**Service-Oriented Arden-Syntax-Based Clinical Decision Support**

<table>
<thead>
<tr>
<th>Journal:</th>
<th>AMIA 2010 Annual Symposium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID:</td>
<td>AMIA-1182-A2009</td>
</tr>
<tr>
<td>Manuscript Type:</td>
<td>Paper</td>
</tr>
<tr>
<td>Date Submitted by the Author:</td>
<td>15-Mar-2010</td>
</tr>
<tr>
<td>Complete List of Authors:</td>
<td>Fehre, Karsten; Medical University of Vienna, Section for Medical Expert and Knowledge-Based Systems; Medexter Healthcare GmbH Mandl, Harald; Medexter Healthcare GmbH Adlassnig, Klaus-Peter; Medical University of Vienna, Section for Medical Expert and Knowledge-Based Systems; Medexter Healthcare GmbH</td>
</tr>
</tbody>
</table>

[AMIA logo and Manuscript Central watermark]
Abstract

We describe a software architecture for a clinical decision support system based on Arden Syntax for Medical Logic Systems. Its main components are an Arden Syntax compiler, an engine, a server with an attached data and knowledge services center, and an integrated development environment. Interoperability with external medical information systems is achieved by standard web services. The functionality and scalability of this architecture were proven in large routine applications for early detection and automated monitoring of hospital-acquired infections at the Vienna General Hospital. These systems, known as Moni/Surveillance-ICU and -NICU, daily receive clinical, laboratory, and patient care data from 15 wards. About 225 Arden Syntax medical logic modules process the data and establish whether certain definitions of hospital-acquired infections are fulfilled. The system is well accepted by the hospital’s infection control team and has proven its benefits in the above mentioned setting.

Introduction

Clinical decision support systems (CDSSs) are software programs developed to support physicians and other health professionals in patient care. Extensive efforts have been made at medical institutions and in software companies to establish clinically useful, well accepted and viable CDSSs. However, given the growing complexity of clinical workflows and the stringency of working hours for medical staff, care must be taken by the institutions deploying CDSSs to ensure that the systems become a fluid and integral part of the clinical work process. To this end, some CDSSs have met with varying degrees of success while others suffer from persistent problems which impede or prevent their successful adoption.

A typical CDSS consists of three parts:

- **A medical knowledge base (KB)** which is usually derived either from interviews with physicians or other health professionals, or through automated knowledge acquisition. The KBs referred to in this report were derived from interviews with physicians.

- **An inference engine** which combines a patient’s medical data from various sources with the medical knowledge in the KB and infers proposals in respect of diagnosis, therapy, prognosis, or patient management.

- **A communication mechanism** that permits the software system to channel the input data into the system and - after processing - present the results back to the user.

Background

**Arden Syntax for Medical Logic Systems**

Following [1], the general evolution of software architectures for CDSSs may be divided into four phases: stand-alone systems, integrated systems, standard-based systems, and systems based on service models.

Our approach falls into the category of standard-based systems because it is based on the Arden Syntax for Medical Logic Systems, approved as a standard by the American National Standards Institute (ANSI) and further developed and maintained by the Health Level Seven (HL7) organization [2].

An Arden Syntax KB consists of a set of units known as Medical Logic Modules (MLMs), each of which contains sufficient logic for at least a single medical decision. An Arden Syntax MLM is a hybrid between a production rule and a procedural formalism and is designed to evaluate a single patient’s data at a time.

A more complex KB usually contains several MLMs which interact with each other by calling them to obtain some functionality or by importing declarations from them. Furthermore, to enable interaction with the surrounding environment (e.g., for loading patient data from the host information system’s database), there are read-and-write statements which can be configured by curly braces expressions. These are the Arden Syntax constructs to interact with the surrounding environment. MLMs can be triggered by a direct call or after a specific event. The evaluation of an MLM after such an event may start immediately, or delayed, or periodically (e.g., to monitor the course of a specific laboratory value).
In general, it should be possible to transfer KBs between medical institutions to share clinical knowledge. For this purpose it may be necessary to adapt curly braces expressions to the new environment’s specific conditions. This makes any MLM transfer inflexible and cumbersome. Approaches to alleviate or resolve this problem have been described in [3, 4]. One suggestion is to establish a standard data model, such as HL7 Reference Information Model (RIM), in which members of an institution can map their individual and local vocabulary and database schema. By this process an MLM can phrase its queries towards such a standard data model, which then allows those MLMs to be transferred to other institutions at which the same data model had been implemented.

For a more detailed description of Arden Syntax see [5], which includes a tutorial for the novice Arden Syntax user.

**Service-Oriented Architecture**

A service-oriented architecture is essentially a collection of services capable of mutually communicating with each other. Such communication covers simple data transport as well as coordination of two or more services to perform a more complex task.

