AIM: An XML-Based ECA Rule Language for Supporting a Framework for Managing Complex Information

Essam Mansour, Kudakwashe Dube and Bing Wu

School of Computing, Dublin Institute of Technology, Kevin Street, Dublin 8, Ireland.
<firstname>.<surname>@dit.ie

Abstract. This paper presents an XML-based event-condition-action (ECA) rule language, AIM, for supporting the SEM framework and approach to the computer-based incorporation of best practice in daily work and the subsequent management of the resulting complex information. SEM framework provides knowledge and information management support in terms of three planes: the specification plane, the execution plane and the manipulation plane. AIM language is an assembly of declarative language modules for supporting the three planes of the SEM framework and envisages its use within the context of XML and databases.

Keywords: ECA rule paradigm, XML language, active database, clinical practice guidelines, information management

1 Introduction

The formalization and computerization of best practice and its subsequence incorporation into an organisation's computer-based information systems results in *complex information* whose management poses a serious computing challenge. In intensive care applications, a medical patient plan is an example for the *complex information* that is produced by incorporating the healthcare best practice, clinical guidelines, into the disease management. The domain users are interested in incorporating the best practice and managing the *complex information* as one distinct entity- as it exists in the domain- and at high and declarative level. However, most of computerized approaches, which are utilized to incorporate best practice into application activities, focus only on incorporating the best practice at the level of: 1) individuals rules and triggers, such as in active database research [1, 2]; 2) processes that create a list of administrative actions based on the user's criteria, such as in workflow based approaches [3]; 3) decision making by utilizing a decision model or AI technique, such as in [4]. These approaches create gaps between the domain users and the computerized best practice. For example, it is difficult for doctors to review or modify the medical patient plan at the level of triggers or processes. Best practice needs to be specified using a suitable language based on an appropriate computational formalism. The language and computational formalisms used should be chosen

specifically for its ability to support easy incorporation of best practice with domain information databases as well as its ability to support computer-based execution mechanisms. The dynamic nature of best practice demands that specification languages and execution mechanisms should allow on-the-fly manipulation. Furthermore, the query and information retrieval facilities in information systems also need to be provided with respect to the formalised best practice. This work follows a generic, comprehensive and unified information management framework, called SEM [5], developed by the authors in order to address these challenges. The unified framework allows information to be managed comprehensively in three planes for the specification, execution and manipulation with each plane being able to be integrated with the other two planes.

This paper addresses the problem of providing a comprehensive language to support the unified framework, SEM, for managing complex information arising from the incorporation of best practice into computer-based applications. In this paper, a high level XML-based declarative language, called AIM, for supporting the SEM framework is presented. The rest of this paper is organised as follows: Section 2 outlines related work; Section 3 presents the AIM language requirements; Section 4 discusses the AIM language specification component, AIMSL; Section 5 presents the complex information model in AIM; Section 6 presents the AIM language query and manipulation component, AIMQL; Section 7 outlines the implementation progress and future work; Section 8 summarises and concludes this paper.

2 Related Work

Languages that support the incorporation of best practice into computer information systems has continued to attract a lot of research attention. The formalisation and computerisation of best practice and expert knowledge in the area of database administration has been achieved through using a XML-based rule language to specify business policies that govern user database access rights [6]. Thus, using computerised best practice and expertise, non-IT personnel could be allowed to perform functions that are normally performed by a database administrator. Further typical examples are found in the areas of business activity management (BAM) [7] as well as in automated e-business negotiation with emphasis on goals, policies, strategy and plans for decisions and actions [8] in which best practice is formalised and specified for the purpose of computerisation.

ECA-RuleML [9] constitutes work on an XML-based rule language that focuses on the logic programming framework for supporting specifications of event-condition-action (ECA) rules that are integrated with derivation rules and integrity constraints. In the area of active XML, the ECA rule paradigm [2] is incorporated into the XML to support active behaviour over XML data. Several active XML languages have been produced, such as Active XQuery [10], and An Event-Condition-Action language for XML [11]. These languages incorporate the best practice into the application activities at the level of separated rules and triggers, which makes difficulties to domain users to review or modify. The AIM language is unique in its provision of a unified framework that caters not only for creation of specifications but also for their

execution and subsequent manipulation and querying within a comprehensive information management context.

