Exploring Complexity in Health: An Interdisciplinary Systems Approach
A. Hoerbst et al. (Eds.)

doi:10.3233/978-1-61499-678-1-379

A Conformance Test Suite for Arden Syntax Compilers and Interpreters

Klaus-Hendrik WOLF[a,1] and Mike KLIMEK[a]
[a]*Peter L. Reichertz Institute for Medical Informatics University of Braunschweig – Institute of Technology and Hannover Medical School*

Abstract. The Arden Syntax for Medical Logic Modules is a standardized and well-established programming language to represent medical knowledge. To test the compliance level of existing compilers and interpreters no public test suite exists. This paper presents the research to transform the specification into a set of unit tests, represented in JUnit. It further reports on the utilization of the test suite testing four different Arden Syntax processors. The presented and compared results reveal the status conformance of the tested processors. How test driven development of Arden Syntax processors can help increasing the compliance with the standard is described with two examples. In the end some considerations how an open source test suite can improve the development and distribution of the Arden Syntax are presented.

Keywords. Arden Syntax, knowledge representation, test driven development, HL7 standards, decision support system

1. Introduction

The Arden Syntax for Medical Logic Modules (MLM) is a well-established programming language to represent medical knowledge [1]. Health Level 7 (HL7) and the American National Standard Institute (ANSI) have standardized it [2]. Early versions date back to 1990 [3] and the language has been updated through the years to the now current version 2.10. Since version 2.9 Arden Syntax includes language constructs to represent fuzzy logic. Over the years a variety of interpreters and compilers has been developed. Examples include EGADSS [4], Arden/J [5], Medexter's ArdenSuite [6], and Arden2ByteCode [7]. Besides these stand-alone processors, some clinical information systems include the ability to interpret Arden MLMs. Among these systems are Siemens Soarian, AGFA Orbis, Health VISION. In the literature one can find further reports on self-developed systems [8,9,10,11].

After developing the Arden2ByteCode compiler, the authors have often been asked to what extend the compiler covers the specification. While this seems to be an easy question, the answer is difficult and biased. We knew of some syntax features that we willfully omitted in the implementation. For these the answer is easy. For those parts we intended to cover, the process of translating the specification to a compiler can introduce errors that are difficult for the implementer to discover (as we now know).

[1] Corresponding Author: klaus-hendrik.wolf@plri.de

The aim of the presented research is to develop a test suite to test Arden Syntax compilers and interpreters (in the following called Arden processors) against the specification.

To achieve this goal we tried to answer the following questions:

1. How can we translate the Arden Syntax specification to a set of tests?
2. What is a good representation for these tests to test different Arden processors?
3. How do different Arden processors perform when tested?
4. Does the test suite help in the development of Arden processors?

2. Methods and Material

First, we had to decide on the version of the Arden Syntax that we use for the development of our test suite. Based on the available compilers and interpreters that we wanted to test, we chose the version 2.5. With some limitations, the test suite can test Arden processors implementing newer versions of the specification, since the Arden Syntax is mostly backward compatible.

In the first step, we analyzed the specification sentence by sentence to find statements that we can transfer into test cases. We used all parts of the text to generate test. These include the descriptions, stated special cases and given examples.

We decided to use the widely used JUnit testing framework developed for Java to represent the test cases formally. To apply the tests to different Arden processors we constructed Java-interfaces that are easy to adapt to new Arden processors.

With the tests represented in the developed test suite, we needed interfaces to compilers to test them. At first, we tested the Arden2ByteCode compiler, the second contestant was the openly available EGADSS compiler. Furthermore, we build an interface to a commercially available compiler. The author of an experimental PHP-based compiler-interpreter (ARSEN/IC) realized an interface to test his Arden processor [11].

Based on the results from the tests we tried to improve our open source compiler Arden2ByteCode to pass more tests. So did the author of ARSEN/IC.

3. Results

3.1. Generation of the test cases and the test framework

The detailed analysis of version 2.5 of the Arden Syntax specification resulted in a list of 161 test cases. To enhance the reproducibility we documented the exact section, each test case was generated from.

We transformed each test case to a unit test using the JUnit framework. We published the result of this process together with our Arden2ByteCode compiler on GitHub [12].

The repository contains a complete framework that is instantiated for the Arden2ByteCode compiler, but extending it to other compilers is as easy as possible. Anyone who wants to test a given Arden compiler is welcome to use it. We had this extensibility in mind when we designed the framework.

