Artificial Intelligence in Medicine 92 (2018) 95–102

Contents lists available at ScienceDirect

Artificial Intelligence in Medicine

journal homepage: www.elsevier.com/locate/aiim

Accessing complex patient data from Arden Syntax Medical Logic Modules

Stefan Kraus[a,b,*], Martin Enders[a], Hans-Ulrich Prokosch[a,b], Ixchel Castellanos[c], Richard Lenz[d], Martin Sedlmayr[b]

[a] Center for Communication and Information Technology, University Hospital Erlangen, Glückstraße 11, 91054 Erlangen, Germany
[b] Department of Medical Informatics, Biometrics and Epidemiology, Chair of Medical Informatics, Friedrich-Alexander-University Erlangen-Nuremberg, Wetterkreuz 13, 91058 Erlangen-Tennenlohe, Germany
[c] Department of Anaesthesiology, University Hospital Erlangen, Krankenhausstraße 12, 91054 Erlangen, Germany
[d] Department of Computer Science, Chair of Computer Science 6 (Data Management), Martensstraße 3, 91058 Erlangen, Germany

ARTICLE INFO

Keywords:
Arden Syntax
Medical Logic Modules
Curly braces problem
Clinical decision support
Microbiology data

ABSTRACT

Objective: Arden Syntax is a standard for representing and sharing medical knowledge in form of independent modules and looks back on a history of 25 years. Its traditional field of application is the monitoring of clinical events such as generating an alert in case of occurrence of a critical laboratory result. Arden Syntax Medical Logic Modules must be able to retrieve patient data from the electronic medical record in order to enable automated decision making. For patient data with a simple structure, for instance a list of laboratory results, or, in a broader view, any patient data with a list or table structure, this mapping process is straightforward. Nevertheless, if patient data are of a complex nested structure the mapping process may become tedious. Two clinical requirements – to process complex microbiology data and to decrease the time between a critical laboratory event and its alerting by monitoring Health Level 7 (HL7) communication – have triggered the investigation of approaches for providing complex patient data from electronic medical records inside Arden Syntax Medical Logic Modules.
Methods and materials: The data mapping capabilities of current versions of the Arden Syntax standard as well as interfaces and data mapping capabilities of three different Arden Syntax environments have been analyzed. We found and implemented three different approaches to map a test sample of complex microbiology data for 22 patients and measured their execution times and memory usage. Based on one of these approaches, we mapped entire HL7 messages onto congruent Arden Syntax objects.
Results: While current versions of Arden Syntax support the mapping of list and table structures, complex data structures are so far unsupported. We identified three different approaches to map complex data from electronic patient records onto Arden Syntax variables; each of these approaches successfully mapped a test sample of complex microbiology data. The first approach was implemented in Arden Syntax itself, the second one inside the interface component of one of the investigated Arden Syntax environments. The third one was based on deserialization of Extended Markup Language (XML) data. Mean execution times of the approaches to map the test sample were 497 ms, 382 ms, and 84 ms. Peak memory usage amounted to 3 MB, 3 MB, and 6 MB.
Conclusion: The most promising approach by far was to map arbitrary XML structures onto congruent complex data types of Arden Syntax through deserialization. This approach is generic insofar as a data mapper based on this approach can transform any patient data provided in appropriate XML format. Therefore it could help overcome a major obstacle for integrating clinical decision support functions into clinical information systems. Theoretically, the deserialization approach would even allow mapping entire patient records onto Arden Syntax objects in one single step. We recommend extending the Arden Syntax specification with an appropriate XML data format.

* Corresponding author at: Center for Communication and Information Technology, University Hospital Erlangen, Glückstraße 11, 91054 Erlangen, Germany. Tel.: +49 91318546474.
E-mail addresses: stefan.kraus@uk-erlangen.de (S. Kraus), martin.enders@uk-erlangen.de (M. Enders), ulli.prokosch@imi.med.uni-erlangen.de (H.-U. Prokosch), ixchel.castellanos@kfa.imed.uni-erlangen.de (I. Castellanos), richard.lenz@fau.de (R. Lenz), martin.sedlmayr@fau.de (M. Sedlmayr).

https://doi.org/10.1016/j.artmed.2015.09.003
0933-3657/

1. Introduction

Arden Syntax for Medical Logic Systems, maintained by the Health Level 7 organization (HL7), is a standard for encoding and sharing medical knowledge. In Arden Syntax, knowledge is contained in independent modules named Medical Logic Modules (MLMs). An MLM typically contains sufficient knowledge to make a single clinical decision. The inventors of Arden Syntax pursued two design targets: Sharing medical knowledge between institutions, and easy knowledge acquisition [1]. MLMs are evoked by clinical events, typically the storage of a data item into the patient record (data-driven evoking). MLMs are hierarchically structured into sections (categories) and subsections (slots). The behavior of an MLM is mostly determined by four slots. The EVOKE slot defines the events an MLM reacts to, like "storage_of_blood_glucose". The DATA slot isolates those parts of an MLM that interact with the clinical information system, in particular the retrieval of patient data from the medical record. The LOGIC slot contains the actual institution-independent medical knowledge. The ACTION slot specifies the steps to be performed in case the LOGIC slot decides an action is required. Those four slots contain sequences of statements that resemble general purpose programming languages but are considered easier to write and understand for non-programmers [2]. The traditional field of application is the monitoring of clinical events [3]. An MLM could for example send an alert notification to a physician in case a new laboratory result exceeds a critical limit. Although the decision logic inside an MLM is usually simple, consisting of a few conditional statements (for a typical example see [4]), an MLM may nevertheless contain complex logic – without upper complexity limit – capable of processing complex data.

