Replay the Execution History of Rule-based Information

Essam Mansour
International University in Germany
School of Information Technology
Campus 3, D-76646 Bruchsal, Germany
essam.mansour@ieee.org

Hagen Höpfner
International University in Germany
School of Information Technology
Campus 3, D-76646 Bruchsal, Germany
hoepfner@acm.org

Abstract

Software systems in health care, such as disease and medical-record management, or financial applications, such as customer relationship and portfolio management, have very often a temporal nature. Information is specified in form of rules as a pre-step for monitoring the changes of interest in the application. Managing such applications requires to provide replay support for managed information at any specified review period. This paper presents a replay support method for information formalized as rules. This method is based on an XML-based model and a replay query language for the rule-based information. We introduce the model and the language using a clinical case study, and evaluates the storage efficiency of the model and the performance of the query.

1. Introduction

Several applications of information systems are temporal in nature. Examples include health care applications, such as disease and medical-record management, and financial applications, such as customer relationship and portfolio management. Information of these applications is specified in form of rules as a pre-step for monitoring the changes of interest in the application. In health-care, several approaches, such as Arden Syntax [12], HyperCare [8], PLAN [13], and AIM [18], have been presented to specify and execute heath-care information using the Event-Condition-Action (ECA) rules paradigm [21]. Furthermore, financial information has been managed using ECA rule-based approaches, such as [2, 7, 9].

The execution history of the rule-based information represents several information scenes. The ability of replaying these scenes enhances the reporting and decision-support capabilities in the organization. The replay support facilitates the information analysis and mining to discover and understand information trends. Providing a replay support management for the rule-based information as an extension for the modern database systems (DBSs) is the main topic of investigation in this paper.

The starting point for information analysis and mining is to keep the evolution history of such information and to provide the ability to replay this history at a high level. The replay support is to help to find out information, such as the time at which the rule-based information became active, at which a rule was executed, why it was executed, what was the action made, and how many times a rule was executed. Therefore, the replay support provides a motion picture that depicts the evolution of a specific task or activity supported by such information.

There is a need to move the complexity of manipulating and querying the rule-based information and its execution history from user/application code to a high level declarative language. This paper presents a replay support management that is based on XML technologies, such as XML Schema and XQuery, and is supported within the modern DBSs. This replay support management provides a model for representing the rule-based information and capturing its execution history, and a replay query language for replaying the evolution of such information.

The reminder of this paper is organized as follows: Section 2 discusses related work. Section 3 presents an XML-based model for the rule-based information and its execution history. Section 4 introduces our replay query language that provides a high-level reply support for the evolution of the rule-based information, Section 5 presents our developed system that utilizes modern DBSs to implement our model and language. Section 6 includes first experimental evaluation results. Section 7 concludes the paper and gives an outlook on future research.

2. Related Work

In the area of active XML, an event-driven mechanism based on the Event-Condition-Action (ECA) rule paradigm [21] is incorporated into XML to provide an advanced active behavior. We have classified the languages, which have been produced in this area, into three categories: The languages in the *first category* play the same role as the high level SQL trigger standard, such as Active XQuery [6], or an Event-Condition-Action language for XML [5]. These languages support the reactive applications at the level of rules and triggers. The languages in the *second category*, such as AXML [1], Active XML Schemas [23], and XChange [4], utilize the event-driven mechanism to support a specific reactive application, such as the Web content management. The languages of the *third category* use XML only to standardize the rule-based information as individual rules, such as ARML [11].

The RuleML languages aim at providing a standard rule language that is interoperability platform [26, 27]. Wanger in [26] classified the ECA rule paradigm as a subtype of the RuleML language. The RuleML language has been utilized to support semantic Web and business applications, such as in [20, 22]. However, the RuleML languages formalize the rule-based information as individual rules, not as a unified distinct entity that could be instantiated. Languages proposed in both areas, active XML and RuleML, overlook the need to keep and replay the evolution history of the rule-based information as a pre-step to analyze and mine the execution history of such information.

Managing real situations in reactive applications requires temporal support, such as temporal triggers and events, as discussed in HiPAC [25]. The support for temporal condition over the history of non rule-based information was studied by Sistla in [24]. Temporal storage models, such as [3], and temporal query support, such as [14, 19], are proposed in the area of temporal XML. There is a lack of support for querying temporal XML data that represents rule-based information. A temporal support is required to store and retrieve the history of such information.

The replay support management presented in this paper is part of the AIM language, which has been developed by Mansour [15, 16] for specifying, instantiating and maintaining rule-based information. The AIM language is based on XML and ECA rule paradigm [18], and has been implemented using a system, called AIMS [17] that extends the DBS utilities to support the advanced management required for the rule-based information. Our replay support provides a high-level method for playing over again the history of the rule-based information.