3.2. Applying the tests to different Arden compilers

The first contestant for the test framework was the Arden2ByteCode compiler. The first run of the binary release version from 2012-10-02 took 2.5 seconds. Of the 161 tests 33 generated errors. The section 3.3 further describes the investigation of the bugs and some of the fixes we applied to the compiler consequently.

The second Arden processor we implemented the interface for was EGADSS. Due to the processors design solely based only on CDA documents instead of return values, the implementation was a little more difficult. When it finally worked, 141 of 161 tests failed for the EGADSS version dating to 2013-04-17. Among the failing test cases are simple additions like 5 + 5 returning 5, the if-statement, that does not check whether the supplied variable is true, but simply whether it exists, and not working assignments in the data slot. Due to this result we tested how many of the test did at least compile. We found that the compiler successfully compiles 34 out of the 161 test sets.

The third processor we tested was a commercial installation. Of the 161 tests only 22 fail. Five of those fail, because the processor does not support language versions before 2.1, and because language versions after 2.5 are in some cases more permissive.

The fourth contestant was a web-based interpreter written in PHP. This one is different, not only on a design basis, but as it was intended for experiments with the Arden Syntax and not to be fully standard compliant. Therefore, it comes as no surprise that in the first run 45 of the 161 tests failed.

3.3. Using the test results to improve Arden processors

Given the test results for our Arden2ByteCode compiler, we tried to improve its implementation to pass more tests and thereby to better comply with the standard.

We traced many of the failing tests back to the quite complicated handling of primary times. The primary time is an attribute that all variables in Arden Syntax independent of their containing type have. As an example, given an evenly sized set of values, the Arden Syntax standard wants the median's primary time to be equal to the primary time of the latest of the two values in the middle, if they have the same value. A more severe bug was that, due to a copy and paste error in the code, the operators LATEST…FROM and EARLIEST…FROM returned the same result.

Once we knew where the compiler had flaws, it was quite easy to address these issues. After some coding only 14 out of the 161 tests fail. Due to the project's open source nature, the problems and their fixes are documented in the projects changelog on GitHub [11]. Most of the remaining issues are related to the CALL, INCLUDE, and EVOKE-statements that Arden2ByteCode does not fully support due to its design considerations.

The author of ARSEN/IC used the tests results to improve his interpreter as well. Talking to him he stated that the test suite helped him a lot to identify weak spots in his processor and having the ability to analyze the failing tests, change the processor, rerun the test and see the results is quite a game changer. A first debugging session correctly identified those language constructs that deliberately deviate from the standard, and identified two yet unknown bugs where the priority and urgency slots in the knowledge category accepted invalid values. The test suite will become an integral part of the further development of this Arden Syntax implementation.

4. Discussion

4.1. Generation of test cases and test framework

Like the implementation of a compiler, the interpretation of the specification to generate test cases is an activity that a human has to perform and that can result in mistakes. While the process to formulate test is far easier than generating a parser, we cannot guarantee that the produced test suite covers the entire standard and that it does so correctly. On the other hand, it has been easy to match all failing tests to sections of the specification. We double-checked the tests the contesting Arden processors failed and confirmed that these tests conform to our interpretation of the standard.

While mapping the defining text and the examples from the standard, some inconsistencies in the specification arose again. The process itself was quite straightforward so that we do not expect too many mistakes.

We achieved the goal creating a test suite that can be applied to a variety of Arden processors. As a demonstration, we interfaced and tested four different Arden processors. Basing the test framework on the widespread and mature JUnit proved to be a practical decision. To test a new compiler one only has to implement two methods to interface to this new compiler. Due to the operating system independent Java environment, the tests can run on a wide variety of operating systems, reaching many Arden processors. We demonstrated this with the testing of four different processors.

4.2. Performance of different Arden processors

Our tests of different Arden processors showed quite different, yet interesting results. Though none of the tested processor passes all tests, the span is huge. From the EGADSS compiler that fails to run almost all of the tests (141 of 161) to the commercial compiler that passes all but 22 tests of minor importance.

4.3. Supporting the development of Arden processors

Seeing that our compilers fails in 33 of the tests stimulated the development and soon after the test results the compiler successfully runs 19 more tests, making it more conformant to the standard.