3 AIM Language Requirements

The main requirements for AIM language is to support the *SEM framework* at the three planes that cover information and knowledge specification, execution, manipulation, querying and information scenario replays. These requirements are summarised as follows: 1) Language requirements for the Specification Plane**:** AIM language is required to support the specification process, in which the best practice is formally specified ; 2) Language requirements for the Execution Plane*:* with respect to a specific domain scenario, the best practice specification is customized and instantiated to produce the *complex information*, such as producing a medical patient plan for a specific patient using a specific clinical guidelines specification. The *complex information* contains a reactive behaviour that determines the correct reaction for certain situation, such as the medical patient plan gives medical recommendations when the patient temperature is changed. Hence, the best practice specification must be expressive enough to specify behaviour that can be executed within this plane; 3) Language requirement for the Manipulation Plane: The *complex information* is subject to the same manipulation operations, as the domain information, plus some special operations, such as terminate, enable, and disable. The manipulation operations could be included in the behaviour of the *complex information*, or issued by the domain users. The manipulation operations facilitate a) the propagation of the changes from the generic specification to the *complex information*; and b) the maintenance of the complex information. The *complex information* is also subject to the same queries, as the domain information, plus special query support, such as the replay function, which allow dynamic execution scenarios to be re-enacted for the user's review.

Thus, to satisfy the requirements of the SEM framework, AIM language must provide 1) a specification language, which we will call the *AIM Specification Language* (AIMSL**)**; 2) a model for the complex information; 3) a manipulation and query language, which we will call the *AIM Query Language* (AIMQL).

4 AIMSL: The Specification Language for the Complex Information

This section presents the specification component of AIM language, which called AIMSL.

4.1 AIMSL Model and Distinguishing Features

The model of AIM Specification Language (AIMSL) follows the event-condition-action (ECA) rule paradigm. AIMSL model expresses the best practice as modularized sets of rules, which are classified according to functional objectives and

scopes. Fig. 1 illustrates the XML Schema of the AIMSL model. In this schema model the best practice is formally specified as a protocol library, which consists of protocol specifications as well as specifications of global rules whose scope is the entire domain of discourse and one not associated with any protocol. As shown in Fig. 1, the individual protocols one made up of schedules and a set of protocols rules that not associated with any schedule. Each schedule is a set of rules that differs from an ordinary rule set in that it has an entry criteria and the fact that all rules in it are bound together by a common functional objective. Each rule in the specification is deemed to be an ECA rule, which is defined over some relevant domain information attributes. It should also be noted that protocol, schedule and rule element in the schema model has a set of attributes and that each element in the schema is made up of a sequence of a combination of attributes and other elements. Thus, the schema model allows ECA rules to be specified as either a memes of a set or a part of a protocol or a schedule element. It should be pointed out here that the protocol and the schedule are manageable as single units although they are effectively sets of rules. The header is a collection of pieces of release and didactic information. The release part provides information related to specific specification version. The didactic part provides literature related to the best practice; cites references to the source of the knowledge that is encapsulated in the AIMSL specification; and provides explanation.

The AIMSL schema is modularized to provide flexibility in modifying or enriching the AIMSL language to suit several application domains. For example, applications, which demand specific requirements for the condition part, could replace the condition part with its own one.

