Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2011
August 28-31, 2011, Washington, DC, USA

DETC2011-48600

DEXML: A FIRST STEP TOWARD A UML BASED IMPLEMENTATION FRAMEWORK FOR PLCS

Sylvere Krima
National Institute of Standards and Technology
Gaithersburg, Maryland 20899, USA
sylvere.krima@nist.gov
University of Burgundy, LE2i
Dijon, Bourgogne, France

Roch Bertucat
Engisis s.r.l
Rome, Italy
roch.bertucat@engisis.com

Joshua Lubell
National Institute of Standards and Technology
Gaithersburg, Maryland 20899, USA
lubell@nist.gov

Sudarsan Rachuri
National Institute of Standards and Technology
Gaithersburg, Maryland, 20899, USA
rachuri.sudarsan@nist.gov

Sebti Foufou
University of Burgundy, LE2i
Dijon, Bourgogne, 21000, France
sfoufou@u-bourgogne.fr
CSE Dept, Qatar University
Doha, Qatar
sfoufou@qu.edu.qa

ABSTRACT

Data exchange specifications not only must be broad and general to achieve acceptance, but also must be customizable in a controlled and interoperable manner to be useful. The Product Life Cycle Support (PLCS) suite of data exchange specifications (known as DEXs) uses templates to enable controlled customizability without sacrificing breadth or interoperability. DEXs are business context-specific subsets of ISO 10303 Application Protocol (AP) 239, subject to additional constraints imposed by the templates. A PLCS template defines how AP239 entities and their attributes will be instantiated using an externally-defined controlled vocabulary defined in a Reference Data Library. Template instantiations are defined using an Instantiation Path (IP) specified using an arcane syntax that must be manually written by the template developer. The PLCS information model is formally defined in the ISO 10303 EXPRESS language, but there is no formalism used at the template level. A challenge for newcomers to PLCS is to dive into and understand all the bespoken, non-standardized and PLCS-specific technologies (domain-specific languages and software) used to develop and implement the templates. DEXML presents an approach based on the Unified Modeling Language (UML) enabling the use of mainstream software and technologies to develop and implement DEXs, reducing the need for nonstandard and unfamiliar languages and tools.

INTRODUCTION

Product Lifecycle Management (PLM) is the integration of “people, data, processes, and business systems” to “provide a product information backbone for companies and their extended enterprise.” [1] PLM is complex due to the large number of actors, lifecycle stages, and domains involved throughout a product’s existence. Consequently, PLM systems must manage an enormous quantity and variety of documents and data. Examples include assemblies of collections of parts, product configurations, maintenance tasks, and documentation associated with a product such as analysis results or requirements. Central to management of this information is the product model itself, defined as “the representation of a product in terms of parameters that reflect its descriptive and performance characteristics." [2] Because of its importance to PLM, product modeling has been an intense area of research, resulting in frameworks such as the Core Product Model (CPM) [3] and the "Methodology and tools Oriented to Knowledge based engineering Applications" (MOKA). [4]

CPM is an extensible conceptual representation of a product and is not tied to any particular engineering domain or implementation technology. CPM is intended to serve as a basis for both extension and specialization meeting domain-specific requirements. Extensions and specializations differ in how they refine the conceptual model. Extensions are achieved by adding new concepts to the initial conceptual model, thus increasing the conceptual model's scope. An example of an extension to CPM is the Open Assembly Model, which adds concepts for representing assembly structure and kinematics information. [5] Specializations, on the other hand, add domain-specific semantics to the concepts initially present in the model, resulting in a model that is no longer purely conceptual but rather is tied to a particular application area or business context. An example of CPM specialization is the NIST Design Repository, a software architecture and set of interfaces for editing and browsing product models stored in design repositories. [6]

MOKA, a predecessor and influence on the CPM research, employs an approach combining product modeling with formal logic and ontology. Bock et al [7] have recently proposed a more rigorous ontological approach to product modeling, enabling development of languages using terminology comfortable for engineers, yet enjoying the benefits of ontology and open world semantics.

