# An Intelligent Educational Metadata Repository[◊]

[*]Nick Bassiliades[§], [*]Fotios Kokkoras[¥], [*]Ioannis Vlahavas, [**]Dimitrios Sampson

[*] Dept. of Informatics
Aristotle University of Thessaloniki
54006 Thessaloniki, Greece
{nbassili,kokkoras,vlahavas}@csd.auth.gr

[**] Informatics and Telematics Institute
1 Kyvernidou Str.
54639 Thessaloniki, Greece
sampson@ath.forthnet.gr

## Abstract

Recently, several standardization efforts for e-learning technologies gave rise to various specifications for educational metadata, that is, data describing all the "entities" involved in an educational procedure. The internal details of systems that utilize these metadata are still an open issue since these efforts are primarily dealing with "what" and not "how". In this chapter, under the light of these emerging standardization efforts, we present X-DEVICE, an intelligent XML repository system for educational metadata. X-DEVICE can be used as the intelligent back-end of a WWW portal on which "learning objects" are supplied by educational service providers and accessed by learners according to their individual profiles and educational needs. X-DEVICE transforms the widely adopted XML binding for educational metadata into a flexible, object-oriented representation and uses intelligent second-order logic querying facilities to provide advanced, personalized functionality. Furthermore, a case study is presented, in which learning object metadata and learner's profile metadata are combined under certain X-DEVICE rules in order to dynamically infer customized courses for the learner.

[◊] Part of the work presented in this chapter was partially financially supported by the European Commission under the IST No 12503 Project "KOD - Knowledge on Demand" through the Information Society Technologies Programme (IST).

[§] Supported by a scholarship from the Greek *Foundation of State Scholarships* (F.S.S. - I.K.Y.).

[¥] Supported by a scholarship from the *Informatics and Telematics Institute*.

# 1. INTRODUCTION

Educational applications are among the most promising uses of the World Wide Web. There is already a large amount of instructional material on-line, most of it in the form of static multimedia HTML documents, some of which are enriched with a level of interactivity via Java-based technologies. Unfortunately, the majority of these approaches has been built on general-purpose, non-educational standards and fails to utilize Web's great potential for distributed educational resources that are easily located and interoperate with each other.

Although computers have been used in education for almost 20 years, in the form of Computer based Training (CBT) and Computer Assisted Instruction (CAI), the level of sophistication of such systems was rather primitive, since they were primarily dealing with the sequencing of the instructional material. In contrast to this, research in information technology assisted education has reached sophisticated levels during 90's, taking into account issues like pedagogy, individual learner and interface, apart from the basic educational material organization. Following the recent "e-" trend, these approaches are just beginning to appear on the Internet. The reason for this late adoption is mainly the substantial effort that is required to bring them on the Web since all of them have been designed without the Web in mind.

On the other hand, it is commonplace that our society has already moved away from the "once for life" educational model. The complexity and continuous evolution of modern enterprises' activities requires continuous training of their personnel. The networked community enables the management and enhancement of knowledge in a centralized - yet personal way, while keeping track and merging new intellectual resources into that process.

The above requirements and advances lead us to the "Lifelong Learning" concept. The idea is to integrate the WWW technology with a novel, dynamic and adaptive educational model for continuous learning. The result will be a learning environment that will enable the individual learner to acquire knowledge just in time, anytime and anywhere, tailored to his/her personal learning needs.

In our days, there is an ongoing, large-scale effort on improving the interoperability over the Internet. The eXtended Mark-Up Language (XML - [43]) is massively used to define the semantics of the information required to achieve the so-called *Semantic Web* idea [44]. In the domain of education, these efforts are primarily dealing with the definition of educational metadata, that is, data describing all the "entities" involved in an educational process. The internal details of systems that utilize these metadata are still an open issue, since these efforts are primarily dealing with "what" and not "how". Although in an early stage of development, these standardization efforts will enable the various educational resource developers to create autonomous, on-line educational material that will be used by multiple tutorials, will operate independently of any single tutorial and will be adjusted to each learner's individual needs.

In this chapter, under the light of these emerging standardization efforts, we present X-DEVICE [7], an intelligent XML repository system for educational metadata. X-DEVICE can be used as the intelligent back-end of a WWW portal on which "learning objects" are supplied by educational service providers and accessed by learners according to their individual profiles and educational needs. X-DEVICE transforms the widely adopted XML binding for educational metadata into a flexible, object-oriented representation and uses intelligent second-order logic querying facilities to provide advanced, personalized functionality. Furthermore, a case study is presented, in which learning object metadata and learner's profile metadata are combined under certain X-DEVICE rules in order to dynamically infer customized courses for the learner.

The outline of this chapter is as follows: Section 2 refers to the standardization efforts in the education and describes the current trends in intelligent approaches in e-Learning; Section 3 overviews the management of XML data and documents; Section 4 describes the storage model of X-DEVICE, i.e. it describes how XML data are mapped onto the object data model of X-DEVICE; Section 5 presents the X-DEVICE deductive rule language for querying XML data through several examples on querying educational metadata objects. Section 6 presents an e-Learning case-study, which combines in an intelligent way metadata from different information models in order to dynamically generate a course, adapted to an individual learner. Finally, Section 7 concludes this chapter and discusses future work. There are five Appendices in this chapter that contain the DTDs for the various educational metadata used throughout this chapter, their equivalent X-DEVICE object schemata, a sample XML document with learner information, its X-DEVICE representation, and finally, the syntax of the X-DEVICE rule language.

## 2. PREPARING THE BASE FOR ADVANCED e-LEARNING

The term e-Learning is used to describe a wide range of efforts to provide educational material on the web. These efforts include a diversity of approaches, ranging from static HTML pages with multimedia material to sophisticated interactive educational applications accessible on-line. The state-of-the-art in this latter category is the "Adaptive and Intelligent Web-based Educational Systems" (AIWES). As the term indicates, such approaches have their roots in the fields of Intelligent Tutoring Systems (ITS) and Adaptive Hypermedia Systems (AHS). Actually, most of the current implementations are web-enabled adaptations of earlier stand-alone systems. The main features of AIWES are [8]:

*Adaptive Curriculum Sequencing*: The material that will be presented to the learner is selected according to his learning request that is initially stated to the system. The sequencing refers to various levels of granularity; from which concept (topic, lesson) should be presented next to which is the next task (exercise, problem) the user should deal with. Either type of sequencing if performed based on the learner's model, that is, the perception that the system has about the learner's current knowledge status and goals.

*Problem Solving Support*: We can identify three levels of support. In "*Intelligent analysis of learner's solution*" the system waits for the final solution and responds with the errors the student has made. In "*Interactive Problem Solving Support*" the system is continuously monitoring the learner and is capable of giving hints, signaling errors or even auto-executing the next step of an exercise. Finally, in "*Example based Problem Solving Support*" the system is able to suggest relevant problems that the user has successfully dealt with in the past.

*Adaptive Presentation*: This feature refers to the system's ability to adapt the content of the supplied curriculum to the learner's preferences.

*Student Model Matching*: This is a unique feature in AIWES. It allows the categorization of the learners to classes with similar educational characteristics and the use of this information for collaborative problem solving support and intelligent class monitoring.

None of the currently available e-learning systems delivers the advanced functionality described earlier. Most of the current e-learning approaches are limited to simple hyperlinks between content pages and "portal pages" which organize a set of related links. This happens because most of the existing educational material can be easily published on-line thanks to MIME data format standards. This is very important since educational material is expensive to create in terms of cost and time. This reusability however does not exist at the content, tutorial, or pedagogical levels.

One interesting approach on e-Learning is described in [34] where a framework called Model for Distributed Curriculum (MDC) is introduced. MDC uses a topic server architecture to allow a Web-based tutorial to include a specification for another tutorial where the best fit to this specification will automatically be found at run time. A specific reasoning mechanism towards this functionality is not presented though.

Another approach in e-Learning, which takes into account personalization aspects, is the DCG [42]. This is a tool that generates individual courses according to the learner's goals and previous knowledge and dynamically adapts the course according to the learner's success in acquiring knowledge. DGC uses "concept structures" as a road-map to generate the plan of the course.

The lack of widely adopted methods for searching the Web by content makes difficult for an instructor or learner to find educational material on the Web that addresses particular learning and pedagogical goals. In addition, the lack of standards prevents the interoperability of educational resources. Towards this direction and under the aegis of the IEEE Learning Technology Standards Committee (LTSC) [20], several working groups are developing technical standards, recommended practices and guides for software components, tools, technologies and design methods that facilitate the development, deployment, maintenance and interoperation of computer implementations of educational components and systems. Two of the most important LTSC groups are the Learning Object Metadata (LOM) group and the Learner Model (LM) group.

The LOM working group is dealing with the attributes required to adequately describe a learning object [21], that is, any digital or non-digital mean, which can be used during technology supported learning. Such attributes include type of object, author, owner, terms of distribution, format, requirements to operate, etc. More over, LOM may also include pedagogical attributes, such as teaching or interaction style, mastery level, grade level, and prerequisites. The Learner Model Group is dealing with the specification of the syntax and semantics of attributes that will characterize a learner and his/her knowledge abilities [22]. These will include elements such as knowledge, skills, abilities, learning styles, records, and personal information.

Another working group, whose work is referenced later in this chapter, is the Content Packaging group, which is trying to define a single unit of transmission for the media components (text, graphics, audio, video) and all the supporting material of a learning object. This will enable, among others, the activation of learning content with a single click of a URL in a browser.

Since no information is included in these standards on how to represent meta-data in a machine-readable format, the IMS Global Learning Consortium developed a representation of LOM in XML [25]. Similar bindings have been developed by IMS for the learner model metadata (LIP - [24]) and the content packaging metadata (CP - [23]). For technical details the reader can refer to Appendix A of this chapter.

One of the most ambitious efforts on e-Learning that makes use of educational metadata is the Advanced Distributed Learning initiative [3]. Recently, ADL released the SCORM (Sharable Courseware Object Reference Model) that attempts to map existing learning models and practices so that common interfaces and data may be defined and standardized across courseware management systems and development tools.

Another approach on utilizing educational metadata in personalized e-learning is CG-PerLS [28], a knowledge based approach for organizing and accessing educational resources. CG-PerLS is a model of a WWW portal for learning objects that encodes the learning technologies metadata in the Conceptual Graph knowledge representation formalism, and uses related inference techniques to provide advanced, personalized functionality. CG-PerLS allows learning resource creators to manifest their material, client-side learners to access these resources in a way tailored to their individual profile and educational needs, and dynamic course generation based on fine or coarse grained educational resources.

All the above standardization efforts will require enough time to mature. More time will be required to build systems to conform to these specifications. The internal details of such systems are an open issue since these standardization efforts are primarily dealing with "what" and not "how". Meanwhile, everyday more and more educational material is becoming available. Therefore, there is an urgent need for methods to efficiently organize what is available today and what will become available in the near future, before the educational resource providers conform to the results of the standardization efforts.

The X-DEVICE, which is presented in the following sections, is an XML-based, intelligent educational metadata repository system. X-DEVICE translates the DTD definitions of the XML binding of the educational metadata, into an object database schema that includes classes and attributes. The XML metadata are translated into objects of the database and stored into an underlying object-oriented database. In its current state, X-DEVICE can handle LOM, LIP and CM metadata and perform reasoning over them using second-order logic querying facilities. It can be used as the intelligent back-end of a WWW portal on which "learning objects" are supplied by educational service providers and accessed by learners according to their individual profiles and educational needs.

## 3. MANAGING XML DATA

XML is the currently proposed standard for structured or even semi-structured information exchange over the Internet [43]. However, the maintenance of this information is equally important. Integrating, sharing, re-using and evolving information captured from XML documents are essential for building long-lasting applications of industrial strength.

There exist two major approaches to manage and query XML documents. The first approach uses special purpose query engines and repositories for semi-structured data ([9], [17], [33], [30], [35]). These database systems are built from scratch for the specific purpose of storing and querying XML documents. This approach, however, has two potential disadvantages. Firstly, native XML database systems do not harness the sophisticated storage and query capability already provided by existing database systems. Secondly, these systems do not allow users to query seamlessly across XML documents and other (structured) data stored in database systems.

Traditional data management has an enormous research background that should be utilized. The second approach to XML data management is to capture and manage XML data within the data models of either relational ([40], [14], [16], [39]), object-relational ([41], [27]), or object databases ([47], [38], [36], [12]). Our system, X-DEVICE, stores XML data into the object database ADAM [18], because XML documents have by nature a hierarchical structure that better fits the object model. Also references between or within documents play an important role and are a perfect match for the notion of object in the object model. This better matching between the object and document models can also be seen in the amount of earlier approaches in storing SGML multimedia documents in object databases ([9], [37], [1], [2]).

