eHealth Beyond the Horizon – Get IT There
S.K. Andersen et al. (Eds.)
IOS Press, 2008

An Archetype-based Testing Framework

Rong CHEN[a,b,1], Sebastian GARDE[c], Thomas BEALE[c], Mikael NYSTRÖM[a], Daniel KARLSSON[a], Gunnar O. KLEIN[d], Hans ÅHLFELDT[a]

[a] *Department of Biomedical Engineering, Linköping University, Sweden*
[b] *Cambio Healthcare Systems, Sweden*
[c] *Ocean Informatics UK, London*
[d] *Karolinska Institutet, Sweden*

Abstract. With the introduction of EHR two-level modelling and archetype methodologies pioneered by openEHR and standardized by CEN/ISO, we are one step closer to semantic interoperability and future-proof adaptive healthcare information systems. Along with the opportunities, there are also challenges. Archetypes provide the full semantics of EHR data explicitly to surrounding systems in a platform-independent way, yet it is up to the receiving system to interpret the semantics and process the data accordingly. In this paper we propose a design of an archetype-based platform-independent testing framework for validating implementations of the openEHR archetype formalism as a means of improving quality and interoperability of EHRs.

Keywords. Computerized Medical Record Systems, EPR-CPR-EMR, Standards, Knowledge-based systems, Archetypes, openEHR, EHR, Testing, Semantic Interoperability

Background

The innovation of two-level modelling of Electronic Health Records (EHRs) and archetypes [1] pioneered by openEHR [2] and standardized by CEN/ISO [3] brings us one step further towards semantic interoperability of EHRs [4]. Instead of being hard-coded into proprietary software by software developers, clinical content models are expressed in the Archetype Definition Language (ADL) [5] and authored by the clinical professionals themselves. Archetypes are used at runtime by EHR systems to validate user data entry and query fine-grained data in the EHR. Archetype-based EHR systems are highly adaptive and can evolve when clinical requirements change over time since volatile clinical requirements are captured in archetypes while software systems are built using only the stable openEHR information model and archetype language. Archetypes are expressed in a standardized formal language so they are machine-interpretable and can be shared between systems. This makes the semantics of EHR data available not only to other EHR systems but also to surrounding systems.

[1] Corresponding Author: Rong Chen, Department of Biomedical Engineering, Linköping University, Sweden; E-mail: rong.chen@imt.liu.se

Archetypes can be considered similar to software written in a declarative programming language. They define what should be recorded in an information repository, without defining procedural semantics. Both the validation and initial creation functions can be realized by a software component based on the Reference Model (RM) and Archetype Object Model known in openEHR as a 'kernel', using the semantics defined in the archetype formalism. In an archetype-based system, we can consider the archetypes and the kernel together as the entity providing the information semantics of the system. Thus, it is desirable to develop unit tests for each archetype that is developed according to its domain model together with the kernel to use it.

Since a 'kernel' is crucial to any archetype-based system, any defects or misinterpretation of the specifications in the implementations could impede archetype-based data processing and data exchange between systems. It is therefore important to validate the kernel implementations to ensure archetype semantics are carried out correctly in different implementations across platforms. Traditional hand-crafted unit test code in its classic form is not useful, since it is implementation- and platform-specific. Additionally, since archetypes are shareable and platform-independent, it seems reasonable to have a platform-independent way of testing archetypes and archetype-processing software. This leads to the following requirements of a testing framework for archetype-based systems:

1) Tests are platform-independent and implementation-independent;
2) Tests are archetype-based and preferably auto-generated.

This paper presents a platform-independent archetype-based testing framework that fulfils these requirements.

1. Methods

In software engineering, unit testing is a procedure used to validate that individual units of software are working properly. It creates a 'safety net' for the software when it evolves throughout its lifespan. We used the design patterns from Beck's unit testing framework [6] and applied it to archetypes system in a platform-independent way.

2. Results

2.1. Design overview

The core requirement for this design is to find a platform-independent way for testing different kernel implementations of archetype-based systems. The scope of the test should cover the core functions of the kernel component, initially including archetype based data creation, validation and path based querying. The testing framework consists of three logical parts (Figure 1).

- The first part is test fixture representation and generation. A test fixture refers to a fixed state in order to run a test and expect a particular outcome. In the context of testing archetype-based systems, a test fixture is equivalent to a state of a reference model object.

- The second part is test case representation and generation. A test case is a unit of testing logic for a specific test scenario. It includes references to test fixtures, the kernel operation to test and the expected result from invocation.
- A test runner can load test cases and test fixtures, execute the specified kernel operation against a particular kernel implementation with given input values and compare the returned value with the expected result to decide whether the test has passed or not.

