Arden Syntax–based decision support systems are not widely used in current medical information systems, despite the importance placed on them in the literature and their endorsement by standards organizations such as the American Society of Testing and Machinery (ASTM), Health Level 7 (HL7), and the American National Standards Institutes (ANSI, which approved Arden Syntax without giving a number to the standard). The major likely reasons include the difficult tasks of writing Arden Syntax compilers and integrating a decision support system with both the Arden compiler and the underlying information system.

This paper presents a design and implementation approach of an Arden Syntax medical logic module (MLM)–based decision support system that uses an object-oriented model and the dynamic linking features of the Java platform. The portability to a target information system is achieved by the addition of appropriate adapter components, which they call mappers. These mappers are dynamically selected using explicit and implicit elements of MLMs. Furthermore, they can help translate data from the clinical information system representation into the representation needed by an MLM. This was validated by an experiment in two clinical units. Also, the authors propose a convention to name signals that trigger other MLMs (called intermediate states) so that they remain unique to each institution. The authors implemented this architecture in their clinical system and in an XML-based medical record application that has been used experimentally in their urology and nephrology departments. The Tetrasys company that provided the medical record was able to incorporate their runtime without modifications, and typical MLM execution time was less than 1 sec.

This paper presents a design and implementation approach of an Arden Syntax medical logic module (MLM)–based decision support system that uses an object-oriented model and the dynamic linking features of the Java platform. The approach enables the dynamic use of data mapping objects (which we call mappers) to accommodate the differences in data availability and representation among different institutions. This should lead to a faster and more reliable sharing of the medical knowledge encoded in MLMs. To demonstrate the portability advantage of this approach, this Arden/J framework has been used to implement Arden MLMs both in our clinical system and in an XML-based medical record application that has been used experimentally in our urology and nephrology departments. The Tetrasys company that provided the medical record was able to incorporate their runtime without modifications, and typical MLM execution time was less than 1 sec.

Arden Syntax–based decision support systems are not widely used in current medical information systems, despite the importance placed on them in the literature and their endorsement by standards organizations such as the American Society of Testing and Machinery (ASTM), Health Level 7 (HL7), and the American National Standards Institutes (ANSI, which approved Arden Syntax without giving a number to the standard). The major likely reasons include the difficult tasks of writing Arden Syntax compilers and integrating a decision support system with both the Arden compiler and the underlying information system.

Affiliation of the authors: Biostatistics and Medical Informatics Department, Henri Mondor Hospital, Creteil, France.

This work was made possible by grants from France Télécom.

Correspondence and reprints: Harry C. Karadimas, MD, MS, CHU Henri Mondor, AP-HP, 51 Avenue du Marechal de Lattre de Tassigny, 94010 Creteil, France; e-mail: <harry.karadimas@hmn.ap-hop-paris.fr>.

Received for publication: 6/24/01; accepted for publication: 3/5/02.

Abstract The Arden Syntax was introduced more than 10 years ago, but it is still not in widespread use. One reason might be that for each particular architecture and information system, a different Arden Syntax compiler must be written as well as a program for the runtime execution of the medical logic modules (MLMs). The authors have designed and implemented an architecture that increases the portability of Arden Syntax rules, using the Java platform. The portability to a target information system is achieved by the addition of appropriate adapter components, which they call mappers. These mappers are dynamically selected using explicit and implicit elements of MLMs. Furthermore, they can help translate data from the clinical information system representation into the representation needed by an MLM. This was validated by an experiment in two clinical units. Also, the authors propose a convention to name signals that trigger other MLMs (called intermediate states) so that they remain unique to each institution. The authors implemented this architecture in their clinical system and in an XML-based medical record application that has been used experimentally in their urology and nephrology departments. The Tetrasys company that provided the medical record was able to incorporate their runtime without modifications, and typical MLM execution time was less than 1 sec.

data and events from major medical record implementations existing at that time. In April 1992, the Arden Syntax was approved by the American Society of Testing and Machinery (ASTM), under the number E-1460-92, making it the first standardized syntax available for writing medical decision-support rules.