Capturing XML data in traditional DBMSs alone does not suffice for exploiting the facilities of a database system. Effective and efficient querying and publishing these data on the Web is actually more important since it determines the impact this approach will have on future Web applications. There have been several query language proposals ([45], [13], [2], [11], [10], [19]) for XML data. Furthermore, recently the WWW consortium issued a working draft proposing XQuery [46], an amalgamation of the ideas present in most of the proposed XML query languages of the literature. Most of them have functional nature and use path-based syntax. Some of them ([13], [2], [11]), including XQuery [46], have also borrowed an SQL-like declarative syntax, which is popular among users.

The X-DEVICE system is a deductive object-oriented database that stores XML documents as objects and uses logic-based rules as a query language for XML data [7]. X-DEVICE integrates high-level, declarative rules (namely deductive and production rules) into an active OODB that supports only event-driven rules [15], built on top of Prolog. This is achieved by translating each high-level rule into one event-driven rule. The condition of the declarative rule compiles down to a set of complex events that is used as a discrimination network that incrementally matches the rule conditions against the database.

X-DEVICE automatically maps XML document DTDs to object schemata, without loosing the document's original order of elements. XML elements are represented either as first-class objects or as attributes based on their complexity. The deductive rule language of X-DEVICE uses special operators for specifying complex queries and materialized views over the stored semi-structured data. Most of these operators have a second-order syntax (i.e. variables range over class and attribute names), but they are implemented by translating them into first-order rules (i.e. variables can range over class instances and attribute values), so that they can be efficiently executed against the underlying deductive object-oriented database.

The advantages of using a logic-based query language come from their well-understood mathematical properties. The declarative character of these languages also allows the use of advanced optimization techniques. Logic has also been used for querying semi-structured documents, in WebLog [29] for HTML documents and in F-Logic/FLORID [31] and XPathLog/LoPix [32] for XML data.

## 4. THE X-DEVICE STORAGE MODEL

The X-DEVICE system [7] incorporates XML documents with a schema described through a DTD into an object-oriented database. Specifically, X-DEVICE parses XML documents, which describe learning resources and learners' information (including their DTD definitions), and transforms them as follows: DTD definitions are translated into an object database schema that includes classes and attributes, while XML data are translated into objects of the database. Generated classes and objects are stored within the underlying object database.

The mapping between a DTD and the object-oriented data model uses the object types presented in Figure 1, and is done as follows:

*Elements* are represented as either object attributes or classes. More specifically:

- If an element has PCDATA content (without any attributes), it is represented as an attribute of the class of its parent element. The name of the attribute is the same as the name of the element and its type is string.
- If an element has either a) children elements, or b) attributes, then it is represented as a class that is an instance of the `xml_seq` meta-class. The attributes of the class include both the attributes of the element and its sub-elements. The types of the attributes of the class are determined as follows:

```
class xml_elem
attributes
  alias       (attribute-xml_elem, set, optional)
  empty       (attribute, set, optional)

class xml_seq
is_a          xml_elem
attributes
  elem_ord    (attribute, list, optional)
  att_lst     (attribute, set, optional)

class xml_alt
is_a          xml_elem
```

Figure 1. X-DEVICE object types for mapping XML documents.

- Simple character (PCDATA) sub-elements correspond to attributes of string type, except when they have also attribute elements. In this case, the element is represented as a class, the element attributes as class attributes, and the element content as a string attribute called `content`.
- Element attributes correspond to object attributes of string type.
- Children elements or sub-elements that are represented as objects correspond to object reference attributes.

*Attributes* of elements are represented as object attributes. The types of the attributes are currently only strings or object references, since DTDs do not support data types. Attributes are distinguished from sub-elements through the `att_lst` meta-attribute.

In Appendix B the OODB schemata for the educational metadata used throughout this chapter are presented. The DTDs for these educational metadata are shown in Appendix A. More details for the XML-to-object mapping algorithm of X-DEVICE can be found in [7]. Appendix C shows an XML document that conforms to the LIP DTD (see Appendix A) and Appendix D shows how this document is represented in X-DEVICE as a set of objects.

There are more issues that a complete mapping scheme needs to address, except for the above mapping rules. First, elements in a DTD can be combined through either sequencing or alternation. *Sequencing* means that a certain element must include *all* the specified children elements with a specified order. This is handled by the above mapping scheme through the existence of multiple attributes in the class that represents the parent element, each for each child element of the sequence. The order is handled outside the standard OODB model by providing a meta-attribute (`elem_ord`) for the class of the element that specifies the correct ordering of the children elements. This meta-attribute is used for returning results to the user in the form of an XML document, or by the query mechanism.

On the other hand, *alternation* means that *any* of the specified children elements can be included in the parent element. Alternation is also handled outside the standard OODB model by creating a new class for each alternation of elements, which is an instance of the `xml_alt` meta-class and it is given a system-generated unique name. The attributes of this class are determined by the elements that participate in the alternation. The types of the attributes are determined as in the sequencing case. The structure of an alternation class may seem similar to a sequencing class, however the behavior of alternation objects is different, because they must have a value for exactly one of the attributes specified in the class (e.g. see Appendix D, instances of class `activity_alt1`).

The alternation class is always encapsulated in a parent element. The parent element class has an attribute with the system-generated name of the alternation class, which should be hidden from the user for querying the class. Therefore, a meta-attribute (`alias`) is provided with the aliases of this system-generated attribute, i.e. the names of the attributes of the alternating class. Mixed content elements are handled similarly to alternation of elements, whereas the plain text elements are represented as string attributes, with the name `content`.

Another issue that must be addressed is the mapping of the occurrence operators for elements, sequences and alternations. More specifically, these operators are handled as follows:

- The "star"-symbol (*) after a child element causes the corresponding attribute of the parent element class to be declared as an optional, multi-valued attribute.
- The "cross"-symbol (+) after a child element causes the corresponding attribute of the parent element class to be declared as a mandatory, multi-valued attribute.
- The question mark (?) after a child element causes the corresponding attribute of the parent element class to be declared as an optional, single-valued attribute.
- Finally, the absence of any symbol means that the corresponding attribute should be declared as a mandatory, single-valued attribute.

The order of children element occurrences is important for XML documents, therefore the multi-valued attributes are implemented as lists and not as sets.

Empty elements are treated in the framework described above, depending on their internal structure. If an empty element does not have attributes, then it is treated as a PCDATA element, i.e. it is mapped onto a string attribute of the parent element class. The only value that this attribute can take is `yes`, if the empty element is present. If the empty element is absent then the corresponding attribute does not have a value. On the other hand, if an empty element has attributes, then is represented by a class. Finally, unstructured elements that have content ANY are not currently treated by X-DEVICE, therefore we have transformed all the elements with ANY structure in the DTDs of the educational metadata into PCDATA elements.

## 5. THE X-DEVICE DEDUCTIVE QUERY LANGUAGE

Users can query the stored XML documents using X-DEVICE, by: a) submitting the query through an HTML form, b) submitting an XML document that encapsulates the X-DEVICE query as a string, or c) entering the query directly in the text-based Prolog environment. In any of the above ways, the X-DEVICE query processor executes the query and transforms the results into an XML document that is returned to the user.

In this section, we give a brief overview of the X-DEVICE deductive rule language. More details about X-DEVICE can be found in [5], [6], [7].

### 5.1. First-Order Syntax and Semantics

The syntax for X-DEVICE deductive rules is given in Appendix E. Rules are composed of condition and conclusion, whereas the condition defines a pattern of objects to be matched over the database and the conclusion is a derived class template that defines the objects that should be in the database when the condition is true. The first rule in *Example 1* defines that an object with attributes identifier=ID and title=T exists in class lp_related_resources if there is an object with OID G in class general with attributes identifier=ID, title=T and the string 'Logic Programming' inside the attribute keyword.

*Example 1*

```
if G@general(identifier:ID,title:T,keyword ∍ 'Logic Programming')
then lp_related_resources(identifier:ID,title:T)

if LP@lp_related_resources(identifier:ID) and
   G@general(identifier=ID) and
   L@lom(general=G,relation ∍ R) and
   R@relation(kind='Requires',resource:RS) and
   RS@resource(identifier:ID1\=ID) and
   G1@general(identifier=ID1,title:T1)
then lp_related_resources(identifier:ID1,title:T1)
```

Class lp_related_resources is a derived class, i.e. a class whose instances are derived from deductive rules. Only one derived class template is allowed at the THEN-part (head) of a deductive rule. However, there can exist many rules with the same derived class at the head. The final set of derived objects is a union of the objects derived by the two rules. For example, the transitive closure of the set of educational resources required from 'Logic Programming' resources is completed with the second (recursive) rule of *Example 1*, which recursively adds to the result resources that are required by the resources that are in the result already. The second rule is very complicated because it must retrieve the OID of the `lom` objects that have already been placed in the result, through their text identifier. In *Example 13* we will present a more comprehensive version of this set of rules.

The syntax of the basic rule language is first-order. Variables can appear in front of class names (e.g. `LP`, `G`), denoting OIDs of instances of the class, and inside the brackets (e.g. `ID`, `T`, `R`), denoting attribute values (i.e. object references and simple values, such as integers, strings, etc). Variables are instantiated through the ':' operator when the corresponding attribute is single-valued, and the '϶' operator when the corresponding attribute is multi-valued. Since multi-valued attributes are implemented through lists (ordered sequences) the '϶' operator guarantees that the instantiation of variables is done in the predetermined order stored inside the list.

Conditions also can contain comparisons between attribute values, constants and variables (e.g. `kind='Requires'`). Negation is also allowed if rules are safe, i.e. variables that appear in the conclusion must also appear at least once inside a non-negated condition. Furthermore, the use of arbitrary Prolog goals inside the condition of rules is allowed in X-DEVICE. In this way the system can be extended with several features, which are outside of the language and therefore cannot be optimized. For example, in the first rule of *Example 1* the search for the keyword 'Logic Programming' is case-sensitive. If a case-insensitive search is required instead, then the following rule calls upon a Prolog predicate (either built-in or user-defined) that transforms the keywords in capital case and then compares them to `'LOGIC PROGRAMMING'`.

```
if G@general(identifier:ID,title:T,keyword ϶ K) and
   prolog{upper_case(K,K1), K1=='LOGIC PROGRAMMING'}
then lp_related_resources(identifier:ID,title:T)
```

A query is executed by submitting the set of stratified rules (or logic program) to the system, which translates them into active rules and activates the basic events to detect changes at base data. Data then are forwarded to the rule processor through a discrimination network (much alike in a production system fashion). Rules are executed with fixpoint semantics (semi-naive evaluation), i.e. rule processing terminates when no more new derivations can be made. Derived objects are materialized and are either maintained after the query is over or discarded on user's demand. X-DEVICE also supports production rules, which have at the THEN-part one or more actions expressed in the procedural language of the underlying OODB.

The main advantage of the X-DEVICE system is its extensibility that allows the easy integration of new rule types as well as transparent extensions and improvements of the rule matching and execution phases. The current system implementation includes deductive rules for maintaining derived and aggregate attributes. Among the optimizations of the rule condition matching is the use of a RETE-like discrimination network, extended with re-ordering of condition elements, for reducing time complexity and virtual-hybrid memories, for reducing space complexity [4]. Furthermore, set-oriented rule execution can be used for minimizing the number of inference cycles (and time) for large data sets [5].

## 5.2. Extended Language Constructs for Querying XML Data

The deductive rule language of X-DEVICE supports constructs and operators for traversing and querying tree-structured XML data, which are implemented using second-order logic syntax (i.e. variables can range over class and attribute names) that can also be used to integrate schemata of heterogeneous databases [6].

These XML-aware constructs are translated into a combination of a) a set of first-order logic deductive rules, and/or b) a set of production rules that their conditions query the meta-classes of the OODB, they instantiate the second-order variables, and they dynamically generate first-order deductive rules.

Throughout this section, we will demonstrate the use of X-DEVICE for querying XML data through examples on educational metadata objects. The DTDs for these metadata are shown in Appendix A, while their equivalent X-DEVICE object schemata are shown in Appendix B. More details about the translation of the various XML-aware constructs to the basic first-order rule language can be found elsewhere ([7]).

*Path Expressions*

X-DEVICE supports several types of path expressions into rule conditions. The simplest case is when all the steps of the path must be determined. For example, the second rule of *Example 1* can be expressed as follows:

*Example 2*

```
if LP@lp_related_resources(identifier:ID) and
   L@lom(identifier.general=ID, kind.relation='Requires',
          identifier.resource.relation:ID1\=ID) and
   G1@general(identifier=ID1,title:T1)
then lp_related_resources(identifier:ID1,title:T1)
```

The path expressions are composed using dots between the "steps", which are attributes of the interconnected objects, which represent XML document elements. The innermost attribute should be an attribute of "departing" class, i.e. `relation` is an attribute of class `lom`. Moving to the left, attributes belong to classes that represent their predecessor attributes. Notice that we have adopted a right-to-left order of attributes, contrary to the C-like dot notation that is commonly assumed, because we would like to stress out the functional data model origins of the underlying ADAM OODB [18]. Under this interpretation the chained "dotted" attributes can be seen as function compositions.

