Transformation of ARIS Markup Language to EPML

Jan Mendling†, Markus Nüttgens‡
†Vienna University of Economics and BA
jan.mendling@wu-wien.ac.at
‡Hamburg University of Economics and Politics
nuettgens@hwp-hamburg.de

Abstract: Heterogeneous and proprietary interchange formats pose a major problem for business process management. This applies in particular to processes that have been modelled as Event-Driven Process Chains (EPCs). This paper addresses heterogeneous representations of EPCs. We take ARIS Markup Language (AML)[1], the proprietary interchange format of ARIS Toolset, as a starting point and discuss its advantages and shortcomings. Afterwards, we propose a mapping from AML to EPC Markup Language (EPML) and discuss the implementation of these mappings as a XSLT transformation program. The mappings reveal that all major AML elements can be transformed to similar EPML elements. Furthermore, the XSLT program makes EPC models managed by ARIS Toolset available in EPML format.

1 Introduction

Business process modeling (BPM) is supported by various graphical modeling tools. Currently, there are at least 14 major tools for BPM available on the market [Je04]. Most of these tools use heterogeneous proprietary formats for import and export. That is in particular true for tools that support modelling of Event-Driven Process Chains (EPC). As a consequence, moving model data from one tool to another requires non-trivial transformations and detailed knowledge about the data formats of each tool involved. This is a major road block for integrating different BPM tools. A recent survey by Delphi Group identifies the lack of a commonly accepted interchange format as one of the major hindrances for BPM [De03].

This heterogeneity of proprietary data formats has been the major motivation for the definition of EPML (EPC Markup Language) [MN02, MN03, MN04a]. EPML aims to serve as a tool-neutral, XML-based *interchange format* for EPC business process models. Accordingly, it is related to other tool-neutral interchange formats like OMG's XML Metadata Interchange (XMI) [Ob03], the Petri Net Markup Language (PNML) [BCvH$^+$03] or the XML Process Definition Language (XPDL) [Wo02] proposed by the Workflow Management Coalition. Furthermore, EPML can serve as an *intermediary format* between hetero-

[1]ARIS, ARIS Toolset, and AML are trademarks of IDS Scheer AG. The use of registered names and trademarks in this paper does not imply that such names are free for general use.

geneous tools. This is especially economic when the number of tools is large. Instead of defining bilateral transformations between every pair of tools, the usage of an intermediary format reduces the number of transformations from $O(n^2)$ to $O(n)$ [WHB02]. EPML is especially suited for this purpose, because it represents EPCs in a generic, tool-neutral manner. When the interchange format is good to read and understand (for readability see [SW01, AN02, MN04b]) software developers can faster write transformations programs from and to that format, e.g. with XSLT [Cl99]. EPML has been designed to comply with the design principle of readability [MN04a] in order to allow fast and easy development of transformation programs.