3. A Model for the Rule-Based Information

The rule-based information is specified in a generic form as a skeleton for generating plans suiting a specific domain entity. An example is the generic specification of the test ordering protocol developed for diabetic patients, from which several patient plans are generated to suit particular patients. For more details, the reader is referred to [18, 16, 15].

This section presents a model, called DRDoc, for formalizing the plans generated for specific domain entities and their evolution as a temporal XML document. DRDoc stands for *D*ynamic *R*ule-based *Doc*ument. It consists of two main parts; an *active part* and *passive part*.

The *active part* represents the reactive behavior of the rule-based information, and is formalized as rules, which are coded as a trigger or several triggers. These triggers are used to register the rule-based information in the system. The *passive part* represents the descriptive information, which represents the states of the rule-based information and its evolution. This part is subject to actions that log the execution history of the rule-based information. Therefore, the DRDoc document grows over time. The *passive part* is modelled as time-varying information.

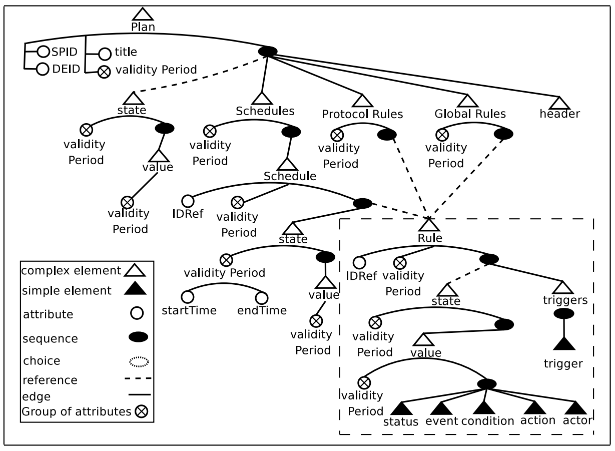

Figure 1. The XML Schema of DRDoc.

3.1. The XML Schema of DRDoc

The main part of the DRDoc model is the *rule* element as shown in Figure 1 that illustrates the XML Schema of the model. The rules representing the rule-based information are classified into groups (*schedule, protocol* and *global*) according to their objective and scope. Each *rule* element contains a trigger or several triggers. The *state* element of a rule records several values that show the status of the rule, such as *generated*, *registered*, *executed* or *inactive*, the actual runtime evaluation of the rule components, *event*, *condition* and *action*, and who participated or causing the execution of the rule (*actor*). However, the *state* element under the *schedule* and *plan* elements consists of value representing the state and validly period of such state.

The *evolution history* of the rule-based information is presented as *state* element. The validity period represents a time period, during which a component existed as a part of the DRDOC document. A temporal XML support is needed to realize the DRDOC model. The *header* element provide descriptive information, such as the person in charge of the plan. Each plan is generated from a specific skeletal plan, whose ID is *SPID*, for a particular domain entity, whose ID is *DEID*, with a title, such as *patient plan*, and has a period within it the plan was valid.

3.2. A Maintenance Mechanism

The ECA rule paradigm as implemented in database systems is a promising technology for supporting the execution method of the DRDOC model. The DBSs provide support for the active mechanism using triggers. Once the rules of the DRDOC document are registered (installed) in the DBS, the DBS becomes in charge of executing the triggers representing these rules. Utilizing the database triggering mechanism facilitates the integration of the DRDOC model into the system managing the domain information, such as the patient information system that manages the electronic health care record.

The DRDOC model and its component are joined with a validity period. That period refers to the period of existence, in which the component considered as part of the DRDOC document. It is assumed that the valid time, which is the time when the fact was true in the reality, is equal to the transaction time, which is the time when the fact was stored in the database. The validity period is represented as a tuple,$<$ start time, end time $>$. If the component has the validity period $< 5, NOW >$, the component is currently part of the DRDOC document since the time point 5.

Assume at time T_2, the state of a component having a state S_1 with validity period $< T_1, NOW >$, is changed to S_2. Then the new state is added to the component with validity period $< T_2, NOW >$ and the validity period of the old state will be $< T_1, T_2 >$. That means at time T_2, the state of the component is changed from S_1 to S_2. The validity period of a component, such as *schedule* or *plan*, is equal to $<$ minimum (start time), maximum (end time) $>$ of its sub-components.