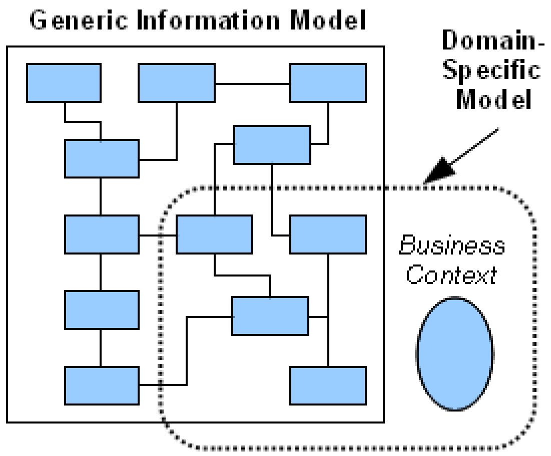


Figure 1. PLM modeling approach.

A common theme of these product modeling research results is that they present a multilayered modeling architecture for PLM combining a common metamodel with support for domain-specific customizations. This idea, shown in Figure 1, accommodates multiple business contexts while allowing for a common, agreed-upon semantics to be shared. As the figure illustrates, constructing a model meeting the requirements of a specific business domain requires two operations. First, the appropriate subset of the common metamodel must be identified. Second, business context information must be added to the identified subset.

These results are relevant to developers of PLM exchange standards because, like product and process modeling, exchange standards development for PLM benefits from a multilayered approach. A data exchange schema for PLM must describe in a computer-interpretable fashion the information to be transferred between systems. Because of the many information types, potential relationships between digital objects, and the need to cover the whole product lifecycle, a PLM exchange schema must be generic, broad, and comprehensive. And Figure 1 applies to PLM standards as well in that the creation of an exchange schema for a particular domain requires determining the correct subset of the comprehensive exchange schema, and then adding the necessary business context.

But a PLM standards architecture needs more than just a common base-level exchange schema. It also requires a mechanism for applying potentially any business context to that schema in such a way that business context-specific schemas are interoperable not only with respect to the base-level exchange schema, but also with one another. This "inter-context" interoperability requires that the application of business context be controlled and also traceable. In other words, business context must be applied in an unambiguous, uniform, and rigorously documented manner.

The rest of this paper focuses on a particular PLM standards architecture, Product Life Cycle Support (PLCS), which aims to enable controlled customizability without sacrificing breadth or interoperability. Although PLCS has a powerful, flexible mechanism for balancing interoperability with extensibility, the nonstandard and unfamiliar languages and tools currently used to develop PLCS exchange specifications are a barrier to widespread adoption. We first describe PLCS and its present day usage. Next we focus on templates, critical elements of PLCS in that they manage the application of a business context to the underlying PLCS schema. Templates in effect encode the relationship between the generic information model (i.e., PLCS schema) and business context shown in Figure 1. We then present DEXML, our approach based on the Unified Modeling Language (UML) [8] which enables the use of mainstream software and technologies to use PLCS to develop and implement business-specific exchange specifications, thus removing some of the PLCS obstacles. Next we discuss our implementation of DEXML, and end with some concluding remarks.

PLCS DATA EXCHANGE SPECIFICATIONS

ISO 10303-239 – Product Life Cycle Support – (PLCS) [9] is a STEP (**St**andard for the **E**xchange of **P**roduct model data) [10] Application Protocol (AP239) that supports the representation of the information involved in the whole lifecycle of a product. AP239's wide scope and agnosticism with respect to business context makes it broadly applicable but hard to understand, implement and use as a whole. To overcome this, the creators of PLCS provide a customizable architecture to allow users to work with a subset of the original information model. This approach is somewhat analogous to the conformance classes used in other STEP APs to represent a subset of the AP's information model in order to support some specific use cases. But rather than specify an inflexible and static set of conformance classes, the PLCS architecture enables the definition of business context-specific subsets of AP239 called DEXs (**D**ata **Ex**change specifications). DEXs use templates [11] to define how PLCS entities and their attributes will be instantiated, enabling customizability without sacrificing breadth or interoperability. Instantiation uses an externally-defined controlled vocabulary defined in a Reference Data Library (RDL). Template instantiations are defined using an Instantiation Path (IP). The IP uses a procedural language that describes, in a computer interpretable fashion, the information instantiations performed by a template.