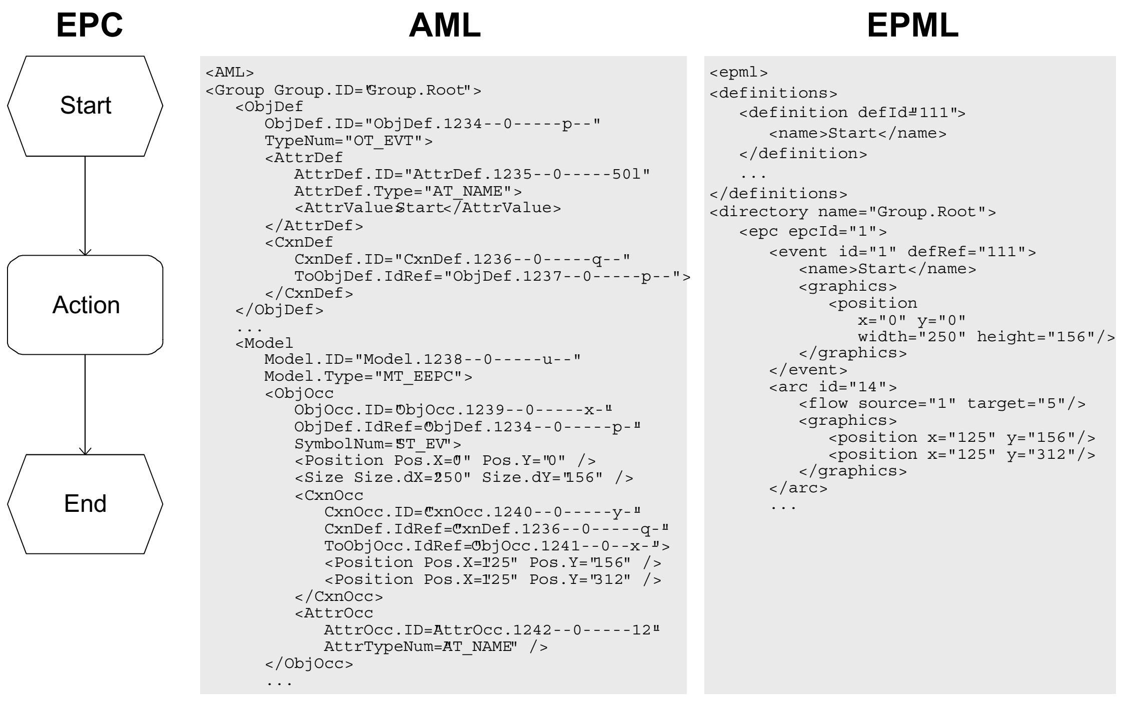
The EPC method [KNS92] is very much related to ARIS (Architecture of Integrated Information Systems) [Sc00] and IDS Scheer AG as one of its major supporters. Today, many EPC business process models are maintained by the help of ARIS Toolset of IDS Scheer AG. ARIS Toolset supports import and export of business process models in a proprietary XML-based interchange format which is called ARIS Markup Language (AML) [ID03]. Although AML also builds on XML it is much more difficult to understand and to transform than data available in EPML. This is a major hindrance for the AML-based integration of ARIS Toolset with other tools and for the automated information extraction from process models available in AML. Accordingly, a transformation from cryptic AML to EPML is desirable in order to leverage the reuse to EPC model data in different applications and to make numerous EPC models available in EPML. Moreover, the transformation allows insight into the expressive power of EPML relative to AML.

The rest of the paper is structured as follows. In Section 2 we will give an introductory example to illustrate AML and EPML. Section 3 will present and discuss AML in detail. Section 4 will introduce EPML and an extension of EPML that is capable to represent non control flow aspectsof EPC models. In Section 5 we will define transformations from AML to EPML. Furthermore, we present an XSLT program that automates these transformations. Section 6 concludes the paper and gives some outlook on future research.

2 AML and EPML by Example

Figure 1 gives the example of a very simple EPC business process model and parts of its representation both in AML and in EPML. The code wants to give a first impression of AML and EPML. Here, we only give some short explanations, details are given in the subsequent sections.

In *AML* the start event is split up in two syntax element: the object definition (`ObjDef`) represents the logical event and the object occurrence (`ObjOcc`) the appearance of the logical event in the graphical diagram. Models (`Model`) are organized in groups (`Group`) and control flow arcs are represented by logical `CxnDef` and graphical `CxnOcc` elements attached to their source object. In *EPML* the start event is captured by an `event` element, the arc is a separate `arc` element. EPC models (`epc`) are arranged in directories (`directory`). A logical representation of the start event is declared in a `definition` right after the root element. Further details on both formats will be given in the following sections.



```
<AML>
<Group Group.ID="Group.Root">
   <ObjDef
      ObjDef.ID="ObjDef.1234--0-----p--"
      TypeNum="OT_EVT">
      <AttrDef
         AttrDef.ID="AttrDef.1235--0-----50l"
         AttrDef.Type="AT_NAME">
         <AttrValue>Start</AttrValue>
      </AttrDef>
      <CxnDef
         CxnDef.ID="CxnDef.1236--0-----q--"
         ToObjDef.IdRef="ObjDef.1237--0-----p--">
      </CxnDef>
   </ObjDef>
   ...
   <Model
      Model.ID="Model.1238--0-----u--"
      Model.Type="MT_EEPC">
      <ObjOcc
         ObjOcc.ID="ObjOcc.1239--0-----x-"
         ObjDef.IdRef="ObjDef.1234--0-----p-"
         SymbolNum="ST_EV">
         <Position Pos.X="0" Pos.Y="0" />
         <Size Size.dX="250" Size.dY="156" />
         <CxnOcc
            CxnOcc.ID="CxnOcc.1240--0-----y-"
            CxnDef.IdRef="CxnDef.1236--0-----q-"
            ToObjOcc.IdRef="ObjOcc.1241--0--x-">
            <Position Pos.X="125" Pos.Y="156" />
            <Position Pos.X="125" Pos.Y="312" />
         </CxnOcc>
         <AttrOcc
            AttrOcc.ID="AttrOcc.1242--0-----12"
            AttrTypeNum="AT_NAME" />
      </ObjOcc>
      ...
```

```
<epml>
<definitions>
   <definition defId="111">
      <name>Start</name>
   </definition>
   ...
</definitions>
<directory name="Group.Root">
   <epc epcId="1">
      <event id="1" defRef="111">
         <name>Start</name>
         <graphics>
            <position
               x="0" y="0"
               width="250" height="156"/>
         </graphics>
      </event>
      <arc id="14">
         <flow source="1" target="5"/>
         <graphics>
            <position x="125" y="156"/>
            <position x="125" y="312"/>
         </graphics>
      </arc>
      ...
```