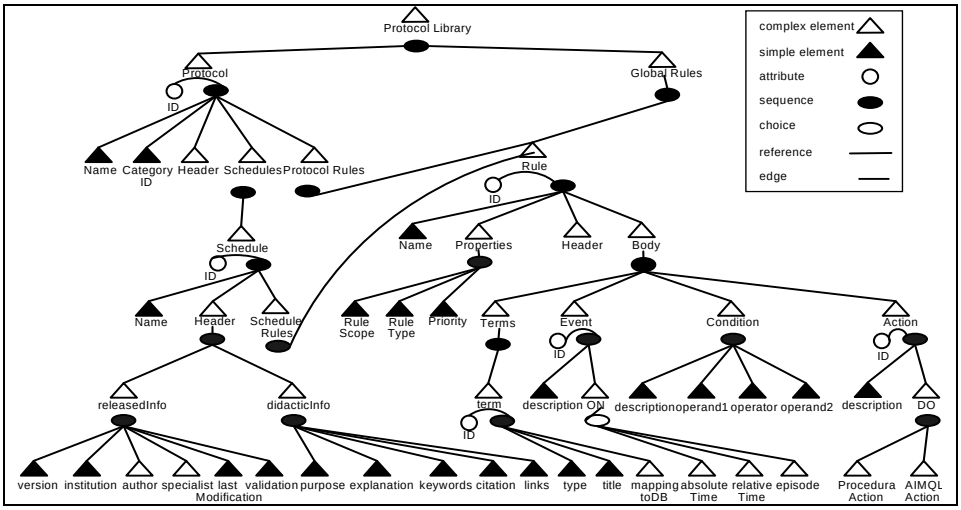

Fig. 1. The XML Schema for the AIMSL sub-language.

4.2 ECA Rules and Temporal Features in AIMSL

In AIMSL model, the rule schema consists of elements (*name, properties, header, and body*) and the id attribute. The element *properties* determines the scope, type and

priority of the rule. The scope specifies whether the rule is a global, protocol, or schedule rule. The rules, according to their event, are classified into to type static or dynamic rules. A static rule performs an action subject to the occurrence of a time event. A dynamic rule is a rule, whose event is a non-temporal event. The body consists of elements (*Terms, Event, Condition, and Action*). General terms are used in specifying the rule event, condition, and action. The element *Terms* specifies a general term and maps it into particular data items according to the utilized database schema. The term might be of type event or element. Consider as example, Rule 1: on two days after patient admission, order the blood test. AIMSL rule specification for Rule 1 will contains term, whose title and type are patient admission and event, respectively. The type event means this term will be used in the specification of the event of Rule 1. If the term is of type event, it will be mapped into database operation(s). If the term is of type element, it will be mapped into database attribute.

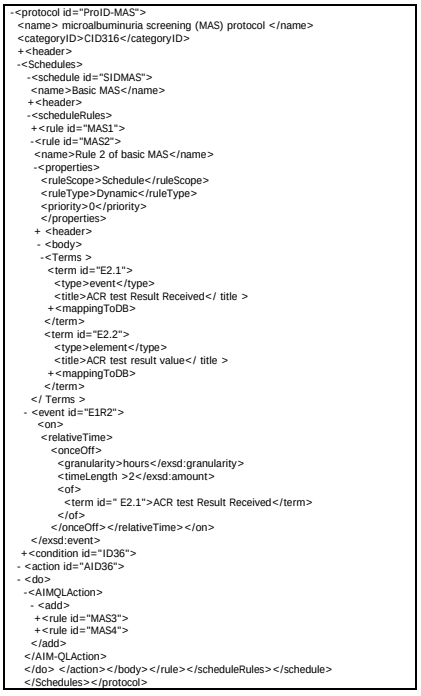

Fig. 2. The MAS protocol specified using AIMSL.

The event part of the AIMSL rules might be of type episode; absolute time; or relative time. The type episode means a domain event, such as “on patient admission”, or “on receiving ACR test result”. The event of type absolute time or relative time is a temporal event. The relative time event is a time event happening once or repeatedly and its time is related to a term of type event. Consider as example for once off event, on day 2 of patient admission and 2 hours before the completion time. Consider as example for repetitive event, every 3 days after patient admission for 10 times, or every 10 hours before the operation time. The relativeTime element has a complex type composed of a choice between two elements, namely, onceOff and every. The absolute time is such as first of June 2008.

The condition part is expressing a simple condition consisting of two operands and an operator. The term of type element could be used as an operand to express a condition, such as ACR test result is greater than 25. The action part might be a procedural action, such as sending email, or an AIMQL action for manipulating or querying the complex information. More complicated conditions and composite events are considered as part of the future work.

4.3 Example

Fig. 2 illustrates an example for an AIMSL specification of a simplified version of the microalbuminuria screening (MAS) protocol, which has a schedule containing two rules, MAS1 and MAS 2, as shown below. MAS2 defines a set of clinical

recommendation that should happen two hours after the result of the required test in MAS1 is received. As shown in Fig. 2, the action of the rule MAS2 adds two rules to the specification, MAS3 and MAS4. The both rules are similar to the rule MAS1, but they fire on day 6 and day 38 of the patient admission, respectively.