DEX developers currently have to manually write the IP using an arcane syntax and without the help of software tools. This lack of tools results in errors (syntax error, data type inconsistency) and inconsistencies (between the templates). In this paper we present a new approach that defines the templates and the IP using UML.

The PLCS information model is formally defined in a data modeling language called EXPRESS (ISO 10303-11) [12] but there is no formally standardized language used at the template level. PLCS template information models are defined in a non-standard variant of EXPRESS-G, a graphical presentation of the EXPRESS language defined in ISO 10303-11. But the EXPRESS-G variant does not provide information about the IP. The IP is defined separately using the aforementioned textual format lacking robust, mainstream tool support. Using a formalized mechanism based on UML to represent the IP is a way to make DEXs easier to develop and implement.

The Object Management Group (OMG) has standardized two graphical languages for representing ordered sets of processes (instantiations in our context): the Business Process Modeling Notation (BPMN) [13] and the UML activity diagram. BPMN is designed only to formally represent the workflows involved during an activity and does not provide any mechanism to represent an information model. This lack of information model representation is the reason why BPMN does not appear as the best candidate for our purpose. On the other hand, UML activity diagrams describe the business workflows of a component in a system, which makes them a good candidate for representing the instantiation of a template. Moreover, the UML activity diagram can be used with a UML class diagram to add type information to the data exchanged between different processes.

DEXML, our implementation, employs UML activity diagrams to enable the use of mainstream software and technologies to develop and implement DEXs, reducing the need for nonstandard and unfamiliar languages and tools. In the next section, we describe PLCS DEX templates and their representation in UML in greater detail. We then discuss the DEXML implementation, an important aspect of which is a mapping of the AP239 information model from EXPRESS to UML. The mapping is needed to achieve consistency at the language level between the AP239 information model and template process models represented in UML.

WHAT IS A TEMPLATE?

As mentioned earlier, a template specifies an IP. This IP describes the template's instantiation, invocation, and usage. But this IP is only one component of a template. A template additionally contains:

- A textual documentation that describes the role of the template
- A textual description of the input parameters and output
- A graphical information model expressed in an EXPRESS-G based graphical language
- Some uniqueness constraints
- One or more instance diagram(s)

Now let us consider the template *Assigning_reference_data*, [14] whose information model is shown in Figure 2. This template describes classification of something, where the class's definition is specified in an external RDL. Because classification is fundamental to the usage of AP239, DEXs use this template more than any other. Readers familiar with EXPRESS-G will notice that this diagram includes the following non-EXPRESS-G annotations:

- The textual annotation beginning with the '^' character describes output parameters. For example, the *External_class* entity (in the bottom left of the picture) contains the **^*ext_class*** annotation, meaning that the template will create an instance of the *External_class* entity, instance named ***ext_class***.
- The blue arrows are used to bind input parameters to attributes of entities. The blue arrow in the bottom right of the picture means that the user needs to provide an input parameter called ***assigning_reference_data.ecl_id*** which will be used to set up the value of the id attribute of the instance called ***ext_class_lib***.

More information on this extended EXPRESS-G notation can be found in the PLCS Technical Description online document [11]. The only software supporting the extensions is a freeware third-party plug-in [15][1] for an obsolete and no-longer-maintained version of a commercial diagramming software package.