Figure 1: An example of AML and EPML representation of an EPC.

3 AML Format of ARIS Toolset

AML and a so-called ARIS-Export DTD is the proprietary XML interchange format of ARIS Toolset. This section refers to the ARIS-Export.dtd and describes a subset of its syntax elements and their semantics. For a complete introduction to AML see [ID03].

An AML file starts with an `AML` element as the root element. General information like time of creation, name of the ARIS database, language, and font style is stored in subelements of `AML`. The `Group` element, also a subelement of `AML`, is a container for all model-related information. In ARIS Toolset each `Group` element refers to a directory folder of the ARIS Explorer. A `Group` must have a unique `Group.ID` attribute and it may have multiple `AttrDef`, `ObjDef`, `Model` or further `Group` subelements as children. When the `Group` and its related directory have a name, ARIS Toolset stores it in an `AttrDef` (attribute definition) subelement whose `AttrDef.Type` attribute is set to `AT_NAME`. This is typical for AML. Every specific information of objects is stored in `AttrDef` or `AttrOcc` subelements of these objects (see Figure 2).

Another principle idea of ARIS Toolset reflected in AML is the separation between definition and occurrence: each model element is first defined in an abstract way and later referenced as an occurrence in a model. This allows one logical object to be included with e.g. two occurrence in a model. Accordingly, the `Model` element contains `ObjOcc` (object occurrence) elements that refer to `ObjDef` (object definition) elements. The `ObjDef` element provides an abstract definition of an object. It has a unique `ObjDef.ID` attribute and a `TypeNum` attribute that refers to an object type, like e.g. EPC function or EPC event. Its `LinkedModels.IdRefs` attribute provides a list of ID-references to linked models.

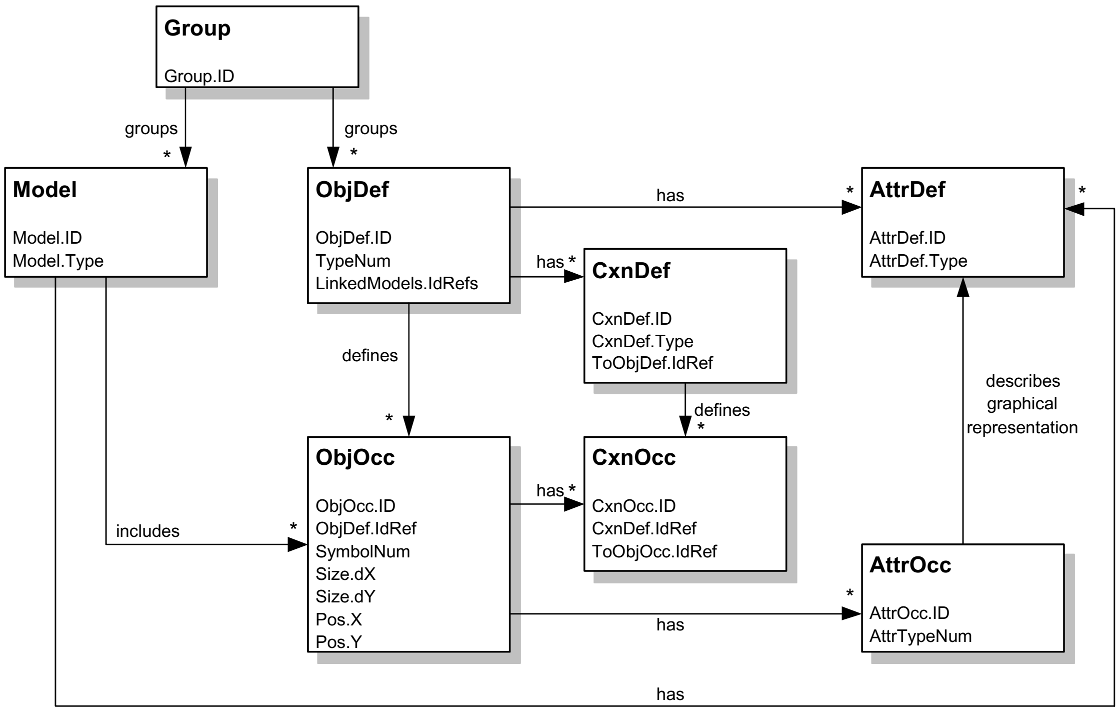


Figure 2: A UML class diagram showing a part of the AML metamodel.