Rule MAS1: **ON** day 2 of the patient admission,
DO order the test albumin creatine ratio (ACR).

Rule MAS2: **ON** 2 hours after receiving the result of test ACR
IF the ACR result is greater than 25
DO order ACR test twice on days
number 6 and 38 of the patient admission

5 The Complex Information Model in AIM

This section outlines a model for the *complex information* (CI). The CI model is presented in terms of the life-cycle, and structure.

5.1 Life-cycle of the Complex Information

During the life-cycle of CI, the CI goes through state transitions, as shown in Fig. 3. These states are predefined and context-sensitive. The context-sensitive means that the CI's state is affected by changes in the domain information. When the CI is generated, it should be authorized to be registered or installed. In the registered state, all rules of the CI are installed in the system. In this state, no rule has fired yet. The CI moves to the "active" state once at least one rule is fired. The state "active" includes two sub-states, "waiting" and "executing". In the "waiting" state, at least one rule is fired and the other rules are waiting for events that are of interest to the CI. In the "executing" state, at least one rule is being executed. Once the rule execution completes, the CI returns to the "waiting" state. The CI might be transited from "active" state to "inactive", "terminated", or "completed" states. "inactive" state means that all the CI rules become disabled. The CI might be transited from "inactive" to "active" state. That means enabling the rules of the CI. "terminated" state means that all the CI rules removed from the system, but are not removed from the CI itself. When all the enabled rules in the CI are completed that means the CI is in the "completed" state. The "completed" state of the CI could be determined by a domain user, who is in charge of the CI. After the CI had become in the "completed" state, all the CI rules are removed from the system. It could be decided to re-register the CI again, after it had been terminated or completed.

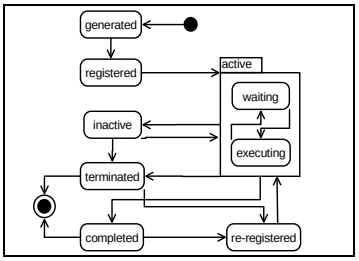

Fig. 3. A state diagram for the CI life-cycle.

5.2 Complex Information Schema

The CI consists of two main parts an active part and the passive part. The active part represents the reactive behaviour derived from the AIMSL specification. The passive part represents the descriptive information, state of the CI and its evolution since it has been created. The passive part is subject to actions that log the execution history of the CI. Therefore, CI grows over time. CI is subject to dynamically changes in order to suit the current conditions and constrains of interest to the domain user. Fig. 4 illustrates the XML Schema for the CI model in AIM. As shown in Fig. 4, the active part of the CI is represented as rules. Each rule is coded as a trigger or several triggers. The trigger(s) are used to register the rule in the system. The rest of the XML Schema shows the passive part. The passive part of the CI is modelled as time-varying information. The model captures the valid times of the fact recorded under the CI. That is leading to temporal relations among the CI and its components. On the other hand, the passive part shows the components of the CI, the validity period of their existence as a part of the CI, and their states. This model produces a temporal XML document, such as the document depicted in Fig. 5.

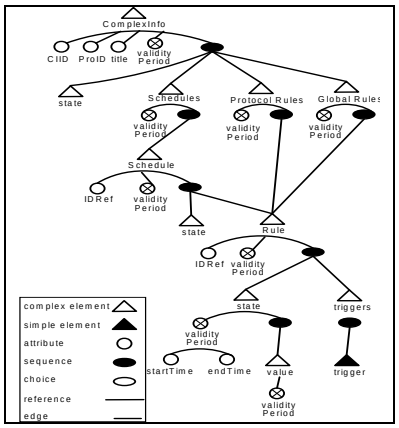

Fig. 4. The XML Schema for the Complex Information Model.

5.3 Example

A medical patient plan could be generated based on the specified protocol shown in Fig. 2. In the generation process, the rule body (`terms`, `event`, `condition` and `action`) is used to generate a trigger, which could be encoded using SQL, SQL/XML, or XQuery triggering language. Choosing the triggering language depends on the type of the database used to store the domain information, whether it is a relational or XML database.

