reference information model with the aid of clinical content definitions explicitly expressed in Archetypes. These research areas are of high interest at the moment, and collaboration on these using the open source implementations are ongoing.

The open source implementation creates synergies between EHR system developers for collaboration on common EHR software components. More access to end user base and beta testers, and focused development efforts on common parts of EHR systems will most likely result in more reliable and re-usable software components that are building blocks of future EHR systems. With enough core building blocks, a common health information platform can be established, which provides essential features for any full-blown EHR system, e.g. standards compliant interfaces and semantic interoperability.

Even the application level presentation layer can benefit from generic screen form generation components [7]. EHR applications could then be built rapidly on top of the common platform with clinically defined archetypes and certain application level customizations in ways previously unimagined. Also the EHR application development can focus on solving application related problems rather than generic EHR problems. We believe this would result in a paradigm shift in how to develop EHR applications in the near future, which would dramatically increase the productivity of EHR development and therefore lead to more sustainable solutions [8] than the current way of building EHR systems.

Conclusion

The *open*EHR Java Reference Implementation project has been very useful for validation and improvement of the *open*EHR design specifications. The Open Source Software development model used by the project fits very well with the purpose of publicly available specifications. This is a sustainable way of providing major healthcare standards for interoperability that we argue should be used more.

The project creates synergies between EHR developers and encourages software re-use aiming to provide building blocks for EHR applications. Not only would this reduce costs of the software and lead to more reliable health information systems, but also improved interoperability between systems. We believe this can contribute to the overall sustainability of health information systems.

Acknowledgements

We want to firstly thank the *open*EHR Foundation for their pioneering work and Thomas Beale in particular for his invaluable guidance on the implementation work and review of the manuscript. We also want to express our appreciation of the UCL team for good collaboration and Yin Su Lim in particular for her meticulous efforts, the Linköping Medical Informatics group. and last but not least, all the early *open*EHR implementers who have tested the Java components and came back with feedback and patches. We are also grateful for the financial support for some of this work from the Semantic Mining EU- project and from Cambio Healthcare Systems.

References

[1] Beale T, Heard S, Kalra D, Lloyd D. Architecture Overview. http://svn.openehr.org/specification/BRANCHES/Release-1.1-candidate/publishing/architecture/overview.pdf [cited 2006 December 03]

[2] Beale T. Archetypes: Constraint-based domain models for future-proof information systems. In: Eleventh OOPSLA Workshop on Behavioral Semantics

[3] Garde S, Knaup P, Hovenga E, Heard S. Towards Semantic Interoperability for Electronic Health Records: Domain Knowledge Governance for openEHR Archetypes. Accepted for Methods of Information in Medicine 2007

[4] Chen R. openEHR Reference Model Java ITS. http://svn.openehr.org/specification/BRANCHES/Release-1.1-candidate/publishing/its/Java/openEHR-JavaITS.pdf [cited 2006 December 03]

[5] Sundvall E, Qamar R, Nyström M, Forss M, Petersson H, _hlfeldt H, A Rector. Integration of Tools for Binding Archetypes to SNOMED CT. Proceedings of SMCS2006; 1-3 Oct, Copenhagen, Denmark, p 64-68

[6] Sundvall E, Nyström M, Forss M, Chen R, Petersson H, _hlfeldt H. Graphical Overview and Navigation of Electronic Health Records in a prototyping environment using Google Earth and openEHR/Archetypes. Accepted by Medinfo 2007

[7] H van der Linden, Schuler T, Chen R, Talmon J. Generic screen representations for future proof systems, is it possible? Accepted by Medinfo 2007

[8] Garde S, Hullin CM, Chen R, Leslie H, Heard S, Schuler H, Gränz J, Knaup P, Hovenga E. Towards Sustainability of Health Information Systems: How Can We Define, Measure and Achieve It. Accepted by Medinfo 2007

Address for Correspondence

Rong Chen,
Dept. of Biomedical Engineering,
Linköping University,
SE-581 85 Linköping,
Sweden.
E-mail: rong.chen@imt.liu.se