These can be used e.g. for hierarchical refinement of functions. `ObjDef` elements may have multiple `AttrDef` and multiple `CxnDef` subelements. `CxnDef` elements represent arcs between objects. Each `CxnDef` has a unique `CxnDef.ID` attribute, a `CxnDef.Type` attribute, and a `ToObjDef.IdRef` attribute which represents the target of the arc. Depending on the `CxnDef.Type` attribute the arc may represent control flow, information flow, or different kinds of semantic association between the objects.

A `Model` has, among others, a unique `Model.ID` and a `Model.Type` attribute. The model type, like e.g. EPC, refers to the allowed set of objects. The `Model` element may contain `AttrDef` elements to store model specific information and `ObjOcc` elements to represent graphical elements in a visual model. An object occurrence has among others a unique `ObjOcc.ID` attribute and a reference to an object definition via the `ObjDef.IdRef` attribute. The `SymbolNum` attribute refers to a graphical icon that is used to represent the object in the visual model. An EPC function would be e.g. represented by a green rectangle with radiused edges. An `ObjOcc` element may have subelements that describe its size and its position in the visual model. Furthermore the `AttrOcc` element defines how information attached via an `AttrDef` is visually represented in a model. It has a unique `AttrOcc.ID` attribute and an `AttrTypeNum` attribute that refers to its type. This type provides a syntactical link between an `AttrOcc` and an `AttrDef` element of two associated `ObjOcc` and `ObjDef` elements. Similar to object definitions `ObjOcc` may also have multiple `CxnOcc` elements. Each of them has a unique `CxnOcc.ID` attribute and a `CxnDef.IdRef` reference to an arc definition and a reference to the target of the arc via an `ObjOcc.IdRef` attribute.

The AML metamodel is not bound to EPCs. It can represent any kind of objects whose type and icon references are understood by the application. This is a very flexible solution and allows for an easy extension of ARIS Toolset with new kinds of models and objects. Yet, there are some problems in both the design of the metamodel and its representation in AML:

Cryptic Element Names: AML uses cryptic names for some of the elements, e.g. `CxnOcc` for an arc in a model. This contradicts domain-independent XML guidelines proposed e.g. by ANSI X12 [AN02] or SWIFT [SW01] that both suggest to use telling names and no abbreviations. Such naming conventions provide for a better readability of the data and consequently for a simpler development of applications using that data.

Arc Representation: AML is to our best knowledge the only XML interchange format for BPM that uses adjacency sub-element lists representing arcs as child elements of the source node (see [MN04b]). This has some conceptual implications. Using this representation does not allow to have arcs that are not connected to a source node. Nevertheless, it could make sense to have such arcs when an incomplete model should be stored.

Separation of Definition and Occurrence: AML strictly separates abstract definition of objects and graphical representation in models. This is motivated by using one logical object multiple times in a model. As a consequence, information is split up in two XML elements while even if there is actually only one logical object. This provokes the question whether to put certain information in the definition or in the occurrence. Some design decisions of AML can be questioned in this context, e.g. it is not clear why there needs to be a `CxnDef` attached to an object definition. It would be sufficient to represent arcs in models only.

Beyond that, there is a flaw in the way how AML is used by ARIS Toolset. The types of object definitions and the icons of object occurrences are stored in the `TypeNum` and the `SymbolNum` attribute. The values of these attributes are neither enumerated in the DTD, nor do they have a telling name. An EPC function has e.g. an object type `OT_FUNC` and a symbol type `ST_FUNC`. As these predefined type values are not documented in the DTD, the developer has to find out about their meaning by analyzing AML code of process models. That fact contributes to AML's limited readability. This shortcoming does not really count if one wants to exchange models only between different ARIS Toolset implementations. But as soon as these models have to be moved to other applications or have to be used in a different context, non-trivial transformations are needed. The limited readability of AML, then, is a road block for developing transformation programs. The following section will explain the EPML representation of EPCs and an extension to capture non control flow aspects of business process models.