Patient data must be mapped onto Arden Syntax data types in order to process them. The creators of Arden Syntax realized that due to the lack of standards both for electronic patient records and interfaces of storage systems [5], it would be at least tedious, if not impossible, to find a standard method for accessing patient data. Instead of an abstraction layer they provided a pair of curly braces to encapsulate any parameters for mapping data between an MLM and a clinical system. As the mapping process itself is potentially tedious and must be implemented by any institution for any system, this aspect of Arden Syntax is referred to as the "curly braces problem". The Arden Syntax data type system was deliberately kept simple and corresponds to that of medical patient records (for a description of data types typically contained in medical records see [6]). Each primitive data type is a pair of a value and a timestamp (the latter termed "primary time" in Arden), since an observation like "blood glucose is 35 mg/dL" is of limited use without information about the time when this value occurred. When retrieving patient data from a database query, timestamps inside the resulting data structure must be set according to those inside the patient record. Initially, ordered lists (collections of primitive types) were the only compound data type. Nesting of lists was not permitted; therefore it was not possible to natively represent patient data with a nested structure. For most MLMs with simple decision logic such as alerting functions these plain lists were absolutely sufficient. However, for more complex applications like processing clinical guidelines, plain lists proved to be insufficient [7,8]. Consequently, starting with version 2.5 of the standard, the limitation to plain lists was overcome by the introduction of objects to permit complex data structures. Objects in Arden Syntax must not be confused with those from object oriented programming languages since there are no methods and no inheritance concept. They only provide a way to group data items and access them by name with nesting permitted.

Even though such object definitions opened up new possibilities for managing nested data structures by grouping we still identified barriers to easily assign complex data items, such as microbiology data, to Arden Syntax variables in previous implementation projects [9]. Complex data are any data with a structure beyond flat lists or tables. A list of glucose values is not complex and may easily be mapped onto the Arden Syntax list data type in a single step. A table containing all patients currently on the ward (e.g., with columns "patientname", "casenumber", "birthdate", etc.) is also not complex and, in versions supporting the object data type, may easily be mapped onto an appropriate Arden Syntax data structure such as a list of table row objects. Microbiology data at our institution, however, are complex as they have a multi-level tree structure as displayed in Fig. 1. HL7 messages are usually complex. Entire patient records are complex, and their subparts may themselves be complex. In addition, data from external systems may be complex since MLMs are not necessarily limited to process data from the patient record only. Thus the objective of this paper was to illustrate the data mapping limitations of the current Arden Syntax versions, to present the results of our investigations towards mapping concepts for complex patient data structures, and finally to propose an extension of Arden Syntax based on one of these approaches. Additionally, we benchmarked the performance of the investigated approaches as speed is known to be of paramount importance for effective clinical decision support, and complex data may be of a large size.

2. Materials and methods

2.1. Current mapping approaches

While the Arden Syntax specification leaves the actual database access to the particular institution, it nevertheless specifies the general behavior for reading patient data by means of two statements: READ (see specification [10], 11.2.1) for mapping patient data onto one or more lists of primitive types, and READ AS (see specification,

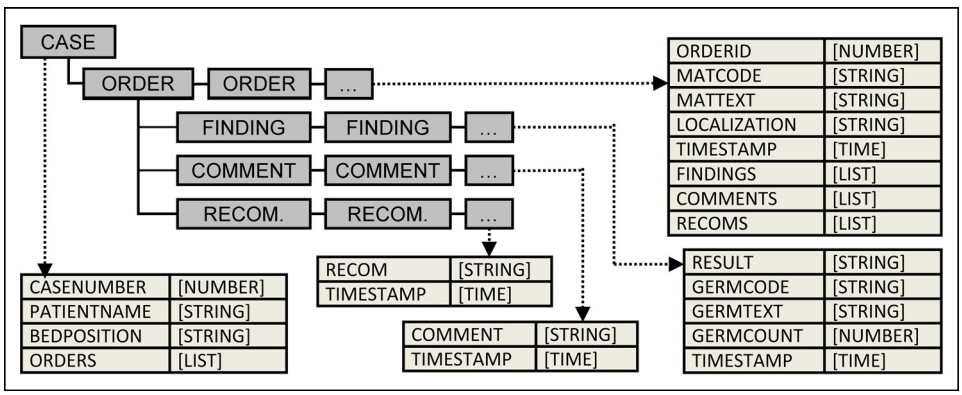


Fig. 1. Basic structure of microbiology data.

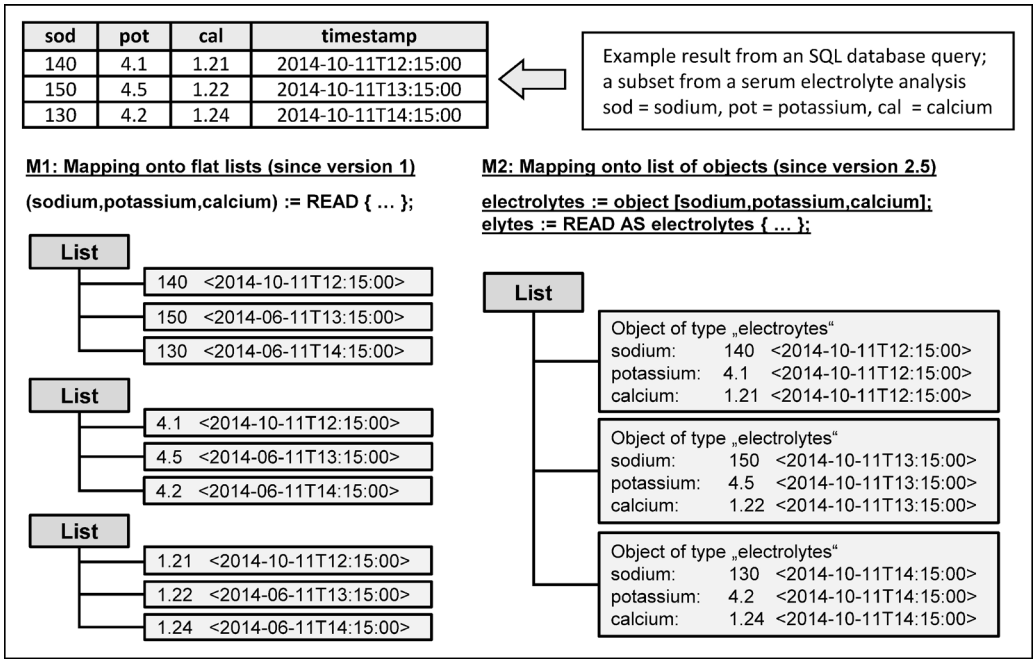


Fig. 2. Data mapping approaches provided by Arden Syntax.