The same holds true for ARSEN/IC that was never intended to be fully standard conformant, but improved a lot by being able to easily and reproducibly assess the problems.

Adapting the framework to support a new compiler is quite easy. Only two Java-functions have to be adapted. Running the test suite on an existing compiler is a matter of seconds and results in an immediate measure of its standard conformance. We supplied the test results and the test suite to the producers of the tested Arden processors and hope that they will be able to improve their tools with the results and the test suite as well.

Anyone, who is interested in testing an Arden processor, can download the full test suite from our GitHub page. We hope that the test suit will help others who have implemented or will implement Arden processors. We have to mention that one does not have to be the producer of the Arden processor to test it.

4.4. Supporting the development of standard

We could show that the test suite we have primarily developed to test the conformance of our compiler to the standard can be used to test and improve different Arden processors. Given that the test suite is itself a translation of the standard to a representation that is easy to read and understand, we hope that it will be helpful to improve the standard itself. There are some inconsistencies in the standard i.e. between text and examples.

As with the Arden2ByteCode compiler, we publish the test suite open source. This time we have chosen the business friendly BSD license. It would be nice to have an officially acknowledged test suite for the Arden Syntax that anyone could use to measure the compliance of an implementation to the standard. The use of more standard conforming implementations would reduce the cost of transferring MLMs from one installation to the other and thereby help the intention of the standard to foster the transfer of medical knowledge.

Acknowledgements

We thank Stefan Kraus, the author of ARSEN/IC, for implementing the interface to his Arden processor, testing it and supplying feedback concerning his experiences with the test suite.

References

[1] G. Hripcsak, P. Ludemann, T.A. Pryor, O.B. Wigertz, P.D. Clayton, Rationale for the arden syntax, *Comput. Biomed. Res.* **27** (4) (1994), 291–324.
[2] Health Level Seven, Arden Syntax for Medical Logic Systems, Version 2.5, 2005.
[3] G. Hripcsak, P.D. Clayton, T.A. Pryor, P. Haug, O.B. Wigertz, J.V.der Lei, The arden syntax for medical logic modules, *in: Proc. Annu. Symp. Comput. Appl. Med. Care,* (1990), 200–204.
[4] G. McCallum, EGADSS: A Clinical Decision Support System For Use in a Service-oriented Architecture, Master of Science, Computer Science, University of Victoria, University of Victoria, Dept. of Computer Science, (2006), http://dspace.library.uvic.ca/handle/1828/2296.
[5] H.C. Karadimas, C. Chailloleau, F. Hemery, J. Simonnet, E. Lepage,Arden/J: an architecture for MLM execution on the java platform, *J. Am. Med. Inform. Assoc.* **9**(4) (2002), 359–368.
[6] M. Samwald, K. Fehre, J. de Bruin, K.P. Adlassnig. The Arden Syntax standard for clinical decision support: experiences and directions. *J Biomed Inform*. **45**(4) (2012), 711-8.
[7] M. Gietzelt, U. Goltz, D. Grunwald, M. Lochau, M. Marschollek, B. Song, K.-H. Wolf, Arden2ByteCode: A one-pass Arden Syntax compiler for service-oriented decision support systems based on the OSGi platform. *Comput Methods Programs Biomed.* **106**(2) (2012), 114-25.
[8] R.A. Kuhn, R.S. Reider, A C++ framework for developing medical logic modules and an arden syntax compiler, *Comput.Biol. Med.* **24**(5) (1994) 365–370.
[9] P. Ludemann, Mid-term report on the arden syntax in a clinical event monitor, *Comput. Biol. Med.* **24**(5) (1994) 377–383.
[10] S. Kraus, C. Drescher, M. Sedlmayr, I. Castellanos, H. U. Prokosch, D. Toddenroth. Using Arden Syntax for the creation of a multi-patient surveillance dashboard. *Artif Intell. Med.* (2015) pii: S0933-3657(15)00126-8.
[11] S. Kraus, M. Enders, H. U. Prokosch, I. Castellanos, R. Lenz, M Sedlmayr. Accessing complex patient data from Arden Syntax Medical Logic Modules. *Artif Intell Med.* (2015) Sep 12. pii: S0933-3657(15).
[12] M. Klimek and K.-H. Wolf, JUnit-based test suite for the Arden Syntax version 2.5. (2016) *source code, available from:* https://github.com/PLRI/arden2bytecode/tree/master/test/arden/tests/specification