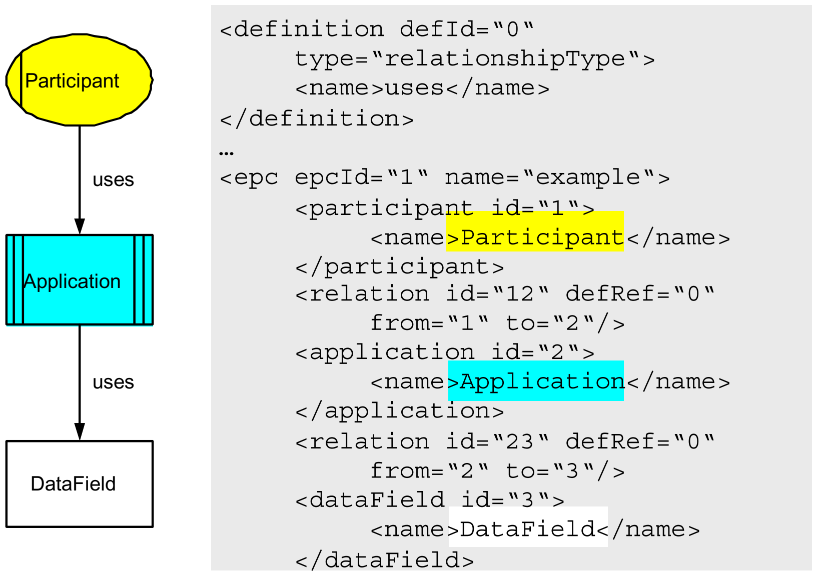
4 EPML and Non Control Flow Aspects

EPML is a XML-based tool-neutral interchange format for EPC business process models [MN02, MN03, MN04a]. The `epml` element is the root of every EPML file. It contains among others a `directory` element that can nest further directories and `epc` models. Each of these models is identified by a unique `epcId` and a `name` attribute. An `epc` element is a container for multiple control flow elements like `event`, `function`, `processInterface`, as well as `and`, `or`, and `xor` connectors, and multiple control flow `arc` elements. Each of these elements is identified by a unique `Id` attribute and a `name` element. The `function` and the `processInterface` element may include `ToProcess` elements. The latter has a `linkToEpcId` attribute representing a logical pointer to a subprocess of a function or to a subsequent process of a process interface. Each `arc` has a `flow` element whose `source` and `target` attributes represent the source and the target of the control flow arc. All EPC elements may have a `graphics` element. This element may contain `position`, `fill` (not applicable for arcs), `line`, and `font` visualization information. For control flow elements the `position` element specifies the `x` and the `y` position of the top left corner of a bounding box. Its size is indicated via the attributes `width` and `height`. Control flow arcs may have multiple `position` elements, each representing a point of a polyline. In the most simple case there are two position elements to represent the start point and the end point of the arc. For further details and further syntax elements of EPML we refer to [MN04a].

In general there are two categories of information that are frequently added to a business process model. First, *attributes* represented as *(name,value)* pairs can be used to attach statistical or configurational data to a process or to process objects. Second, various *objects* involved in the execution of a business process are frequently displayed as icons in the visual process model. Dedicated elements of EPML have been defined to represent both these kinds of information (see [MN04a]). Yet, a clear separation of textual attributes and graphical object icons like proposed by AML is also desirable for EPML in order to provide for a better tool orientation. As a consequence, we propose in the following some modifications to the way such additional information is represented in EPML.

The new Version 1.1 of the EPML Schema renames the former elements `view`, `unit`, and `unitReference` to `attributeTypes`, `attributeType` as well as `attribute` elements, respectively. Arbitrary `attributeTypes` can be declared as top-level elements of an EPML file. Single `attribute` elements can be attached to `epc` elements and to all its child elements like e.g. `function` elements. The `attributeType` may have a `description` and it must have a unique `typeName` attribute. This type name is referenced in the `typeRef` attribute of an `attribute` element. The attribute type declaration provides for a consistent naming of extension attributes used by individual tools.

Moreover, the new EPML version adds non-control flow elements which can be displayed in a graphical EPC process model. Figure 3 shows an example of these new EPC subelements. A participant uses an application and the application uses a certain data field e.g. to execute a task. The relationship is declared in a `definition` element at the top of the EPML file.



```
<definition defId="0"
     type="relationshipType">
     <name>uses</name>
</definition>
…
<epc epcId="1" name="example">
     <participant id="1">
          <name>Participant</name>
     </participant>
     <relation id="12" defRef="0"
          from="1" to="2"/>
     <application id="2">
          <name>Application</name>
     </application>
     <relation id="23" defRef="0"
          from="2" to="3"/>
     <dataField id="3">
          <name>DataField</name>
     </dataField>
```

Figure 3: Example of new EPC subelements.