11.2.2) for mapping patient data onto a list of objects. The behavior of both statements is illustrated in Fig. 2, using an example result from a SQL database query containing laboratory results from a serum electrolyte analysis. Approach M1 maps each column onto a corresponding Arden Syntax list. The primary time of each list element is set according to the timestamp column. Within a typical MLM, a READ statement requests only a single parameter and therefore maps a single column of values from a database result set onto an Arden Syntax list. Mapping multiple columns onto a set of lists may render the code inside an MLM difficult to read since the knowledge engineer has to care for accessing correspondent elements by keeping track of their positions through index variables. Approach M2 (enabled by the introduction of the READ AS statement in Arden Syntax Version 2.5), introduced alongside the object statement, maps each table row onto an electrolytes object as defined by a previous object statement, and returns a list of such objects. Each object has three attributes (sodium, potassium, calcium) containing the particular values together with the particular timestamps as primary times. This approach allows to easily process multi-column database results in a comfortable way. Note that with both mapping approaches the column names of the database result are lost. In case of M1 because lists do not support names at all; in case of M2 because attribute names are extracted from the previous object declaration ("sod" becomes "sodium", and so on).

2.2. The setting

Between 2006 and 2012, a commercial patient data management system (PDMS) was implemented in 9 intensive care units (ICU) of the University Hospital Erlangen (UHER), a tertiary care hospital with 1400 beds [11]. Since this PDMS (Integrated Care Manager, ICM©, Dräger Medical, Lübeck, Germany) lacked features for clinical decision support, the UHER information technology department and the Erlangen Chair of Medical Informatics entered a development cooperation with Dräger in order to integrate a commercial Arden Syntax environment (ArdenSuite©, Medexter Healthcare, Vienna, Austria) into the PDMS [9]. The inherent limitations of the PDMS required the implementation of several workarounds. The PDMS does not yet provide direct database access for third party systems such as an external Arden Syntax engine. However, it provides an export interface to generate doctors' letters and textual reports. We applied this interface to replicate patient data required by MLMs into a separate proxy database, which is entity-attribute-value (EAV) modeled (for a detailed explanation of this data model see [6]). Database access from MLMs to the proxy database was enabled by a self-written abstraction layer based on prepared statements. In addition, the PDMS lacks a trigger mechanism to notify the Arden Syntax engine about the storage of data items, which is an indispensable feature for data-driven MLM execution. Therefore, such a mechanism has been implemented in our proxy database. Processing of microbiology findings within MLMs required another workaround. These data are contained in HL7 messages sent by our laboratory information system, but our PDMS currently stores only extracted contents as plain text, thus inhibiting automatic processing by MLMs. Consequently, we duplicated the HL7 messages on our communication server so that they can be processed using a self-written parser which extracts the desired information and stores it as structured data inside our proxy database. As microbiology data are not sparse and have a regular structure, we chose to model them in a conventional database schema rather than in an EAV model. The basic structure of microbiology data at our institution is shown in Fig. 1. Each patient's microbiology data is a list of orders with their associated findings. Thus, attached to each order is a list of findings, a list of comments entered into the laboratory system, and a list of DRG coding recommendations relevant for billing.

In this initial situation our investigation was prompted by new clinical demands. First, ICU clinicians wanted to generate multi-patient overviews with filtered or highlighted observations according to their clinical importance. As the PDMS does not natively support such flexible visualization features, clinicians required MLMs to process these structured microbiology data. Second, clinicians demanded immediate alerts about critical events. Up to now, alert messages are delayed for up to 10 min due to the required data replication into the proxy database. To satisfy these demands, we searched for a reusable, generic approach to map arbitrary subsets of patient records, or even sets of whole patient records, onto a single Arden Syntax data structure, and for a way

to map entire HL7 messages onto Arden Syntax objects in order to monitor time-critical laboratory results directly from the HL7 messaging stream on our communication server.

2.3. Approaches to map complex clinical data

Our study started with the specification of the intended mapping result, a list of microbiology objects represented by the Arden Syntax object data type introduced in version 2.5. Each object should contain a single patient's entire microbiology data, with the tree structure inside the patient record preserved. Subsequently we analyzed the interface components of three different Arden Syntax engines to compare approaches for handling the curly braces problem: The commercial system used in the clinical routine at UHER (ArdenSuite [12], supporting all versions of Arden Syntax), an open source system developed at the University of Braunschweig (Arden2ByteCode [13], supporting version 2.5), and a PHP: Hypertext Preprocessor (PHP) web application developed by one of the authors for research purposes (ARSEN/IC, supporting version 2.8).

We identified three different basic approaches to map complex data by analyzing both the available application programming interface (API) documentation and the source code of the particular interface components, and implemented them using ARSEN/IC as this system provides both native support of mapping approach M2 and a built-in database abstraction layer. We created a database snapshot of microbiology data for all patients currently admitted to the largest of our ICUs as a data sample for testing the implementation. To validate the results of each mapping approach we wrote an MLM called STRUCTUREPRINTER which creates a tree view (see Fig. 3) of any data structure it receives as the argument by recursive descent, using the comparison operators (see specification, 9.6) and the relevant object operators (see specification, 9.18.3 and 9.18.4). As an alternative application example, and to fulfil the clinical requirement of immediate alerting, we mapped entire HL7 messages onto Arden Syntax objects, using an approach provided by the commercial ArdenSuite as part of its "Arden Syntax Server Protocol".