The IP describes how to use and instantiate this information model. The IP uses a procedural language similar to that of ISO10303 SC4 reference paths [16] and specifies:

- Input parameters of a template which correspond to the user input

[1] Mention of commercial or third party products or services in this paper does not imply approval or endorsement by NIST, nor does it imply that such products or services are necessarily the best available for the purpose.

- Reference parameters of a template which correspond to the instances created by the template
- Assignment of a value to an attribute of an entity
- Invocation of other templates
- Instantiation of entities
- The ordering of assignments, template invocations, and entity instantiations

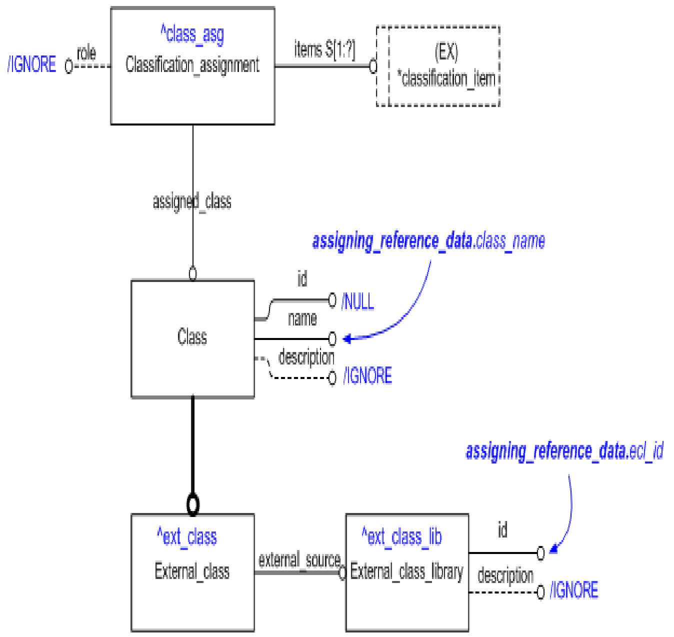


Figure 2. Information model for Assigning_reference_data.

DEXML

To overcome the drawbacks resulting from the use of non-standard technologies, DEXML presents a first step toward a UML-based implementation framework for PLCS. DEXML uses UML because it meets the requirements we have identified to replace the existing technologies and tools in use. Moreover, UML enjoys strong software support.

The main goal of DEXML is to provide an implementation framework that is easier to understand and use for both newcomers to PLCS and current PLCS users. To reach this goal, DEXML meets the following benefits, through the use of UML:

- Decreases complexity. Development of a template is done within a single UML authoring tool.
- Increases software choices. Any UML2-compliant tool can be used.
- Avoids redundancy. A single process defines the whole template, as opposed to multiple overlapping processes.
- Uses modern, widely supported, and standardized technologies.
- Represents the IP graphically using UML, enabling the IP to be linked to a UML class diagram.
- Facilitates code generation. Many UML tools can create programming language code from UML models.
- Enables data type checking at the definition level. Using a class diagram to represent the data in an activity diagram means that type checking is performed by the UML tool during the creation of the activity diagram.

The current version of UML provides all the functionality needed to satisfy the previously listed requirements. UML 2 specifies 14 diagram types classified as structure, behavior and interaction diagrams.

DEXML uses the UML diagram types as follows:

- The UML class diagram is used to represent the AP239 information model.
- The UML activity diagram, which can represent any sequence of processes as well as input and output parameters used or generated by these processes, is used to define the IP of a template.
- The UML profile diagram, which can redefine the semantics of any UML element, is used to extend the semantics of the activity diagram to represent the IP.

The UML profile is an important component of DEXML since it redefines/extends the semantics of some UML elements. To develop a DEXML UML profile we first map the IP language syntax elements to UML constructs having similar semantics, and then create new UML elements through the profile for the IP elements that do not have a match in UML. As it turns out, all IP elements naturally correspond to UML elements except for entity instantiation, which does not have an exact UML equivalent. The mapping is described in Table 1.