In contrast to ARIS Toolset that offers about 100 different icons and object type, EPML restricts itself to three objects: `dataField`, `participant`, and `application`. These three process objects are also found in both XPDL [Wo02] and, with different names, the Architecture of Integrated Information Systems (ARIS) [Sc00]. All these three elements have a `name` and a `description` element and they are identified by a unique `id` attribute. Furthermore, they may have `graphics` and `attribute` elements. Relationships between these elements or between these elements and control flow elements are represented by `relation` elements. A `relation` is a directed edge between an element whose `id` is referenced in the `from` attribute and another element referred to in the `to` attribute. The `relation` element is related to the `arc` element. Yet, the syntactical distinction between both allows to easily identify control flow with `arc` elements. The `relation` elements may have multiple `graphics` elements. They can also contain `attribute` elements. Furthermore, the `defRef` attribute of a relation must reference a `defId` of a `definition` element. The definition is meant to describe the semantics of the relationship. Including these new elements an `epc` may now have `function`, `event`, `processInterface`, `and`, `or`, `xor`, and `arc` elements as well as `participant`, `application`, `dataField`, and `relation` elements as children.

5 Transformation from AML to EPML

In this section we will present the transformation from AML to EPML. This transformation has been implemented as an XSLT program. For further information see *http://wi.wu-wien.ac.at/˜mendling/EPML*. In the following we will illustrate the transformation subdivided into five aspects: generation of positive integer identifiers, generation of EPML header fields, mapping of navigation structure, mapping model elements, and mapping model element subelements (see Table 1). To avoid confusion we use the namespace prefixes `aml`, `epml`, and `xsl` in the text.

Table 1: Mapping of AML elements to EPML elements

AML	EPML
`aml:Model.ID`	`epml:epcId`
`aml:ObjDef.Id`	`epml:defId`
`aml:ObjOcc.Id`	`epml:id`
`aml:AttrDef.Type`	`epml:defId`
`aml:ObjDef`	`epml:definition`
`aml:AttrDef`	`epml:attributeType`
`aml:Group`	`epml:directory`
`aml:Model`	`epml:epc`
`aml:ObjOcc`	different EPML elements
`aml:CxnOcc`	`epml:arc` or `epml:relation`
`aml:Pos.X`	`epml:x`
`aml:Pos.Y`	`epml:y`
`aml:Size.dX`	`epml:width`
`aml:Size.dY`	`epml:height`

EPML requires identifying attributes to have positive integer values. As AML identifiers are of string type there has to be a mapping. Figure 4 illustrates how a list of *(identifier,position)* pairs can be generated. An XSLT node set of e.g. all `aml:ObjOcc` is processed via a `xsl:for-each` loop. Each pair of an identifier and its position in the node set is written to the list. Later in the transformation program this list can be queried via a `xsl:select` of an XSLT `xsl:value-of` statement (see Figure 4).

```
<xsl:variable name="ObjOccTable">
     <xsl:for-each select="//*[name()='ObjOcc']">
          <xsl:value-of select="@ObjOcc.ID" />
          <xsl:value-of select="concat(' ',position(),' ')" />
     </xsl:for-each>
</xsl:variable>
```

```
<xsl:value-of select="substring-before(substring-after(
     substring-after($ObjOccTable,$Id),' '),' ')" />
```

Figure 4: A list of ObjOcc.ID-Position pairs and a related query.

EPML header fields include definition and attribute type elements. First, if there are two or more `aml:ObjOcc` elements that share an `aml:OccDef` element a corresponding definition element has to be included to represent such a relationship. Subelements of EPC models have to reference the `epml:defId` attribute of a definition in their `epml:defRef` attribute. Second, definition elements have to be added for non control flow relationships. Accordingly, `epml:relation` elements reference the respective `epml:defId` in their `epml:defRef` attribute. Third, attribute type elements have to be added for all types of attributes used in the EPC models. These attributes have to reference the correct `epml:typeId` in their `epml:typeRef` attribute. For all these elements and their identifiers dedicated *(identifier,position)* lists have to be stored in variables similar as described above for `aml:ObjOcc.ID` attributes.