3.3. An Example

We assume that there is a skeletal plan for diabetic patients, which consist of two rules MAP_1 and MAP_2 grouped at the same schedule. The rule MAP_1 is to be fired two hours after patient admission to order an Albumin/Creatinine Ratio (ACR) test for the patient. The rule MAP_2 is to be fired once the result of the ACR test is received, if the result is greater than 25 then repeat the ACR test every two days after patient admission, as an action. This action is formalized as a new rule (MAP_3).

For each diabetic patient, a patient plan is generated from the generic plan by customizing the rules to a particular patient, i.e. using the admission time of the patient. It is assumed that the patient plans were registered at time point 1, and the result of the ACR test is received three hours after patient admission. The result was 33 and is greater than 25.

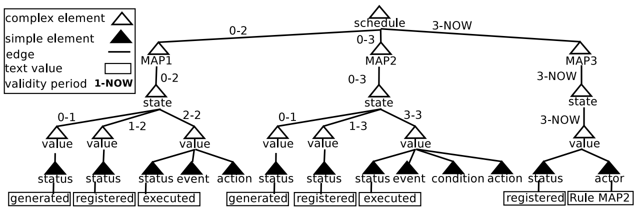

Figure 2. A part of a patient plan.

According to this scenario, the patient plan was maintained as shown in Figure 2 that illustrates part of the patient plan four hours after the patient admission. The rule MAP_1 and MAP_2 were generated at time point zero and registered at time point 1. The *generated* status is a system-defined status that happens at the generation time of an entity-specific plan. Therefore, there is no need to the other elements, such as *actor* or *event*. The rule MAP_1 was fired two hours after patient admission. Therefore, the status *registered* of rule MAP_1 was valid from 1 to 2. The status *executed* was added with validity period 2 to 2. The actual evaluation of the *event* and *action* of MAP_1 were recorded. The rule MAP_2 was fired three hours after patient admission. Therefore, the status *registered* of rule MAP_2 was valid from 1 to 3. The status *executed* was added with validity period 3 to 3. The evaluation of the *event, condition*, and *action* of MAP_2 were recorded. The rule MAP_3 was added at time point 3 and it is recorded that MAP_2 caused such modification.

4. A Replay Query Support

This section presents our replay query language, which called AIMQL (**A**dvanced **I**nformation **M**anagement **Q**uery **L**anguage), and examples that demonstrate the AIMQL capabilities. The rule-based information and its execution history are represented and stored as DRDOC document, which is a temporal XML document compatible with the XML model. Therefore, any XQuery engine could be used to query them. However, querying the DRDOC documents demands special query operators, which are capable of querying the history at a declarative and high level.

4.1. AIMQL Replay Language

The AIMQL replay language is a language that plays over again the history of the DRDOC documents to show in details the actions that were carried out during the execution of the rule-based information. Figure 3 illustrates the XML Schema of the AIMQL replay query, which consists of main three clauses *REPLAY*, *SHOW* and *WHERE*.

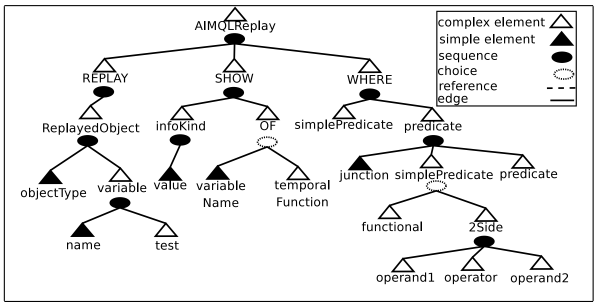

Figure 3. The AIMQL replay query structure.

The *REPLAY* clause indicates the element that is subject to be replayed. The replay queries are applied only to the *plan*, *schedule* and *rule* elements, which are called *replayable* elements. Two or more *replayable* elements might be joined (combined) in order to produce the query result. By mapping the replay query into XQuery script, the utilized XQuery engine is to be in charge of managing the join operation.

As shown in Figure 3, the *REPLAY* element consists of a sequence of *ReplayedObject* elements, which consists of a sequence of elements (*objectType* and *variable*). The *objectType* element is of token type restricted to values (*plan, schedule* and *rule*). The *variable* element consists of a sequence of elements (*name* and *test*). The *name* element refers to the variable name, which should be unique. The *test* element is used to specify a condition that restricts the variable to a specific *plan, schedule* or *rule*.

The *SHOW* clause determines which pieces of information are to be returned. As shown in Figure 3, *SHOW* element consists of a sequence of elements (*infoKind* and *OF*). The *OF* element specifies *replayable* elements, whose replayed parts are to be specified using the *infoKind* element and to be returned as a part of the query result.