Table 1. Mapping from IP to UML

IP	UML
Template	Activity
Input parameter	Activity parameter node: UML allows classifying a node as an input of an activity.
Reference parameter	Activity parameter node: UML allows classifying a node as an output of an activity.
Attribute assignment	Object flow, which defines an assignment from a node to another.
Next action	Action flow, which defines the order of the activities/actions.
Template call	Call behavior action, which allows the reuse of an activity within another.
Entity instantiation	Create Object Action, which has a similar semantics but it does not fully match our need, as it does not accept inputs. We address this issue in the “Implementation” section of our paper.

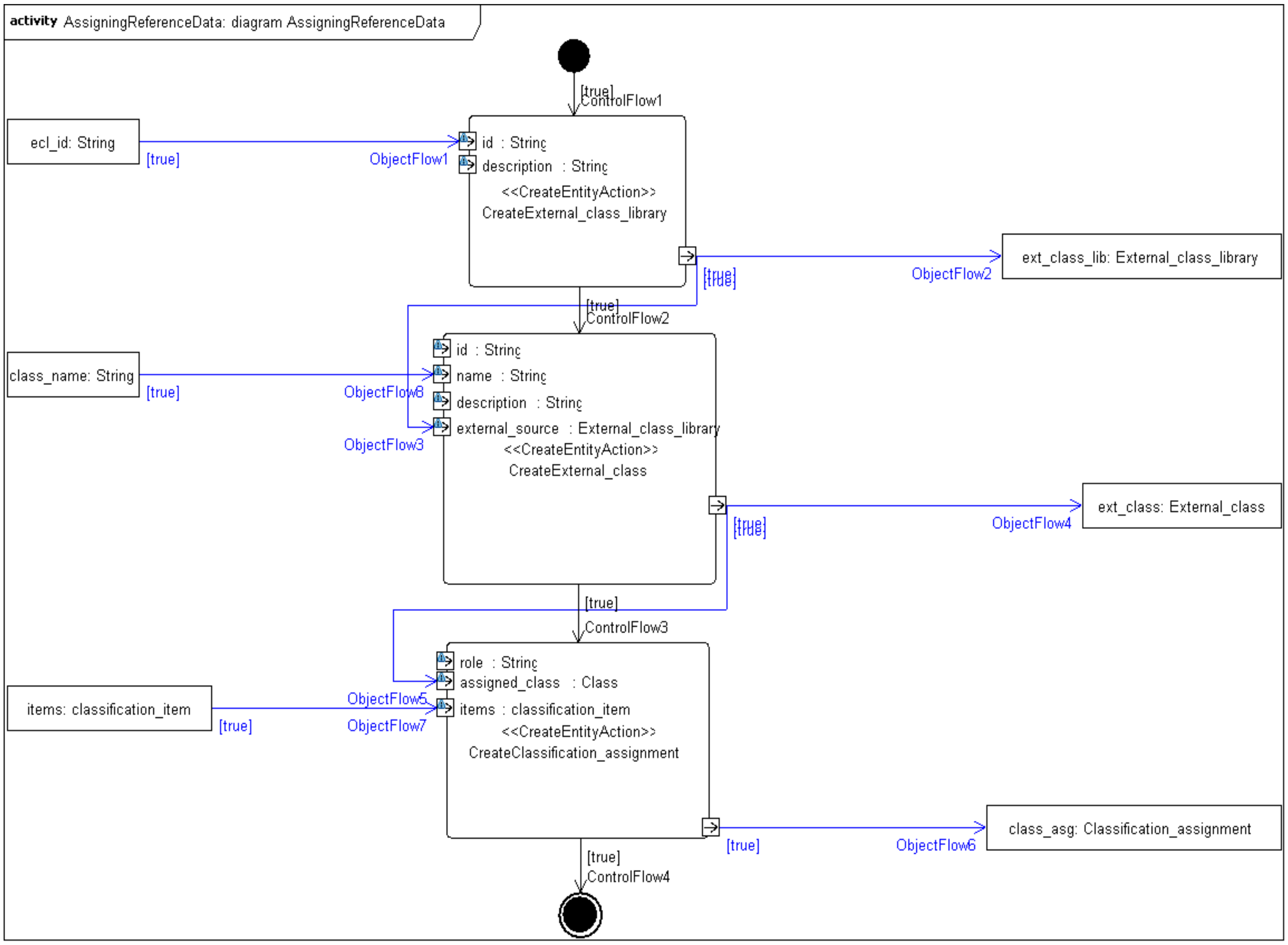


Figure 3. DEXML representation of the Assigning_reference_data IP.

Figure 3 shows the resulting activity diagram-based DEXML representation of the *Assigning_reference_data* template [14], after mapping of its IP. In this figure, input parameters are represented by the boxes on the left side. The boxes on the right side represent the reference parameters. All these parameters are classified using the AP239 class diagram obtained by transforming the AP239 EXPRESS schema to UML (as discussed later). Blue arrows represent the object flow, and black arrows represent the control flow (which defines the order of processing). Object flows are used both for binding the input parameters to the attributes of the instances and also the reference parameters to the instances created. The first instance created by CreateExternal_class_library is an instance of *External_class_library* and is bound to the **ext_class_lib** reference parameter through ObjectFlow2. The **id** attribute of this instance is bound to the **ecl_id** input parameter through ObjectFlow1. Once these operations are performed, ControlFlow2 indicates the next operation to perform: CreateExternal_class.

IMPLEMENTATION OF A DEXML EDITOR