Our first approach to map microbiology data onto Arden Syntax objects, named "Arden approach", was to build the intended data structure from "inside" an MLM. We used a top-down approach based entirely on the mapping approach M2 described above. We defined a set of object data types (Fig. 4A) according to the basic tree structure of microbiology data (Fig. 1). The last attribute ORDERS from data type CASETYPE serves as an anchor for the orders

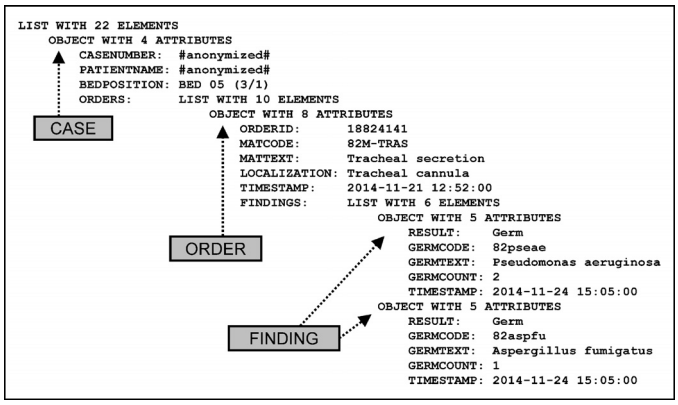


Fig. 3. Detail from a tree view created by MLM STRUCTUREPRINTER.

sub-level, just as the last three attributes from data type ORDERTYPE (FINDINGS, COMMENTS, RECOMS) serve as anchors for the findings, comments, and recommendations sub-levels. We created the intended data structure by using nested loops as shown in Fig. 4B. Left-hand parameters inside the curly braces are the names of prepared statements. Right-hand parameters, marked with a prefixed dollar symbol, are MLM variables to be resolved and inserted into the prepared statements. We integrated the nested loops into a separate MLM that may be called by any other MLM requiring microbiology data.

Our second approach was to perform data mapping from "outside" an MLM. This approach was basically the same as with the Arden approach, except that the code was not implemented in Arden Syntax but in the PHP programming language of the systems interface component. We used the particular API methods to create the intended data structure, i.e., the methods to create data types, add list elements and set attributes of objects, and thus termed it the "API approach". Database access from within the interface component was performed using an open source database abstraction layer. As with the Arden approach described above, each sub-level was processed inside a nested loop.

Our third approach, called the "deserialization approach", was based on the proprietary Arden Syntax Server Protocol provided by ArdenSuite, specifying an Extensible Markup Language (XML) data format as displayed in Fig. 5. We transformed microbiology data to the XML-based data structure defined by this protocol, and called an MLM using this XML structure as the argument. As

A

```
casetype     := object [casenumber,patientname,bedposition,orders];
ordertype    := object [id,matcode,mattext,localization,timestamp,findings,recoms,comments];
findingtype  := object [result,germcode,germtext,germcount,timestamp];
commenttype  := object [comment,timestamp];
recomtype    := object [recommendation,timestamp];
```

B

```
for case in caselist do
    casenumber   := case.casenumber;
    orders       := read as ordertype {mibiorders|$casenumber};
    for order in orders do
        orderid          := order.id;
        findings         := read as findingtype {mibifindings|$orderid};
        order.findings   := findings;
        comments         := read as commenttype {mibicomments|$orderid};
        order.comments   := comments;
        recommendations  := read as recomtype {mibirecommendations|$orderid};
        order.recoms     := recommendations;
    enddo;
    case.orders  := orders;
enddo;
```

Fig. 4. Arden approach based on nested loops (A = object declarations, B = nested loops performing the actual mapping process).

```
<LIST>
  <OBJECT OBJDECLNAME="CASETYPE">
    <FIELD NAME="CASENUMBER"><NUMBER>#anonymized#</NUMBER></FIELD>
    <FIELD NAME="PATIENTNAME"><STRING>#anonymized#</STRING></FIELD>
    <FIELD NAME="BEDPOSITION"><STRING>BED 05 (3/1)</STRING></FIELD>
    <FIELD NAME="ORDERS">
      <LIST>
        <OBJECT OBJDECLNAME="ORDERTYPE">
          <FIELD NAME="ORDERID"><NUMBER>18824141</NUMBER></FIELD>
          <FIELD NAME="MATCODE"><STRING>82M-TRAS</STRING></FIELD>
          <FIELD NAME="MATTEXT"><STRING>Tracheal secretion</STRING></FIELD>
          <FIELD NAME="LOCALIZATION"><STRING>Tracheal cannula</STRING></FIELD>
          <FIELD NAME="TIMESTAMP"><TIME>2014-11-21 12:52:00</TIME></FIELD>
          <FIELD NAME="FINDINGS">
            <LIST>
              <OBJECT OBJDECLNAME="FINDINGTYPE">
                <FIELD NAME="RESULT"><STRING>Germ</STRING></FIELD>
                <FIELD NAME="GERMCODE"><STRING>2pseae</STRING></FIELD>
                <FIELD NAME="GERMTEXT"><STRING>Pseudomonas aeruginosa</STRING></FIELD>
                <FIELD NAME="GERMCOUNT"><NUMBER>2</NUMBER></FIELD>
                <FIELD NAME="TIMESTAMP"><TIME>2014-11-24 15:05:00</TIME></FIELD>
              </OBJECT>
              <OBJECT OBJDECLNAME="FINDINGTYPE">
                <FIELD NAME="RESULT"><STRING>Germ</STRING></FIELD>
                <FIELD NAME="GERMCODE"><STRING>82aspfu</STRING></FIELD>
                <FIELD NAME="GERMTEXT"><STRING>Aspergillus fumigatus</STRING></FIELD>
                <FIELD NAME="GERMCOUNT"><NUMBER>1</NUMBER></FIELD>
                <FIELD NAME="TIMESTAMP"><TIME>2014-11-24 15:05:00</TIME></FIELD>
              </OBJECT>
```

Fig. 5. Detail from an XML representation of microbiology data (proprietary Arden Syntax Server Protocol).

with the other approaches, we used the MLM STRUCTUREPRINTER to verify the correctness of the mapping. To further investigate this approach, we integrated a data mapper into ARSEN/IC, allowing serialization [14] of any data structure to the XML-based Web Distributed Data Exchange (WDDX) format [15], and integrated another mapper which performs the same process backwards, i.e., deserializing WDDX to Arden Syntax data structures. We opted for WDDX because it is natively supported by PHP, and ARSEN/IC stores compiled MLMs in this format.