OF might refer directly to the *replayable* element, which should be defined in the *REPLAY* clause, or by using one of the temporal functions supported by AIMQL. The *OF* element consists of a sequence of elements (*variableName*, *temporalFunction*). The *variableName* element refers to one of the defined variables under the *REPLAY* clause. The *temporalFuction* element calls one of the supported temporal functions, such as *overlaps, meets, first, valid* and *cast*.

The *infoKind* element consists of at least one *value* element, which is restricted to the values (*when*, *why*, *who*, *how*, and *what*). These values are used as an indicator to specify the information of interest to the user, as follow:

the *when* value is an indicator to show the validity period;

the *why* value is an indicator to show the event that causes the firing, and the condition evaluated in order to execute the rule;

the *who* value is an indicator to show the actor participating in performing the rule;

the *how* value is an indicator to show the action carried out;

the *what* value is an indicator to show the corresponding specification (the skeletal plan), from which the plan is generated.

The *WHERE* clause includes a comparison predicate, which is used to restrict the number of elements returned by the query. The *WHERE* clause eliminates all rows from the result set where the comparison predicate does not evaluate to true. As shown in Figure 3, *WHERE* element has a complex type that consists of a sequence of elements(*simplePredicate* and/or *predicate*). The *simplePredicate* element specifies a simple predicate that calls one of the temporal function, or a two side predicate that is consists of two operands connected by an operator. The *predicate* element is used in the case of dealing with composite predicate that is consists of two simple predicates connected by a junction, which is *and* or *or*. The *predicate* element is a recursive element that calls itself in order to support *N* number of composite predicates.

4.2. Examples

AIMQL is to be used to find out information that assists the analysis and decision-making process. AIMQL queries specify declaratively what is the user is interested to review. Several examples are depicted in Figure 4.

In replay query 1, it is required to retrieve the history of the patient plan no *(@DEID,@SPID) (X,PID)* over the period from *TP1* to *TP2*. In this query the variables *X, PID, TP1*, and *TP2* are to be replaced with appropriate values. The replay query returns *N* versions of the plan over the mentioned period. Replay query 2 retrieves the the first version of the plan no (*@DEID,@SPID) (X,PID)*.

In replay query 3, it is required to retrieve How many times was the rule *R* of the schedule *S* of the plan (*X,PID*) executed, and why. The *OF* element specifies the replayable information using the function *count*, which counts the states, whose value is *executed*, of the rule *R* of the schedule *S* of the plan no (*X,PID*). This query shows *how* and *why* parts of the replayable information. So, the actual evaluation of the *event* and *condition* elements are to be shown for each execution of *R*. The replayed period is the period, at which *R* was executed, as specified using the function *meet*.

replay query 1

```
REPLAY PLAN p1
SHOW When OF p1
WHERE p1[@DEID = X and @SPID = PID]
      and p1.overlaps(valid(TP1,TP2))
```

replay query 2

```
REPLAY PLAN p1
SHOW When OF FIRST(p1)
WHERE p1[@DEID = X and @SPID = PID]
```

replay query 3

```
REPLAY RULE plan[@DEID = X and @SPID = PID]
            //schedule[@IDREF=S]/rule[@IDREF=A] R
SHOW How, Why OF count(R.state[value/status='executed']
WHERE R.meet(valid(R.state[value/status='executed']))
```

replay query 4

```
REPLAY PLAN p1,p2
SHOW When, How, Why OF p1
WHERE p1[@DEID = X1 and @SPID = PID1] and
      p2[@DEID = X2 and @SPID = PID2] and
      NOT(p1.precedes(valid(p2.state[value=ST])))
```

Figure 4. AIMQL replay queries.

Replay query 4 replays the history of plan no *(X1,PID1)* after the validity period of the state *ST* of the plan no *(X2,PID2*). In this query the variables *X1, PID1, ST, X2,* and *PID2* are to be replaced with appropriate values. This replay query returns the versions of the plan no *(X1,PID1*), whose validity period does not precede the validity period of the state *ST* of the plan no *(X2,PID2*). This query helps in comparing the progress of two different patients, to who the same generic plan is applied.

5. A Prototype System

We have utilized the modern DBSs that provide XML data management and triggering mechanism, as discussed in [17, 18], to develop a proof-of-concept system, called *AIMS* [17], for managing the rule-based information. AIMS maps the AIMQL replay queries into XQuery scripts that are to be executed by the DBS.

