

Modeling Interorganizational Workflows with XML Nets

Kirsten Lenz

*Institute of Information Systems
J.W. Goethe-University
D-60054 Frankfurt/Main, Germany
lenz@wiwi.uni-frankfurt.de*

Andreas Oberweis

*Institute of Information Systems
J.W. Goethe-University
D-60054 Frankfurt/Main, Germany
oberweis@wiwi.uni-frankfurt.de*

Abstract

Efficient interorganizational business processes in the field of e-commerce require the integration of electronic document and data interchange and interorganizational processes. We propose to support interorganizational business processes by so-called XML nets, a new kind of high-level Petri nets. XML nets are a formal, graphical modeling language that allows to model both the flow of XML documents and the business process. XML nets are based on GXSL, a graphical XML schema definition language, and its related graphical XML document manipulation language (XManiLa). They can be directly executed by a workflow engine. Advantage of the formal foundation of XML nets can be taken e.g. when analyzing a (global) interorganizational workflow model: XML nets support the process of identifying relevant process fragments, assigning them to the appropriate organizational units, and thus help deriving an improved, process-oriented organizational structure.

1. Introduction

Efficient interorganizational business processes in the field of e-commerce require the integration of electronic document interchange and interorganizational processes: From the document point of view, there is a need for electronically interchanging structured documents and data between the organizations involved. From a process-oriented perspective, modeling, analysis, and automated execution of distributed workflows gain more and more importance. Moreover, the advantages of workflow driven electronic document interchange are relevant for the intra-organizational process management, especially with the emerging integration of mass data in databases and document management. The proposed XML nets, a new kind of high-level Petri nets, are a formal, graphical modeling language that allows to model both the flow of XML documents and the business process. They are based on

GXSL, a graphical XML schema definition language, and the XML document manipulation language XManiLa.

XML nets can be directly executed by a workflow engine and therefore allow for a detailed analysis and simulation of distributed business processes. Additionally, XML nets support the process of identifying relevant process fragments and assigning them to the appropriate organizational units, thus help to derive an improved, process-oriented organizational structure.

Related concepts for graphical XML data modeling languages and XML query languages can be found for example in [2], [5], [7], and [18]. Many of the XML net concepts proposed in this paper rely on so-called SGML nets [20], a variant of Petri nets for the modeling of workflows based on SGML documents.

Our paper is structured as follows: First, we give a brief introduction to XML, Petri nets, and workflow management. In the next section, we present a graphical XML schema definition language and derive a graphical XML document manipulation language. Finally, we propose XML nets for the intra- and interorganizational workflow driven document interchange.

1.1. Extensible markup language

The Extensible Markup Language (XML) [9] is a document declaration standard proposed by the World Wide Web Consortium (W3C, <http://www.w3.org/>). It is a simplification of the Standard Generalized Markup Language (SGML, ISO 8879). An XML document type definition (DTD) allows to specify a document type with problem specific markups as a class of XML documents conforming to that specific DTD, i.e. a set of documents that follow these document design rules. Figure 1 shows a simple DTD for an employee document with personnel information, a list of projects she or he works in and a list of the skills.

A corresponding valid document for the employee Mary L. Miller is shown in Figure 2.

```
<!-- DTD for employee documents-->
<!ELEMENT employee (empname,
    work_in_proj*, skill*)
<!ATTLIST employee empid ID #REQUIRED
    sex (male|female) "female">
<!ELEMENT empname (title?,
    (firstname|abbreviation)+,
    lastname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT abbreviation (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT work_in_proj (#PCDATA)>
<!ATTLIST work_in_proj projid
    IDREF #IMPLIED>
<!ELEMENT skill (#PCDATA)>
```