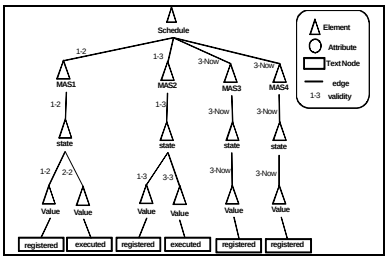

Fig. 5. A part of the patient plan on day number 4 of patient admission.

Assume 1) the medial patient plan is registered at time point 1; and 2) the result of ACR test is received on day 3 and its value is greater than 25. The action of MAS2 of the patient plan adds two new rules, MAS3 and MAS4, and then these changes are logged in the patient plan. Fig. 5 illustrates a portion of the patient plan on day 4. This portion has the history of the patient plan and its execution.

6 AIMQL: A High Level Query and Manipulation Language for the Complex Information

There is a need to move the complexity of manipulating and querying the best practice specified using AIMSL and its corresponding *complex information* from user/application code to a high level declarative language. AIMQL is a high level XQuery-based language provides facilities to perform manipulation operation, and advanced queries, such as replaying dynamic execution scenarios of the *complex information*.

6.1 Requirements

The main functional requirements of AIMQL are to assist in: 1) Manipulating the AIMSL specification and *complex information* (CI). The changes are made to AIMSL specification might be required to be propagated to the corresponding CI; and 2) Retrieving this information. This includes the ability to replay the CI or a specific part of it within specific time period. There are general functional requirements that should be also provided to AIMQL. These requirements are: 1) Declarativity, AIMQL should be declarative. It should be independent of any particular platform or query evaluation strategy. 2) Temporal Support, it should be able to record the history of executing the CI reactive behaviour and to query it. 3) XQuery-based, the AIMSL specification and CI are represented as XML document. Therefore, AIMQL should be based on XQuery. 4) Convenient for humans to read and write, this could be achieved using an XML-based graphical tool that assists in generating AIMQL query and browsing it. XML is easy to be generated using tools and easy to be converted to human readable format using a stylesheet language, such as XSL. Using XML in representing AIMQL provides a compatibility with AIMSL, and assists in managing the *complex information* remotely, using Web services.

6.2 Extensions to XQuery

Several extensions to XQuery are required in order to achieve the AIMQL requirements as following: 1) *Manipulation Operations*: AIMQL introduces seven manipulation operations (expressions). These expressions includes add, remove, modify, activate, deactivate, terminate and Fire. The AIMQL manipulation operations are distinguished in the sense that they not only potentially modify the AIMSL specification or CI, but also propagate the modification to the corresponding CI documents and modify the corresponding triggers created in the system. Furthermore, the manipulation expressions log the changes occurring to CI documents; 2) *Query Support*: AIMQL provides support to query AIMSL specification and CI document, as the domain information, plus special query capabilities, replay function and temporal query support for CI document, which is a temporal XML document.

AIMQL introduces a new functionality called replay. AIMQL replay query is a query that plays over again the history of the complex information to show in details the actions that cause changes on the complex information and how it evolved over time. Fig. 6 illustrates examples for AIMQL replay queries. These queries are presented here as patterns for AIMQL replay queries over medial patient plans, for short plans.

Replay Pattern 1:
Retrieve the history of the schedule no S1 of the plan no X, when the state of
the rule no R of schedule S2 was ST.
--AIMQL----------
REPLAY Complex Information CI
SHOW When, How, Why OF CI.schedule[@id = S1] CIS
Where CI[@CIID = X] and
CIS.overlaps
(valid(CI.schedule[@id=S2]\rule[@id=R]\state[value = ST]))
--Pattern Result----------
This replay pattern returns the versions of Schedule no S1 of the complex information no X, such that the validity of the version overlaps the validity period of the state ST of rule R in schedule S2.

Replay Pattern 2:
Replay the plans of category no CAT, which was working through out the past Y
days.
--AIMQL----------
REPLAY Complex Information CI
SHOW When, How, Why OF CI
Where CI.cast("day") >= Y
and CI.meets(NOW)
and CI[@catID=CAT]
--Pattern Result----------
This replay pattern returns the versions of the plans of category CAT, whose validity period meets the current time, and whose age is greater than or equal Y days.

Fig. 6. AIMQL replay patterns.

