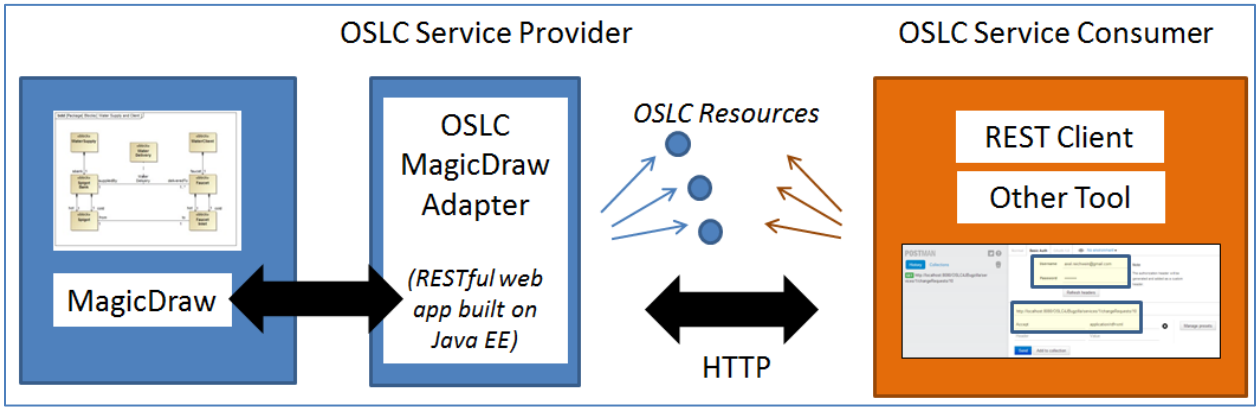
Extending the OSLC MagicDraw Adapter to expose additional MagicDraw SysML Concepts as OSLC Resources

By Axel Reichwein, March 24, 2016



Steps to expose an additional MagicDraw SysML concept through the OSLC MagicDraw adapter

1. **Define** new metaclass in SysML Ecore metamodel describing the MagicDraw SysML concept to be exposed through OSLC
2. **Perform** automatic code generation of Java class describing the corresponding OSLC resource
3. **Implement** the mapping between the MagicDraw SysML concept and the corresponding OSLC resource by using the MagicDraw API
4. **Implement** RESTful web services to expose the MagicDraw SysML concept in RDF/XML and in HTML by using Java EE and JSP