We measured the execution speed by repeating each data mapping approach 1000 times on a server virtual machine (Windows 2008 server with 8 GB main memory and 2 Intel Xeon X5675 processors), with each pass followed by a random delay within 1 to 3 s. The benchmarks for all three approaches were performed in parallel to create a considerable server load. Patient data were retrieved from a database server within the local network. Additionally, we measured the peak memory usage of each approach. Our data sample for testing and benchmarking comprised 22 patients, with a total of 352 orders, 690 findings, 67 comments, and 91 recommendations.

Finally, we utilized the Arden Syntax Server Protocol to map entire HL7 messages containing laboratory results onto Arden Syntax objects. We integrated a self-written Python program into our communication server, which transforms HL7 messages to appropriate XML structures consisting of nested objects. We tested several variations regarding the naming of objects and attributes, and decided to use an easy to understand convention. We used the names from the HL7 specification, i.e., “segment”, “field”, “component” and “subcomponent”, and attached consecutive numbers as suffixes. Therefore, a subcomponent inside a HL7 message, passed to an MLM as the argument, would be accessed with an expression like

hl7message.segment5.field3.component1.subcomponent3.

3. Results

Each of the three investigated Arden Syntax engines provides an interface component for the connection to a clinical system. Besides trigger methods to evoke or call MLMs (Fig. 6A), such interface components also provide methods that are implicitly called each time the control flow inside an MLM reaches a statement associated with a curly braces expression such as a READ statement (Fig. 6B). Such an evaluation method receives the content of the specific curly braces expression as the argument and returns an Arden Syntax data structure which eventually is assigned to a variable.

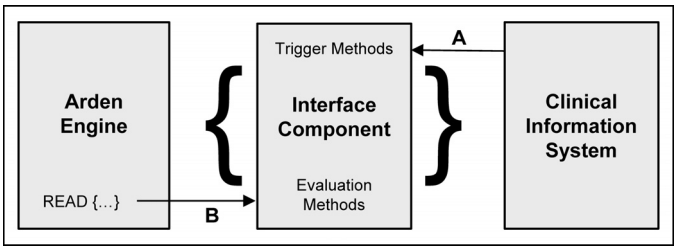


Fig. 6. Interface component of an Arden Syntax engine.

#anonymized#, discharged 01/05/2015: No microbiology orders

#anonymized#, discharged 01/11/2015
- **Wound swab (abdomen) 01/05/2015: Candida albicans complex**
- **Wound swab (abdomen) 01/05/2015: Enterococcus faecium**
- Sputum 01/02/2015: No evidence of yeast-like fungi
- Nose/throat swab 01/02/2015: No evidence of staphylococcus aureus

#anonymized#, currently bed 23 (5/7)
- **Tracheal secretion 01/03/2015: Candida albicans complex (C. albicans / dubliensis)**
- **Tracheal secretion 01/03/2015: Staphylococcus aureus**
- **Bronchial secretion 01/03/2015: Candida albicans spp.**
- Blood culture, aerobic and anaerobic 01/09/2015: No bacterial growth after 5 days

Fig. 7. Details from an MLM generated overview of microbiology findings.

All investigated systems utilize the same approach regarding the READ statement as described above (Fig. 2, M1). They provide an evaluation method returning one or more lists; this method is to be implemented by the particular institution.) The approaches slightly differ regarding the READ AS statement (Fig. 2, M2). Arden2Bytecode and ARSEN/IC implicitly perform approach M2, transforming a database query result into a list of non-nested objects with attribute names corresponding to the previous object declaration. Therefore, both systems will not return objects with a deeper structure. However, both systems permit the implementation of alternative behavior. The commercial system utilizes a slightly different approach by adding the object type to the argument list of the evaluation method. This allows returning nested objects, but mapping approach M2 is to be implemented by the customers themselves. In summary, all three systems are basically capable of mapping approach M2 with or without nested object structures. Moreover, each system provides a built-in abstraction layer for SQL databases, although this is currently not required by the Arden Syntax standard. ArdenSuite additionally provides a proprietary mapping approach termed Arden Syntax Server Protocol. This approach allows MLMs to be called with an XML data structure as the argument, using a web service interface. The data structure is implicitly deserialized to a congruent Arden Syntax data structure and may be assigned to an Arden Syntax variable using the ARGUMENT statement inside the DATA slot.

Based on the results of our analysis, we determined and implemented three different approaches to map our microbiology data test sample onto Arden Syntax data structures. All three mapping approaches resulted in exactly the same data structure: An Arden Syntax list containing Arden Syntax objects, as visualized in Fig. 7, which shows a detail from an MLM generated overview based on this data structure. The MLM processes all patients of a surgical ICU who are currently admitted or were discharged within the past seven days. Positive findings are printed in bold.

Our approaches for mapping complex data can be summarized as follows:

- *Arden approach*: The Arden approach required the definition of five appropriate Arden Syntax object types as described in Fig. 4A. The entire mapping process was performed by nested loops, requiring 14 lines of code altogether, as shown in Fig. 4B. Each

tree level was created by a READ AS statement implementing mapping approach M2.