A service is a well-defined and self-containing function which does not depend on the internal state of other services or the current context. As such a service only has to be visible through a public interface, it is seen as a black box. As long as the service is available within a certain environment, it does not matter where the service-running host is located.

A web service is a service with a Uniform Resource Identifier (URI) whose public interfaces and data bindings are described by the Extensible Mark-up Language (XML) [6]. System frameworks based on functionalities providing web services apply standard communication protocols such as XML, Service-Oriented Architecture Protocol (SOAP), or Web-Service Description Language (WSDL). They use the Hypertext Transfer Protocol (HTTP) as their transport mechanism.

**Methods**

**Arden Syntax Compiler and Engine**

The present Java-based web service system that fully implements the Arden Syntax specification [2] consists of two essential components: an Arden Syntax compiler and an Arden Syntax engine. The Arden Syntax compiler has implemented the complete Arden Syntax and compiles given MLMs into Java code, which is executable on the corresponding Arden Syntax engine. The Arden Syntax compiler generates a signature containing information for each compiled MLM about the following: name, institution, and events. The Arden Syntax engine is able to run compiled MLMs (by MLM calls) and can handle events by executing all MLMs pertaining to the specific event. Moreover, the Arden Syntax engine approves the transfer of external objects resulting from the evaluation of curly braces expressions in the respective custom-specific mode.

Moreover, an Arden Syntax engine sub-component known as the MLM manager maintains the compiled MLMs and conceals the specific implementation of the MLM storage from the engine. In other words, the engine itself only takes over incoming requests from the framework (calls for MLMs or events), asks the MLM manager for the corresponding MLM objects, and invokes them in the correct order. Thus, the Arden Syntax engine and the MLM manager jointly handle the requests from the framework, which in our case is the respective medical host information system.

The Arden Syntax engine, the Arden Syntax compiler and the MLM manager constitute a stand-alone system which can be run on a single host to verify, compile and run MLMs.

**Arden Syntax Server**

As Arden Syntax is meant to work closely with the host information system, the two must be connected. We developed a Java-based Arden Syntax server which encapsulates the Arden Syntax engine and provides interfaces to the external systems. The Arden Syntax server conforms to an Arden Syntax host interface (AHI) - a Java interface which defines communication between the Arden Syntax engine and the Arden Syntax server. Additionally, the Arden Syntax server has to fulfill the following tasks:

- **Concurrency**: The Arden Syntax server can handle concurrent calls from outside by executing the supposed MLMs in parallel or by scheduling them in correct order.
- **Multiple Executions**: The Arden Syntax server permits multiple executions of MLMs with only one call.
- **Asynchronous Calls**: A requesting instance may use the Arden Syntax server to call an MLM and retrieve the results of the execution later on.
- **Independence from Java**: By providing an external interface which communicates with the
environment via XML messages, external resources do not have to use Java for their implementation.

To handle requests from the Arden Syntax MLMs to the host information system, the Arden Syntax server was developed to process curly braces expressions and to return the results of executed MLMs to the host system. In doing so, the Arden Syntax engine and the Arden Syntax server use the Arden-Syntax-to-Host part of the AHI. The Arden Syntax server may consult internal resources or query external hosts to handle the Arden Syntax engine requests.

By definition, an Arden Syntax server may not only send requests to a host but also receive requests from it. Such requests are MLM calls or event transmissions. An external host may send such a message to the Arden Syntax server, which will use the Host-to-Arden-Syntax part of the AHI to route the corresponding call to the Arden Syntax engine. In other words, an Arden Syntax server may implement both a client and a server component.

An Arden Syntax server connected to a KB, as shown in Figure 1, has to implement both parts of the AHI. In contrast, a system only designed to provide an interface to read information from a database must only implement the Arden-Syntax-to-Host part. To illustrate this distinction, systems implementing one of the parts of the AHI are labeled as host in Figure 1.

Figure 1 shows the general structure of a CDSS based on our software. All hosts and Arden Syntax servers in such a framework implement a communication protocol (which is described later).

Graphical User Interface (GUI)-connected hosts allowing the end user to start specific calculations or to call explicit MLMs will present the results of such operations to the user. A host system is also able to trigger MLMs or events automatically. On the knowledge provider side there are several Arden Syntax servers, each connected to a specific KB.

If an MLM that is called in the above-mentioned process contains a read phrase in a curly braces expression, the Arden Syntax server has to interpret this expression and provide the required data to the MLM. Alternatively, the Arden Syntax server may delegate the read call to a database-connected host on the customer side. Such a database-connected host usually implements only the read functionality of the AHI – a step that can be executed by host database specialists. This permits separation of data storage from the logic held by the KB-containing Arden Syntax server.