The navigation structure of AML and EPML is very similar. In AML the `aml:group` element corresponds to the `epml:directory` element. Moreover, the `aml:model` element is mapped to an `epml:epc` element. The `aml:Model.ID` attribute maps to an `epml:epcId` attribute following the mechanism described above concerning identifiers. Figure 5 illustrates how the name of a directory or a model is retrieved from an AML element. It is stored in the `aml:AttrValue` element of an `aml:AttrDef`. The corresponding `aml:AttrDef.Type` is AT_NAME.

```
<xsl:attribute name="name">
     <xsl:value-of select="./*[name()='AttrDef']/
          *[name()='AttrValue'][../@AttrDef.Type='AT_NAME']" />
</xsl:attribute>
```

Figure 5: Mapping names from AML to EPML.

The subelements of `epml:epc` are derived from `aml:ObjOcc` and `aml:CxnOcc` elements. The type of `aml:ObjOcc` elements can be identified via its `aml:SymbolNum` attribute and the `aml:TypeNum` attribute of its corresponding `aml:ObjDef` element. Table 2 gives an overview of the mapping rules. For the `aml:CxnOcc` element two mappings have to be distinguished. First, if the source and the target of the edge are both EPC control flow elements, then the edge maps to an `epml:arc` element. Second, if at least the source or the target of the edge is not an EPC control flow element, then the edge maps to an `epml:relation` element.

The different subelements of `epml:epc` may contains further information. We use the case of an `epml:function` to explain the mappings. Similar to identifiers and names of directories as described above the `id` attribute and the `name` element of a function can be generated from the AML elements. Moreover, the `graphics` element of a function includes position information. The corresponding `epml:x`, `epml:y`, `epml:width`, and `epml:height` attributes can be generated using `aml:Pos.X`, `aml:Pos.Y`, `aml:Size.dX`, and `aml:Size.dY` attributes, respectively. Linked EPC business process models captured in the `epml:linkToEpcId` attribute can be extracted from `aml:LinkedModels.IdRefs` attributes. Finally, `aml:AttrDef` and `aml:AttrOcc` elements map to `epml:attribute` elements.

The proposed mapping from AML to EPML permits the following conclusions. First, EPML is capable to capture the essential AML concepts without loss of information. This has been demonstrated by the implementation of the described XSLT transformation program. Second, EPML uses a more intuitive representation of EPCs than AML. This rather subjective statement is supported by two facts: EPML uses telling element names inspired

Table 2: XSLT-Mapping of AML's `ObjOcc` elements to EPML elements

AML's `ObjOcc`	EPML
`aml:TypeNum='OT_FUNC' and aml:SymbolNum!='ST_PRCS_IF'`	`function`
`aml:TypeNum='OT_FUNC' and aml:SymbolNum='ST_PRCS_IF'`	`processInterface`
`aml:TypeNum='OT_EVT'`	`event`
`aml:TypeNum='ST_OPR_XOR_1'`	`xor`
`aml:TypeNum='ST_OPR_AND_1'`	`and`
`aml:TypeNum='ST_OPR_OR_1'`	`or`
`contains(aml:TypeNum,'_ORG') or contains(aml:TypeNum,'_PERS') contains(aml:TypeNum,'_EMPL')`	`participant`
`contains(aml:TypeNum,'_APP') or contains(aml:TypeNum,'_CMP') or contains(aml:TypeNum,'_MOD') or contains(aml:TypeNum,'_PACK')`	`application`
`contains(aml:TypeNum,'_CLS') or contains(aml:TypeNum,'_INFO') or contains(aml:TypeNum,'_KPI') or contains(aml:TypeNum,'_LST') or contains(aml:TypeNum,'_OBJ') or contains(aml:TypeNum,'_TERM')`	`dataField`

by the generic names of EPC syntax elements. Furthermore, EPC models transformed to EPML are less than half as large as the original AML files. This intuitive and generic representation of EPCs in EPML contributes to its readability. As a consequence, building EPML-based applications is easier than relying on AML. We estimate that thoroughly understanding and deciphering AML's names and abbreviations takes at least half a week, considering that AML object types and symbol types are not documented in the AML manual. Third, the transformation program makes available EPC models managed by ARIS Toolset for EPML-based applications, like EPC Tools [CK04] or EPML2SVG [MBN04].