5.1. Conceptual Architecture

The conceptual architecture of *AIMS* is illustrated in Figure 5. AIMS was implemented using DB2 and Java. The main components of *AIMS* are the *Complex Information Manager*, *Rule Manager*, *Information Manager*, and *Communication Manager*. The *Complex Information Manager* supports the management of the rule-based information at a high level. The domain users and information providers,

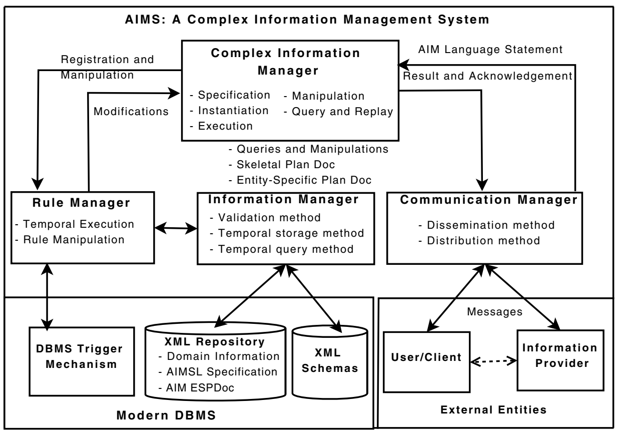

Figure 5. the AIMS conceptual architecture.

such as patient information systems or stock items systems, deal with *Complex Information Manager* through the *Communication Manager*. The *Rule Manager* extends the triggering mechanism of the DBS to support the advanced features of the rule-based information, and recording the execution history. The *Information Manager* extends the XML support provided by DBSs to provide temporal support and utilizes the DBS to validate and store the DRDoc document using its XML Schema shown in Figure 1. One of the main roles of *Information Manager* is to map the AIMQL queries into XQuery scripts.

5.2. Mapping Replay Queries into XQuery

The AIMQL replay query is a declarative query, which means that the user does not need to know the structure of the rule-based information. The translator knows the structure (elements and attributes) of the DRDoc documents. The translator generates the equivalent XQuery script that constructs a new XML document as a result of the AIMQL replay query. Each part of an AIMQL replay query is translated into its corresponding XQuery. The AIMQL replay query consists of *REPLAY*, *SHOW*, and *WHERE*.

The AIMQL variables are defined in the *REPLAY* clause to access specific elements that are subject to be replayed. These elements must be of type *plan*, *schedule*, or *rule*. The variables might be restricted using a specific condition, which might be specified in the *REPLAY* or *WHERE* clauses. If the variable appears in the *SHOW* clause that means there is a need to set up an iteration through the element associated to the variable. For example, if a variable *P1* is of type *plan*, that needs to set up an iteration through the *plan* element and its sub-elements, such as *schedule* and *rule*. Setting up iterations through these sub-element is implicitly demanded.

The XQuery provides the *FOR* clause support the iteration. Consequentially, the variables appear in the *SHOW* clause are defined within an XQuery *FOR* clause. However, if variables do not appear in the *SHOW* clause, its expressions in the AIMQL replay query are defined within an XQuery *LET* clause, which binds the variable to a specific value. For example, the AIMQL replay query 4 defines two variables of type *plan*, *p1* and *p2*, as shown in Figure 4. The variable *p1* is used in the *SHOW* clause. The variable *p2* is used only in one expression in the *WHERE* clause.

XQuery uses functions, such as *doc* and *collection*, to access XML documents from within a query [28]. In DB2, an XQuery can obtain input data by calling a function named *db2-fn:xmlcolumn* with a parameter that identifies the table name and column name of an XML column in a DB2 table [10]. The tables representing the AIMS XML repository are AIM_ESPlan_TAB, AIM_Protocol_TAB, and AIM_GlobalRules_TAB. These tables include attributes of XML type to store the rule-based information formalized using the DRDOC model.

```
XQUERY
declare namespace xsd =”http://www.w3.org/2001/XMLSchema”;
for $p1 in
db2-fn:xmlcolumn(’AIM_ESPlan_TAB.ESPDOC’)//Plan[@DEID=X and @SPID=PID]
where(
( xsd:dateTime($p1/@startTime) = xsd:dateTime($p1/@startTime) )
)
return
<Plan domainEntity_ID=”{$p1/@DEID}” protocol_ID=”{$p1/@SPID}”
startTime=”{$p1/@startTime}” endTime=”{$p1/@endTime}”>
{for $PState in $p1/state
return <state startTime=”{$PState/@startTime}” endTime=”{$PState/@endTime}”>
{ $PState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)] }</state> } {for $PSches in $p1/schedules
return <schedules startTime=”{$PSches/@startTime}” endTime=”{$PSches/@endTime}”>
{ for $sch in $PSches/schedule
where (
( xsd:dateTime($sch/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <schedule IDREF=”{$sch/@IDREF}” startTime=”{$sch/@startTime}” endTime=”{$sch/@endTime}”>
<scheduleRules startTime=”{$sch/scheduleRules/@startTime}” endTime=”{$sch/scheduleRules/@endTime}”>
{
for $rul in $sch/scheduleRules/rule
where (
( xsd:dateTime($rul/@startTime) = xsd:dateTime($p1/@startTime) )
)
return <rule IDREF=”{$rul/@IDREF}” startTime=”{$rul/@startTime}” endTime=”{$rul/@endTime}”>
{for $RState in $rul/state
return <state startTime=”{$RState/@startTime}” endTime=”{$RState/@endTime}”>
{
$RState/value[( ( xsd:dateTime($p1/@startTime) <= xsd:dateTime(@endTime) ) and
( xsd:dateTime(@startTime) = xsd:dateTime($p1/@startTime) )
)]
}</state>
}</rule>
}</scheduleRules></schedule>
}</schedules>
} </Plan>
```