Each participant in the Arden Syntax meeting developed and published its implementation of the new standard (Columbia-Presbyterian, Salt Lake City LDS, Regenstrief Hospital, and a European Hospital in Linköping, Sweden). In 1998, the Object Management Group (OMG) issued a recommendation requesting system developers to take this standard into account in their solutions. The Arden Syntax is now also offered by several vendors of clinical information systems (CISs, e.g., Health Vision, SMS, Micromedex, HBOC, IDX, Eclipsys) as an addition to their systems.

In 1999, HL7 took over the management of the Arden Syntax standard, producing the second version. This version enhanced and clarified certain constructs. It should soon become an ANSI standard (it has been “ANSI approved”). It was also designed to be backward compatible with the initial version. Different shortcomings of the MLM model and Arden Syntax were described, explaining its limiting use in medical practice.

**MLM Data Element Representation**

Representation of the external data elements used in MLMs is a very difficult task. More important, the definition is currently not reusable by another institution than the one that issued the MLM, unless the other institution has the same representation of external data elements and has those elements available in its CIS. For example, an MLM author writing for many research institutions out of the standardization effort. One of the goals of the Arden meeting was to devise a syntax that would be implemented appropriately in each different system. Most systems at the time of the meeting consisted of already existing decision support systems, which all had their own languages, concepts, representation of events, queries, etc. However, an immediate consequence of this specification is that MLM implementers usually start their work by taking the model of their information system and writing a wrapper library around it, which forms the runtime environment for MLM execution. A new compiler must then be written to translate Arden Syntax into executable machine code (or any intermediate form) specific to the runtime environment.

The code associated with a data element can be unclear if not commented. For example, at Columbia-Presbyterian Medical Center (CPMC), data element descriptions consist of one ore more concept numbers. When the MLM is ported to another institution, the external data element codes must be redefined for the target institution. This task involves extra work from the institution that imports the MLM, and it affects the way the MLM works.

However, an institution may also be forced to change its own knowledge representation standards over time (like, in our case, using International Classification of Diseases [ICD] version 10 instead of ICD version 9). This results in records with different versions of the in-house representation standard and, therefore, complex maintenance problems. In this case, MLMs work only if they refer to recent records. However, if they query data in old records as well as in new ones, a different mechanism has to be used. When the number of MLMs grows, additional problems may occur, because of shortcomings in the standard, which does not describe mechanisms for MLM grouping, inter-MLM communication, and uniform representation of data elements. However, if they query data in old records as well as in new ones, a different mechanism has to be used. When the number of MLMs grows, additional problems may occur, because of shortcomings in the standard, which does not describe mechanisms for MLM grouping, inter-MLM communication, and uniform representation of data elements.
The implementation at CPMC uses a virtual machine. This implies that the Arden compiler produces specific code for this virtual machine; the produced compiled code can be reused, provided that the virtual machine is rewritten for the target information system. Another practical result is that MLMs can be added without recompiling or linking. However, we do not know a port of this particular virtual machine to another institution using another machine type. (An IBM mainframe is used at CPMC.)

The implementation at Linköping University was the first to use an object model. This model avoids the explicit generation of intermediate machine code and simplifies compiler design. The compiler can write C++ source directly, and this C++ is then compiled to produce the executable MLM. However, every time an MLM is added to the system, a compile-and-link process has to be executed, and the system has to be restarted. Also, there is no published object model for the generated MLM code.

Knowledge Maintenance Problems

The design choice of having stand-alone MLMs for atomic knowledge still leaves some problems open:

- When adding an MLM, the implementer must ensure that no MLM exists that addresses the same issue (or an overlapping issue), which would result in getting two similar, redundant alerts.

- When an MLM calls one or more other MLMs, the MLMs are tied together in a logical way. To keep these related MLMs together, one must ensure that they are in the same file when they are moved from one CIS to another. This is not easy, especially when some MLMs are used repeatedly by many other MLMs. Suggestions of more convenient packaging mechanism would be desirable. For the moment, only the institution’s name is used to achieve this grouping, although this is not described explicitly in the standard.