A standard is of benefit only if people and/or applications use it. The more widely a standard is adopted, the more it acquires value, and the more it enables data exchange and collaboration. To promote the usage of the standards, it is critical that their implementations are based on mainstream technologies and common languages. The implementation of DEXML follows this principle and therefore consists of a UML representation of the PLCS data model, a UML profile, and a plug-in for Topcased [17], an open source UML tool.

UML representation of the PLCS data model

Reeper [18] is a set of Ruby [19] tools for manipulating ISO EXPRESS data models. One of these tools allows mapping from an EXPRESS file to a UML2 XML Metadata Interchange (XMI) [20] file derived using the ISO 10303-25 standard [21]. In our DEXML editor, the UML2 representation of the PLCS data model is generated with Reeper.

During our development with Reeper, we discovered that XMI/UML tool implementations differ from vendor to vendor, so none of the implementations really interoperate yet. As a result, we had to tweak the Reeper-generated mapping from EXPRESS to UML2 to meet the requirements of our UML software. We shared our modifications with the Reeper developers for possible inclusion in a future Reeper release.

UML profile

The UML profile for DEXML extends the semantics of the UML element "Create Object Action." UML defines "Create Object Action" as "an action that creates an object that conforms to a statically specified classifier and puts it on an output pin at runtime." [8] Create Object Action does not permit input parameters (input pins). It simply exposes the created instance through an output parameter (output pin).

Our DEXML implementation creates a subclass of Create Object Action called Create Entity Action. Create Entity Action enables DEXML to specify input parameters when the entity is created. These input parameters allow for assignment of values to the attributes needed for the instantiation of the AP239 entity. We formally define Create Entity Action as an action that creates an instance that conforms to a statically-specified classifier (AP239 entity) with initial parameters passed through the input pins and made available through an output pin at runtime. Figure 4 shows Create Entity Action being used to create an instance of the AP239 entity *Identification_Assignment*.