Another case in path expressions is when the number of steps in the path is determined, but the exact step name is not. In this case, a variable is used instead of an attribute name. This is demonstrated by the following example, which returns the identifiers of all direct sub-elements of `lom` objects that have `'Logic Programming'` in their keywords.

*Example 3*

```
if L@lom(identifier.E:ID,keyword.general ∍ 'Logic Programming')
then identifiers(identifier:ID)
```

Variable `E` is in the place of an attribute name, therefore it is a second-order variable, since it ranges over a set of attributes, and attributes are sets of things (attribute values). Deductive rules that contain second-order variables are always translated into a set of rules whose second-order variable has been instantiated with a constant. This is achieved by generating production rules, which query the meta-classes of the OODB, instantiate the second-order variables, and generate deductive rules with constants instead of second-order variables. More details can be found in [6] and [7]. In this example, two such bindings actually exist for variable `E`, namely elements `general` and `metadata`.

The most interesting case of path expressions is when some part of the path is unknown, regarding both the number and the names of intermediate steps. This is handled in X-DEVICE by using the "star" (*) operator in place of an attribute name. Such path expressions are called "generalized". *Example 3* can be re-written using the "star" (*) operator as:

*Example 4*

```
if L@lom(identifier.*:ID,keyword.general ∍ 'Logic Programming')
then identifiers(identifier:ID)
```

The above query has different semantics from *Example 3* because it involves any sub-tree with an `identifier` leaf of any length originating from `lom` objects. Actually, the `identifier.*` path is resolved with the following three concrete paths:

```
identifier.general
identifier.metadata
identifier.resource.relation
```

Sometimes, one of the steps of the path expression involves a recursive element, i.e. an element that contains other elements of the same type, as for example the element `definition` of the LIP objects (Appendix A). *Example 5* illustrates the use of recursive elements in path expressions. Specifically, the rule in *Example 5* retrieves the names of all modules that have been taken by some Mr. John Smith in the first year of the curriculum. Since `definition` is a recursive element, it is not determined at which depth of the tree the required constants will be found. Thus, the star (*) symbol after the `definition` element. Notice that once the appropriate level of the definition elements is found (where the `'Curriculum'` constant exists), it is known that `'Module'` definitions are exactly one step further.

*Example 5*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year1',
                 tyvalue.typename='Curriculum',definition ∋ D2) and
   D2@definition(tyvalue.typename='Module',
                 indexid.referential.contentype:ModName)
then modules(module:ModName)
```

*Ordering Expressions*

X-DEVICE supports expressions that query an XML tree based on the ordering of elements. Here we will demonstrate all types of ordering expressions through several examples. *Example 6* retrieves the first two modules that Mr. John Smith took during his second year.

*Example 6*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year2',
                 tyvalue.typename='Curriculum',
                 definition ∋=<2 D2) and
   D2@definition(indexid.referential.contentype:ModName)
then modules(module:ModName)
```

The $\ni_{=<2}$ operator is an absolute numeric ordering expression that returns the first two elements of the corresponding list-attribute. More such ordering expressions exist for every possible position inside a multi-valued attribute. The operator $\ni_{=n}$ that returns the n-th element in a sequence has the shortcut notation $\ni_{n}$. When there are multiple such expressions in a path expression, then there is another shortcut notation, which is demonstrated with *Example 7* that retrieves the subject of the second lecture of the third module that Mr. John Smith attended to during his second year.

*Example 7*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year2',
                 tyvalue.typename='Curriculum',
                 fielddata.definitionfield₂.definition₃:Lecture) and
then lecture(subject:Lecture)
```

Except for the absolute numeric ordering expressions, there are also relative ordering expressions, which are demonstrated with the following two examples. The first one (*Example 8*) retrieves the subjects of the lectures that Mr. John Smith attended to *between* the lectures of `'Boolean Logic'` and `'FET Transistors'`. Notice that the query also determines the name of the module, which must be `'Electronics_101'`.

*Example 8*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Electronics_101',
                 definitionfield ∋ L1) and
   D1@definition(definitionfield ∋ L2) and
   L1@definitionfield(fielddata='Boolean Logic') and
   L2@definitionfield(fielddata='FET Transistors') and
   D1@definition(definitionfield ∋between(L1,L2) L) and
   L@definitionfield(fielddata:Lecture) and
then lecture(subject:Lecture)
```

The operator `between(L1,L2)` is a relative ordering expression that returns all elements in a sequence after the one with an OID identified by the instantiations of the variable `L1` and before the ones with OID `L2`.

In some cases, the relative ordering expression coexists with an absolute numeric ordering expression, which is called a complex ordering expression. *Example 9* demonstrates this by retrieving the subjects of the first two lectures that Mr. John Smith attended to *after* the lectures of `'Boolean Logic'` during the `'Electronics_101'` module.

*Example 9*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contenttype='Electronics_101',
                 definitionfield ∋ L1) and
   L1@definitionfield(fielddata='Boolean Logic') and
   D1@definition(definitionfield ∋{after(L1), =<2} L) and
   L@definitionfield(fielddata:Lecture) and
then lecture(subject:Lecture)
```

The operator ∋`{after(I),=<2}` is a complex ordering expression that consists of the relative ordering expression `after(L1)` followed by the absolute numeric ordering expression `=<2`.

*Exporting Results*

So far, only the querying of existing XML documents through deductive rules has been discussed. However, it is important that the results of a query can be exported as an XML document. This can be performed in X-DEVICE by using some directives around the conclusion of a rule that defines the top-level element of the result document.

When the rule processing procedure terminates, X-DEVICE employs an algorithm that begins with the top-level element designated with one of these directives and navigates recursively all the referenced classes constructing a result in the form of an XML tree-like document. More details can be found in [7]. *Example 10* demonstrates how XML documents (and DTDs) are constructed in X-DEVICE for exporting them as results. Actually, *Example 10* is the same with *Example 5*; the only difference being the keyword `xml_result` around the derived class.

*Example 10*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contenttype='Year1',
                 tyvalue.typename='Curriculum',definition ∋ D2) and
   D2@definition(tyvalue.typename='Module',
                 indexid.referential.contenttype:ModName)
then xml_result(modules(module:ModName))
```

The keyword `xml_result` is a directive that indicates to the query processor that the encapsulated derived class (`modules`) is the answer to the query. This is especially important when the query consists of multiple rules. In order to build an XML tree as a query result, the objects that correspond to the elements must be constructed incrementally in a bottom-up fashion, i.e. first the simple elements that are towards the leaves of the tree are generated and then combined into more complex elements towards the root of the tree. The above query produces an awkward forest of `modules` elements, with the following DTD:

```
<!DOCTYPE modules [
  <!ELEMENT modules (module)>
  <!ELEMENT module (#PCDATA)>
]>
```

To see why the above is an awkward result, regard the input XML document of Appendix C, which will produce the following answer document:

```
<modules>
  <module>Electronics_101</module>
</modules>
<modules>
  <module>Maths_101</module>
</modules>
```

A better-looking XML document would be one that encapsulates both `module` elements in the same (root) `modules` element, like the following:

```
<modules>
  <module>Electronics_101</module>
  <module>Maths_101</module>
</modules>
```

The above XML document, where an element has multiple occurrences of the same sub-element type, conforms to the following DTD, instead of the one presented above:

```
<!DOCTYPE modules [
  <!ELEMENT modules (module*)>
  <!ELEMENT module (#PCDATA)>
]>
```

In order to construct the above tree-structured DTD, then we should use the `list` construct in the rule conclusion to wrap the `module` elements inside the top-level element `modules`:

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year1',
                 tyvalue.typename='Curriculum',
                 tyvalue.typename.definition='Module',
                 indexid.referential.contentype.definition:ModName)
then xml_result(modules(module:list(ModName)))
```

The `list(ModName)` construct in the conclusion denotes that the attribute `module` of the derived class `modules` is an attribute whose value is calculated by the aggregate function `list`. This function collects all the instantiations of the variable `ModName` and stores them under a strict order into the multi-valued attribute `module`. More details about the implementation of aggregate functions in X-DEVICE can be found in [5].

In order to produce an even more structured DTD with the `module` element not being a mere PCDATA element, but having an internal structure as well, then X-DEVICE offers a wrapping construct as a shortcut notation:

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year1',
                 tyvalue.typename='Curriculum',
                 tyvalue.typename.definition='Module',
                 indexid.referential.contentype.definition:ModName)
then xml_result(modules(module(name:ModName)))
```

This would produce the following slightly more complicated DTD:

```
<!DOCTYPE modules [
  <!ELEMENT modules (module*)>
  <!ELEMENT module (name)
  <!ELEMENT name (#PCDATA)>
]>
```

which corresponds to the following result document:

```
<modules>
  <module>
    <name>Electronics_101</name>
  </module>
  <module>
    <name>Maths_101</name>
  </module>
</modules>
```

Another directive for constructing XML documents is `xml_sorted`, which is similar to `xml_result` and is used for sorting the elements of the result according to a group of element values specified in the rule head. *Example 11* repeats the rule of *Example 10*, sorting the result according to the module name.

*Example 11*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 definition*.activity:D1) and
   D1@definition(indexid.referential.contentype='Year1',
                 tyvalue.typename='Curriculum',
                 tyvalue.typename.definition='Module',
                 indexid.referential.contentype.definition:ModName)
then xml_sorted([ModName],modules(module(name:ModName)))
```

*Alternative Attribute Expressions*

The presentation of X-DEVICE so far assumed that path expressions contain steps that refer only to normal XML elements. However, step expressions can also refer to element attributes, using the (^) symbol as a prefix before the name. *Example 12* retrieves all the comments in the English language found anywhere inside the LIP object of Mr. John Smith.

*Example 12*

```
if L@learnerinfo(text.formname.identification='Mr. John Smith',
                 comment.*:C) and
   C@comment(content:Text,^xml_lang='en')
then xml_result(comments(comment:list(Text)))
```

The system attribute is another special attribute that can be used in X-DEVICE. *Example 13* demonstrates the use of special attributes. Recall the two rules of *Example 1* that retrieve the identifiers and titles of all educational resources required for "teaching" Logic Programming. The first rule collects all the resources that directly contain the `'Logic Programming'` keyword, while the second rule recursively adds to the result resources by retrieving the OID of the `lom` objects that have already been placed in the result, through their text identifier. Things would be easier if one could record in the result class `lp_related_resources` the OID of the recorded `lom` object, so that the second rule knows where to begin. However, we would not like this `lom` OID to appear in the final result. This is achieved by prefixing the name of the "hidden" attribute with an exclamation mark (!). In this way, it is considered a system attribute that will not appear in the result.

*Example 13*

```
if L@lom(identifier.general:ID,title.general:T,
           keyword.general ∋ 'Logic Programming')
then xml_result(
           lp_related_resources(!orig_lom:L,identifier:ID,title:T))

if LP@lp_related_resources((!orig_lom:L) and
   L@lom(kind.relation='Requires',
         identifier.resource.relation:ID1) and
   L1@lom(identifier.general=ID1,title.general:T1)
then lp_related_resources(!orig_lom:L1,identifier:ID1,title:T1)
```

Notice that although both rules refer to the same derived class `lp_related_resources`, only one of them contains the `xml_result` directive. However, this is not a strict language rule; it does not matter if several rules contain the `xml_result` or any other result directive, as long as the following constraints are satisfied:

- Only one type of result directive is allowed in the same query.
- Only one derived class is allowed at the result.

## 6. E-LEARNING CASE STUDY

In this section we present an e-learning case-study, which demonstrates the functionality of the X-DEVICE system as an intelligent educational metadata repository. Specifically, the system (Figure 2) maintains LOM objects submitted by learning resource providers as well as LIP objects of its registered users (learners). Users submit their learning requests through a Web interface and the system dynamically generates a Content Package. The latter provides access to the suggested educational resources that cover the learning request.

X-DEVICE uses deductive rules to suggest educational material by intelligently combining information of educational resources (LOM objects) with information about an individual learner (LIP objects). Specifically, we assume that the LIP objects of the database include past learner activities in terms of LOM objects that they have "learned". This may sound as an oversimplification, but we can assume that when new learners are added to the system their past activities expressed in various "agreed" formats are "translated" to the format required by our system, i.e. sets of LOM objects that were used during these past activities. Furthermore, we assume that when a learner has a new learning goal (e.g. `'Logic Programming'`), this is represented in the goal sub-tree of his/her LIP object. The objective of this case-study is to construct a content package that includes all the required LOM objects for achieving the

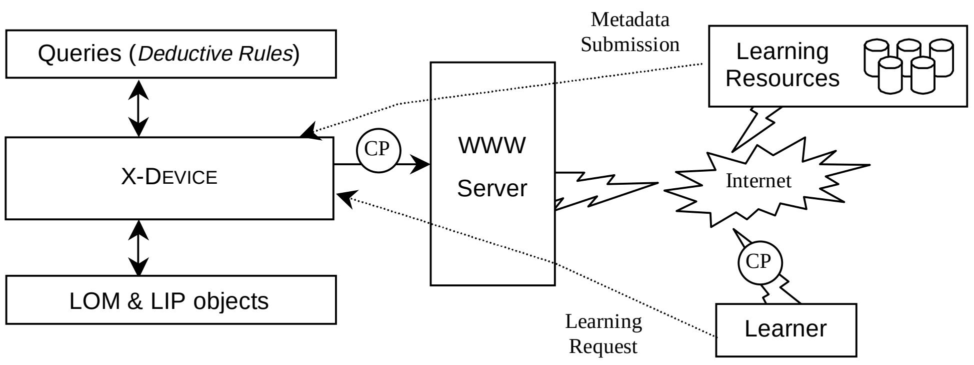

Figure 2. The architecture of the X-DEVICE system.

new learning goal, except of those that have already been "taken" by the learner in the past. The outline of the constructed CP tree and the rules that generate it are shown in Figure 3.

In order to identify the LOM objects that can achieve the learning goal we first match the goal directly with the title of the LOM object (*Rule 1*). If the match fails, then we try to find LOM objects whose keywords are contained within the learning goal (*Rule 2*), e.g. `'Logic'`, `'Programming'`. A more sophisticated approach would use ontologies to better resolve the user's learning goal. The information found with either the above ways is maintained within the user-defined class `top_level`.

*Rule 1*