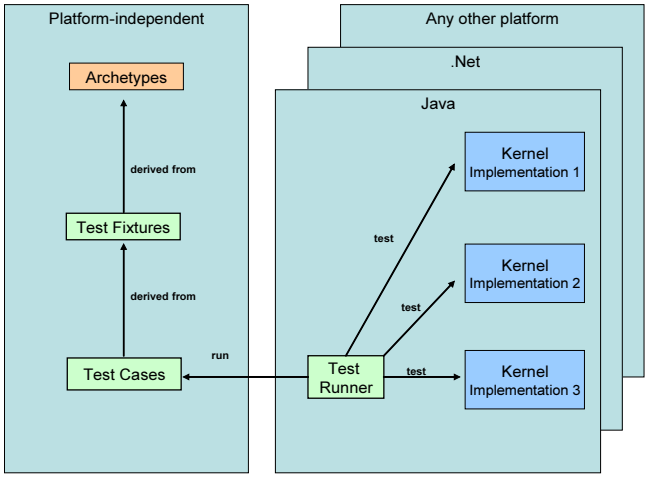

Figure 1. System diagram of archetype-based testing framework

2.2. Test fixtures

One requirement on the design of this testing framework is that it should be platform-independent so that it can be used to validate different openEHR kernel implementations on different platforms. This requires the representation format of test fixtures to be platform-impendent. We considered using XML for this, but due to the ambiguities in handling common container data structures like List and Hash, the data Archetype Definition Language (dADL) was used instead. Since dADL is also used to represent archetypes, it is expected to be supported by all archetype-based systems.

One of the difficulties of archetype-based testing is that the number of possible states of RM that are valid according to an archetype can be so large that it is close to impossible to test all combinations. A good strategy of generating test fixtures is needed to have good coverage of valid states of RM and at same time not yield too many fixtures. A reasonable set of test data objects should at least cover typical data that could occur in a production EHR context, as well as some of the likely exceptions.

One design idea is to automatically generate fixtures based on pre-defined rules. Rules can be specific to any type of constraint or object node. Constraints in archetypes can be divided into two categories: 1) object constraints, which are to do with object occurrences and cardinality etc; 2) leaf constraints i.e. constraints on primitive data types such as Integer and String. Test fixtures can be generated by examining constraints on each object node level. For example, one rule can exclude any optional attributes while another can include them. On the leaf constraint level, valid values can be selected by inspecting the constraints. For example, for the integer range "1..10" constraint, a specific rule can include both boundary values and a random value in the

range. The total number of auto-generated fixtures can grow rapidly depending on selected variations and the depth of the object tree. By choosing a good strategy using appropriate rules, it is possible to get a good balance between manageable size of the test fixture and representativeness of the data. It is possible to calculate the total number of fixtures to be generated using a specific set of rules in advance. This gives the possibility to tailor the rules so that a suitable amount of fixtures can be obtained. Another design idea is to combine both manually selected special instances and auto-generated fixtures. This way, test fixtures that are known or suspected to cause errors can be included even if they are excluded by auto-generation rules.

2.3. Test cases

Each test case includes specifications for: 1) the kernel operation to test; 2) the input values to the kernel operation if required; 3) the expected return result from the kernel operation. Both the input values and the expected result can be a reference to a test fixture if a complex object instance is required. In order to test the kernel thoroughly, it is necessary to include not only test cases that expect the kernel operation to return with a result but also those that expect the kernel operation to abort with an error message. Since test fixture generation is based on archetypes, it is reasonable to group test cases around archetypes. It also makes sense to automate the generation of test specifications using existing text fixtures.

Creation of test cases for archetype based data creation can be automated by inspecting test fixtures and computing the input values. For a given reference model object, one test case can be created to test if the same object can be created by the kernel using computed input values and several other test cases can be created with deliberately invalid values to cause a data creation failure from the kernel with anticipated error messages. To generate test cases for data validation, it is possible to use a valid test fixture as is and expect the kernel validation function to return a positive answer. To create a test case that causes a negative result from the kernel, the test fixture needs to be modified to make it invalid. This can be achieved by modifying the object tree in such a way that it violates certain constraints. Path-based query test cases can be generated by extracting all valid paths from a given fixture and then querying the fixture using a reference kernel implementation to obtain the correct values used as the expected results in the test case.