Figure 6. The XQuery generated to query 2.

5.3. An Example

Figure 6 depicts an the XQuery script for the AIMQL replay query 2 shown in Figure 4. This XQuery script returns a complete DRDOC document that represents the initial plan of the domain entity **X** and this plan is generated from the protocol *PID*. The XQuery statement in DB2 starts with the key word *XQuery*, as shown in Figure 6. It is needed to define the name space used to execute this query, which is the standard W3C XML Schema. This name space is defined using the key word *declare*.

This query has only one variable, *p1*, of type *plan*. Using the variable template, a *FOR* clause is generated to define the XQuery variable **$p1** that iterates over the plan, whose *DEID* attribute is *X* and *SPID* attribute is *PID*. The function *db2-fn:xmlcolumn* is used to access the plans stored in the *AIM_ESPlan_TAB.ESPDOC* table.

6. Evaluation

This section discusses the experiments that evaluate the storage efficiency and the execution performance of DRDOC and AIMQL, respectively. These experiments have been tested on Debian 4, a Linux system, and an Intel Pentium III processor machine, whose configuration is one Gigabyte RAM and 40 Gigabyte hard disk.

6.1. The storage efficiency

The DRDOC document is a temporal XML document that records all the changes produced by updating the DRDOC document. Most of these changes add a new state to an element of the DRDOC document. For example, executing the rule MAP_3 every two days adds a new *executed* state under the *rule* element. These changes might be also adding a new rule, such as MAP_2 that adds a new rule, MAP_3. Consequentially, the storage management of the DRDOC document is of critical importance and the main factor of the AIMS storage management performance.

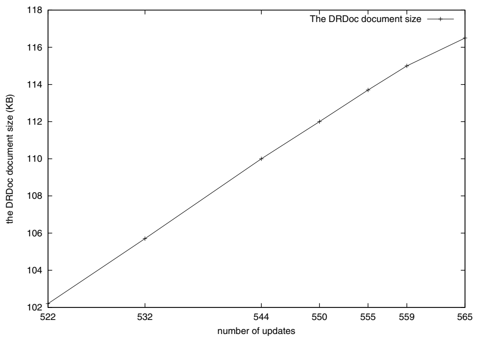

Figure 7. The storage efficiency.

This experiment compares the size of the DRDOC document with the number of updates that take place in them. The growing in the plan size is almost linear to the number of updates, as shown in Figure 7. This linearity assists in estimating the DRDOC document size after *N* number of updates, such that most of the updates are changes on the rule state. Such change is represented using a *value* element of almost fixed size, see Figure 1. In conclusion, the AIMS storage management is stable to the number of updates.

6.2. The Query Performance

AIMS translates the AIMQL replay queries into a pure XQuery scripts, which is to be executed by the DB2 XQuery engine. DB2 provides different tools, such as *db2batch*, to analyze the runtime performance of queries. The *db2batch* returns the elapsed time spent for executing the given query. The size of the DRDOC document is of critical importance and the main factor of the AIMS query performance because the DRDOC documents grow over time.

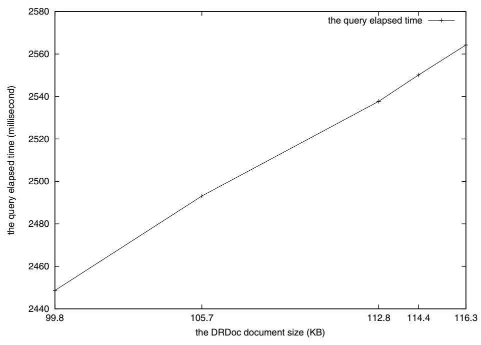

Figure 8. The query performance.

