Executing medical logic modules expressed in ArdenML using Drools

Chai Young Jung,1 Katherine A Sward,1,2 Peter J Haug1,3

ABSTRACT

The Arden Syntax is an HL7 standard language for representing medical knowledge as logic statements. Despite nearly 2 decades of availability, Arden Syntax has not been widely used. This has been attributed to the lack of a generally available compiler to implement the logic, to Arden’s complex syntax, to the challenges of mapping local data to data references in the Medical Logic Modules (MLMs), or, more globally, to the general absence of decision support in healthcare computing. An XML representation (ArdenML) may partially address the technical challenges. MLMs created in ArdenML can be converted into executable files using standard transforms written in the Extensible Stylesheet Language Transformation (XSLT) language. As an example, we have demonstrated an approach to executing MLMs written in ArdenML using the Drools business rule management system. Extensions to ArdenML make it possible to generate a user interface through which an MLM developer can test for logical errors.

INTRODUCTION

Clinical decision support systems (CDSS) are computer-based systems that represent medical knowledge. Clinicians use CDSS to identify patients with specific conditions, propose treatments based on evidence, and prevent mistakes through alerts and reminders.12 While there are many ways to represent knowledge, this project is focused on rule-based systems, in which knowledge is expressed as declarative statements (rules). Knowledge management (creating, validating, and updating collections of rules) remains a significant challenge.3 It is particularly difficult to exchange knowledge between institutions which may have different computer systems, different rule languages, and different structures and terminologies for clinical data.3 4 The traditional response to such heterogeneity is to promote a standard.

A key standard for the representation and exchange of rule-based, medical knowledge is the Arden Syntax for Medical Logic Modules (MLMs).5 6 Arden Syntax is managed by Health Level Seven International (HL7) and certified by the American National Standards Institute. It was adopted as a standard in 1992; however, like many other logic representations, it has not been widely used in clinical care. Reported issues with Arden include the ‘curly braces’ problem.7–9 In the absence of a universally accepted terminology and clinical data structure, data are accessed through a mapping to site-specific data representations5 7–9 that is isolated within curly braces.

Arden provides logic for decision making which is relatively complex. Syntactic variants allow logic statements to be more readable and understandable by medical personnel, but result in opportunities for alternative expressions of similar logic.11 Arden Syntax generally requires a complex interpreter or a compiler to convert MLMs to a form that can be used by an actual system. While a number of vendors have created Arden-based systems, the variation among information systems currently in use, combined with variable ways of expressing similar logic, has resulted in compilers or interpreters customized for each system.12–18

Although the logic expressed in MLMs can be restricted to ‘if–then’ constructs, Arden Syntax is often written using procedural components. In procedural approaches, logical statements are mixed with instructions about what statement to evaluate next.2 Processing in Arden Syntax is assumed to flow sequentially through the MLM from beginning to end. This beginning to end processing assumption is in contradistinction to production rule systems,1 2 19 which are constructed on the model of rules as declarative knowledge.

Much clinical knowledge can be thought of as declarative knowledge.20 21 Rules in a declarative model are invoked by pattern matching—a rule is invoked when the conditions (the ‘if’ part) are matched by facts currently present in working memory. Additional syntax may support sequences of rule activations, but nested and branching logic are discouraged.22 While production rule systems are a more constrained environment in which to describe medical knowledge, they have advantages in terms of implementation and of logic management. Production rules are interpreted using a rules execution application, which may be configured as a service accessible to any clinical system.22 Some production rule systems provide internal memory management, rule repositories, and access to additional inferencing tools.

We used ArdenML (Arden Syntax Markup Language, a non-mandatory schema for expressing Arden Syntax in XML) in an effort to bridge the gap between the constraints of a production rule model, and a user-friendly rule authoring environment. Here we describe the extension of an earlier version of ArdenML to accommodate Arden version 2.7. We explore the proposed advantages of an XML representation through translation of Arden-based MLMs into run-time logic that can be executed in Drools (http://www.jboss.org/drools), an open source production rule system developed and supported by the JBoss community.20 23 24
METHODS
Sailors et al proposed levels of XML granularity for Arden, from 0 (wrapping existing MLMs inside a single XML construct) to 4 (encoding down to the level of operators and operands). The schema published in the Arden Syntax version 2.7 manual is at level 2 (encoding most of the categories and slots, but leaving the detailed logic uncoded). Subsequently, Kim et al published an XML schema that can express MLMs through level 4; the level 2 (encoding most of the categories and slots, but leaving the detailed logic uncoded). Kim's level 4 XML schema to correspond to Arden Syntax version 2.7. A future update is expected to replace the current schema in the Arden Syntax manual and serve as an optional way to express MLMs.

The Arden Syntax consists of an ordered set of categories, subdivided into slots. We focused on the Knowledge category, which contains the decision making logic. The slots in the Knowledge category share a set of statements and operators. We simplified the structure for operators to name and argument(s); however, statements required an individualized structure. These changes are anticipated to facilitate creating an ArdenML document and will support making Extensible Stylesheet Language Transformations (XSLTs) to transform ArdenML to other forms.