The Arden Syntax

The first step in the standardization process focused on the control structures of the rules; it was decided to delay the standardization of data elements until the different data nomenclature efforts converged toward a clearer consensus. Therefore, the data element descriptions were separated from the rest of the syntax. Their content format was left open (with minor restrictions) to the implementer’s choice. However, the translation of an external data element (represented by the character string inside the curly braces) from one institution to another with a different data representation is difficult, tedious, and prone to error. This leads to what is now known in literature as the “curly braces problem.”

Another choice made by the Arden group was to delay the standardization of the rule-chaining mechanisms, defining only the possibility of one MLM calling another, in a programming subroutine–like manner. This made “atomic modularity” possible, which means that every MLM is mostly independent of all other MLM, except for the few ones it calls directly, as one MLM cannot (and, in fact, should not) trigger another MLM. However, although defined superficially only, provision was made to allow an MLM to write to the patient database. This led to an MLM chaining methodology using “intermediate states.” This methodology consists of special database representations that, when written into the database, trigger one or several other MLMs. Musen and others, however, criticized the intermediate states approach, which can lead to knowledge engineering problems.

There is also no methodology (provided by the standard) to group together logically related MLMs (that have similar goals, for example). Thus, MLMs chained indirectly by intermediate states must be kept together by some external, ad hoc methodology. The only possible criterion for grouping is the institution (given that all authors of the institution always use exactly the same string in the “institution” slot). MLMs also contain keywords and bibliographic references to help with indexing and management. However, nothing guarantees that for a specific set of MLMs, the same subset of MLMs retrieved by a query based on keywords will be retrieved through the same query in a different institution.

Knowledge Validation Problems

Customization or adaptation of knowledge can be difficult when all the knowledge must be contained in a single MLM. Medical data representation can be different across specialties, explaining the difficulty of representing knowledge relevant to different specialties in a single MLM.

Design Objectives

When we planned the implementation of Arden Syntax–based MLMs, we realized that most parts of an MLM-compliant system do not depend on the implementation. Then, we tried to model the neces-
We discovered that the most important part to be designed was not the compiler itself but the runtime of the MLM system. We set the following design goals:

- Build an object model of the external elements (event, data, message, destination, etc.) used in the Arden Syntax. An adequate object model leads to the production of a portable and generic runtime and a unique, portable compiler.
- Separate internal objects that exist independent of the CIS, from objects that are adapters to CIS data elements, which we call mappers, and make writing mappers simple.
- Build an environment for stand-alone rule syntax checking and execution.
- Respect the institution-specific syntax while allowing the mixing of MLMs from different institutions with no or minimal rewriting.
- Achieve fast execution performance, since alert systems should be responsive.

Material

To implement the compiler, we used the Java platform for all developments, using Sun’s Java Development Kit JDK 1.3.0 and the Java CUP (“Construction of Useful Parsers, ”a YACC-equiva-

lent compiler construction tool) from Princeton University. The compiler was written using traditional, well-known compilation techniques.19

Arden Syntax MLM Element Object Model

Building an object model for portability implies finding an object model for MLMs and, more specifically, a model of the data sources. Then, implementing data sources in a target information system is only a matter of writing mappers for each data source. Queries from the MLM are translated to the target system, and results are translated back to the encoding expected inside the MLM. Similarly, external event monitors, specific to an institution, must be supported, which means the generic model of the MLM runtime environment serves as the starting point.

Accessing Specific Host Elements Through Mappers

Our implementation proposes a semi-automatic external data element adaptation mechanism. It uses a special tagging format for external element description and enables the automatic selection of the appropriate mappers.

When the compiler encounters the description of an external data element (the part inside the curly braces), it includes it in the generated code. The MLM, when loaded for the first time, uses these descriptions to first load mapper objects from the MLM host. These mapper objects are object factories that must be written specifically for a given CIS. These mediators understand the query in the MLM, translate it to the host CIS, and translate the result back to the MLM so that it can be used inside the MLM logic.