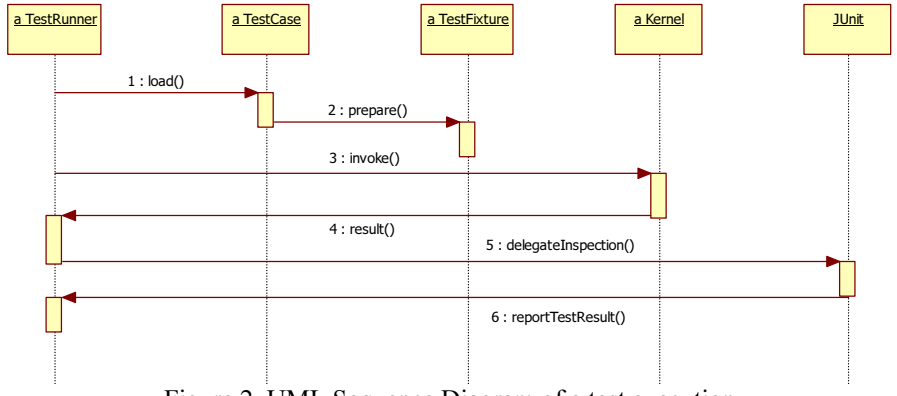

Figure 2. UML Sequence Diagram of a test execution

2.4. Test runners

A test runner can load test fixtures and use test cases as specifications to carry out actual tests against an individual kernel implementation. A typical use case is as follows: a test runner takes a test case and loads according test fixtures with specified input data and expected result. Then it invokes the specified kernel operation according to the test case, takes the output from the kernel and compares it with the specified result to decide whether the test passes or not (Figure 2). The result inspection part can usually be delegated to the underlying unit testing framework, e.g. JUnit[2]. The parsing of test fixtures is done by a dADL parser, which usually exists for an openEHR system. Because a test runner needs to interact with a kernel component through direct method invocations, it needs to be platform specific. But if a kernel implementation can be accessed in a platform-independent way through for example Web Services, even this part of the testing framework can be made platform-independent.

2.5. Implementation

We verified this design by prototyping it using available open source components from the openEHR Java Reference Implementation project [7]. Test fixtures and test cases (Listing 1) are initially manually created based on the blood pressure archetype[3]. We plan to add support for archetype-based automated generation of test fixtures and test cases later on. A Java test runner will be made available via the openEHR Java project[4].

Listing 1. Test case specification in XML for testing path based query

```
<PathQueryTestCase>
   <DataInstance>blood_pressure-001.dadl</DataInstance>
   <Test>
      <Path>/observation/data/events[1]/data/items[1]/value/magnitude</Path>
      <Expected>120.0</Expected>
   </Test>
   <Test>
      <Path>/observation/data/events[1]/data/items[2]/value/magnitude</Path>
      <Expected>90.0</Expected>
   </Test>
</PathQueryTestCase>
```

3. Discussion

Compared with traditional testing frameworks, the key innovation of this design is that the test fixtures and test cases are archetype-based and therefore independent of any particular archetype kernel implementation on any computing platform. Because archetypes are machine-interpretable, it is possible to automate the generation of test fixtures and test cases based on pre-defined rules.

One important aspect of system interoperability is to reduce ambiguity of data semantics. The archetype formalism has delivered on providing the semantics to EHR

[2] http://junit.org

[3] http://www.openehr.org/svn/knowledge/archetypes/dev/adl/openehr/ehr/entry/observation/openEHR-EHR-OBSERVATION.blood_pressure.v1.adl

[4] http://www.openehr.org/projects/java.html

and surrounding systems. But it is still up to actual implementations to interpret the archetype semantics and apply them during EHR processing. Differences between different archetype formalism implementations pose a potential threat to EHR interoperability. The archetype-based testing framework we propose in this paper can be used for validating different archetype implementations to help ensure interoperability between different systems and platforms.

With the introduction of EHR two-level modelling and archetype methodology, we are entering a new era of developing health information systems. Such systems are more open, adaptive and collaborative than ever before. They are based on standardized information models, shared clinical content models and open sourced core components. This will not only increase the total productivity in the field of health information systems, but also enable us to develop more advanced clinical systems using the semantics intended by the original clinical content models. We believe that EHRs will be central to this new paradigm and their semantics, expressed as archetypes and their interpreters, the kernels, will be of vital importance to this. The correctness of the archetype implementation in different archetype-based systems is crucial to the quality and interoperability of EHRs. An archetype-based platform-independent testing framework is a first attempt to address this issue.

The testing framework is extendable. When the kernel functionality evolves, support for new kernel operations can be added. This can be achieved by adding new operation types in the test case expression format and mapping it to the new kernel function invocations within the test runner. As a next step, we will explore the selection of the best strategy for archetype-based generation of test fixtures and test cases. It is desirable to have an open source implementation of a test fixture generation tool, a test case authoring tool and one test runner for each major computing platform.

4. Conclusion

The platform-independent archetype-based testing framework we propose here is potentially useful for validating different archetype formalism implementations as it can ensure that the semantics of archetypes are interpreted correctly and utilized faithfully across different systems and platforms. Thus this work contributes to the aim of semantic interoperability of EHRs.

Acknowledgements

This work was kindly supported by Cambio Healthcare Systems, Sweden.

References

[1] Beale T, Heard S. Archetype Principles in The openEHR Foundation Release 0.95. In: Beale T and Heard S, eds. The openEHR foundation, 2005
[2] Kalra, D., T. Beale, et al. (2005). The openEHR Foundation. Stud Health Technol Inform 115: 153-73.
[3] EN 13606-1:2007 Health informatics - Electronic health record communication - Part 1: Reference model, CEN - European Committee for Standardization. Also published as ISO 13606-1:2008, ISO-International Organization for Standardization.
[4] Garde, S., P. Knaup, et al. (2007). Towards semantic interoperability for electronic health records. Methods Inf Med 46(3): 332-43.
[5] Beale T, Heard S. Archetype Object Model. (2006), The openEHR Foundation.
[6] Beck, K. Simple Smalltalk Testing: With Patterns, http://www.xprogramming.com/testfram.htm, [cited 2007 November 12]
[7] Chen, R. and G. Klein (2007). The openEHR Java reference implementation project. In Proceedings of Medinfo 2007, Stud Health Technol Inform 129: 58-62.