1. New Metaclass in SysML Ecore Metamodel

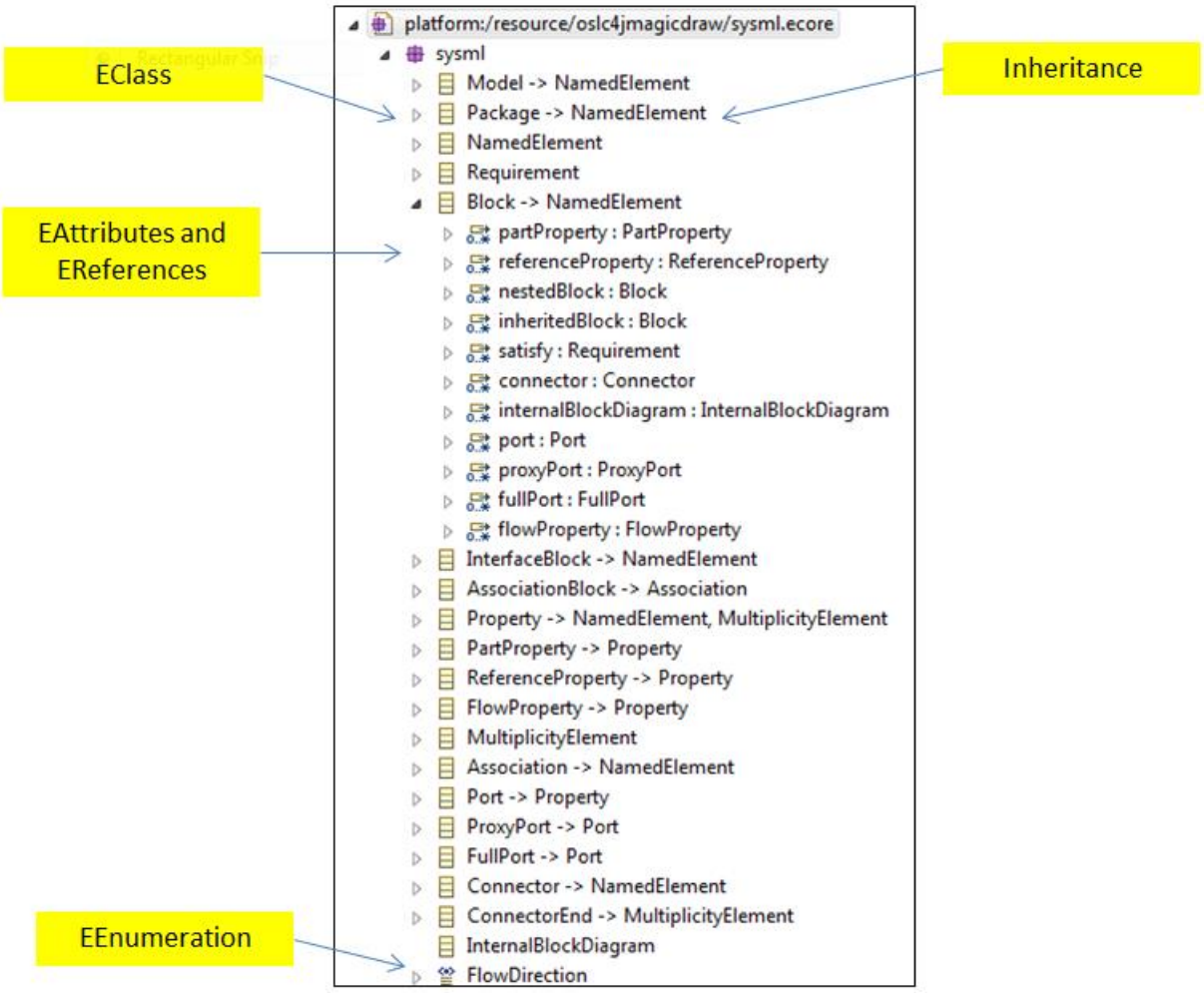
OSLC4J (http://wiki.eclipse.org/Lyo/LyoOSLC4J) is a Java toolkit for building Open Services for Lifecycle Collaboration providers and consumers. It includes:

- annotations to decorate Java objects with OSLC attributes
- annotations to assist with resource preview UIs (compact rendering)
- built-in support for service provider and resource shape documents
- libraries to simplify service provider and consumer development

- sample applications, including an OSLC Registry (catalog) application and a Change Management provider implementation
- Tests for the sample applications to complement the Lyo OSLC Test Suite

OSLC4J allows to handle OSLC resources as simple Plain Old Java Objects (POJOs). A new SysML concept can for example be handled as an OSLC resource if the SysML concept is mapped to a Java class with OSLC annotations. Adding the OSLC annotations to the class definition and to the method signatures is time-consuming and error-prone. Mistakes in defining the OSLC annotations can be difficult to identify. A model-driven approach was therefore performed in which SysML concepts were first defined in an easily customizable metamodel in Ecore, based on which the Java classes with OSLC annotations were then generated.

The first step in adding a new SysML concept to the OSLC MagicDraw adapter is to define a new metaclass with appropriate attributes and references in the SysML metamodel in Ecore. The figure below shows for example the attributes and references of the Block metaclass which inherits the attributes and references from the NamedElement metaclass.