Our goal was not only to express the current version of Arden Syntax using XML but also to develop a means to execute and verify MLMs written in ArdenML. We chose to operationalize ArdenML MLMs in JBoss Drools. Drools uses the Rete algorithm to process rules, and so is quite fast. The Rete algorithm is a highly efficient pattern matching system designed for iteratively matching facts to rules in production rules systems, using a network of nodes (patterns based on condition statements). Drools is an open source product which supports knowledge sharing and customizability. Logic and the mechanisms to execute the logic are separate. A Drools rule is expressed in a declarative style, in simple if-then (when-then) statements that operate over an object-oriented fact model.

We developed an approach for transforming ArdenML MLMs using XSLT (figure 1). Transforming the rules to a Drools rule file was only part of the process needed to generate a run-time environment. For testing, we needed a user interface as well. We also needed an application to invoke the Drools inference engine. We assumed for this study that a user would supply all data directly through a simple user interface. While recognizing that, in a live environment, data will likely be obtained from queries to the hospital’s information system, this approach allowed us to create a simple test environment. We modified the ArdenML schema to support automatic user interface generation by adding XForms elements (http://www.w3.org/TR/xforms) to the ArdenML schema for elements that invoke local data queries (such as those containing Read, ReadAs, and Destination). An XSLT reads the MLM and generates code for this user interface (figure 2).

We also needed to address Arden Syntax data types and variables. Arden Syntax supports primitive data types such as Boolean, number, or string, like any programming language. Complex data types may also be used, and object types were introduced into version 2.5. Time is an important element in medical logic. Arden Syntax typically uses a single variable to represent both the value and the measurement time. Time computations are implied with logical expressions such as ‘within 6 months’. To accommodate this we created a run-time library for ArdenML based on Java.

RESULTS
Figure 5A displays a simple example for an HgA1C reminder written by Arden Syntax. HgA1C is a laboratory test used to monitor patients who have diabetes mellitus. Only the knowledge category is shown.

We manually translated the Arden MLM into ArdenML. Figure 5B shows the same HgA1C reminder MLM written in ArdenML. Once the MLM is written in ArdenML, we can transform it into another format using XSLT. We constructed three XSLT stylesheets to support creation of a user interface, the Drools rule file, and an application to drive execution of the rules.

Differences between procedural approaches and declarative approaches present a challenge in converting Arden Syntax to Drools rules. One challenge is procedural syntax such as ‘for loops’; another challenge is managing the flow between rules.
Although it is possible to change procedural syntax to declarative syntax (e.g., changing a series of ‘if–then–elseif’ statements to multiple ‘if–then’ statements), it is less clear how to enforce the order of rule evaluation implied by the sequence of statements. In a purely declarative approach, rules do not have an inherent sequence of firing. However, the order of rule evaluation can be a crucial element in the overall logic. On the other hand, Drools supports both simple mechanisms for prioritization and the JSR-94 standard. Since it is Java based, the simplest approach is to use the Java syntax. For this first study, we implemented most of the logic as Java code in the ‘then’ clause, allowing Drools to behave more like structured programming. In addition, Arden Syntax has features that are handled differently in Java. For example, the ‘and’ operator of Arden Syntax works differently from the ‘and’ operator of Java because the null state is handled differently. To accommodate the object type introduced in Arden Syntax version 2.5, we required that all operators have an ArdenMLObject parameter.

Figure 2  A user interface for testing a Medical Logic Module (MLM). Display and data entry components are defined in the curly braces part of Arden data definition statements. An Extensible Stylesheet Language Transformation (XSLT) converts this syntax into JSP code defining relevant fields on a web-based data entry form.

Figure 3  (A) A simple rule in Arden Syntax. Only the knowledge category is shown in full. This Medical Logic Module (MLM) reminds the clinician of the need for routine monitoring of HgA1C in patients with diabetes. (B) The same MLM expressed in ArdenML. The XML-based format for MLM expression provides an opportunity to manipulate the logic and to convert it into other forms.
**DISCUSSION**

We updated the XML schema for MLMs corresponding to Arden Syntax version 2.7. We developed a run-time library based on Java and a method of automated transformation using XSLT to execute MLMs written by ArdenML using the JBoss, Drools engine. We successfully created Drools-compatible files. We anticipate that our approach of generating rules, application, and user interface will provide a means to quickly review and test MLMs written in ArdenML. This approach supports knowledge management by providing a means for clinicians to author and verify ArdenML rules syntactically and logically. We expect this will support shareable medical knowledge.

Our approach has limitations. We were only able to address the curly braces problem locally, because it is related to a specific execution environment. MLMs are triggered when an event happens. Events are a system specific to execution environment. MLMs are triggered when an event interface (GUI) to author MLMs, and will support the transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments.

We are realizing this capability through the development of an transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments. Ongoing work includes developing an transformation to other medical knowledge representations and run-time environments.

**REFERENCES**