```
if L@learnerinfo(text.formname.identification:LName,
                  tyvalue.typename.goal='Education',
                  tyvalue.typename.status.goal='New',
                  short.description.goal:Subject) and
   L1@lom(title.general=Subject,identifier.general:ID)
then top_level(learner:LName,goal:Subject,lom_id:ID,
                lom_title:Subject,!org_learner:L,!org_lom:L1)
```

*Rule 2*

```
if L@learnerinfo(text.formname.identification:LName,
                  tyvalue.typename.goal='Education',
                  tyvalue.typename.status.goal='New',
                  short.description.goal:Subject $ K) and
   not T1@tmp_elem1 and
   L1@lom(keyword.general ∋ K,title.general=LTitle,
           identifier.general:ID)
then top_level(learner:LName,goal:Subject,lom_id:ID,
                lom_title:LTitle,!org_learner:L,!org_lom:L1)
```

Notice that the above two rules keep also the OIDs of the original LOM and LIP objects

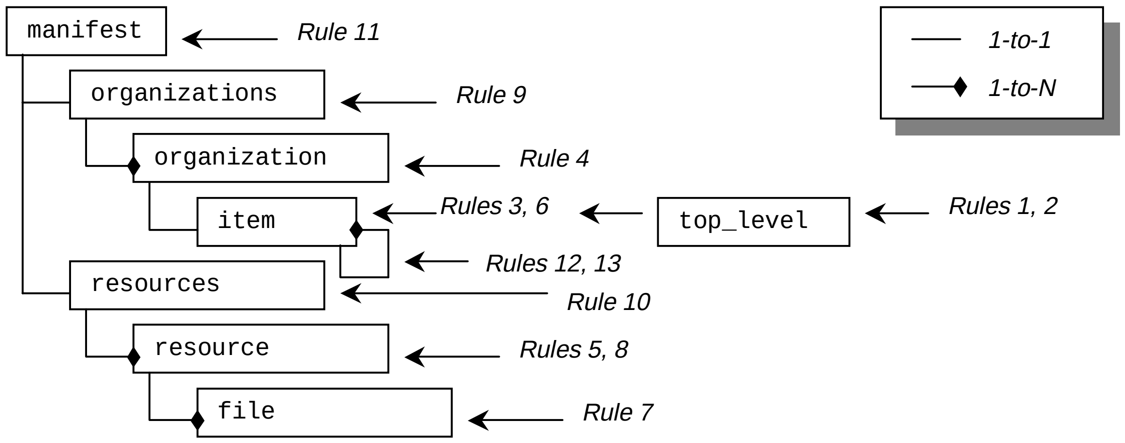

Figure 3. Outline of the case-study

that initiated the whole procedure, in order to use them in subsequent retrieval of information about these objects. This is achieved by using system attributes, which will be hidden away from the final XML result. Also notice the use of the ($) operator, which searches if its right-hand-side argument (string) is a sub-string of its left-hand-side argument.

Subsequently, the "discovered" top-level LOM objects are added to the content package as organizations and items. Specifically, each top-level LOM object is considered to be an alternative study suggestion for the learning goal and will be represented by a separate organization element. Each of these organization elements will have just one item sub-element, which also represents the corresponding top-level LOM object. The LOM objects that are related to these top-level objects will be recursively "hanged" below these top-level item elements.

*Rule 3* creates the top-level item elements for each of the top-level LOM objects. Before the item object is created, the rule condition ensures that the same LOM object has *not* been "attended" by the learner in the past, by checking his/her activity records for completed educational activities that their source is LOM and their ID matches the ID of the "discovered" object. Activities can be nested, thus we use the recursive element `activity*`. Furthermore, there may be cases where "discovered" top-level LOM objects are connected through 'Required' relations. In order to avoid repetitions of items in the CP tree, if the same LOM object has already been recorded as an item object, it is not created again. Subsequently, *Rule 4* creates an organization object for each top-level LOM object and points to the corresponding item object. Notice that we use a Prolog predicate to generate a unique identifier for the organization object by combining the string of the goal with a number.

*Rule 3*

```
if TL@top_level(lom_id:ID,lom_title:LTitle,
                !org_learner:L,!org_lom:L1) and
   not L@learnerinfo(tyvalue.typename.activity*='Education',
               tyvalue.typename.status.activity*='Completed',
               source.sourcedid.learningactivityref.activity*='lom',
               id.sourcedid.learningactivityref.activity*=ID) and
   not I@item(^identifier=ID,!org_learner=L)
then item(^identifier:ID,title:LTitle,!org_lom:L1,!org_learner:L)
```

*Rule 4*

```
if TL@top_level(lom_id:ID,goal:Subject) and
   I@item(^identifier=ID) and
   prolog{generate_id(Subject,OID)}
then organization(^identifier:OID,title:Subject,item:list(I))
```

For each organizational item a corresponding resource object that holds information about physical resources of the LOM object must be created and linked to the item object. *Rule 5* creates the resource object and stores information about the type of the resource. Notice that the OID of the corresponding item object is kept for using it later (*Rule 6*) for cross-referencing the resource from the item object. Notice that *Rule 6* is a, so-called, derived-attribute rule [5], which defines the value that an attribute of an *existing* object should have.

*Rule 5*

```
if I@item(!org_lom:L) and
   L@lom(format.technical:Type) and
   prolog{generate_id(resource,RID)}
then resource(^identifier:RID,^type:Type,!org_item:I)
```

*Rule 6*

```
if R@resource(^identifier:RID,!org_item:I)
then I@item(^identifierref:RID)
```

The resource object is not complete until it is linked with the physical file information of the corresponding LOM object, which we assume corresponds to the `location` element of the `technical` sub-tree of the LOM object. However, the `file` sub-element of `resource` is an object itself, so it must first be created (*Rule 7*) and then linked to the `resource` object (*Rule 8*).

*Rule 7*

```
if R@resource(!org_item:I) and
   I@item(!org_lom:L) and
   L@lom(location.technical:File)
then file(^href:File,!org_resource:R)
```

*Rule 8*

```
if F@file(!org_resource:R)
then R@resource(file:list(F))
```

Finally, the `organization` and `resource` objects must be linked together into a `manifest` object. First, all the `organization` objects are linked together into one `organizations` object (*Rule 9*), according to the DTD of CP (see Appendix A). The same also is done for the `resource` objects (*Rule 10*). Finally, *Rule 11* generates the top-level `manifest` object, generating an identifier from the learner's name. Notice the use of the `xml_result` directive to indicate the top-level element of the result XML document.

*Rule 9*

```
if O@organization
then organizations(organization:list(O))
```

*Rule 10*

```
if R@resource
then resources(resource:list(R))
```

*Rule 11*

```
if TL@top_level(learner:LName) and
   R@resources and
   O@organizations and
   prolog{generate_id(LName,MID)}
then xml_result(manifest(^identifier:MID,
                         organizations:O,resources:R))
```

Now that the result tree has been created, it is time to navigate through the LOM objects that are required by the top-level LOM objects, check if they have been encountered by the learner during his past activities, create the corresponding items, and finally, link them to their parent item/LOM object. *Rule 12* creates the items and keeps the OID of their parent item object, which is used by *Rule 13* to link the two items. Notice that *Rule 12* checks if the item objects to be created have already been created for the same learner. This could happen when multiple LOM objects have common 'Required' LOM objects.

*Rule 12*

```
if I@item(!org_lom:L,!org_learner:L2) and
   L@lom(kind.relation='Requires',
         identifier.resource.relation:ID1) and
   L1@lom(identifier.general=ID1,title.general:T1) and
   not L2@learnerinfo(tyvalue.typename.activity*='Education',
               tyvalue.typename.status.activity*='Completed',
               source.sourcedid.learningactivityref.activity*='lom',
               id.sourcedid.learningactivityref.activity*=ID1)
   not I1@item(^identifier=ID1,!org_learner=L2)
then item(^identifier:ID1,title:T1,!org_lom:L1,!org_learner:L2,
          !parent_item:I)
```

*Rule 13*

```
if I1@item(!parent_item:I)
then I@item(item:list(I1))
```

Although the resulting class (`manifest`) has already been created in *Rule 11*, it does not matter to continue deriving objects, because the order of rule execution is determined by stratification and the results will be exported after the logic program reaches a fixpoint. The creation of the `resource` objects that correspond to the recursively added item objects is taken care by rules already defined (*Rule 5* - *Rule 8*).

The XML document that will be created by the previous set of rules (Figure 3) will be packaged along with the corresponding physical resources (e.g. files) and sent to the learner. The DTD of this document is a subset of the full CP DTD (see Appendix A).