- *API approach*: The API approach, to be implemented in the programming language of the particular interface component, strongly resembles the Arden approach, building the intended data structure by nested loops. However, due to the number of API methods to be called in order to create objects and to set attribute values, this approach requires a much larger code.
- *Deserialization approach*: The deserialization approach is not part of the Arden Syntax standard. Currently, it is a built-in function provided by the commercial Arden Syntax Server that is part of ArdenSuite. To use this approach, data must be provided in XML format as shown in Fig. 5. Investigations of this approach using ARSEN/IC, including the performance and memory usage measurements, were based on WDDX as the data format and the corresponding built-in deserializer from PHP. In any case, patient data provided as XML are deserialized to Arden Syntax data structures. A deserializer can be used by means of the INTERFACE and CALL statements of Arden Syntax. In case of ARSEN/IC, the following lines of code are required in the DATA slot in to deserialize the content of an XML file:

```
mapper:= INTERFACE {XmlFileMapper};
<identifier>:= CALL mapper WITH <filename>;
```

Our performance measurements revealed that the Arden approach was the slowest one, taking an average of 497 ms for the entire data structure to be created. The API approach took an average of 382 ms, the deserialization approach, based on WDDX, an average of 84 ms. All approaches significantly differ from each other ($p < 2.2e-15$). Both the Arden and the API approach consumed about 3 MB of main memory, while the deserialization approach consumed about 6 MB. The results are displayed in Fig. 8.

Utilizing the deserialization approach for mapping entire HL7 message onto Arden Syntax objects proved successful. The XML data structure generated on our communication server was passed to the Arden Syntax server provided by ArdenSuite, where it was deserialized to a single Arden Syntax object and passed to an MLM as the argument. Each data item in the message was successfully accessed using the naming convention described above. Fig. 9 shows an example message that is mapped to a variable ("hl7message") containing the message as a nested object. A glucose value of 147 [mg/dL] is assigned to a variable using a path expression corresponding to its position (suffixes starting at 1, according to the counting of list elements in Arden Syntax). The glucose value is explicitly converted to the NUMBER data type, which is required to compare it to a numeric threshold.

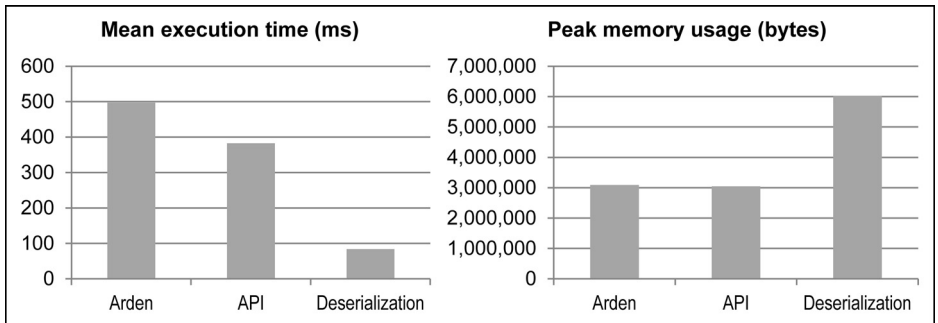
4. Discussion

Clinical event monitoring presupposes access to patient data [3,16]. Therefore, the importance of the task of mapping patient data onto Arden Syntax data types cannot be overestimated; MLMs are "desperately seeking data" [17]. Since the data type system of the early versions of Arden Syntax was limited to flat lists as the only compound data type, an approach for mapping complex data was not necessary; Arden Syntax was literally one-dimensional. Traditional MLMs typically processed data elements from medical patient records, e.g., laboratory values, which could be mapped onto lists. The introduction of objects into the type system [18] opened up many new fields of application for MLMs, but also entailed new challenges for data mapping. Later versions (2.5 and higher) are capable of processing complex data such as tree structured microbiology data or possibly even random sets of entire patient records. Even the traditional limitation to a single patient focus retained by most Arden Syntax engines, such as the model implementation at the Columbia Presbyterian Medical Center (CPMC) [19], may have to be overcome e.g., for infection outbreak monitoring.

We identified three different approaches for mapping complex data: An Arden approach, building the intended data structure inside an MLM; an API approach, building the intended data structure inside the interface component of an Arden Syntax engine using the particular API methods; and a deserialization approach, deserializing an XML data structure onto a congruent Arden Syntax data structure. The deserialization approach appears to be the most promising, because it is generic insofar that it can map arbitrary data structures as long as they are in an appropriate XML format. The other two approaches are not generic insofar that the code must be tailored to their structure, which must be known in advance. Nevertheless, the API approach could be implemented in a more generic way by deviating slightly from the specified behavior of the READ AS statement. If clinical data are stored in EAV format, which is the case with many clinical information systems, the EAV format might be extended to EAV/CR (EAV with classes and relationships; for a detailed introduction see Nadkarni [6]). Adding a metadata repository to the database schema allows the grouping of data items to objects, enabling nested data structures that could easily be mapped onto congruent Arden Syntax data structures. In this case, whenever the control flow within an MLM reaches a statement like "READ AS electrolytes", the interface component would look up the structure of "electrolytes" in the metadata repository rather than in a previous object declaration. Nevertheless, in case of conventionally modeled microbiology data at our institution as described above, as well as with HL7 messages, this approach could not be applied because these data are not EAV modeled.

The Arden approach, although not generic, has the advantage of not requiring any technical prerequisites besides Arden Syntax versions 2.5 and higher. It does not require any additional programming work inside the interface component. Mapping complex data is therefore possible for any knowledge engineer who is familiar with nested loops and knows the structure of the data to be mapped. As the code performing the mapping is entirely in Arden Syntax, it may be transferred to another institution for the purpose of knowledge transfer, where it is to be adjusted according to the local database schema. We recommend integrating the nested loops performing the data mapping into a separate MLM. In this case, one could say the DATA slot is outsourced into a "data loader MLM".