Figure 1 illustrates this process in our CIS with the mapper to our laboratory, patient identification, admission discharge and transfer, diagnosis-related groups, and procedure codes databases. The MLM sends specific queries with query parameters (QP). This specific query is translated into a query suitable for our CIS (QIT). The query is issued to the appropriate database, using the translated query parameters (TQP). The results of the query (QR) are translated back (RIT) to the MLM’s data representation (TQR). These results are then available to the MLM.

Object Data Model for Implementers

To integrate the Arden/J runtime in an external information system, some objects have to be written by the implementer. We present here some concepts that prevail for the Arden/J runtime design and only
a few relevant objects from the object hierarchy. The base classes relevant to the porting are summarized in Figure 2. The framework/library is available free (including source code and instructions) to academic institutions.20

Arden Runtime Item Categories

- From the Arden Syntax specification version 1, five external item types (declared in the “data” slot) can be distinguished—events (indicated by the “event” keyword); result data (inside “read” clauses); action destinations (indicated by the “destination” keyword); messages for a destination (indicated by the “message” keyword); and other MLMs (indicated by the “MLM” keyword). Arden Syntax version 2 adds another external item type—external functions (indicated by the “interface” keyword).
- The event item has a type but no value.
- The data source item, when queried, returns a collection of time-stamped rows; each row can be an individual value or a list of values.
- The destination item can be translated to an e-mail address, a database destination, or whatever medium the implementer chose.
- The message item is written to a destination item. Its data type is, therefore, relevant only to the target system and destination.

Mapper Objects

For the Arden/J runtime, the implementer must provide mapper objects, which implement the QAIF-Mapper interface. Mappers use a general hierarchic qualifying mechanism to tag in a unique fashion any external data element used inside the syntax. The qualifiers are written using a syntax similar to C++ and XML namespaces. The first qualifier is always the special string “qaif:” which stands for Qualified Arden Item Format. The second qualifier is used to select a mapper. The other qualifiers, separated by colons, and the query by itself remain specific to the institution that manages them. However, they may come from a different institution than the one that issued the MLM.

An example is “qaif:hmn1:diag:icd9:412 Old myocardial infarction.” Item hmn1 is used to select a mapper to the Henri Mondor Information System. Item diag is a Henri Mondor mapper-specific tag that indicates that a diagnostic will be queried, and ICD-9 indicates that the expected coding of the diagnostic will be an ICD-9 code. ICD-9 is a qualifier for the ninth revision of the International Classification of Diseases, and “412” is the code of the disease.

The textual description that follows the code is ignored by the hmn1 mapper; it is kept for human readability. This description forms a simple indirec-
tion mechanism that gives the MLM host object enough information to instantiate a correct mapper object. This object is able to interpret the code “412”. It instantiates an object that represents the exact concept of this code in the local nomenclature used in the CIS (for example “I252” if ICD-10 is used). The mapper object is then ready to query the patient database.

For internal nomenclature translations (in our example, ICD-9 to ICD-10), it is a matter of implementation whether the nomenclature item object internally uses a generalized nomenclature like Galen or UMLS or implements only a few nomenclatures with code equivalences. If the code cannot be mapped (for example, an ICD-9 code that has disappeared from ICD-10 is being used here), a Java exception is thrown. (A “thrown” Java exception is an error-handling mechanism that allows efficient error recovery.) The MLM host signals that it can not import the MLM. The internal nomenclature translation problems remain, and these problems have already been documented in the literature.15

**Arden Implementation Objects**

Mappers return objects that implement a Java interface as required for the item: QADataItem for data items, QAEventItem for event items, QADestinationItem for destination items. The runtime implementation uses external objects through these interfaces exclusively. Messages are not processed by the Arden/J runtime, so they do not need to implement a particular interface.