```
<!ELEMENT manifest (organizations, resources)>
<!ATTLIST manifest
  identifier ID #REQUIRED>
<!ELEMENT organizations (organization*)>
<!ELEMENT organization (title?, item*)>
<!ATTLIST organization
  identifier ID #REQUIRED>
<!ELEMENT item (title?, item*)>
<!ATTLIST item
  identifier ID #REQUIRED
  identifierref CDATA #IMPLIED>
<!ELEMENT resources (resource*)>
<!ELEMENT resource (file+)>
<!ATTLIST resource
  identifier ID #REQUIRED
  type CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT file EMPTY>
<!ATTLIST file
  href CDATA #REQUIRED>
```

The case-study presented here does not deal with what the user does in order to study the suggested LOM objects and what happens after each of the suggested item gets completed. However, we assume that the studied LOM objects are added to the activity sub-tree of the learner's LIP object, so that the next time the learner has a new learning goal they will not be considered again (see *Rule 3*).

## 7. CONCLUSIONS AND FUTURE WORK

In this chapter, we presented X-DEVICE, an intelligent repository system for educational metadata. X-DEVICE transforms the widely adopted XML binding for LOM, LIP and CP educational metadata, into a flexible, object-oriented representation and uses intelligent second-order logic querying facilities to provide advanced, personalized learning content access. As demonstrated by the case-study, X-DEVICE can be used as the intelligent back-end of a WWW portal on which "learning objects" are supplied by educational service providers and accessed by learners according to their individual profiles and educational needs.

X-DEVICE stores an XML document into an OODB by automatically mapping the schema of the XML document (DTD) to an object schema and XML elements to database objects, treating them according to their structure complexity, without loosing the relative order of elements in the original document. Furthermore, X-DEVICE employs a powerful deductive rule query language for expressing queries over the stored XML data. The deductive rule language has certain constructs (such as second-order variables, general path and ordering expressions) for traversing tree-structured data that were implemented by translating them into first-order deductive rules.

Comparing X-DEVICE with other XML query languages (e.g. XQuery) seems that the high-level, declarative syntax of X-DEVICE allows users to express everything that XQuery can express, in a more compact and comprehensible way, with the powerful addition of fixpoint recursion and second-order variables. Furthermore, users can also express complex XML document views, a fact that can greatly facilitate customizing information for e-learning, as it was demonstrated by the case-study.

Concerning the functionality of X-DEVICE as an educational portal, we plan to incorporate the learner's assessment results described in the IMS Question & Test Interoperability Specification (QTI - [26]). This information can be used to improve the overall knowledge transfer from the system to the learner by preferring to serve him/her with a specific learning object from a set of similar ones, based on the assessment results obtained by learners with similar profiles. This will require substantial research on aspects related to the learner's model.

Furthermore, the suitability of the suggested learning content would be improved by using ontologies to better resolve the end-users' learning request. Further improvement could be achieved by using curriculum description information. In its current state, X-DEVICE assumes a curriculum description defined indirectly by the relations between learning objects. On the other hand, curriculum information contains higher-level pedagogical knowledge and as a result can lead to a better teaching strategy for the suggested educational material.

## 8. REFERENCES

[1] Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., Siméon J., Querying Documents in Object Databases, *Int. J. on Digital Libraries,* 1(1): 5-19 (1997)

[2] Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J.L., "The Lorel Query Language for Semistructured Data," *Int. Journal on Digital Libraries,* 1(1), pp. 68-88, 1997.

[3] ADL, "*Sharable Courseware Object Reference Model (SCORM)*", Version 1.1, Jan-2001, (http://www.adlnet.org).

[4] Bassiliades N. and Vlahavas I., "Processing Production Rules in DEVICE, an Active Knowledge Base System", *Data & Knowledge Engineering*, Vol. 24(2), pp. 117-155, 1997.

[5] Bassiliades N., Vlahavas I., and Elmagarmid A.K., "E-DEVICE: An extensible active knowledge base system with multiple rule type support", *IEEE TKDE*, 12(5), pp. 824-844, 2000.

[6] Bassiliades N., Vlahavas I., Elmagarmid A.K., and Houstis E.N., "InterBaseKB: Integrating a Knowledge Base System with a Multidatabase System for Data Warehousing," *IEEE TKDE*, (to appear) 2001.

[7] Bassiliades N., Vlahavas I., Sampson D., Using Logic for Querying XML Data, submitted for publication.

[8] Brusilovsky P., Adaptive and Intelligent Technologies for Web-based Education, in C.Rollinger and C.Peylo (eds.), Kunstliche Intelligenz, Special Issue on Intelligent Systems and Technology, 1999, 4.

[9] Buneman P., Davidson S. B., Hillebrand G. G., Suciu D., A Query Language and Optimization Techniques for Unstructured Data, *Proc. ACM SIGMOD Conf.*, 1996, pp. 505-516.

[10] Buneman P., Fernandez M., Suciu D., "UnQL: A query language and algebra for semistructured data based on structural recursion," *VLDB Journal*, 9(1), 2000.

[11] Chamberlin D., Robie J., and Florescu D., "Quilt: an XML Query Language for Heterogeneous Data Sources," *Int. Workshop WebDB,* pp. 53-62, 2000.

[12] Chung T.-S., Park S., Han S.-Y., and Kim H.-J., Extracting Object-Oriented Database Schemas from XML DTDs Using Inheritance, K. Bauknecht, S.K. Madria, G. Pernul

(Eds.), *Proc. 2nd Int. Conf. EC-Web 2001*, Munich, Germany, 2001, LNCS 2115, pp. 49-59.

[13] Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D., "A Query Language for XML," *WWW8 / Computer Networks*, 31(11-16), pp. 1155-1169, 1999.

[14] Deutsch A., Fernandez M.F., Suciu D., "Storing Semistructured Data with STORED," *ACM SIGMOD Conf.*, pp. 431-442, 1999.

[15] Diaz O., Jaime A., "EXACT: An Extensible Approach to Active Object-Oriented Databases", *VLDB Journal*, 6(4), pp. 282-295, 1997.

[16] Florescu D. and Kossmann D., "Storing and Querying XML Data using an RDMBS," *IEEE Data Eng. Bulletin*, 22(3), pp. 27-34, 1999.

[17] Goldman R., Widom J., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *Proc. Int. Conf. VLDB*, 1997, pp. 436-445.

[18] Gray P.M.D., Kulkarni K.G., and Paton N.W., *Object-Oriented Databases, A Semantic Data Model Approach*, Prentice Hall, London, 1992.

[19] Hosoya H., Pierce B., "XDuce: A Typed XML Processing Language," *Int. Workshop WebDB*, pp. 111-116, 2000.

[20] IEEE Learning Technology Standards Committee (LTSC), http://ltsc.ieee.org/

[21] IEEE P1484.12/D6.0, "*Draft Standard for Learning Object Metadata*", Feb-2001.

[22] IEEE P1484.2/D7, *"Draft Standard for Learning Technology - Public and Private Information (PAPI) for Learners (PAPI Learner)"*, Nov-2000.

[23] IMS Global Learning Consortium, Content Packaging (CP) Specification, Version 1.1.2, Final Specification, http://www.imsglobal.org/content/packaging/index.html

[24] IMS Global Learning Consortium, Learner Information Package (LIP) Specification, Version 1.0, Final Specification, Mar 2001, http://www.imsglobal.org/profiles/index.html

[25] IMS Global Learning Consortium, Learning Resource Meta-data (LOM) Specification, Version 1.2, Final Specification, May 2001, http://www.imsglobal.org/metadata/index.html

[26] IMS Global Learning Consortium, Question & Test Interoperability Specification, Version 1.1, Feb-2001, http://www.imsproject.com/question

[27] Klettke M. and Meyer H., "XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics," *Int. Workshop WebDB*, pp. 63-68, 2000.

[28] Kokkoras F., Sampson D.G., and Vlahavas I., CG-PerLS: Conceptual Graphs for Personalized Learning Systems, to be presented at *8th Panhellenic Conf. on Informatics*, Cyprus, Nov. 2001

[29] Lakshmanan L.V.S., Sadri F., Subramanian I. N., A Declarative Language for Querying and Restructuring the WEB. RIDE-NDS 1996: 12-21

[30] Lucie Xyleme, A Dynamic Warehouse for XML Data of the Web, *IEEE Data Engineering Bulletin*, 24(2), June 2001, pp. 40-47.

[31] Ludäscher B., Himmeröder R., Lausen G., May W., Christian Schlepphorst, Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective, *Information Systems*, Vol. 23, No 8, 1998, pp. 589-613.

[32] May W.: XPathLog: A Declarative, Native XML Data Manipulation Language. IDEAS 2001: 123-128

[33] McHugh J., Abiteboul S., Goldman R., Quass D., Widom J., Lore: A Database Management System for Semistructured Data, *ACM SIGMOD Record*, 26(3),pp. 54-66, 1997.

[34] Murray T., A Model for Distributed Curriculum on the WWW, *Journal of Interactive Media in Education*, 5 (1998) (http://www-jime.open.ac.uk/98/5).

[35] Naughton J., et al, The Niagara Internet Query System, *IEEE Data Engineering Bulletin*, 24(2), June 2001, pp. 27-33.

[36] Nishioka S., Onizuka M., Mapping XML to Object Relational Model, *Proc. Int. Conf. on Internet Computing*, pp. 171-177, 2001.

[37] Ozsu M.T., Iglinski P., Szafron D., El-Medani S., and Junghanns M., “An Object-Oriented SGML/HyTime Compliant Multimedia Database Management System,” presented at ACM Multimedia, Seattle, WA, USA, 1997.

[38] Renner A., "XML Data and Object Databases: A Perfect Couple?", *Proc. Int. Conf. on Data Engineering*, pp. 143-148, 2001.

[39] Schmidt A., Kersten M.L., Windhouwer M., Waas F., "Efficient Relational Storage and Retrieval of XML Documents," *Int. Workshop WebDB*, pp. 47-52, 2000.

[40] Shanmugasundaram J., Tufte K., Zhang C., He G., DeWitt D.J., and Naughton J.F., “Relational Databases for Querying XML Documents: Limitations and Opportunities,” *Int Conf. VLDB*, pp. 302-314, 1999.

[41] Shimura T., Yoshikawa M., and Uemura S., “Storage and Retrieval of XML Documents Using Object-Relational Databases,” *Proc. Int. Conf. on Database and Expert Systems Applications*, Florence, Italy, 1999, pp. 206-217.

[42] Vassileva J., "Dynamic Course Generation on the WWW", *8th World Conference on AI in Education (AI-ED97)*, Knowledge and Media in Learning Systems, Kobe, Japan, 1997.

[43] W3 Consortium, Extensible Markup Language (XML) 1.0 (2$^{nd}$ Edition), Recommendation, Oct 2000, http://www.w3.org/TR/REC-xml.

[44] W3 Consortium, Semantic Web Activity Statement, http://www.w3.org/2001/sw/Activity.

[45] W3 Consortium, XML Path Language (XPath) Ver. 1.0, Recommendation, Nov 1999, http://www.w3.org/TR/xpath

[46] W3 Consortium, XQuery 1.0: An XML Query Language, Working Draft, June 2001, http://www.w3.org/TR/xquery.

[47] Yeh C.-L., “A Logic Programming Approach to Supporting the Entries of XML Documents in an Object Database,” *Int. Workshop PADL*, pp. 278-292, 2000.

## Appendix A. DTDs for LOM, LIP and CP Objects

This appendix contains the DTDs for the LOM (Learning Object Metadata) [25], LIP (Learner Information Packaging) [24] and CP (Content Packaging) [23] objects that are used throughout this chapter. Notice that, due to space limitations, only a part of the LIP DTD that is used in chapter is actually shown. Furthermore, some of the DTDs have been simplified for presentation purposes.

**LOM**

```
<!ELEMENT lom (general?, lifecycle?, metametadata?, technical?, educational?, rights?,
relation*, annotation*, classification*)>
<!ELEMENT general (identifier?, title?, catalogentry*, language*, description*, keyword*,
coverage*, structure?, aggregationlevel?, extension?)>
<!ELEMENT catalogentry (catalog, entry, extension?)>
<!ELEMENT lifecycle (version?, status?, contribute*, extension?)>
<!ELEMENT contribute (role, centity*, date?, extension?)>
<!ELEMENT date (datetime?, description?)>
<!ELEMENT metametadata (identifier?, catalogentry*, contribute*, metadatascheme*, language?,
extension?)>
<!ELEMENT technical (format*, size?, location*, requirement*, installationremarks?,
otherplatformrequirements?, duration?, extension?)>
<!ELEMENT requirement (type?, name?, minimumversion?, maximumversion?, extension?)>
<!ELEMENT duration (datetime?, description?)>
<!ELEMENT educational (interactivitytype?, learningresourcetype*, interactivitylevel?,
semanticdensity?, intendedenduserrole*, context*, typicalagerange*, difficulty?,
typicallearningtime?, description?, language*, extension?)>
<!ELEMENT typicallearningtime (datetime?, description?)>
<!ELEMENT rights (cost?, copyrightandotherrestrictions?, description?, extension?)>
<!ELEMENT copyrightandotherrestrictions (source, value)>
<!ELEMENT relation (kind?, resource?, extension?)>
<!ELEMENT resource (identifier?, description?, catalogentry*, extension?)>
<!ELEMENT annotation (person?, date?, description, extension?)>
<!ELEMENT classification (purpose?, taxonpath*, description?, keyword*, extension?)>
<!ELEMENT taxonpath (source?, taxon?)>          <!ELEMENT keyword (#PCDATA)>
<!ELEMENT taxon (id?, entry?, taxon?)>          <!ELEMENT coverage (#PCDATA)>
<!ELEMENT structure (source, value)>            <!ELEMENT source (#PCDATA)>
<!ELEMENT type (source, value)>                 <!ELEMENT value (#PCDATA)>
<!ELEMENT name (source, value)>                 <!ELEMENT version (#PCDATA)>
<!ELEMENT aggregationlevel (source, value)>     <!ELEMENT installationremarks (#PCDATA)>
<!ELEMENT status (source, value)>               <!ELEMENT otherplatformrequirements (#PCDATA)>
<!ELEMENT role (source, value)>                 <!ELEMENT typicalagerange (#PCDATA)>
<!ELEMENT interactivitytype (source, value)>    <!ELEMENT location (#PCDATA)>
<!ELEMENT learningresourcetype (source,         <!ATTLIST location
value)>                                            type (URI | TEXT) #IMPLIED>
<!ELEMENT interactivitylevel (source, value)>   <!ELEMENT identifier (#PCDATA)>
<!ELEMENT semanticdensity (source, value)>      <!ELEMENT extension (#PCDATA)>
<!ELEMENT intendedenduserrole (source, value)>  <!ELEMENT catalog (#PCDATA)>
<!ELEMENT context (source, value)>              <!ELEMENT language (#PCDATA)>
<!ELEMENT difficulty (source, value)>           <!ELEMENT vcard (#PCDATA)>
<!ELEMENT cost (source, value)>                 <!ELEMENT datetime (#PCDATA)>
<!ELEMENT kind (source, value)>                 <!ELEMENT metadatascheme (#PCDATA)>
<!ELEMENT purpose (source, value)>              <!ELEMENT format (#PCDATA)>
<!ELEMENT centity (vcard)>                      <!ELEMENT size (#PCDATA)>
<!ELEMENT person (vcard)>                       <!ELEMENT minimumversion (#PCDATA)>
<!ELEMENT title (#PCDATA)>                      <!ELEMENT maximumversion (#PCDATA)>
<!ELEMENT entry (#PCDATA)>                      <!ELEMENT id (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