Jenders and Shah [20] described an earlier attempt to process microbiology data at the CPMC. The objective of their work was to monitor nosocomial infection outbreaks. The authors developed a two-phase system. The first phase was to monitor microbiology results by MLMs that extracted relevant data and stored them in a repository. The second phase was to perform a cross-patient analysis by counting organisms within timeframes by means of a statistical monitor, which was initially used to monitor the correct function of the CPMC Arden Syntax installation itself [21]. The reason why their monitoring approach could not entirely be performed in Arden Syntax was the fact that the CPMC installation, like many others, was limited to a single patient focus, thus prohibiting a cross-patient analysis within an MLM. Nevertheless, even if this installation had supported multi-patient MLMs, there would still have been the problem of mapping multi-patient microbiology results onto Arden Syntax data structures in the DATA slot, because at that time the object data type was not supported. Although it would be technically possible to process populations of patients in MLMs based on flat lists, handling the code may soon become tedious. We presume that it would be expedient to use a "one case – one object" approach to create a list of microbiology objects in the DATA slot, in line with our approaches described above. Currently,



	Arden	API	Deserialization
Mean execution time (ms)	497.4	382.7	83.9
Median execution time (ms)	468	358.8	78
Minimum execution time (ms)	452.4	327.6	78
Maximum execution time (ms)	1326.1	1248	109.2
Standard deviation execution time (ms)	80.2	73.5	7.6
Peak memory usage (bytes)	3,090,512	3,048,392	6,018,096

Fig. 8. Results of performance and memory usage measurements.

we do not monitor infection outbreaks by MLMs at our institution, but this is a matter of discussion. Thanks to the cross-patient capabilities of our installation and the support of the object data type we do not expect technical problems.

Utilizing this cross-patient capability together with the described mapping approach allowed us to process the data in a convenient way inside MLMs. Finally, the initial clinical requirement to display a multi-patient view of microbiology data has been fulfilled. The MLM displaying microbiology data is user-driven, i.e., triggered by button click; performance was thus an important aspect (according to the first commandment of effective clinical decision support: "speed is everything" [22]). In the case of data-driven monitoring MLMs, a short delay within a range of seconds is usually unproblematic. In case of user-driven MLMs, with the clinician waiting at the computer for the result to be displayed, execution time must not exceed a few seconds in order to maintain user acceptance. With an average execution speed of less than 1 s, each of the three mapping approaches proved to be fast enough. As expected, the Arden approach was slower than the API approach. We were surprised by the high performance of the deserialization approach which was several times faster than the API approach. Along with the advantage of being generic, this aspect makes this approach even more promising. A striking observation is that the memory usage of the deserialization approach is twice as large as that of the other approaches, reflecting the way many XML libraries work. In a first step, they transform the entire XML data structure into a main memory tree structure according to the Document Object Model (DOM) [23]. In a second step, they traverse the resulting DOM tree to create a congruent Arden Syntax data structure; thus two large memory structures exist. In the case of the other approaches, the Arden Syntax data structure is constructed successively during a series of database access operations, resulting in only one large memory structure. This also relates to the striking differences in execution time. The deserialization approach is based on fast memory access, while the other approaches are based on considerably slower database access. Of course, our measurements refer to a single Arden Syntax engine in our specific environment only. But we presume similar results would be achieved in different settings, because the API methods of the investigated Arden Syntax engines used to construct Arden Syntax data types are very similar, and memory access will probably be faster than database access. Moreover, the API approach will probably be faster than the Arden approach, simply because Arden Syntax code can hardly execute faster than the underlying programming language.

Arden Syntax was intended for knowledge transfer between institutions, thus reusability of the deserialization approach in different healthcare settings is a matter of interest. A necessary first step would be to specify a data format and a method to map it onto Arden Syntax data types. For our test implementation inside ARSEN/IC we used the WDDX deserialization operator from PHP, because it is natively supported by that language and requires only a single line of code. However, the data format is not limited to XML but may also be JavaScript Object Notation (JSON) or any other data format permitting nested data structures. In the case of our local setting, JSON would be of great interest, as the vendor of our PDMS announced the integration of a JSON-based data access interface. Nevertheless, if we were to propose a data format for integration into the Arden Syntax standard, we would recommend using the one included in the Arden Syntax Server Protocol provided by the commercial ArdenSuite from Medexter Healthcare. This

```
MSH|^~\&|RADIANCE|ABL835|||20141115000100||ORU^R01|1421|P^not present|2.2
PID|1||||Doe^Jane||19891109|F|||||||DE
PV1||N|M1I|||||||||||||||||1234567890
OBR|1||1060262008^Blood Gas Analysis|1^Blood Gas Analysis^^^BG Examination…
OBX|1|ST|GluM^Glu^M||147|mg/dL||N|||F||||30
```

```
DATA:
hl7message := argument;
glucose := hl7message.segment5.field6 as number;
```

Fig. 9. Accessing HL7 message contents from MLMs.

XML-based format is straightforward and clearly tailored towards the data type system of Arden Syntax, making it easier to use than more generic formats like WDDX. The only slight adjustments we would suggest are to rename the element "field" to "attribute" and the attribute "objdeclname" to "type" to ensure compliance with the naming convention in the Arden Syntax specification. This format already supports Fuzzy Arden Syntax [24,25] by providing an attribute for the degree of applicability. Implementing a mapper for this format is routine programming work, and a variety of libraries exist for many programming languages to assist the programmer. The Arden Syntax standard stipulates that each institution has to implement a local solution for data mapping. However, it appears reasonable for Arden Syntax engines to provide such mapping solutions out of the box, to avoid that multiple institutions need to reinvent the wheel. It may be worth discussing the integration of a new language construct for the purpose of deserialization into the standard. As ARSEN/IC supports the integration of new statements and operators for research purposes, we added a unary operator using a filename as the argument, calling a data mapper, and returning an Arden Syntax data structure. In our opinion, the approach to leave the curly braces problem entirely to the specific institution appears slightly anachronistic.