**Event Monitors**

Event monitors are implemented separately and must be declared to the MLM host. New event monitors can easily be added to the system (if a new laboratory system is added, for example). They are managed concurrently, so that every different event source can use an individual event monitor without conflicting with other event monitors.

**System Description**

**The Arden/J Compiler**

Our compiler produces one Java class source file for one MLM. The Java source files must then be compiled using traditional Java compilers. This step is automatic. The intermediate Java compilation phase allows the generation of efficient optimized byte code. The code generated from the MLM can not be optimized much, as is the case in all languages with dynamic data types (i.e., discovered at runtime). However, in practice, most of the execution time was taken by the database queries (see benchmark in Table 1), which is consistent with other reported experiences.8,16,21–23 Java classes have the advantage of being linked at runtime, in contrast with standard C++ classes that must be linked explicitly. This means that additional classes (in our case mapper objects, event monitors, and even compiled MLMs) can be added in their binary form by administrators by simply being copied in a directory and declared in a configuration file. No supplemental development tools are needed to add new classes to the system. This should enable several authors to test their MLMs iteratively and concurrently on a running system, without restarting the system every time.

**The Arden/J Runtime**

The Arden/J runtime is the necessary part used to run the compiled MLM. Active MLM classes are listed in a configuration file and loaded at startup time. Figure 3 shows the schematic interaction of the different runtime objects. The clinical event monitors (CLEM) send events (E), which trigger MLMs (M). MLMs that are candidates for one or more executions are scheduled in an execution agenda, which can be implemented in an external database or in memory. The MLM queries data from the CIS databases (db) and then may write to a destination (D). Typical destinations are an e-mail address or the patient data-

![Figure 3](https://academic.oup.com/jamia/article-abstract/9/4/359/695751/b1a213df)  
Figure 3 Arden/J runtime objects and their interaction with the clinical information system. CLEM indicates clinical event monitor; E, event; M, medical logic module; D, destination; db, database.
base. The writing to a destination can also produce events. The runtime does not allow the addition of a newly compiled MLM while the system is running, but this is technically possible and is planned for a future version. For the moment, the Arden/J runtime has to be stopped and restarted to take new MLM into account.

**Status Report**

**Arden/J Software Package**

The result of our design, modeling, and development work is the Arden/J framework. It consists of a portable compiler, a generic MLM runtime kernel, and a minimal implementation for stand-alone use (without connection to a CIS) for educational purposes. Specific implementation code was written for two platforms—the Dossier Médical Actif (DMA) medical record system, developed by the Tetrasys company, and the CIS at the Henri Mondor Hospital.

**Integration to the Henri Mondor Information System**

The Arden/J runtime is integrated to our CIS and responds to ADT (admission, discharge, transfer) events, diagnosis code storage events, medical procedure code storage events, and laboratory result dependent events. The mappers have a separate configuration for the databases they query. We have distinct databases for ADT information, diagnostic, medical procedures, and laboratory results. On some legacy tables used for ADT information, we were not allowed to implement a triggering mechanism, so our implementation of the clinical event monitor has to regularly query the database for ADT events.

Some execution statistics have been measured and are summarized in Table 1. They were measured on a Pentium 300 running Windows NT, and the queries were executed on our ADT database, which has nearly 700,000 records in an Oracle database system, running on a cluster of three Unix servers. They show that for an MLM, the execution time is mostly taken by the database queries, and, furthermore, query execution time increases with data model complexity.

**Integration into the Dossier Médical Actif XML Patient Record**

Experimentation has been conducted on the DMA, a semi-structured medical record built around Java and XML technologies (from the Tetrasys company) to illustrate the use of mappers for accessing data outside our CIS. We integrated the XML medical record to our laboratory results server and developed an interface to incorporate the Arden/J runtime. In the medical record, we were able to quickly create new clinical item types, their graphical user interface (GUI) representation, and associated MLMs. The Arden item mediators specific to the DMA were developed separately by Tetrasys using the mapper application programming interface. The mappers make direct queries to the document object model (DOM) tree and translate the data into results suitable for MLM execution.