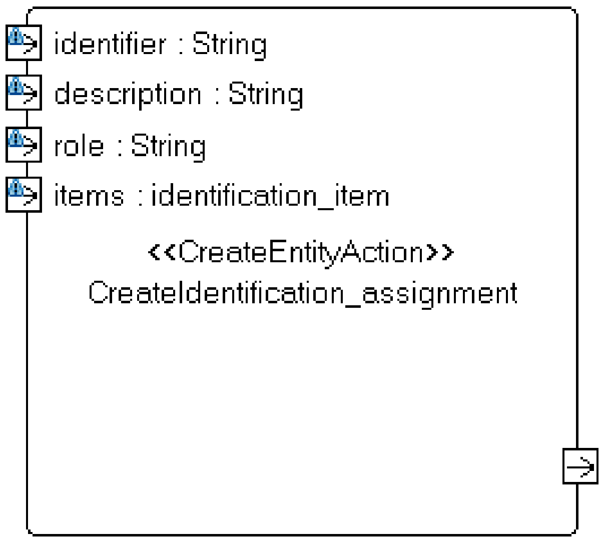


Figure 4. Example of Create Entity Action.

Topcased plug-in

Topcased is an open source software environment providing methods and tools for critical embedded systems development. It is based on the Eclipse [22] Integrated Development Environment (IDE) open source project. Topcased provides a graphical environment for UML2 diagrams development, implemented on top of the Eclipse UML2 component. An Eclipse plug-in extends the initial set of functionalities of the Eclipse IDE

We developed a Topcased plug-in for DEXML to generate and apply our UML profile to the AP239 UML model. Figure 5 shows the menu added by the DEXML plug-in to Topcased.

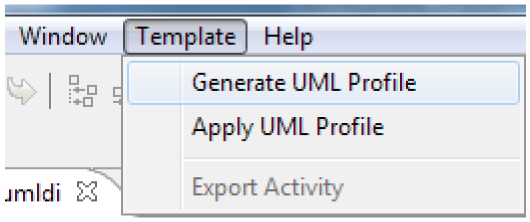
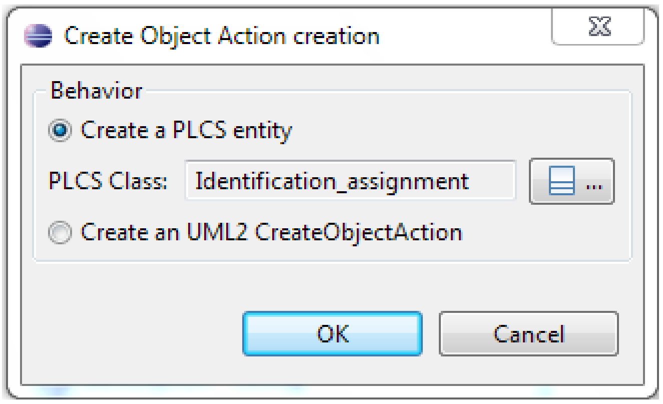


Figure 5. New menu added to Topcased.

The plug-in assists the user in the creation of an AP239 entity. Figure 6 shows the dialog that appears in Topcased when the user creates a Create Object Action. It allows the user to select the AP239 entity. Once the user confirms the selection, the plug-in automatically creates the corresponding Create Entity Action together with the input pins representing the attributes of the selected AP239 entity (as shown in Figure 4).



[✓ Create a PLCS entity]
PLCS Class: Identification_assignment
Create an UML2 CreateObjectAction

Figure 6. Dialog box for Create Object Action.

CONCLUSION AND FUTURE WORK

The PLCS/DEX architecture is analogous to the notion of specialization in the product modeling research discussed in the Introduction. PLCS goes a step further, however, in that through the use of templates, DEXs – specializations of the AP239 schema – are assured to be interoperable with one another. The templates also aggregate lower level concepts into higher level concepts, easing DEX development. [23] DEXML's goal is to advance AP239 adoption by providing a robust, standards-based framework for developing templates and implementing PLCS (through DEXs). In this paper we have shown a first step toward such a framework by providing a graphical, UML-based representation of the IP. UML is simpler and more reliable for PLCS developers and implementers to use than the bespoken technologies currently available. We have demonstrated how DEXML addresses issues with the current DEX development technologies. DEXML can be used with any UML tool that supports UML profiles.