7 Implementation and Future Work

A combination of the ECA rule paradigm, XML, and database systems has been adopted as seamlessly integrated and easily incorporated technologies in the implementation method for AIM language. The poof-of-concepts implementation of the first vision of AIM language is currently in progress. The AIM language is being implemented using DB2, java, and XML technologies, such as XQuery and Web services. The DB2 XML database is used to store specifications based on AIMSL schema. Several complex information documents could be generated from a specific AIMSL specification based on AIM model for the *complex information*. The main phase in generating a complex information document is the mapping of the rules into triggers. The temporal events, which discussed in Section 4.2, are not supported by the available DBMSs. We have extended the triggering mechanism of DBMSs to support the temporal events. The instantiation and execution of specifications is based on SQL trigger mechanism in DB2. However, our extension is a generic approach that could be applied to other DBMSs. The mapping between AIMSL and the SQL trigger language is being developed using Java, SQL in DB2.The AIMQL replay queries are transformed into our temporal XQuery language that is under implementation.

8 Summary and Conclusion

Most languages for the formalisation and specification of best practice are designed outside the context of a comprehensive management framework, sometimes leading to difficulties in managing the specified information or the integration of specifications with domain information. This paper has presented a high level XML-based declarative language, called AIM, for supporting the SEM framework. The SEM framework manages best practice through three planes covering specification, execution, manipulation and querying. AIM language aims at enabling the specification, behaviour execution, manipulation and querying of the *complex information* arising from handling computerised best practice. AIMSL, the specification component of AIM language, uses the ECA rule paradigm to allow the formalisation and specification of best practice to be incorporated into an event-driven

mechanism using XML. The AIM model for *complex information* has been discussed. AIMQL is the manipulation and query component of AIM and is based on the XQuery language with promises to extend it with temporal facilities. Work on implementing the AIM language and evaluating it within the domain of clinical guideline management is currently on-going.

References

1. Dayal Umeshwar, Barbara T. Blaustein, Alejandro P. Buchmann, et al., The HiPAC Project: Combining Active Databases and Timing Constraints. SIGMOD Record, 1988. 17: p. 51-70.
2. Jennifer Widom and Stefano Ceri, Active Database Systems: Triggers and Rules For Advanced Database Processing. 1996: Morgan Kaufmann.
3. A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, et al., Developing scientific workflows from heterogeneous services. SIGMOD Rec., 2006. 35(2): p. 22-28.
4. M. Yusof Azwina, An on-line purchasing and decision support system for distributed retail chain stores, in Proceedings of the 6th international conference on Electronic commerce. 2004, ACM Press: Delft, The Netherlands.
5. Bing Wu, Essam Mansour, and Kudakwashe Dube. Complex Information Management Using a Framework Supported by ECA Rules in XML. in submitted to: International RuleML Symposium on Rule Interchange and Applications (RuleML 2007). 2007. Orlando, Florida.
6. Ajay Gupta, Manish Bhide, and Mukesh Mohania. Towards Bringing Database Management Task in the Realm of IT non-Experts. in Proceedings of the 19th International Conference on Data Engineering (ICDE'03). 2003: IEEE Computer Society.
7. Jun-Jang Jeng, Henry Chang, and Jen-Yao Chung. A Policy Framework for Business Activity Management. in IEEE International Conference on E-Commerce Technology (CEC'03). 2003.
8. Haifei Li, Stanley Y. W. Su, and Herman Lam, On Automated e-Business Negotiations: Goal, Policy, Strategy, and Plans of Decision and Action. Journal of Organizational Computing and Electronic Commerce, 2006. 16(1): p. 1-29.
9. A Paschke. ECA-RuleML/ECA-LP: A Homogeneous Event-Condition-Action Logic Programming Language. in Int. Conf. of Rule Markup Languages (RuleML'06). 2006. Athens, Georgia, USA.
10. Angela Bonifati, Daniele Braga, Alessandro Campi, et al. Active XQuery. in Proceedings of the 19th International Conference on Data Engineering ICDE. 2002. San Jose (California).
11. James Bailey, Alexandra Poulovassilis, and Peter T. Wood. An Event-Condition-Action Language for XML. in The 12th International World Wide Web Conference, www. 2002. Hawaii.