This experimentation was successfully carried out in the nephrology and the urology units. Each new item is added to the DOM tree, either externally by the processing of an external XML message containing laboratory data information, or internally when the user validated input from the user interface. Each new item can trigger an MLM if this item is declared in the MLM. The total time from the reception of the item to the display of the alert is always less than 5 sec and is in most cases not noticeable by the user (less than a second). Interviews showed that the physicians were pleased with the integration of the alerts in their medical practice, because 1) they were initially afraid that the inference of MLM would slow down the use of the medical record, which was not the case; and 2) they were not disturbed by unnecessary alerts, because these alerts were few (three on serum creatinine and urea interpretation, and two on data inconsistencies), and they were made very specific to avoid false positives.

**Stand-alone Use for Educational Purposes**

When the Arden/J runtime is started in stand-alone mode, it reads a test patient database (i.e., a simple text file containing a list of values) in memory and displays a simulation window (Figure 4). The window allows entry of additional data, triggering of
events directly or through the addition of data, and simulation of time. This mode is useful for several reasons:

- Teaching the Arden Syntax using an interactive tool
- Testing MLMs directly using adequate test data (in a text file prepared for the session, or hand entered by the student who can test the behavior of the MLM directly)
- Testing the behavior of MLMs that have a specific temporal sequence, without having to wait for the whole sequence to execute

It is important that the same environment be used in education, development, and implementation, so that the required technical skills are the same.

**Discussion and Perspectives**

**Difference from Previously Published Work**

As in the implementation at Linköping University, an object orientation framework is used. This allows us to generate code that resembles the original MLM and implements an instruction set orthogonal with the Arden data types. This makes it easier to adhere strictly to the standard. However, we propose a CIS object model, simple and generic, sufficient to execute an MLM, and the methodology to adapt this generic runtime model to a particular CIS, by providing specific interface code and adapter objects. This model is demonstrated within three environments—for use in our CIS; for use with the DMA record from Tetrasys; and for educational use with a database represented by a text file and events triggered directly by a student using a GUI.

The use of Java as a system platform also has several advantages. Java is implemented using a virtual machine. The use of a virtual machine brings the same portability benefits as the implementation at Columbia-Presbyterian hospital, with the additional benefits of Java standardization and industry support. The strong object orientation of the Java system allowed us to cleanly separate our implementation classes from external classes and allowed the reuse and extension of our runtime classes by other developers outside of our institution. All these different classes communicate clearly and efficiently through Java interfaces and abstract classes. Java’s garbage collector, hiding of pointers, and exception mechanisms also enhance the reliability of the system in comparison with most C++ based systems. A simple indirection mechanism allows the dynamic selection of specific mappers that can understand item representations other than the native ones. The translation of these representations to the native data scheme is clearly not part of our work and must be done by the implementer. However, even when only a simple one-to-one translation table is possible between two systems, our approach is still quicker and less error-prone than re-mapping the same item by hand every time a new update of an MLM is received.

**Acceptance by Physicians**

Our practitioners fully understood the rules we had jointly designed when they were encoded in the Arden Syntax. However, they did not envision all the implications of this kind of decision support system. A few MLMs that were thought to be useful, about abnormal results warning (urea and creatinine mainly), proved to be unnecessary and annoying for most physicians, whereas other users did not mind the alerts and wanted to keep the MLMs active. Currently, only very critical alerts (e.g., creatinine rising fast after a renal transplantation) are kept active. The other alerts were degraded to simple information.
messages that do not require reading acknowledgment. To indicate this, the MLM urgency slot has a value of 1, which is the minimum value allowed for this slot. In the future, we will have to add a more precise description of the medical profile of the referring physician in our CIS. This will enable a more precise adaptation of the MLM.