**LIP**

```
<!ELEMENT learnerinformation (comment?, contentype?, (identification | goal | qcl | activity |
competency | transcript | accessibility | interest | affiliation | securitykey | relationship
| ext_learnerinfo)*)>
<!ATTLIST learnerinformation
   xml:lang CDATA "en">
<!ELEMENT identification (comment?, contentype?, (formname | name | address | contactinfo |
demographics | agent)*, ext_identification?)>
```

```
<!ELEMENT goal (typename?, comment?, contentype?, date*, priority?, status?, description?,
goal*, ext_goal?)>
<!ELEMENT qcl (typename?, comment?, contentype?, title?, organization?, registrationno?,
level?, date*, description?, ext_qcl?)>
<!ELEMENT activity (typename?, comment?, contentype?, date*, status?, units?,
(learningactivityref | definition | product | testimonial | evaluation)*, description?,
activity*, ext_activity?)>
<!ELEMENT name (typename?, comment?, contentype?, partname*)>
<!ELEMENT description (short | long | full)+>
<!ELEMENT full (comment?, media+)>
<!ELEMENT contentype (comment?, (referential | temporal | privacy)+, ext_contentype?)>
<!ELEMENT referential (sourcedid | indexid | (sourcedid, indexid))>
<!ELEMENT status (typename?, date?, description?)>
<!ELEMENT partname (typename?, text?)>
<!ELEMENT date (typename?, datetime, description?, ext_date?)>
<!ELEMENT organization (typename?, description?)>
<!ELEMENT level (text, level?)>
<!ELEMENT evaluation (typename?, comment?, contentype?, evaluationid?, date*, evalmetadata?,
objectives*, status?, noofattempts?, duration*, result*, description?, evaluation*,
ext_evaluation?)>
<!ELEMENT testimonial (typename?, comment?, contentype?, date*, description?,
ext_testimonial?)>
<!ELEMENT definition (typename?, comment?, contentype?, definitionfield*, description?,
definition*, ext_definition?)>
<!ELEMENT evalmetadata (typename?, evalmetadatafield+)>
<!ELEMENT objectives (comment?, (media | contentref)+, ext_objectives?)>
<!ATTLIST objectives
   view (All | Administrator | AdminAuthority | Assessor | Author | Candidate |
InvigilatorProctor | Psychometrician | Scorer | Tutor) "All">
<!ELEMENT result (comment?, ((interpretscore | score)* | result*))>
<!ELEMENT product (typename?, comment?, contentype?, date?, description?, ext_product?)>
<!ELEMENT formname (typename?, comment?, contentype?, text?)>
<!ELEMENT learningactivityref (sourcedid | text)+>
<!ELEMENT relationship (typename?, comment?, contentype?, tuple?, description?,
ext_relationship?)>
<!ELEMENT duration (fieldlabel, fielddata)>
<!ELEMENT tuple (tuplesource, tuplerelation, tupledest+)>
<!ELEMENT media (#PCDATA)>
<!ATTLIST media
   mediamode (Text | Image | Video | Audio | Applet | Application) #REQUIRED
   contentreftype (uri | entityref | Base-64) "Base-64"
   mimetype CDATA #REQUIRED>
<!ELEMENT tysource (#PCDATA)>
<!ATTLIST tysource
   sourcetype (imsdefault | list | proprietary | standard) "imsdefault"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT tuplesource (sourcedid?, indexid)>
<!ELEMENT tuplerelation (typename, text?)>
<!ELEMENT tupledest (sourcedid?, indexid)>
<!ELEMENT units (unitsfield+)>
<!ELEMENT typename (tysource?, tyvalue)>
<!ELEMENT comment (#PCDATA)>
<!ATTLIST comment
   xml:lang CDATA "en"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT source (#PCDATA)>
<!ATTLIST source
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT id (#PCDATA)>
<!ATTLIST id
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT short (#PCDATA)>
<!ATTLIST short
   xml:lang CDATA "en"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT long (#PCDATA)>
<!ATTLIST long
   xml:lang CDATA "en"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT fielddata (#PCDATA)>
<!ATTLIST fielddata
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT datetime (#PCDATA)>
<!ATTLIST datetime
   e-dtype NMTOKEN #FIXED "dateTime">
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
   uri CDATA #IMPLIED
   xml:lang CDATA "en"
   entityref ENTITY #IMPLIED
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT title (#PCDATA)>
<!ATTLIST title
   xml:lang CDATA "en"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT registrationno (#PCDATA)>
<!ATTLIST registrationno
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT fieldlabel (typename)>
<!ELEMENT sourcedid (source, id)>
<!ELEMENT indexid (#PCDATA)>
<!ATTLIST indexid
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT evaluationid (#PCDATA)>
<!ATTLIST evaluationid
   e-dtype NMTOKEN #FIXED "ID">
<!ELEMENT noofattempts (#PCDATA)>
<!ATTLIST noofattempts
   e-dtype NMTOKEN #FIXED "int">
<!ELEMENT evalmetadatafield (fieldlabel,
fielddata)>
<!ATTLIST evalmetadatafield
   xml:lang CDATA "en">
<!ELEMENT contentref (#PCDATA)>
<!ATTLIST contentref
   e-dtype NMTOKEN #FIXED "ID">
```

```
<!ELEMENT tyvalue (#PCDATA)>
<!ATTLIST tyvalue
   xml:lang CDATA "en"
   e-dtype NMTOKEN #FIXED "string">
<!ELEMENT interpretscore (fieldlabel,
fielddata)>
<!ELEMENT score (fieldlabel, fielddata)>
<!ELEMENT definitionfield (fieldlabel,
fielddata)>
<!ELEMENT unitsfield (fieldlabel, fielddata)>
<!ELEMENT ext_goal (#PCDATA)>
<!ELEMENT ext_evaluation (#PCDATA)>
<!ELEMENT ext_learnerinfo (#PCDATA)>
<!ELEMENT ext_contentype (#PCDATA)>
<!ELEMENT ext_activity (#PCDATA)>
<!ELEMENT ext_date (#PCDATA)>
<!ELEMENT ext_definition (#PCDATA)>
<!ELEMENT ext_identification (#PCDATA)>
<!ELEMENT ext_objectives (#PCDATA)>
<!ELEMENT ext_product (#PCDATA)>
<!ELEMENT ext_qcl (#PCDATA)>
<!ELEMENT ext_relationship (#PCDATA)>
<!ELEMENT ext_testimonial (#PCDATA)>
```

**CP**

```
<!ELEMENT manifest (metadata?, organizations,
resources, manifest*)>
<!ATTLIST manifest
   identifier ID #REQUIRED
   version CDATA #IMPLIED>
<!ELEMENT organizations (organization*)>
<!ATTLIST organizations
   default IDREF #IMPLIED>
<!ELEMENT organization (title?, item*,
metadata?)>
<!ATTLIST organization
   identifier ID #REQUIRED>
<!ELEMENT item (title?, item*, metadata?)>
<!ATTLIST item
   identifier ID #REQUIRED
   isvisible CDATA #IMPLIED
   parameters CDATA #IMPLIED
   identifierref CDATA #IMPLIED
   a-dtype NMTOKENS "isvisible boolean">
<!ELEMENT metadata (schema?, schemaversion?)>
<!ELEMENT schema (#PCDATA)>
<!ELEMENT schemaversion (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT resource (metadata?, file+,
dependency*)>
<!ATTLIST resource
   identifier ID #REQUIRED
   type CDATA #REQUIRED
   href CDATA #IMPLIED>
<!ELEMENT resources (resource*)>
<!ELEMENT file (metadata?)>
<!ATTLIST file
   href CDATA #REQUIRED>
<!ELEMENT dependency EMPTY>
<!ATTLIST dependency
   identifierref CDATA #IMPLIED>
```

## Appendix B. X-DEVICE OBJECT SCHEMATA FOR EDUCATIONAL METADATA

This appendix contains the object schemata in X-DEVICE for the DTDs of the various educational objects that are used throughout this chapter (see Appendix A). Notice that, due to space limitations, only a small part of the object schema for the LIP DTD is actually shown, which is necessary for understanding the XML document of Appendix C and Appendix D.

**LOM**

```
xml_seq lom
   attributes
      general          (general, single, optional)
      lifecycle        (lifecycle, single, optional)
      metametadata     (metametadata, single, optional)
      technical        (technical, single, optional)
      educational      (educational, single, optional)
      rights           (rights, single, optional)
      relation         (relation, list, optional)
      annotation       (annotation, list, optional)
      classification   (classification, list, optional)
   meta_attributes
      elem_ord         [general, lifecycle, metametadata, technical, educational, rights,
                          relation, annotation, classification]
xml_seq general
   attributes
      identifier       (string, single, optional)
      title            (string, single, optional)
      catalogentry     (catalogentry, list, optional)
      language         (string, list, optional)
      description      (string, list, optional)
      keyword          (string, list, optional)
      coverage         (string, list, optional)
      structure        (structure, single, optional)
      aggregationlevel (aggregationlevel, single, optional)
      extension        (string, single, optional)
   meta_attributes
```

```
      elem_ord          [identifier, title, catalogentry, language, description, keyword,
                           coverage, structure, aggregationlevel, extension]
xml_seq lifecycle
   attributes
      version           (string, single, optional)
      status            (status, single, optional)
      contribute        (contribute, list, optional)
      extension         (string, single, optional)
   meta_attributes
      elem_ord          [version, status, contribute, extension]
xml_seq metametadata
   attributes
      identifier        (string, single, optional)
      catalogentry      (catalogentry, list, optional)
      contribute        (contribute, list, optional)
      metadatascheme    (string, list, optional)
      language          (string, list, optional)
      extension         (string, single, optional)
   meta_attributes
      elem_ord          [identifier, catalogentry, contribute, metadatascheme, language,
                           extension]
xml_seq technical
   attributes
      format            (string, list, optional)
      size              (string, single, optional)
      location          (location, list, optional)
      requirement       (requirement, list, optional)
      installationremarks  (string, single, optional)
      otherplatformrequirements  (string, single, optional)
      duration          (duration, single, optional)
      extension         (string, single, optional)
   meta_attributes
      elem_ord          [format, size, location, requirement, installationremarks,
                           otherplatformrequirements, duration, extension]
xml_seq educational
   attributes
      interactivitytype       (interactivitytype, single, optional)
      learningresourcetype    (learningresourcetype, list, optional)
      interactivitylevel      (interactivitylevel, single, optional)
      semanticdensity         (semanticdensity, single, optional)
      intendedenduserrole     (intendedenduserrole, list, optional)
      context                 (context, list, optional)
      typicalagerange         (string, list, optional)
      difficulty              (difficulty, single, optional)
      typicallearningtime     (typicallearningtime, single, optional)
      description             (string, single, optional)
      language                (string, list, optional)
      extension               (string, single, optional)
   meta_attributes
      elem_ord                [interactivitytype, learningresourcetype, interactivitylevel,
                                 semanticdensity, intendedenduserrole, context,
                                 typicalagerange, difficulty, typicallearningtime,
                                 description, language, extension]
xml_seq rights
   attributes
      cost                    (cost, single, optional)
      copyrightandotherrestrictions    (copyrightandotherrestrictions, single, optional)
      description             (string, single, optional)
      extension               (string, single, optional)
   meta_attributes
      elem_ord                [cost, copyrightandotherrestrictions, description, extension]
xml_seq relation
   attributes
      kind                    (kind, single, optional)
      resource                (resource, single, optional)
      extension               (string, single, optional)
   meta_attributes
      elem_ord                [kind, resource, extension]
xml_seq annotation
   attributes
      person                  (person, single, optional)
      date                    (date, single, optional)
      description             (string, single, mandatory)
      extension               (string, single, optional)
   meta_attributes
      elem_ord                [person, date, description, extension]
xml_seq classification
```

```
   attributes
      purpose                (purpose, single, optional)
      taxonpath              (taxonpath, list, optional)
      description            (string, single, optional)
      keyword                (string, list, optional)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [purpose, taxonpath, description, keyword, extension]
xml_seq catalogentry
   attributes
      catalog                (string, single, mandatory)
      entry                  (string, single, mandatory)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [catalog, entry, extension]
xml_seq contribute
   attributes
      role                   (role, single, mandatory)
      centity                (centity, list, optional)
      date                   (date, single, optional)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [role, centity, date, extension]
xml_seq requirement
   attributes
      type                   (type, single, optional)
      name                   (name, single, optional)
      minimumversion         (string, single, optional)
      maximumversion         (string, single, optional)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [type, name, minimumversion, maximumversion, extension]
xml_seq rights
   attributes
      cost                   (cost, single, optional)
      copyrightandotherrestrictions    (copyrightandotherrestrictions, single, optional)
      description            (string, single, optional)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [cost, copyrightandotherrestrictions, description, extension]
xml_seq resource
   attributes
      identifier             (string, single, optional)
      description            (string, single, optional)
      catalogentry           (catalogentry, list, optional)
      extension              (string, single, optional)
   meta_attributes
      elem_ord               [identifier, description, catalogentry, extension]
xml_seq taxonpath
   attributes
      source                 (string, single, optional)
      taxon                  (taxon, single, optional)
   meta_attributes
      elem_ord               [source, taxon]
xml_seq taxon
   attributes
      id                     (string, single, optional)
      entry                  (string, single, optional)
      taxon                  (taxon, single, optional)
   meta_attributes
      elem_ord               [id, entry, taxon]
xml_seq langstring
   attributes
      content                (string, single, mandatory)
      xml_lang               (string, single, optional)
   meta_attributes
      elem_ord               [content]
      att_lst                [xml_lang]
xml_seq location
   attributes
      content                (string, single, mandatory)
      type                   (string, single, optional)
   meta_attributes
      elem_ord               [content]
      att_lst                [type]
```