This experiment compares the query execution elapsed time with the size of DRDOC documents, which accessed in the query. The experiment is achieved using a complicated query, which accesses a DRDOC document and scans it three time for calculating the number of executing its rules and returning the recent instance of the plan represented in this document. The query runtime performance is almost linear to the size of the DRDOC document participating in the query, as shown in Figure 8.

7. Conclusion and Outlook

This paper has presented a replay support management for information formalized as rules. This replay support management is based on an XML-based model, called DRDOC, and a replay query language, called AIMQL. The DRDOC model formalizes the rule-based information and its execution history as one temporal XML document. The AIMQL replay language is a language that plays over again the history of the DRDOC documents to show in details the actions that were carried out during the execution of the rule-based information.

The AIMQL is distinguished by the ability to specifying declaratively the reviewed history of the rule-based information at a high level. Furthermore, the AIMQL is able to use the components, such as *plan, schedule* or *rule*, of the rule-based information as first class object. The paper highlights a proof-of-concept system, called AIMS, that maps the AIMQL replay queries into XQuery scripts that are to be executed by the DBS. The evaluation of the AIMS storage and query performance shows positive results.

Currently we are doing additional experiments with different workloads and query sets. There are also several future work concerning the replay method presented in this paper. The AIMQL replay queries return a temporal XML document, which represents the replayed execution history of the rule-based information. This replayed information is visualized as a text that could be browsed using any XML or Web browser. This visualization mechanism is primitive and does not provide high level view. It is needed to develop an advanced graphical visualization mechanism to review the replayed information in a way similar to a movie player. Morever, there is a need to develop a method that provides automatic discovery of information from the execution history of the rule-based information. This discovered information can be assist in auditing, analyzing and improving already enacted rule-based information.

References

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: A Data-Centric Perspective on Web Services. In M. Levene and A. Poulovassilis, editors, *Web Dynamics - Adapting to Change in Content, Size, Topology and Use*, pages 275–300. Springer, 2004.

[2] A. Adi, D. Botzer, G. Nechushtai, and G. Sharon. Complex Event Processing for Financial Services. In *Proceedings of the IEEE Services Computing Workshops*, pages 7–12, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.

[3] T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In M. T. Ibrahim and J. K. Revell, editors, *Proceedings of the 11th International Conference on Database and Expert Systems Applications*, volume 1873 of *Lecture Notes In Computer Science*, pages 334–344, London, UK, 2000. Springer-Verlag.

[4] J. Bailey, F. Bry, M. Eckert, and P.-L. Pătrânjan. Flavours of XChange, a Rule-Based Reactive Language for the (Semantic) Web. In A. Adi, S. Stoutenburg, and S. Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3791/2005 of *Lecture Notes in Computer Science*, pages 187–192, Berlin/Heidelberg, 2005. Springer.

[5] J. Bailey, A. Poulovassilis, and P. T. Wood. An Event-Condition-Action Language for XML. In *Proceedings of the 11th international conference on World Wide Web*, pages 486–495, New York, NY, USA, 2002. ACM.

[6] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proceedings of the 18th International Conference on Data Engineering*, page 403, Washington, DC, USA, 2002. IEEE Computer Society.

[7] F. Bry, M. Eckert, P.-L. Pătrânjan, and I. Romanenko. Realizing Business Processes with ECA Rules: Benefits, Challenges, Limits . In J. J. Alferes, J. Bailey, W. May, and U. Schwertel, editors, *Principles and Practice of Semantic Web Reasoning*, volume 4187/2006 of *Lecture Notes in Computer Science*, pages 48–62, Berlin/Heidelberg, 2006. Springer.

[8] P. V. Caironi, L. Portoni, C. Combi, F. Pinciroli, and S. Ceri. HyperCare: a Prototype of an Active Database for Compliance with Essential Hypertension Therapy Guidelines. In D. R. Masys, editor, *Proceedings of the AMIA Annual Fall Symposium*, pages 288–292. Hanley & Belfus, Inc, 1997. available online `http://www.amia.org/pubs/proceedings/symposia/1997/D004024.pdf`.

[9] R. Chandra, S. Arie, and W. A. Haas. Active Databases for Financial Applications. In J. Widom and S. Chakravarthy, editors, *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering*, pages 46–52, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[10] W.-J. Chen, A. Sammartino, D. Goutev, F. Hendricks, I. Komi, M.-P. Wei, and R. Ahuja. *DB2 9 pureXML Guide*. IBM Redbooks, 1st edition, January 2007. available online `http://www.redbooks.ibm.com/redbooks/pdfs/sg247315.pdf`.