Mapping MLMs from Other Institutions

The mappers in our system are loaded at runtime and queried for the particular data source and representations that they understand. For example, our institution, the Henri Mondor Hospital (which we abbreviate “hmn”), provides a mapper that knows how to handle some local items (tagged with “qaif:hmn1:”). We will write mappers that map external sources from other institutions to the data available in hmn. We could, for example, write a mapper that recognizes items from the Columbia-Presbyterian MED nomenclature. We could tag explicitly the data items with a “qaif:cpmc:” tag or declare in the configuration files to have implicitly “qaif:cpmc:” tagged items if no explicit tagging is given and the MLM comes from Columbia-Presbyterian institution (this is inferred using the text in the “institution” slot). The mapper will then try to translate the items to our local data representation, in order to import MLM directly from the CPMC with minimal rewriting.

Use of Intermediate States

The chaining of MLMs is desirable in many cases, mainly when the execution of some task is dependent of the result of another task, in an asynchronous manner, as in guideline action steps, for example. It has been shown that intermediate states allow this type of chaining of MLMs\(^\text{18}\) acting as signals. This methodology can be helpful, if properly applied, despite certain knowledge engineering problems.\(^\text{12}\)

The main problem lies with the localization of these signals; they are meaningful only in the context of the set of MLMs they control. If the set of MLMs related to a set of signals becomes too important, it can be almost impossible to predict the behavior of the system. However, if only one to ten MLMs are relevant to a given signal, without “auto-activation” of an MLM, the activation of the MLMs can be clearly set off. To achieve this, each signal must be fully qualified, so that it can be addressed from any place and from any context.

We propose to use our tagging syntax for the destination that qualifies these signals. However, a more global naming convention must be used in this case. For this purpose, we have chosen a convention directly inspired by the Web URI syntax.

Web URLs have been designed to identify a given resource in a unique, global manner. This convention uses the special name ‘signal’ as the data destination designation, followed by the URI name of the institution, followed by the description of the signal, which is a string in URI format. For example, to signal that chemotherapy must start in the LNH98 trial (a chemotherapy clinical trial for lymphomas), the signal triggering the MLM(s) would be:

\[
\text{start_chem} := \text{destination (qaif:signal:hmn.ap-hop-paris.fr/protocols/lnh98/start_chem)}
\]

This convention could help organize a set of MLMs into a directed activation graph or Petri net by allowing the recognition of the nodes using their common signals. Edges can be built between nodes (MLMs) that declare a signal as a message (start of edge) and nodes that declare a signal as an event (end of edge).

Organization of MLM in Packages

We have borrowed from Java the organization in packages. A package in Java groups related classes. This naming methodology has proved to be convenient and efficient for Java classes, and it also applies well to MLMs. The packages are organized hierarchically, with the inverse URI name of the institution at the beginning. For example, for MLMs related to biochemistry in our institution, we use the package name ‘fr.ap-hop-paris.hmn.biochemistry’. For this purpose, we use the ‘links’ slot with a ‘package’ keyword. For the biochemistry-related MLM, we write:

\[
\text{links: package fr.ap-hop-paris.hmn.biochemistry;}
\]

Conclusion

The use of the Java platform to implement an MLM runtime combined with an adequate object model simplified our implementation of an Arden Syntax-compliant decision support system, compared with a more traditional approach that would use the C or C++ language. The compiler does not have to be rewritten, and the necessary objects that must be written for the target system (mostly for query execution) are clearly defined. The use of a tagging mechanism allows some automatic mapping from one data element representation to another (when this mapping exists), thus diminishing the localization efforts required when sharing MLM. The use of the Java class-loading system allows the addi-
tion of new MLMs and new revisions of existing MLMs at runtime, simplifying concurrent and iterative usage of the system for MLM testing and adapting. We were able to adapt this Arden Syntax–compliant system to three different information systems—our CIS, an XML-based patient medical record, and a CIS simulation for stand-alone use by students. The Arden Syntax standard needs further improvements in external data representation and in MLM control strategies.

The authors thank the Urology and the Nephrology Departments of the Henri Mondor Hospital for evaluating the medical XML record and Tetrasys for providing the DMA XML record system and the Arden/J integration into the DMA. They also thank Omar Bouhaddou for his comments and corrections.

References