Classes `structure`, `aggregationlevel`, `status`, `role`, `type`, `name`, `interactivitytype`, `learningresourcetype`, `interactivitylevel`, `semanticdensity`, `intendedenduserrole`, `context`, `difficulty`, `cost`, `copyrightandotherrestrictions`, `kind`, `purpose` have the following structure:

```
xml_seq ClassName
   attributes
      source                 (source, single, mandatory)
      value                  (value, single, mandatory)
   meta_attributes
      elem_ord               [source, value]
```

Classes `date`, `duration`, `typicallearningtime` have the following structure:

```
xml_seq ClassName
   attributes
      datetime               (string, single, optional)
      description            (description, single, optional)
   meta_attributes
      elem_ord               [datetime, description]
```

Classes `centity`, `person` have the following structure:

```
xml_seq ClassName
   attributes
      vcard                  (string, single, mandatory)
   meta_attributes
      elem_ord               [vcard]
```

**LIP**

```
xml_seq learnerinformation
   attributes
      comment                (comment, single, optional)
      contentype             (contentype, single, optional)
      learnerinformation_alt1 (learnerinformation_alt1, list, optional)
      xml_lang               (string, single, optional)
   meta_attributes
      elem_ord               [comment, contentype, learnerinformation_alt1]
      att_lst                [xml_lang]
      alias                  [identification-learnerinformation_alt1,
                                goal-learnerinformation_alt1, qcl-learnerinformation_alt1,
                                activity-learnerinformation_alt1,
                                competency-learnerinformation_alt1,
                                transcript-learnerinformation_alt1,
                                accessibility-learnerinformation_alt1,
                                interest-learnerinformation_alt1,
                                affiliation-learnerinformation_alt1,
                                securitykey-learnerinformation_alt1,
                                relationship-learnerinformation_alt1,
                                ext_learnerinfo-learnerinformation_alt1]
xml_alt learnerinformation_alt1
   attributes
      identification         (identification, single, optional)
      goal                   (goal, single, optional)
      qcl                    (qcl, single, optional)
      activity               (activity, single, optional)
      competency             (competency, single, optional)
      transcript             (transcript, single, optional)
      accessibility          (accessibility, single, optional)
      interest               (interest, single, optional)
      affiliation            (affiliation, single, optional)
      securitykey            (securitykey, single, optional)
      relationship           (relationship, single, optional)
      ext_learnerinfo        (string, single, optional)
xml_seq activity
   attributes
      typename               (typename, single, optional)
      comment                (comment, single, optional)
      contentype             (contentype, single, optional)
      date                   (date, list, optional)
      status                 (status, single, optional)
      units                  (units, single, optional)
      activity_alt1          (activity_alt1, list, optional)
      description            (description, single, optional)
      activity               (activity, list, optional)
      ext_activity           (string, single, optional)
   meta_attributes
```

```
      elem_ord               [typename, comment, contentype, date, status, units,
                                activity_alt1, description, activity, ext_activity]
      alias                  [learningactivityref-activity_alt1, definition-activity_alt1,
                                product-activity_alt1, testimonial-activity_alt1,
                                evaluation-activity_alt1]
xml_alt activity_alt1
   attributes
      learningactivityref    (learningactivityref, single, optional)
      definition             (definition, single, optional)
      product                (product, single, optional)
      testimonial            (testimonial, single, optional)
      evaluation             (evaluation, single, optional)
      att_lst                [xml_lang]
xml_seq contentype
   attributes
      comment                (comment, single, optional)
      contentype_alt1        (contentype_alt1, list, mandatory)
      ext_contentype         (string, single, optional)
   meta_attributes
      elem_ord               [comment, contentype_alt1, ext_contentype]
      alias                  [referential-contentype_alt1, temporal-contentype_alt1,
                                privacy-contentype_alt1]
xml_alt contentype_alt1
   attributes
      referential            (referential, single, optional)
      temporal               (temporal, single, optional)
      privacy                (privacy, single, optional)
xml_seq referential
   attributes
      referential_alt1       (referential_alt1, single, mandatory)
   meta_attributes
      elem_ord               [referential_alt1]
      alias                  [sourcedid-referential_alt1, indexid-referential_alt1,
                             sourcedid-referential_alt1_seq1, indexid-referential_alt1_seq1]
xml_alt referential_alt1
   attributes
      sourcedid              (sourcedid, single, optional)
      indexid                (indexid, single, optional)
      referential_alt1_seq1  (referential_alt1_seq1, single, optional)
   meta_attributes
      alias                  [sourcedid-referential_alt1_seq1,
                                indexid-referential_alt1_seq1]
xml_alt referential_alt1_seq1
   attributes
      sourcedid              (sourcedid, single, mandatory)
      indexid                (indexid, single, mandatory)
   meta_attributes
      elem_order             [sourcedid, indexid]
xml_seq definition
   attributes
      typename               (typename, single, optional)
      comment                (comment, single, optional)
      contentype             (contentype, single, optional)
      definitionfield        (definitionfield, list, optional)
      description            (description, single, optional)
      definition             (definition, list, optional)
      ext_definition         (string, single, optional)
   meta_attributes
      elem_ord               [typename, comment, contentype, definitionfield, description,
                                definition, ext_definition]
xml_seq sourcedid
   attributes
      source                 (source, single, mandatory)
      id                     (id, single, mandatory)
   meta_attributes
      elem_ord               [source, id]
xml_seq typename
   attributes
      tysource               (tysource, single, optional)
      tyvalue                (tyvalue, single, mandatory)
   meta_attributes
      elem_ord               [tysource, tyvalue]
xml_seq evalmetadatafield
   attributes
      fieldlabel             (fieldlabel, single, mandatory)
      fielddata              (fielddata, single, mandatory)
      xml_lang               (string, single, optional)
   meta_attributes
      elem_ord                [fieldlabel, fielddata]
xml_seq definitionfield
   attributes
      fieldlabel              (fieldlabel, single, mandatory)
      fielddata               (fielddata, single, mandatory)
   meta_attributes
      elem_ord                [fieldlabel, fielddata]
xml_seq fieldlabel
   attributes
      typename                (typename, single, mandatory)
   meta_attributes
      elem_ord                [typename]
xml_seq tysource
   attributes
      content                 (string, single, mandatory)
      sourcetype              (string, single, optional)
      e-dtype                 (string, single, mandatory)
   meta_attributes
      elem_ord                [content]
      att_lst                 [e-dtype, sourcetype]
xml_seq tyvalue
   attributes
      content                 (string, single, mandatory)
      xml_lang                (string, single, optional)
      e-dtype                 (string, single, mandatory)
   meta_attributes
      elem_ord                [content]
      att_lst                 [e-dtype, xml_lang]
```

Classes `comment`, `source`, `id`, `fielddata`, `indexid` have the following structure:

```
xml_seq ClassName
   attributes
      content                 (string, single, mandatory)
      e-dtype                 (string, single, mandatory)
   meta_attributes
      elem_ord                [content]
      att_lst                 [e-dtype]
```

**CP**

```
xml_seq manifest
   attributes
      metadata             (metadata, single, optional)
      organizations        (organizations, single, mandatory)
      resources            (resources, single, mandatory)
      manifest             (manifest, list, optional)
      identifier           (string, single, mandatory)
      version              (string, single, optional)
   meta_attributes
      elem_ord             [metadata, organizations, resources, manifest]
      att_lst              [identifier, version]
xml_seq metadata
   attributes
      schema               (string, single, optional)
      schemaversion        (string, single, optional)
   meta_attributes
      elem_ord             [schema, schemaversion]
xml_seq organizations
   attributes
      organization         (organization, list, optional)
      default              (string, single, optional)
   meta_attributes
      elem_ord             [organization]
      att_lst              [default]
xml_seq organization
   attributes
      title                (string, single, optional)
      item                 (item, list, optional)
      metadata             (metadata, single, optional)
      identifier           (string, single, mandatory)
   meta_attributes
      elem_ord             [title, item, metadata]
      att_lst              [identifier]
xml_seq item
   attributes
      title                (string, single, optional)
      item                 (item, list, optional)
      metadata             (metadata, single, optional)
      identifier           (string, single, mandatory)
      isvisible            (string, single, optional)
      parameters           (string, single, optional)
      identifierref        (string, single, optional)
      a-dtype              (string, list, optional)
   meta_attributes
      elem_ord             [title, item, metadata]
      att_lst              [identifier, isvisible, parameters, identifierref, a-dtype]
xml_seq resources
   attributes
      resource             (resource, list, optional)
   meta_attributes
      elem_ord             [resource]
xml_seq resource
   attributes
      metadata             (metadata, single, optional)
      file                 (file, list, mandatory)
      dependency           (dependency, list, optional)
      identifier           (string, single, mandatory)
      type                 (string, single, mandatory)
      href                 (string, single, optional)
   meta_attributes
      elem_ord             [metadata, file, dependency]
      att_lst              [identifier, type, href]
      empty                [dependency]
xml_seq dependency
   attributes
      identifierref        (string, single, optional)
   meta_attributes
      elem_ord             []
      att_lst              [identifierref]
xml_seq file
   attributes
      metadata             (metadata, single, optional)
      href                 (string, single, mandatory)
   meta_attributes
      elem_ord             [metadata]
      att_lst              [href]
```

## Appendix C. XML DOCUMENT EXAMPLE

This appendix presents an example of an XML document that conforms to the LIP DTD (see Appendix A).

```
<learnerinformation>
   <comment>An example of LIP Activity information.</comment>
   <contentype>
      <referential>
         <sourcedid>
            <source>IMS_LIP_V1p0_Example</source>
            <id>2001</id>
         </sourcedid>
      </referential>
   </contentype>
   <activity>
      <typename>
         <tysource sourcetype="imsdefault"/>
         <tyvalue>Education</tyvalue>
      </typename>
      <contentype>
         <referential>
            <indexid>activity_1</indexid>
         </referential>
      </contentype>
      <definition>
         <typename>
            <tysource sourcetype="imsdefault"/>
            <tyvalue>Course</tyvalue>
         </typename>
         <contentype>
```

```xml
            <referential>
               <indexid>DegreeCourse</indexid>
            </referential>
         </contentype>
         <definition>
            <typename>
               <tysource sourcetype="imsdefault"/>
               <tyvalue>Curriculum</tyvalue>
            </typename>
            <contentype>
               <referential>
                  <indexid>Year1</indexid>
               </referential>
            </contentype>
            <definition>
               <typename>
                  <tysource sourcetype="imsdefault"/>
                  <tyvalue>Module</tyvalue>
               </typename>
               <contentype>
                  <referential>
                     <indexid>Electronics_101</indexid>
                  </referential>
               </contentype>
               <definitionfield>
                  <fieldlabel>
                     <typename>
                        <tyvalue>Lecture1</tyvalue>
                     </typename>
                  </fieldlabel>
                  <fielddata>BooleanLogic</fielddata>
               </definitionfield>
               <definitionfield>
                  <fieldlabel>
                     <typename>
                        <tyvalue>Lecture2</tyvalue>
                     </typename>
                  </fieldlabel>
                  <fielddata>Transistors</fielddata>
               </definitionfield>
            </definition>
            <definition>
               <typename>
                  <tysource sourcetype="imsdefault"/>
                  <tyvalue>Module</tyvalue>
               </typename>
               <contentype>
                  <referential>
                     <indexid>Maths_101</indexid>
                  </referential>
               </contentype>
               <definitionfield>
                  <fieldlabel>
                     <typename>
                        <tyvalue>Lecture1</tyvalue>
                     </typename>
                  </fieldlabel>
                  <fielddata>BooleanLogic1</fielddata>
               </definitionfield>
               <definitionfield>
                  <fieldlabel>
                     <typename>
                        <tyvalue>Lecture2</tyvalue>
                     </typename>
                  </fieldlabel>
                  <fielddata>BooleanLogic2</fielddata>
               </definitionfield>
            </definition>
         </definition>
      </definition>
   </activity>
</learnerinformation>
```