6 Conclusion and Future Work

In this paper we presented an approach to transform EPC business process models available as files conforming to the ARIS Markup Language (AML) to EPC Markup Language (EPML). This transformation has been implemented as an XSLT program (see *http://wi.wu-wien.ac.at/~mendling/EPML*). Such a transformation is on the one hand important from a pragmatic point of view because many EPC business process models are managed by the help of ARIS Toolset and a transformation program makes these models available for EPML-based applications, like EPC Tools [CK04] or EPML2SVG [MBN04]. On the other hand the transformation offers insight into the different representational alternatives for EPCs in XML. EPML uses a more generic and more readable representation for EPCs than AML. This is helpful when building EPML-based applications. Beyond its readability, EPML is still capable to capture all essential model elements that are included in AML. Future research will be dedicated to the analysis of XML-based representation of EPCs in further business process management tools. This will include the development of related transformation programs to leverage EPML as a tool-neutral interchange format for EPCs.

Acknowledgement. The authors would like to thank the anonymous reviewers for there comments on an earlier version of this paper, which helped to improve the presentation of the ideas.

Disclaimer. We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any information about ARIS Toolset and AML contained in this paper. However, we made all possible efforts to ensure that the results presented are, to the best of our knowledge, up-to-date and correct.

References

[AN02] ANSI X12: ASC X12 Reference Model for XML Design. Technical Report Type II - ASC X12C/TG3/2002-xxx. ANSI ASC X12C Communications and Controls Subcommittee. July 2002.

[BCvH 03] Billington, J., Christensen, S., van Hee, K. E., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: W. M. P. van der Aalst and E. Best (eds.), *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands*. volume 2679 of *Lecture Notes in Computer Science*. pages 483–505. 2003.

[CK04] Cuntz, N. and Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: M. Nüttgens and F. J. Rump (eds.), *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg*. 2004.

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[De03] Delphi Group: *BPM 2003 – Market Milestone Report, White Paper*. 2003.

[ID03] IDS Scheer AG: *XML-Export und -Import (ARIS 6 Collaborative Suite Version 6.2 Schnittstellenbeschreibung)*. ftp://ftp.ids-scheer.de/pub/ARIS/HELPDESK/EXPORT/. Juni 2003.

[Je04] Jenz und Partner (ed.): *Business Process Modeling Tools, accessed on 3th July 2004*. http://www.jenzundpartner.de/Resources/Product_Watchlist/product_watchlist.htm. 2004.

[KNS92] Keller, G., Nüttgens, M., and Scheer, A. W.: Semantische Prozessmodellierung auf der Grundlage “Ereignisgesteuerter Prozessketten (EPK)”. Technical Report 89. Institut für Wirtschaftsinformatik Saarbrücken. Saarbrücken, Germany. 1992.

[MBN04] Mendling, J., Brabenetz, A., and Neumann, G.: Generating SVG Graphics from EPML Processes. In: M. Nüttgens and F. J. Rump (eds.), *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg.* 2004.

[MN02] Mendling, J. and Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: M. Nüttgens and F. J. Rump (eds.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany.* pages 87–93. 2002.

[MN03] Mendling, J. and Nüttgens, M.: XML-basierte Geschäftsprozessmodellierung. In: W. Uhr and W. Esswein and E. Schoop (eds.), *Proc. of Wirtschaftsinformatik 2003 / Band II, Dresden, Germany.* pages 161 –180. 2003.

[MN04a] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: J. Mendling and M. Nüttgens (eds.), *Proc. of the 1st GI-Workshop XML4BPM - XML Interchange Formats for Business Process Management, Marburg, Germany, March, 2004.* pages 61–79. 2004.

[MN04b] Mendling, J. and Nüttgens, M.: XML-based Reference Modelling: Foundations of an EPC Markup Language. In: J. Becker and P. Delfmann (eds.), *Referenzmodellierung - Proceedings of the 8th GI-Workshop on Reference Modelling, MKWI Essen, Germany.* pages 51–71. 2004.

[Ob03] Object Management Group: XML Metadata Interchange (XMI). Specification, Version 2.0. Object Management Group. May 2003.

[Sc00] Scheer, A. W.: *ARIS business process modelling.* Springer Verlag. 2000.

[SW01] SWIFT: SWIFT Standards XML Design Rules Version 2.3. Technical Specification (http://xml.coverpages.org/EBTWG-SWIFTStandards-XML200110.pdf). Society for Worldwide Interbank Financial Telecommunication. 2001.

[WHB02] Wüstner, E., Hotzel, T., and Buxmann, P.: Converting Business Documents: A Classification of Problems and Solutions using XML/XSLT. In: *Proceedings of the 4th International Workshop on Advanced Issues of E-Commerce and Web-based Systems (WECWIS).* pages 61–68. 2002.

[Wo02] Workflow Management Coalition: Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0. Workflow Management Coalition. 2002.