Figure 1: Document type definition for employee documents

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM
"http://...empdoc.dtd">
<employee empid="0815" sex="female">
  <empname>
    <firstname>Mary</firstname>
    <abbreviation>L.</abbreviation>
    <lastname>Miller</lastname>
  </empname>
  <work_in_proj projid="328">20
  </work_in_proj>
  <work_in_proj projid="85">10
  </work_in_proj>
  <skill>Java</skill>
  <skill>C++</skill>
  <skill>Oracle 8</skill>
</employee>
```

Figure 2: Valid employee document

In the near future, XML DTDs may be complemented by XML schemas [21], a new standard promoted by W3C which attempts to overcome the limits of a DTD e.g. concerning the data-oriented role of XML in e-commerce applications. XML schemas, a functional superset of DTDs, are written in XML and can therefore be parsed and validated like XML documents. Moreover, they provide a way to specify data types and thereby ensure data integrity for valid XML documents.

1.2. Petri nets

For the modeling of business processes several more or less formal description languages have been proposed. In contrast to other languages, Petri nets [4, 17] combine the advantages of the graphical representation of processes with a formal definition. Beyond visualization of processes Petri nets allow for the analysis and validation of business processes [6]. A simple example of a low-

level Petri net is shown in Figure 3. A planned project is initialized. Afterwards, employees can be assigned to the project. At the begin of a project the project identifier is assigned to the responsible department. After finishing the project, the employee information must be updated: new skills learned in the project have to be added, whereas the project has to be removed from the list of projects the employee works in.

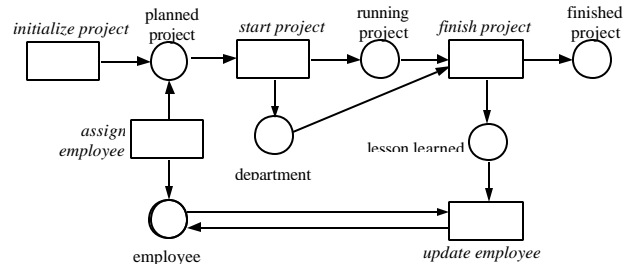


Figure 3: Simple Petri net for the process of project execution

High-level Petri nets such as predicate/transition nets (Pr/T-nets) [12, 15] or nested relation/transition nets (NR/T-nets) [16] integrate behavior- and object-related aspects of workflows. In Pr/T-nets or NR/T-nets the places represent relation schemes according to which the marking of the net assigns a relation to each place. A transition represents a class of operations on the relations in the adjacent places. When a transition occurs, the respective tuples are removed from its input places and inserted into its output places according to the edge inscriptions.

1.3. Workflow Management

Workflow management supports business process management: business processes can be modeled by more or less formal or graphical modeling languages like event-driven process chains [19] or Petri nets. The workflow itself, i.e. a concrete process execution, can be interpreted as an instantiation of the workflow model. The main tasks of workflow management include the specification of the workflow model, its analysis and verification, the execution of the workflow by a suitable workflow architecture, and the development of workflow applications [10].

Petri nets are well suited and frequently proposed for workflow modeling [11]: They provide a restricted number of graphical modeling concepts and possess at the same time formal semantics. Their formal foundation enables the application of theoretical analysis techniques (e.g. to find out whether a certain system state can be reached or not) and the model-driven execution of the workflow. Moreover, they integrate data and behavior

aspects and therefore meet the need to model for example interorganizational workflows based on electronic document interchange. The use of Petri nets at different formalization levels allows for a stepwise refinement of a workflow model. A survey of different approaches for Petri net based business process and workflow management can be found for example in [1].

2. Graphical languages for XML document management

In this section, we introduce the graphical XML schema definition language (GXSL) for graphically specifying XML schemas that represent a DTD. XManiLa, an extension to the graphical XML schema definition language, is proposed for the retrieval and manipulation, especially insertion and deletion, of XML documents.

2.1. XML schema definition with GXSL

The XML document type definition represents a grammar for the declaration of documents. It consists of a set of markup declarations of different types: element, attribute list, entity or notation declaration. Unfortunately, this grammar is textual so that the DTDs for complex objects often may become quite unreadable. The advantages of a graphical schema definition language (like the entity/relationship model for the database design) are missing.

In the following, we propose a graphical schema definition language for the design of XML document types, that we call *graphical XML schema definition language* (GXSL). The presented version of GXSL is DTD-based, i.e. a DTD can be unambiguously derived from the *graphical XML schema* (GXS).

Instead of creating a completely new graphical modeling language for XML document types, we rely on well known data modeling concepts (of the E/R-model and other semantic data models), which all had their impact on the static modeling concepts of the Unified Modeling Language (UML) [3]. The main advantage of UML is that it is a highly accepted integration of well-known modeling concepts and guidelines and that it comprises a generic notation for the derivation of new graphical modeling languages for specific purposes. Due to space limitations, we focus on the logical structure of XML document types and omit XML entity declarations, internal subsets and INCLUDE or IGNORE markups. We also derive a simplification of the GXS conforming to UML which has been adapted to the quick, rather intuitive modeling of XML document types.

The main principles of XML DTDs are classification of documents with comparable structure and aggregation of document components. We present XML documents and their elements by UML classes, hierarchically (i.e. non-recursively) structured by composition or aggregation. In the following, the UML concepts for GXSL are explained in detail.

Simple element types. In general, element type declarations of the XML DTD are represented by classes in GXSL. By simple element types we understand element types without children (i.e. they are not composed of other element types), namely the EMPTY-element type, the ANY-element type and the predefined #PCDATA. When the element type declaration contains the keyword ANY, both other tags and general characters are allowed within the element tags. Empty elements have no content at all. See Figure 4 for the GXSL representation of simple element type declarations in a GXS by stereotyped «any»- and «empty»-classes.

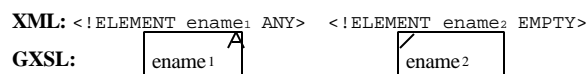


Figure 4: GXSL class representation of the ANY- and EMPTY-element type declaration

#PCDATA stands for *parsed character data* and symbolizes any sequence of general characters that does not contain any tag. The GXSL class «pcdata» has no name and provides its own icon as shown in Figure 5.



Figure 5: GXSL icon for the «pcdata»-class

In addition to element type declarations, a DTD may include attribute list declarations in order to specify element types. The attribute list declaration for an element type *ename* has the following pattern:

```
<!ATTLIST ename aname1 atype1
defaultdecl1
...
anamen atypen defaultdecln>.
```

aname is the attribute name, *atype* the attribute type and *defaultdecl* an optional default declaration. In a GXS, the attribute list is added to the element class (see Figure 6).

The default declaration consists of a default value and/or a default modifier. In Figure 6, the second attribute

of the element type has a default value. Table 1 shows the translation of XML default modifiers into GXSL.

ename
aname ¹ : atype ¹
aname ² : atype ² = default ²
...
aname ⁿ : atype ⁿ

Figure 6: Element class with attribute list

It is possible to restrict the GXS class representation to the element name with or even without the list of the attribute names in a survey diagram.

Table 1: GXSL class attribute for XML attribute declaration with default modifier

XML	GXSL	
#IMPLIED	aname: atype	attribute value may remain unspecified
#REQUIRED	aname [1]: atype	mandatory attribute value, specified by the occurrence indicator
#FIXED "dvalue"	aname: atype = dvalue {frozen}	attribute value must not be changed; default value required

Nested element types. Element types in XML can be nested in order to describe hierarchically structured documents, i.e. the element type declaration refers to other element type declarations of the DTD:

```
<!ELEMENT ename (...)>.
```

UML offers a special kind of association to adequately model the hierarchical relationship between different element types: composition and aggregation. Composition, represented by a filled diamond, is used to express that the child element is a component of exactly one element of the parent element type with existence dependency (e.g. a car consists of four wheels). Aggregation, the weaker form of composition and represented by an open diamond in the GXS (see Figure 7), can be used to express a loose, non dependent 'consists of'-relationship (e.g. a discussion group consists of four people). Additionally, a name and a name direction can be assigned to each association.

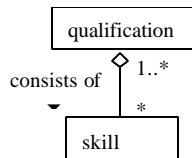


Figure 7: GXS for a nested element type

XML occurrence indicators as appendix of an element name or a group of elements can be translated into a cardinality that can be assigned to the associated subclass(es), see Table 2.

Table 2: GXSL cardinalities for XML occurrence indicators

XML occurrence operator	GXSL cardinality
none	1 mandatory
?	0..1 optional
*	* occurs zero or more times
+	1..* occurs one or more times

The cardinality corresponding to the parent classes are always 1 for the composition and 1..* for the aggregation. For example, the element type declaration `<!ELEMENT qualification (skill*)>` for the qualification of an employee as a set of skills has the GXS in Figure 7.

Choice. The XML element type can be defined as a choice between two or more element types. For example, the DTD contains the element type declaration `<!ELEMENT ename1 (ename2|ename3)>` and respective element type declarations for `ename2` and `ename3`. Within a document which is valid for this DTD there has to appear either an `ename2` element or an `ename3` element inside the `ename1` element (and must not appear both). With GXSL choice can be expressed by a constraint with {or}-condition between the respective alternative subclasses (see Figure 8).

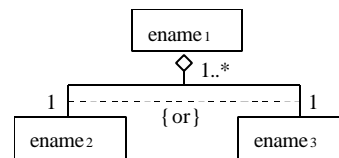


Figure 8: GXS for alternative element types

Sequence. A sequence of elements that have to appear in a predefined order within another element can be declared as a list of element names, separated by commas:

```
<!ELEMENT ename (ename1, ename2, ..., enamen)>.
```

We use the dependency modeling concept of UML to model a sequence with GXSL. Dependency is a (possibly directed) association and indicates sort of a using relationship between elements. In GXSL we introduce a directed dependency including the label {precedes} to order the subclasses of a superclass (see Figure 9) with

the following semantics: The document elements must follow the same order as their element types (respective their cardinality).

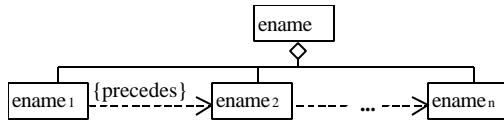


Figure 9: GXS for a sequence of element types

Cascading Nesting. XML allows for element type declarations with cascading nesting, i.e. element types are not declared explicitly but are implicitly given by an expression enclosed in parenthesis. For example, we can declare by $(ename_1, (ename_2|ename_3)+)$ that an element of type $ename_1$ must be followed by an iteration of either an $ename_2$ element or an $ename_3$ element. In this case, we have to introduce an abstract class to model the hierarchy of the element type (see Figure 10). This means splitting the expression into $(ename_1, abstract+)$ and $<!ELEMENT abstract (ename_2|ename_3)>$.

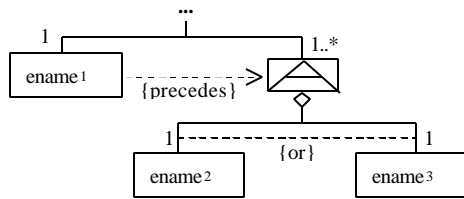


Figure 10: GXS for nested element types with an abstract class

The abstract class is a stereotyped class and has its own icon with an "A"-symbol in it. It is neither possible to attach a class name nor to assign attributes to it. However, the abstract class may have a cardinality.

Referential integrity. Attributes may be of type IDREF or IDREFS. Its values must be included in (or be a subset of respectively) the set of ID attribute values of the referenced element type. The attribute types IDREF and IDREFS therefore implicitly define referential integrity constraints between XML elements of a DTD. The GXSL even allows for explicit referential integrity constraints by {ref}-dependencies (see Figure 11). {ref}-dependencies are labeled by the referencing and the referenced attribute and the *-cardinality for attributes of type IDREFS.

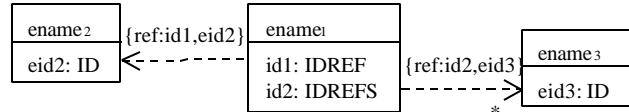


Figure 11: GXSL dependencies for referential integrity constraints

Simplifications of GXSL. In this section, we present some simplifications of GXSL concepts we described above. With these simplifications, GXSL does not completely coincide with UML anymore. But a simplified GXS still provides all necessary information without notational overload and the GXS design becomes more intuitive for those who are familiar with XML.

- The association name and direction can be omitted.
- Cardinality of the superclass of composition or aggregation is fixed and thus can be omitted.
- For GXSL, we declare default cardinality to be '1' (instead of '*' for the UML), which is identical to the default occurrence indicator of XML.
- We propose a stereotyped class and a special icon for the element type declaration $<!ELEMENT ename (\#PCDATA)>$ (Figure 12b) instead of the GXS conforming to UML (Figure 12a). Note that the #PCDATA-element class is not identical to the #PCDATA-class introduced before.

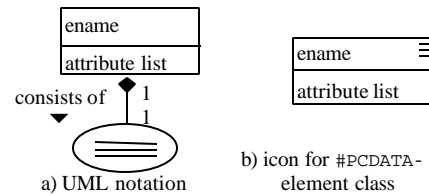


Figure 12: Simplification for #PCDATA-element types

- The constraint for a choice is expressed only by a dashed line. The {or}-condition can be omitted.
- The GXS for a sequence is simplified by omitting the {precedes}-dependencies. Nevertheless, elements of a valid document must appear in the same order as the respective classes in the GXS.
- The label of a {ref}-dependency may be omitted if either class attributes have not been specified in the GXS or both referencing and referenced attributes can be uniquely identified without any label.

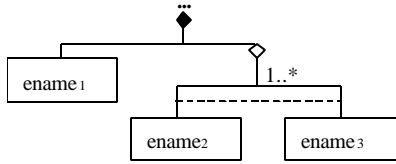


Figure 13: Simplification for cascading nesting of element types referring to Figure 10

- Cascading nesting of elements is represented without any abstract class. We allow for a tree structure of composition and/or aggregation (see Figure 13). The cardinality of the abstract class is written above the respective branch.