## Appendix D. OBJECT-ORIENTED REPRESENTATION OF AN XML DOCUMENT

This appendix presents the X-DEVICE object-oriented representation for the XML document of Appendix C.

```
object         0#source
   instance    source
   attributes
      content     'IMS_LIP_V1p0_Example'
      e-dtype     'string'
object         1#id
   instance    id
   attributes
      content     '2001'
      e-dtype     'string'
object         2#tysource
   instance    tysource
   attributes
      content     ''
      e-dtype     'string'
      sourcetype  'imsdefault'
object         3#tyvalue
   instance    tyvalue
   attributes
      content     'Education'
      e-dtype     'string'
      xml_lang    'en'
object         4#indexid
   instance    indexid
   attributes
      content     'activity_1'
      e-dtype     'string'
object         5#tyvalue
   instance    tyvalue
   attributes
      content     'Course'
      e-dtype     'string'
      xml_lang    'en'
object         6#indexid
   instance    indexid
   attributes
      content     'DegreeCourse'
      e-dtype     'string'
object         7#tyvalue
   instance    tyvalue
   attributes
      content     'Curriculum'
      e-dtype     'string'
      xml_lang    'en'
object         8#indexid
   instance    indexid
   attributes
      content     'Year1'
      e-dtype     'string'
object         9#tyvalue
   instance    tyvalue
   attributes
      content     'Module'
      e-dtype     'string'
      xml_lang    'en'
object         10#indexid
   instance    indexid
   attributes
      content     'Electronics_101'
      e-dtype     'string'
object         11#tyvalue
   instance    tyvalue
   attributes
      content     'Lecture1'
      e-dtype     'string'
      xml_lang    'en'
object         12#fielddata
   instance    fielddata
   attributes
      content     'BooleanLogic'
      e-dtype     'string'
object         13#tyvalue
   instance    tyvalue
   attributes
      content     'Lecture2'
      e-dtype     'string'
      xml_lang    'en'
object         14#fielddata
   instance    fielddata
   attributes
      content     'Transistors'
      e-dtype     'string'
object         15#indexid
   instance    indexid
   attributes
      content     'Maths_101'
      e-dtype     'string'
object         16#fielddata
   instance    fielddata
   attributes
      content     'BooleanLogic1'
      e-dtype     'string'
object         17#fielddata
   instance    fielddata
   attributes
      content     'BooleanLogic2'
      e-dtype     'string'
object         18#sourcedid
   instance    sourcedid
   attributes
      source   0#source
      id       1#id
object         19#typename
   instance    typename
   attributes
      tysource    2#tysource
      tyvalue     3#tyvalue
object         20#referential
   instance    referential
   attributes
      referential_alt1  21#referential_alt1
object         21#referential_alt1
   instance    referential_alt1
   attributes
      indexid     4#indexid
      sourcedid   ∅
      referential_alt1_seq1   ∅
object         22#typename
   instance    typename
   attributes
      tysource    2#tysource
      tyvalue     5#tyvalue
object         23#referential
   instance    referential
   attributes
      referential_alt1  24#referential_alt1
object         24#referential_alt1
   instance    referential_alt1
   attributes
      indexid     6#indexid
      sourcedid   ∅
      referential_alt1_seq1   ∅
```

```
object         25#typename
   instance    typename
   attributes
      tysource    2#tysource
      tyvalue     7#tyvalue
object         26#referential
   instance    referential
   attributes
      referential_alt1  27#referential_alt1
object         27#referential_alt1
   instance    referential_alt1
   attributes
      indexid     8#indexid
      sourcedid   ∅
      referential_alt1_seq1   ∅
object         28#typename
   instance    typename
   attributes
      tysource    2#tysource
      tyvalue     9#tyvalue
object         29#referential
   instance    referential
   attributes
      referential_alt1  30#referential_alt1
object         30#referential_alt1
   instance    referential_alt1
   attributes
      indexid     10#indexid
      sourcedid   ∅
      referential_alt1_seq1   ∅
object         31#typename
   instance    typename
   attributes
      tysource    ∅
      tyvalue     11#tyvalue
object         32#typename
   instance    typename
   attributes
      tysource    ∅
      tyvalue     13#tyvalue
object         33#referential
   instance    referential
   attributes
      referential_alt1  34#referential_alt1
object         34#referential_alt1
   instance    referential_alt1
   attributes
      indexid     15#indexid
      sourcedid   ∅
      referential_alt1_seq1   ∅
object         35#referential
   instance    referential
   attributes
      referential_alt1  36#referential_alt1
object         36#referential_alt1
   instance    referential_alt1
   attributes
      indexid     ∅
      sourcedid   18#sourcedid
      referential_alt1_seq1   ∅
object         37#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [38#contentype_alt1]
      ext_contentype    ∅
object         38#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    20#referential
      temporal       ∅
      privacy        ∅
```

```
object         39#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [40#contentype_alt1]
      ext_contentype    ∅
object         40#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    23#referential
      temporal       ∅
      privacy        ∅
object         41#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [42#contentype_alt1]
      ext_contentype    ∅
object         42#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    26#referential
      temporal       ∅
      privacy        ∅
object         43#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [44#contentype_alt1]
      ext_contentype    ∅
object         44#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    29#referential
      temporal       ∅
      privacy        ∅
object         45#fieldlabel
   instance    fieldlabel
   attributes
      typename    31#typename
object         46#fieldlabel
   instance    fieldlabel
   attributes
      typename    32#typename
object         47#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [48#contentype_alt1]
      ext_contentype    ∅
object         48#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    33#referential
      temporal       ∅
      privacy        ∅
object         49#contentype
   instance    contentype
   attributes
      comment          ∅
      contentype_alt1   [50#contentype_alt1]
      ext_contentype    ∅
object         50#contentype_alt1
   instance    contentype_alt1
   attributes
      referential    35#referential
      temporal       ∅
      privacy        ∅
object         51#definitionfield
   instance    definitionfield
   attributes
      fieldlabel     45#fieldlabel
      fielddata      12#fielddata
```

```
object         52#definitionfield
   instance    definitionfield
   attributes
      fieldlabel     46#fieldlabel
      fielddata      14#fielddata
object         53#definitionfield
   instance    definitionfield
   attributes
      fieldlabel     45#fieldlabel
      fielddata      16#fielddata
object         54#definitionfield
   instance    definitionfield
   attributes
      fieldlabel     46#fieldlabel
      fielddata      17#fielddata
object         55#definition
   instance    definition
   attributes
      typename          28#typename
      comment           ∅
      contentype        43#contentype
      definitionfield   [51#definitionfield,
52#definitionfield]
      description       ∅
      definition        []
      ext_definition    ∅
object         56#definition
   instance    definition
   attributes
      typename          28#typename
      comment           ∅
      contentype        47#contentype
      definitionfield   [53#definitionfield,
54#definitionfield]
      description       ∅
      definition        []
      ext_definition    ∅
object         57#definition
   instance    definition
   attributes
      typename          25#typename
      comment           ∅
      contentype        41#contentype
      definitionfield   []
      description       ∅
      definition        [55#definition,
56#definition]
      ext_definition    ∅
object         58#definition
   instance    definition
   attributes
      typename          22#typename
      comment           ∅
      contentype        39#contentype
      definitionfield   []
      description       ∅
      definition        [57#definition]
      ext_definition    ∅
```

```
object         59#activity
   instance    activity
   attributes
      typename          19#typename
      comment           ∅
      contentype        37#contentype
      date              []
      status            ∅
      units             ∅
      activity_alt1     [60#activity_alt1]
      description       ∅
      activity          []
      ext_activity      ∅object
   60#activity_alt1
   instance    activity_alt1
   attributes
      learningactivityref  ∅
      definition           58#definition
      product              ∅
      testimonial          ∅
      evaluation           ∅
object         61#comment
   instance    comment
   attributes
      content     'An example of LIP Activity
information.'
      e-dtype     'string'
      xml_lang    'en'
object         62#learnerinformation
   instance    learnerinformation
   attributes
      comment           61#comment
      contentype        49#contentype
      learnerinformation_alt1    []
      xml_lang          'en'
object         63# learnerinformation_alt1
   instance    learnerinformation_alt1
   attributes
      identification    ∅
      goal              ∅
      qcl               ∅
      activity          59#activity
      competency        ∅
      transcript        ∅
      accessibility     ∅
      interest          ∅
      affiliation       ∅
      securitykey       ∅
      relationship      ∅
      ext_learnerinfo   ∅
```

# Appendix E. X-DEVICE RULE LANGUAGE SYNTAX

This appendix contains the syntax of the X-DEVICE deductive rule language in BNF notation.

```
<rule> ::= if <condition> then <consequence>
<condition> ::= <inter-object-pattern>
<consequence> ::= {<action> | <conclusion> | <derived_attribute_template>}
<inter-object-pattern> ::= <condition-element> ['and' <inter-object-pattern>]
<inter-object-pattern> ::= <inter-object-pattern> 'and' <prolog_cond>
<condition-element> ::= ['not'] <intra-object-pattern>
<intra-object-pattern> ::= [<inst_expr>'@']<class_expr>['('<attr-patterns>')']
<attr-patterns> ::= <attr-pattern>[','<attr-patterns>]
```

```
<attr-pattern> ::= <attr-expr>['.'<path_expr>] {':'<variable> | <predicates>
                                       | ':'<variable> <predicates>
                                       | <list-operator> <variable>}
<path_expr> ::= <nt-attr-expr> ['.'<path_expr>]
<attr-expr> ::= {<nt-attr-expr>|<t-attr>|<normal-attr>'↑'}
<nt-attr-expr> ::= <nt-attr>[{'*'|<integer>}]
<nt-attr-expr> ::= {'*'|'+'}
<nt-attr> ::= {<normal-attr>|<system-attr>}
<t-attr> ::= {<xml-attr>|<empty-attr>}
<normal-attr> ::= <attr>
<system-attr> ::= '!'<attr>
<xml-attr> ::= '^'<attr>
<empty-attr> ::= '∅'<attr>
<attr> ::= {<attribute>|<variable>}
<predicates> ::= <rel-operator> <value> [{ '&' | ';' } <predicates>]
<predicates> ::= <set-operator> <set>
<rel-operator> ::= '=' | '>' | '>=' | '=<' | '<' | '\=' | '$' | <date-operator>
<date-operator> ::= '$'{'y'|'m'|'d'}
<set-operator> ::= '⊂' | '⊄' | '⊆' | '∈' | '∉' | '⊃' | '\⊃' | '⊇'
<list-operator> ::= '∍' | '\∍'
<list-operator> ::= '∍'<order_expr>
<order_expr> ::= {<abs_order>|<rel_order>|'{'<rel_order>','<abs_order>'}'}
<abs_order> ::= <rel-operator><integer> | <integer>
<rel_order> ::= { 'before' | 'after' }'('<variable>')'
<rel_order> ::= 'between' '('<variable>','<variable>')'
<value> ::= <constant> | <variable> | <arith_expr>
<set> ::= '['<constants>']'
<prolog_cond> ::= 'prolog' '{'<prolog_goal>'}'
<action> ::= <prolog_goal>
<conclusion> ::= <derived_class_template>
<conclusion> ::= {'xml_result' | 'shallow_result'}'('<elem_expr>')'
<conclusion> ::= {'xml_sorted' | 'shallow_sorted'}'('[<group_list>
                                                           ['-'<order_list>]',']<elem_expr>')'
<elem_expr> ::= <derived_class_template>
<elem_expr> ::= <derived_class>'('<derived_class_template>')'
<derived_class_template> ::= <derived_class>'('<templ-patterns>')'
<derived_attribute_template> ::= <variable>'@'{<class>}'('<templ-patterns>')'
<templ-patterns> ::= <templ-pattern> [',' <templ-pattern>]
<templ-pattern> ::= {<normal-attr>|<system-attr>|<xml-attr>}':'{<value> | <aggregate_expr>}
<templ-pattern> ::= <empty-attr>
<aggregate_expr> ::= <aggregate_function>'('<variable>')'}
<aggregate_expr> ::= 'ord_list('<variable>['-'<group_list>['-'<order_list>]]')'
<aggregate_function> ::= 'count'|'sum'|'avg'|'max'|'min'|'list'|'string'
<group_list> ::= '['<variable>[','<variable>]']'
<order_list> ::= '['<ord_symbol>[','<ord_symbol>]']'
<ord_symbol> ::= {'<' | '>'}
<inst_expr> ::= {<variable>|<class>}
<class_expr> ::= {<variable>|<class>}
<class_expr> ::= <inst_expr>'/'<class>
<class> ::= an existing class or meta-class of the OODB schema
<derived_class> ::= an existing derived class or a non-existing base class of the OODB schema
<attribute> ::= an existing attribute of the corresponding OODB class
<prolog_goal> ::= an arbitrary Prolog/ADAM goal
<constants> ::= <constant>[','<constants>]
<constant> ::= a valid constant of an OODB simple attribute type
<variable> ::= a valid Prolog variable
<arith_expr> ::= a valid Prolog arithmetic expression
```