[11] E. Cho, I. Park, S. J. Hyun, and M. Kim. ARML: an active rule mark-up language for heterogeneous active information systems. In M. Schroeder and G. Wagner, editors, *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, volume 60 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2002.

[12] P. D. Clayton, T. A. Pryor, O. B. Wigertz, and G. Hripcsak. Issues and Structures for Sharing Medical Knowledge Among Decision-making Systems: The 1989 Arden Homestead Retreat. In *Proceedings of the 13. Annual Symposium on Computer Applications in Medical Care*, pages 116–121, Los Alamitos, CA, USA, Nov. 1989. IEEE Computer Society Press.

[13] K. Dube, B. Wu, and J. B. Grimson. Using ECA Rules in Database Systems to Support Clinical Protocols. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, volume 2453/2002 of *Lecture Notes in Computer Science*, pages 226–235, Berlin/Heidelberg, 2002. Springer.

[14] D. Gao and R. T. Snodgrass. Temporal Slicing in the Evaluation of XML Queries. In J.-C. Freytag, P. C. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, editors, *Proceedings of the 29th International Conference on Very Large Databases*, pages 632–643, St. Louis, MO, USA, 2003. Morgan Kaufmann Publishers/Elsevier Science. available online `http://www.vldb.org/conf/2003/papers/S19P03.pdf`.

[15] E. Mansour. *A Generic Approach and Framework for Managing Complex Information.* PhD thesis, Dublin Institute of Technology (DIT), 2008. available online `http://arrow.dit.ie/sciendoc/51/`.

[16] E. Mansour, K. Dube, and B. Wu. AIM: An XML-Based ECA Rule Language for Supporting a Framework for Managing Complex Information. In A. Paschke and Y. Biletskiy, editors, *Proceedings of the International Symposium on Advances in Rule Interchange and Applications*, volume 4824/2007 of *Lecture Notes in Computer Science*, pages 232–241, Berlin/Heidelberg, 2007. Springer.

[17] E. Mansour, K. Dube, and B. Wu. Managing complex information in reactive applications using an active temporal XML database approach. In J. Cardoso, J. Cordeiro, and J. Filipe, editors, *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 520–523, 2007.

[18] E. Mansour, B. Wu, K. Dube, and J. X. Li. An Event-Driven Approach to Computerizing Clinical Guidelines Using XML. In *Proceedings of the IEEE Services Computing Workshops*, pages 13–20, Washington, DC, USA, 2006. IEEE Computer Society.

[19] A. O. Mendelzon, F. Rizzolo, and A. A. Vaisman. Indexing Temporal XML Documents. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 216–227, St. Louis, MO, USA, 2004. Morgan Kaufmann Publishers/Elsevier Science. available online `http://www.vldb.org/conf/2004/RS6P1.PDF`.

[20] C. Nagl, F. Rosenberg, and S. Dustdar. VIDRE–A Distributed Service-Oriented Business Rule Engine based on RuleML. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pages 35–44, Washington, DC, USA, 2006. IEEE Computer Society.

[21] N. W. Paton, editor. *Active Rules in Database Systems*. Springer, New York, 1999.

[22] E. Pontelli, T. C. Son, and C. Baral. A Framework for Composition and Inter-operation of Rules in the Semantic Web. In *Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web*, pages 39–50, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.

[23] M. Schrefl and M. Bernauer. Active XML Schemas. In H. Arisawa, Y. K. V. K. H. C. Mayr, and I. Hunt, editors, *Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops*, volume 2465 of *Lecture Notes In Computer Science*, pages 363–376, London, UK, 2001. Springer.

[24] A. P. Sistla and O. Wolfson. Temporal conditions and integrity constraints in active database systems. *ACM SIGMOD Record*, 24(2):269–280, May 1995.

[25] D. Umeshwar, B. T. Blaustein, A. P. Buchmann, U. S. Chakravarthy, M. Hsu, R. Ledin, D. R. McCarthy, A. Rosenthal, S. K. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC Project: Combining Active Databases and Timing Constraints. *SIGMOD Record*, 17(1):51–70, 1988.

[26] G. Wagner, G. Antoniou, S. Tabet, and H. Boley. The Abstract Syntax of RuleML – Towards a General Web Rule LanguageFramework. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, pages 628–631, Washington, DC, USA, 2004. IEEE Computer Society.

[27] G. Wagner, A. Giurca, and S. Lukichev. A General Markup Framework for Integrity and Derivation Rules. In F. Bry, F. Fages, M. Marchiori, and H.-J. Ohlbach, editors, *Principles and Practices of Semantic Web Reasoning*, number 05371 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI).

[28] P. Walmsley. *XQuery*. O'Reilly, 1st edition, March 2007.