Example. We are now able to construct a GXS (in Figure 14) which is equivalent to the employee DTD of Figure 1.

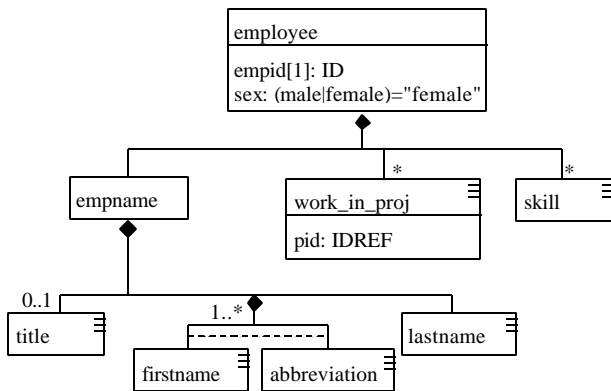


Figure 14: GXS for the employee DTD of Figure 1

Compared to the textual representation in Figure 1, the graphical representation gives a better overview of the defined elements and the logical structure of a valid employee document.

2.2. Manipulation of XML documents with XManiLa

We can use GXSL with some extensions also for querying and manipulating XML documents. The GXSL-based XML document manipulation language is called XManiLa. A GXSL-schema can be seen as a template for a set of XML documents that specifies the structure of matching documents. In order to enable also a content based query, we allow for assigning constants or variables to an element or an attribute. For example, we may want to search for all female employees that have a firstname followed by an abbreviation and the lastname "Miller". The GXS in Figure 15 describes this query.

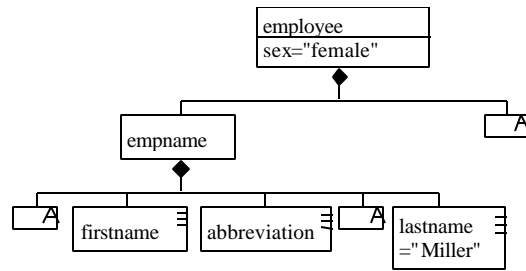


Figure 15: Female employees named Miller with a firstname followed by an abbreviation

XManiLa is not only suited for document retrieval, but also for insertion and deletion (and thereby also for updating). These operations either concern a whole document or elements on lower hierarchy levels. Elements to be inserted or deleted are depicted by a solid line on the left side of the element's rectangle. The insert-and delete-qualification is inherited by all subclasses. For examples of insertion and deletion operations on the employee documents see Figure 16.

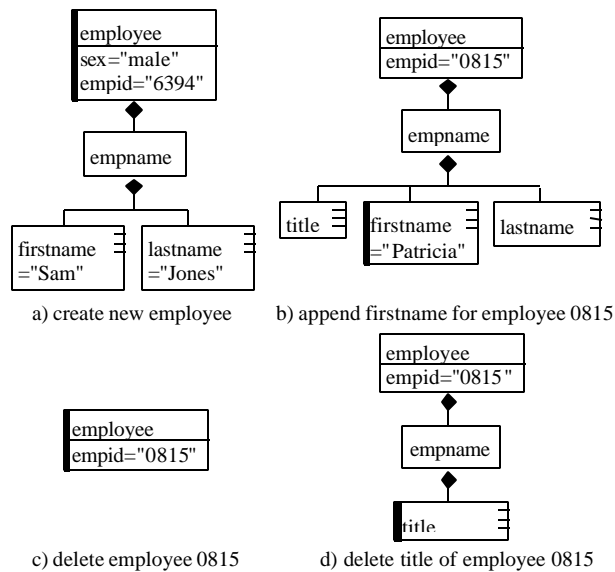


Figure 16: Insertion and deletion operations on the employee documents

3. Interorganizational data interchange based on XML

The interchange of XML documents between organizations can be handled in different ways:

- The XML document can be sent to another organization without any DTD. From the recipient's point of view the document only fulfills the weaker requirements of being well-formed (i.e. has a correct syntax), supplementary information is missing.

- The DTD can be stored internally within the document type declaration of the document and therefore be sent to the recipient together with the document itself. In this case the XML document can be processed anywhere without any supplementary information. However, reuse of the DTD is not supported.
- External DTDs (either public or private) are stored as independent files. They can be identified via uniform resource identifiers (URIs) which must be contained in the document type declaration of the valid XML document. External DTDs can be distinguished through so called namespaces [14]. They are reusable and thus support more efficient document management.

The extent of publishing organizational DTDs depends on many aspects, for example on technological aspects as well as on the organizational strategy. DTDs can be completely hidden to other process participants, bilaterally exchanged or published world wide. For a short introduction into XML and namespaces see for example [8]. In the following, we suppose that the set of process relevant DTDs is known to all participants of the business process.

3.1. XML nets

In the sections above, we have described how to graphically model XML DTDs with GXSL and how to specify document retrieval and document manipulation with XManiLa. We can now combine both techniques for the definition of XML nets.

XML nets, a new kind of high-level Petri nets, are a formal, graphical modeling language that allows to model both the flow of XML documents and the underlying business process. The static components of XML nets (i.e. the places of a Petri net) are labeled with GXSL-schemas representing a DTD. Places can be interpreted as a container for XML documents which are valid for the corresponding DTD. The flow of XML documents is defined by the occurrences of transitions which thereby manipulate (create, change or delete) documents of their adjacent places. The activation of a transition, which is prerequisite for the transition's occurrence, depends on the labels of the adjacent edges constructed with XManiLa and on an optional transition inscription. XML nets have the following characteristics:

- A place is represented by a GXS which identifies the type of documents contained in the place.

- An edge between a place and a transition is labeled with an extended GXS that does not conflict with the schema of the adjacent place. For each instantiation of the variables of the edge label it is possible to decide whether a document of the adjacent place matches the extended GXS or not.
- A transition may be inscribed by a logical expression over all variables that appear in the labels of the adjacent edges. The expression evaluates to either `true` or `false` for an instantiation of the variables.
- The initial marking assigns to each place of the XML net a set of valid XML documents.

The behavior of XML nets is defined by the following rule for a transition being activated for an instantiation of variables (If a variable appears more than once in the vicinity of a transition it must be instantiated with the same value for the same transition occurrence.): All places in the preset of the transition, i.e. with an outgoing edge to the transition, contain (at least) one document that matches with the schema of the adjacent edge under the given instantiation of the variables. All places in the postset of the transition, i.e. with an edge from transition to place, contain (at least) one document that matches with the label of the adjacent edge under the given instantiation of the variables in case of document manipulation or do not contain a matching document in case of document creation. The transition inscription validates to `true` for these documents and the instantiation of the variables.

An activated transition may occur. If a transition occurs, it removes the matching documents or parts of the documents as specified by the edge label from the places in the preset of the transition and inserts new documents into the places in the postset of the transition or new elements into the matching documents.

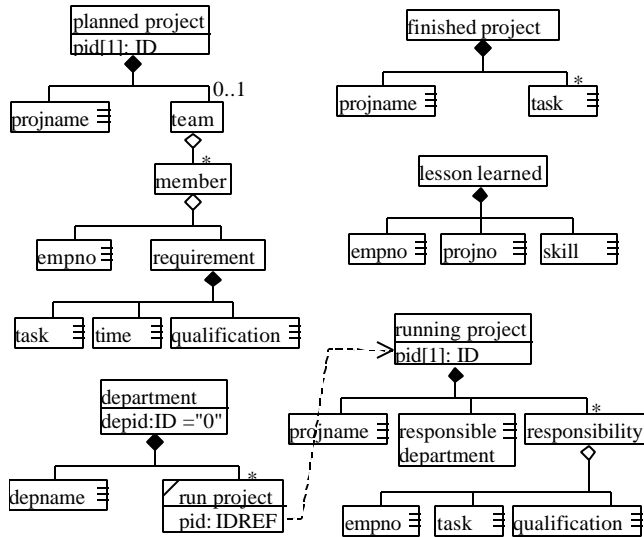


Figure 17: Graphical XML schemas for the project execution

To continue the example of Figure 3, we can now assign to each place a GXS and to each edge an extended GXS as label. Figure 17 shows the schemas for planned, running and finished projects, the department, and the lessons learned. The schema for the employee is shown in Figure 14.

Besides the definition of all process relevant document type definitions, the behavior of the XML net has to be specified in detail for all transitions.

- *initialize project*: A new document of type `planned project` is created.
- *assign employee*: The assignment consists of inserting the employee ID as team member for the project and the time required for the task execution together with the project ID for the employee.
- *start project*: When a project is started, a responsible department is chosen for the project. This is only possible for departments with an identifier that differs from the default identifier 0. The document of type `planned project` is deleted and a new document of type `running project`, including the department ID, created. The project ID is added to the department's list of running projects.
- *finish project*: Delete the respective document from the list of running projects and create a new document with the information about the finished project. Moreover, delete the project ID from the department document. Create a new document that contains all information about the skill that an employee has learned from the project.

- *update employee*: Finally, the information about project participation is deleted in the employee document. At the same time, the new skill is appended to the employee's list of skills.

The XML net corresponding to the simple Petri net of Figure 3 is shown in Figure 18. For the sake of readability, we omitted the GXS place labels. Furthermore, no initial marking is specified.