Even 25 years after the creation of Arden Syntax, the curly braces problem is still an issue as there is still no standard for electronic patient records. And despite all efforts to standardize patient data in form of a reference information model or virtual medical records, the uptake of these approaches in real-world systems is practically nonexistent. The heterogeneity of interfaces to storage systems, however, has been considerably reduced since 1989, and though it is not required by the standard, all three investigated Arden Syntax engines provide support to access SQL databases. An abstraction layer to avoid SQL code inside curly braces can be easily implemented using prepared statements, as in case of our installation. Both the READ and the READ AS statement are well designed and cover the majority of database access operations. Up to now, most of our MLMs process single patients and use READ statements to process flat lists, thus they do not require complex data. Some MLMs, however, process patient populations and make use of READ AS statements to generate table structures. They process admission data, with information derived inside the MLM which is to be attached to that structure at runtime, such as the prototype of a multi-patient dashboard MLM. Clinical demands suggest that population-based MLMs will be of growing interest in the near future and will demand an approach for mapping complex data. Recently (November 2014) ArdenML [26,27] was integrated into the latest version 2.10 of the standard, enabling entire MLMs, including statements and operators, to be expressed in XML. A standardized XML or JSON data format for the deserialization approach would be another meaningful extension to the standard as it would complement the existing READ and READ AS statements with a convenient way to map random data structures of random complexity onto Arden Syntax data types. The application example for mapping an entire HL7 message demonstrated that the deserialization approach works with arbitrary structured data formats.

References

[1] Hripcsak G, Ludemann P, Pryor TA, Wigertz OB, Clayton PD. Rationale for the Arden Syntax. Comput Biomed Res 1994;27(4):291–324.
[2] Samwald M, Fehre K, Bruin F de, Adlassnig K. The Arden Syntax standard for clinical decision support: experiences and directions. J Biomed Inf 2012;45(4):711–8.
[3] Hripcsak G, Clayton PD, Jenders RA, Cimino JJ, Johnson SB. Design of a clinical event monitor. Comput Biomed Res, Int J 1996;29(3):194–221.
[4] Hripcsak G. Writing Arden Syntax medical logic modules. Comput Biol Med 1994;24(5):331–63.
[5] Jenders RA. Decision rules and expressions. In: Greenes RA, editor. Clinical decision support: the road ahead. Amsterdam, Boston: Elsevier Academic Press; 2007. p. 267–80.
[6] Nadkarni PM. Metadata-driven software systems in biomedicine: designing systems that can adapt to changing knowledge. New York, NY: Springer; 2011.
[7] Peleg M, Boxwala AA, Bernstam E, Tu S, Greenes RA, Shortliffe EH. Sharable representation of clinical guidelines in GLIF: relationship to the Arden Syntax. J Biomed Inf 2001;34(3):170–81.
[8] Peleg M, Ogunyemi O, Tu S, Boxwala AA, Zeng Q, Greenes RA, et al. Using features of Arden Syntax with object-oriented medical data models for guideline modeling. Proc AMIA Symp 2001:523–7.
[9] Kraus S, Castellanos I, Toddenroth D, Prokosch H, Bürkle T. Integrating Arden-Syntax-based clinical decision support with extended presentation formats into a commercial patient data management system. J Clin Monit Comput 2014;28(5):465–73.
[10] Arden Syntax for Medical Logic Systems, Version 2.8, Health Level Seven, 2012.
[11] Bürkle T, Castellanos I, Tech H, Prokosch H. Implementation of a patient data management system – an evaluation study of workflow alterations. Stud Health Technol Inf 2010;160(Pt 2):1256–60.
[12] Fehre K, Adlassnig KP. Service-oriented Arden Syntax-based clinical decision support. In: Schreier G, Hayn D, Ammenwerth E, editors. Proceedings of the eHealth 2011. Vienna: OCG; 2011. p. 123–8.
[13] Gietzelt M, Goltz U, Grunwald D, Lochau M, Marschollek M, Song B, et al. ARDEN2BYTECODE: a one-pass Arden Syntax compiler for service-oriented decision support systems based on the OSGi platform. Comput Methods Programs Biomed 2012;106(2):114–25.
[14] Soukop J, Macháček P. Serialization and persistent objects: turning data structures into efficient databases. Berlin: Springer; 2014.
[15] Richards R. Pro PHP XML and web services. Berkeley, New York: Apress; 2006.
[16] Johansson B, Shahsavar N, Ahlfeldt H, Wigertz O. Database and knowledge base integration—a data mapping method for Arden Syntax knowledge modules. Methods Inf Med 1996;35(4–5):302–8.
[17] Hripcsak G, Johnson SB, Clayton PD. Desperately seeking data: knowledge base-database links. Proc Annu Symp Comput Appl Med Care 1993:639–43.
[18] Jenders RA, Corman R, Dasgupta B. Making the standard more standard: a data and query model for knowledge representation in the Arden Syntax. AMIA Annu Symp Proc 2003:323–30.
[19] Hripcsak G, Cimino JJ, Johnson SB, Clayton PD. The Columbia-Presbyterian Medical Center decision-support system as a model for implementing the Arden Syntax. Proc Annu Symp Comput Appl Med Care 1991:248–52.
[20] Jenders RA, Shah A. Challenges in using the Arden Syntax for computer-based nosocomial infection surveillance. Proc AMIA Symp 2001:289–93.
[21] Hripcsak G. Monitoring the monitor: automated statistical tracking of a clinical event monitor. Comput Biomed Res 1993;26(5):449–66.
[22] Bates DW, Kuperman GJ, Wang S, Gandhi T, Kittler A, Volk L, et al. Ten commandments for effective clinical decision support: making the practice of evidence-based medicine a reality. J Am Med Inf Assoc 2003;10(6):523–30.
[23] Harold ER, Means WS. XML in a nutshell. 3rd ed. Sebastopol, CA: O'Reilly; 2004.
[24] Vetterlein T, Mandl H, Adlassnig K. Fuzzy Arden Syntax: a fuzzy programming language for medicine. Artif Intell Med 2010;49(1):1–10.
[25] Vetterlein T, Mandl H, Adlassnig KP. Processing gradual information with Fuzzy Arden syntax. Stud Health Technol Inf 2010;160(Pt 2):831–5.
[26] Kim S, Haug PJ, Rocha RA, Choi I. Modeling the Arden Syntax for medical decisions in XML. Int J Med Inf 2008;77(10):650–6.
[27] Jung CY, Sward KA, Haug PJ. Executing medical logic modules expressed in ArdenML using Drools. J Am Med Inf Assoc 2012;19(4):533–6.