Our next step is to enable a migration path from the current PLCS/DEX architecture to DEXML by developing tools for converting existing templates into activity diagrams. We are also implementing a reverse mapping from the IP activity diagram to the present IP syntax in accordance with Table 1.

ACKNOWLEGMENTS

The authors are grateful to Paul Witherell for his helpful and insightful comments on an earlier draft of this paper. Any remaining mistakes are the authors' sole responsibility.

REFERENCES

[1] PLM Technology Guide. http://plmtechnologyguide.com.

[2] K. Papamichael, H. Chauvet, J. LaPorta and R. Dandridge. Product modeling for computer-aided decision-making. *Automation in Construction 8 (1999) 339-350*. The Netherlands. Elsevier Science B.V. 1999.

[3] Steven J. Fenves, Sebti Foufou, Conrad Bock, Ram D. Sriram. CPM2: A Core Model for Product Data. *Journal of Computing and Information Science in Engineering*. March 2008. Vol. 8.

[4] M. Stokes (Ed.), Managing Engineering Knowledge: MOKA Methodology for Knowledge Based Engineering Applications. Professional Engineering Publishing. 2001.

[5] Sudarsan Rachuri, Young-Hyun Han, Sebti Foufou, Shaw C. Feng, Utpal Roy, Fujun Wang, Ram D. Sriram, and Kevin W. Lyons. A Model for Capturing Product Assembly Information. *J. Comput. Inf. Sci. Eng*. 6, 11 (2006), DOI:10.1115/1.2164451

[6] Simon Szykman and Ram D. Sriram. Design and implementation of the Web-enabled NIST design repository. *ACM Transactions on Internet Technology*. Vol. 6. No. 1. February 2006. Pages 85-116. DOI=10.1145/1125274.1125278

[7] Bock, C.; Zha, X.; Suh, H.; Lee, J. Ontological Product Modeling for Collaborative Design. *Advanced Engineering Informatics*. 24 (2010) 510 - 524. http://www.nist.gov/manuscript-publication-search.cfm?pub_id=822748.

[8] OMG. UML 2.0 Superstructure Specification. http://www.omg.org/cgi-bin/doc?ptc/03-08-02 . 2003.

[9] ISO 10303-239:2005. Industrial automation systems and integration – Product data representation and exchange – Part 239: Application protocol: Product life cycle support.

[10] Sharon Kemmerer. "STEP: The Grand Experience" (Editor of). NIST Special Publication 939. National Institute of Standards and Technology. Gaithersburg, MD, 1999.

[11] PLCS technical description, Templates. http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_template.htm . 2010.

[12] ISO 10303-11:1994. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual.

[13] Object Management Group (OMG). Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/ . 2011.

[14] PLCS template: assigning_reference_data. http://www.plcs-resources.org/plcs/dexlib/data/templates/assigning_reference_data/sys/section.htm#description . 2009.

[15] Eurostep. DEXTemplate. http://www.eurostep.com/global/solutions/download-software.aspx#DEXTemplate . 2006.

[16] ISO TC 184/SC4/N1977:2005(E). Guidelines for the development of mapping specifications. ISO TC 184/SC4 Standing Document. 2005-09-09.

[17] Topcased. The Open-Source Toolkit for Critical Systems. http://www.topcased.org/ . 2011.

[18] Reeper: EXPRESS to UML2 Mapper. http://www.nist.gov/el/msid/reeper.cfm . 2010.

[19] Ruby programming language. http://www.ruby-lang.org/en/

[20] Object Management Group. MOF2.0/XMI Mapping, V2.1.1. 12-01-2007

[21] ISO 10303-25:2005. Industrial automation systems and integration – Product data representation and exchange – Part 25: Implementation methods: EXPRESS to XMI binding.

[22] Eclipse - The Eclipse Foundation open source community website. http://www.eclipse.org/

[23] Keith A. Hunten and Allison Barnard Feeney. Business Objects for Industrial Data Standards. To appear in *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. August 29-31, 2011.