An interorganizational workflow management system should be able to deal with both simply structured data and documents. Although places of predicate/transition nets are interpreted as containers for relations, i.e. structured data of relational databases, predicate/transition nets can also be modeled as XML nets: the GXS corresponding to a relation schema or an edge inscription is a diagram with a one level hierarchy of all attributes. This allows to combine both concepts, data based workflow modeling with predicate transition nets and document based workflow modeling with XML nets.

3.2. Fragmentation of XML nets

The globalization of companies, flexibilization of organizational structures, cooperation between companies and business-to-business commerce, mobile computing, and the development towards distributed business processes require the application of geographically distributed information system technologies. The distribution of a global, interorganizational workflow implies a distribution of the workflow management system on the execution level and on the design level methods for the fragmentation of centralized workflows. The fragmentation of predicate/transition nets as a kind of Petri nets is described in [13] and can easily be transferred to XML nets.

A global XML net can be decomposed into several local net fragments that can be allocated to different execution sites. In general, we decompose Petri nets by

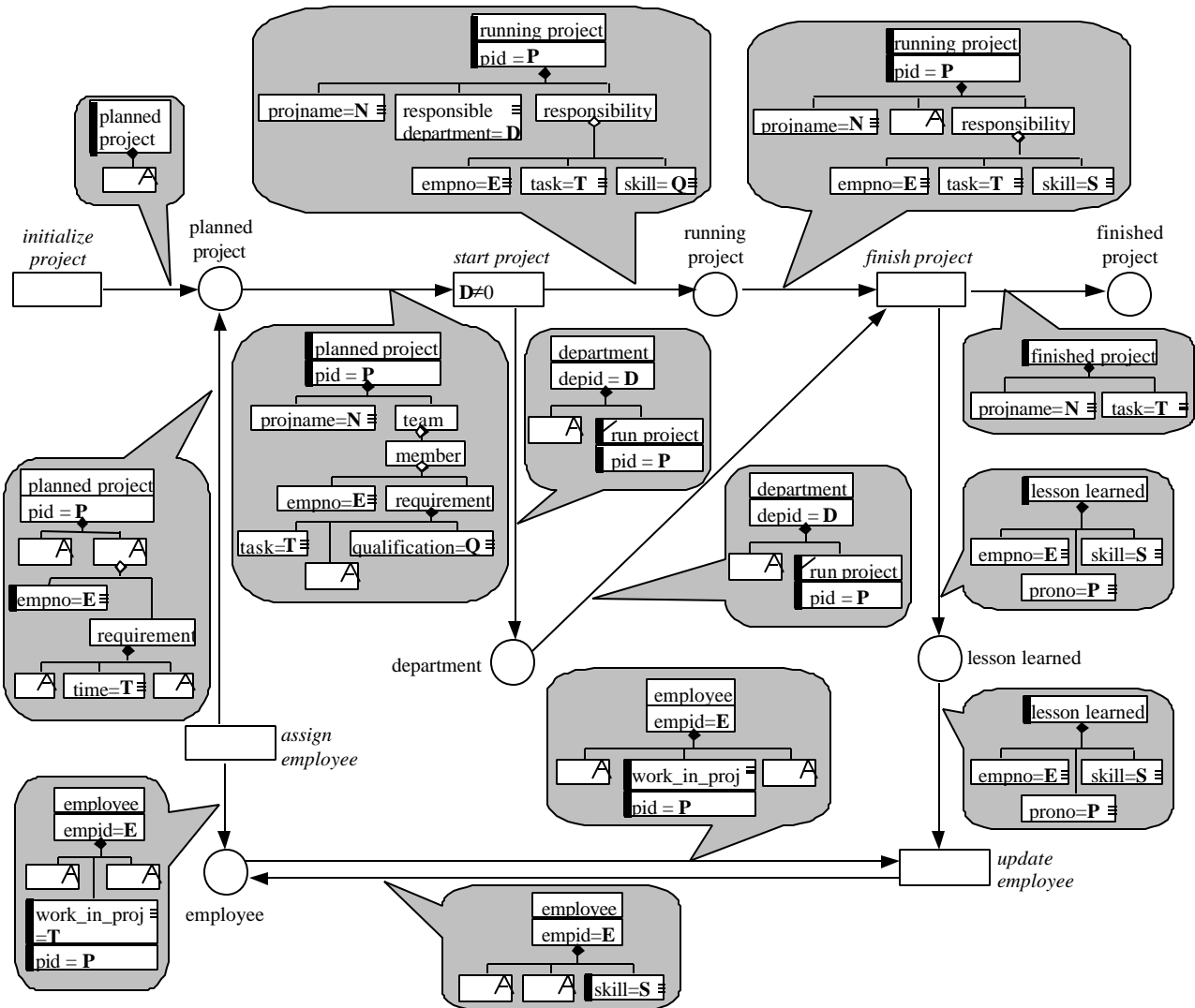


Figure 18. XML net for the process of project execution

splitting the places, i.e. by duplicating them and assigning the copies to the respective fragments. The initial marking of an interface place in the global net (for XML nets a set of valid documents) can be assigned to the corresponding places of the fragments in two ways: by replication of the documents or by allocation to one of the fragments.

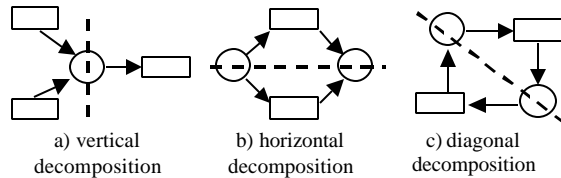


Figure 19: Vertical, horizontal, and diagonal decomposition of Petri nets

In [13] three types of decomposition are described based on the dependencies between the fragments (i.e. the data flow in the global net): vertical, e.g. for sequential processes, horizontal as for alternative processes, and diagonal for decomposition into processes with mutual dependency (see Figure 19 for examples).

Coming back to our workflow example for the project execution, we can identify three different organizational units involved in the execution of the workflow: the management, the department that is responsible for the execution of the specific project and the human resources department (see Figure 20). The arcs describe the document flow and thus the dependencies between the workflow fragments. For example, the project department needs documents from the management for the execution of their part of the workflow (the running project document).

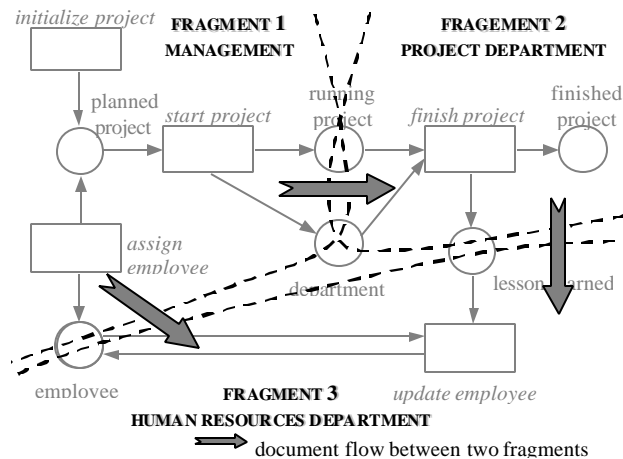


Figure 20: Workflow fragments for the distributed project execution

For the execution of a distributed workflow based on a Petri net several architectures are conceivable, with the degree of distribution ranging from centralized to

completely distributed. Organizational structures, the distribution of workflow relevant data and documents, as well as technical restrictions are crucial criteria for the allocation of the workflow fragments. Whether a workflow engine runs in a central place, is installed at all process participants' sites, or is sent to the site on demand (together with the workflow fragment) depends for example on the size of the workflow engine in relation to the size of the workflow fragments, the average number of workflow fragments to be executed at the site, the maintenance and update cost of the workflow engine, and the flexibility of the workflow fragment allocation. For example, distribution of the workflow engine is a suitable architecture for the workflow support of business-to-business applications because of the small number of participating organizations in contrast to workflows e.g. for business-to-consumer applications like internet shopping malls.

4. Outlook