Communication

Communication between Arden Syntax servers, which establishes an envelope around the Arden Syntax engines that administer Arden Syntax MLMs, is achieved via semi-structured data (XML or HL7-compliant data structures). These structured data provide access to input and output data of the host information system. In some instances, data for Arden Syntax MLMs might just be transferred through Arden Syntax “argument” and “return” statements. Such statements represent input and output operations to an MLM. Our Arden Syntax engine then receives data through an XML structure and passes them on to the “argument” statements (the “return” statement runs the opposite way).
Communication with the Arden Syntax server follows the Arden Syntax server protocol, which is based on sending and receiving documents according to the rules of XML. The general approach of exchanging documents via the given protocol is to return the original document complemented by the results. We defined an XML scheme that specifies such communication between the Arden Syntax servers. This schema proposes a document structure and syntax for each possible message type and the corresponding answer. Figure 2 illustrates a sample document for calling an MLM.

![XML Schema for MLM Call](image)

**Figure 2.** A Sample MLM Call.

The document mentioned in Figure 2 contains an entry labeled “key” which is used to identify the targeted MLM by its name and the creating institution. The “argument” entry contains the payload which is passed to the MLM as parameter. The answer to such a request is the original document extended by the result of the called MLM. In the above case, the resulting document is shown in Figure 3.

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<usecase>
  <medexter version = "1.0" />
  <callMlm>
    <key>
      <mlmName>body mass index</mlmName>
      <institution> … </institution>
    </key>
    <arguments>
      <number>1.75</number> <!--height-->  
      <number>55.125</number> <!--weight-->  
      <date>1977-12-12</date> <!--birth-->  
    </arguments>
  </callMlm>
</usecase>
```

**Figure 3.** A Sample MLM Call Response.

To enable communication between multiple Arden Syntax servers over a network, each XML document is wrapped into a SOAP message which is transported via HTTP. As shown in Figure 4, messages between the participating hosts or Arden Syntax servers are three-layered messages, with Arden Syntax language constructs as their core.

![SOAP Message](image)

**Figure 4.** Wrapped Arden Syntax Communication.

To be complete, each of these participating hosts in the framework has to provide a description of its functionality given by a WSDL-formatted document. With such a description, all subscribers in the network are able to discover the capabilities of the host.

**Results**

Two extended implementations based on the described architecture are in routine clinical operation at the Vienna General Hospital in Austria. The Moni/Surveillance-ICU [7] and -NICU systems were developed and put into operation for early identification and continued monitoring of hospital-acquired infections at the intensive care units (ICUs) of the hospital. It contains two large KBs with about 75 and 150 MLMs, respectively. The first KB is for the detection of hospital-acquired infections in adult patients (12 ICUs with 96 beds), and the second for the detection of hospital-acquired infections in neonatal patients (3 ICUs with 36 beds). The system processes 6,000 to 8,000 data items daily and provides an overview of the development of hospital-acquired infections at these ICUs, to be presented at the hospital’s infection control unit. The gradedness of the results is determined by using fuzzy set theory and fuzzy logic to formalize and propagate the linguistic uncertainty of medical terms and the biological and practical uncertainty of medical conclusions [8]. Both KBs contain complex and elaborate MLMs. This was necessary because the
applied rule sets for determining the presence or absence of hospital-acquired infections in a certain patient are based on complex definitions (derived from CDS, HELICS, and KISS).

The Moni Arden Syntax server includes a database component to temporarily store both medical input data of the patients, intermediate and final results. It contains a reasoning component which provides backward explanation for all the computed results, and an analysis component to log inference steps for analysis and continuous knowledge maintenance.

Thanks to the advantages of the featured service-oriented architecture, both KBs - that for adult ICU patients and that for neonatal patients – coexist smoothly. Moreover, they were installed consecutively. This could be done without changing any of the essential components of the architecture. “Only” a new data input source had to be added.

**Discussion and Conclusion**

We have constructed a system which permits integration of CDSSs into existing hospital environments by removing some of the disadvantages mentioned in [1]. The system was developed as a web service application and fully implements the Arden Syntax specification [2]. An Arden Syntax compiler and an Arden Syntax engine written in Java form the core around which an extended system was built. The latter is highly flexible and can be easily integrated for communication with external host systems. In accordance with the principles of a service-oriented architecture, the individual components can communicate with each other on the basis of the widely used above-mentioned standards.

As shown in [9] and [10], service-oriented architectures are considered to be one of the future approaches for CDSSs. In contrast to the SANDS system described in [11], our system is based on Arden Syntax.

This architecture offers a number of convincing advantages:

- It shares existing KBs with other hospitals, e.g., by sharing a KB-containing Arden Syntax server.
- As all hosts have only to fulfill the AHI, regardless of how they are implemented or where data are stored or how they represent results to the end user, the framework can be easily integrated into any heterogeneous environment.
- Without changing the existing system, new KBs, data sources or GUIs may easily be added.

**References**