In this contribution, we have introduced XML-nets, which integrate behavior- and document-related aspects of workflows. In the future, GXSL might be adopted to XML schemas as soon as ongoing work on XML schemas has lead to a W3C recommendation. Moreover, XManiLa will be extended to querying the sequential order of elements of the same element class.

For the geographical distribution of XML net-based workflows, principles of distributed database management systems may be transferred to the management of distributed XML documents, focussed for example on the management of document replication. Finally, criteria for the allocation of documents, mass data and other relevant resources must be found.

We plan to extend an existing Petri net simulation tool in order to simulate and validate distributed workflows which are modeled as XML nets.

References

- [1] van der Aalst, W.; Desel, J.; Oberweis, A.: *Business Process Management – Models, Techniques and Empirical Studies*, Lecture Notes in Computer Science, Vol. 1806, Springer-Verlag, 2000.
- [2] Bonifati, A.; Ceri, S.: *Comparative Analysis of Five XML Query Languages*, in: SIGMOD Record 29(1), March 2000, pp. 68-79.
- [3] Booch, G.; Rumbaugh, J.; Jacobson, I.: *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

- [4] Brauer, W., Reisig, W., Rozenberg, G. (eds.): *Advances in Petri Nets, Part I*, Lecture Notes in Computer Science, Vol. 254, Springer-Verlag, 1986.
- [5] Ceri, S.; Comai, S.; Damiani, E.; Fraternali, P.; Paraboschi, S.; Tanca, L.: *XML-GL: a Graphical Language for Querying and Restructuring XML Documents*, in: Proc. 8th Int. World Wide Web Conference, Toronto, Canada, 1999.
(<http://www8.org/w8-papers/1c-xml/xml-gl/xml-gl.html>)
- [6] Desel, J.; Erwin, Th.: Modeling, Simulation and Analysis of Business Processes, in [1], pp. 129-141.
- [7] Deutsch, A.; Fernandez, M.; Florescu, D.; Levy, A.; Suciu, D.: *A Query Language for XML*, in: Proc. 8th Int. World Wide Web Conference, Toronto, Canada, 1999.
(<http://www8.org/w8-papers/1c-xml/query/query.html>)
- [8] Eckstein, R.: *XML Pocket reference*, O'Reilly, 1999.
- [9] *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium Recommendation, Technical Report, 10-February-1998.
(<http://www.w3.org/TR/1998/REC-xml-19980210>)
- [10] Jablonski, S.: *Workflow Management Between Formal Theory and Pragmatic Approaches*, in: [1], pp. 345-358.
- [11] Janssens, G.K.; Verelst, J.; Weyn, B.: *Techniques for Modeling Workflows and Their Support of Reuse*, in: [1], pp. 1-15.
- [12] Genrich, H.J.: *Predicate/Transition Nets*, in [4], pp. 207-247.
- [13] Guth, V.; Lenz, K.; Oberweis, A.: *Distributed Workflow Execution Based on Fragmentation of Petri Nets*, in: Traummüller, R.; Csuháj-Varjú, E. (eds.): Proc. 15th IFIP World Computer Congress 'Telecooperation – The Global Office, Teleworking and Communication Tools', Vienna, Budapest, September 1998, pp. 114-125.
- [14] *Namespaces in XML*, World Wide Web Consortium Recommendation, Technical Report, 14-January-1999.
(<http://www.w3.org/TR/REC-xml-names/>)
- [15] Oberweis, A.: *An Integrated Approach for the Specification of Processes and Related Complex Structured Objects in Business Applications*, in: Decision Support Systems, 17, 1996, pp. 31-53.
- [16] Oberweis, A.; Sander, P.: *Information System Behavior Specification by High-Level Petri Nets*, in: ACM Transactions on Information Systems, 14(4), 1996, pp. 380-420.
- [17] Reisig, W.: *Petri Nets: an Introduction*, EATCS monographs 4, Springer-Verlag, 1985.
- [18] Robie, R.; Lapp, J.; Schach, D.: *XML Query Language (XQL)*, in: Proc. of the Query Language Workshop, Cambridge, Massachusetts, 1998.
(<http://www.w3.org/TandS/QL/QL98/pp/xql.html>)
- [19] Scheer, A.-W.; Nüttgens, M.: *ARIS Architecture and Reference Models for Business Process Management*, in [1], pp. 376-389.
- [20] Weitz, W.: *Combining Structured Documents with High-level Petri-Nets for Workflow Modeling in Internet-based Commerce*, Int. Journal of Cooperative Information Systems (IJCIS), 7(4), December 1998, pp. 275-296.
- [21] *XML Schema Part 1: Structures, Part 2: Datatypes*, World Wide Web Consortium Working Draft, Technical Report, 7-April-2000.
(<http://www.w3.org/TR/xmlschema-1/> and <http://www.w3.org/TR/xmlschema-2/>)