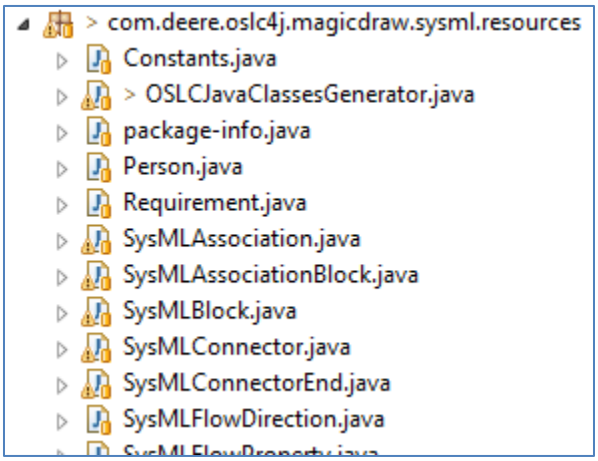


2. Automatic Java Code Generation

The next step is to generate the Java class with annotations which corresponds to the new SysML concept. Run the **OSLCJavaClassesGenerator** class in the *edu.gatech.mbsec.adapter.magicdraw.resources* package to generate Java classes with OSLC annotations based on a SysML metamodel defined in Ecore. Excerpt of the OSLCJavaClassesGenerator class:

```
public class OSLCJavaClassesGenerator {

    static EPackage sysmlPackage;
    static EClass eclass;

    public static void main(String[] args) {

        // load sysml.ecore model
        Resource ecoreResource = loadEcoreModel(URI.createFileURI(new File(
                "sysml.ecore").getAbsolutePath()));

        sysmlPackage = (EPackage) EcoreUtil.getObjectByType(
                ecoreResource.getContents(),
                EcorePackage.eINSTANCE.getEPackage());
        System.out.println(sysmlPackage.getName());

        for (EClassifier eclassifier : sysmlPackage.getEClassifiers()) {
            if (eclassifier instanceof EClass) {
                eclass = (EClass) eclassifier;
                if (!eclass.isAbstract()) {
                    // create annotated Java class
```

The resulting Java classes with annotations will be generated in the **edu.gatech.mbsec.adapter.magicdraw.resources** package.



The **Constants** class in the same package will need to be extended to define Strings associated to the new SysML concept such as:

```
public static String TYPE_SYSML_BLOCK  = SYSML_NAMESPACE + "Block";
```

3. Mapping between MagicDraw SysML Concept and OSLC Resource

As a next step, the MagicDraw API is used to collect all SysML elements of this new type which are contained in MagicDraw projects. This task is performed in the **MagicDrawManager** class of the *edu.gatech.mbsec.adapter.magicdraw.*application package.

If the new SysML concept has no obvious owner, such as SysML packages or blocks which can have either blocks or packages as owners, a method is implemented to first collect all SysML concepts of that new type such as **getAllSysMLBlocks()**. Once the **MagicDrawManager** class has a list of all SysML elements of that new type, the SysML elements need to be mapped into OSLC resources such as through the **mapSysMLBlocks()** method. It is important to keep in mind that a map is then established to link the URI of a SysML element to its corresponding OSLC resource. Example of such a map:

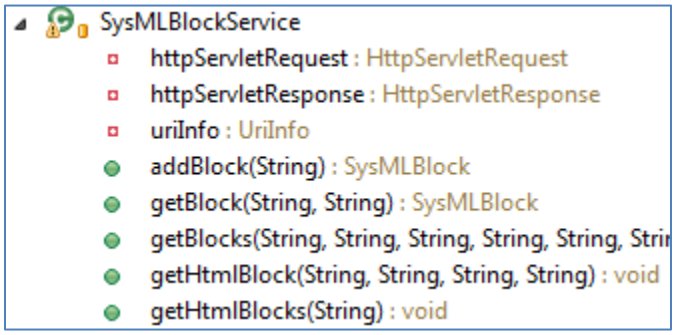
```
static Map<String, SysMLBlock> qNameOslcSysmlBlockMap = new HashMap<String,
SysMLBlock>();
```

If the new SysML concept has an obvious owner, such as SysML part properties which are always owned by SysML blocks, then there is no need to implement a method to collect all SysML elements of that type. Once the owner elements are being mapped to OSLC resources through the **map…()** method, the child elements including the elements of the new SysML type are also being mapped into OSLC resources. The **mapSysMLBlocks()** method for example calls the **mapSysMLPartProperties()** method. Below is a code excerpt of the **mapSysMLBlocks()** method:

```
// SysML Block Parts
mapSysMLPartProperties(mdSysMLBlock, sysMLBlock);

// SysML Block References
mapSysMLReferenceProperties(mdSysMLBlock, sysMLBlock);

// SysML Block Value Properties
mapSysMLValueProperties(mdSysMLBlock, sysMLBlock);

// SysML Block Flow Properties
mapSysMLFlowProperties(mdSysMLBlock, sysMLBlock);

// SysML Block Connectors
mapSysMLConnectors(mdSysMLBlock, sysMLBlock);

// SysML Block Ports
mapSysMLPorts(mdSysMLBlock, sysMLBlock);
```

4. RESTful Web Services to expose MagicDraw SysML Concepts

The last step is to implement the RESTful web services to expose MagicDraw SysML elements as OSLC resources in RDF/XML or HTML. For each SysML element type, there will be a servlet which will have 4 query methods and one creation method as shown in the figure below.

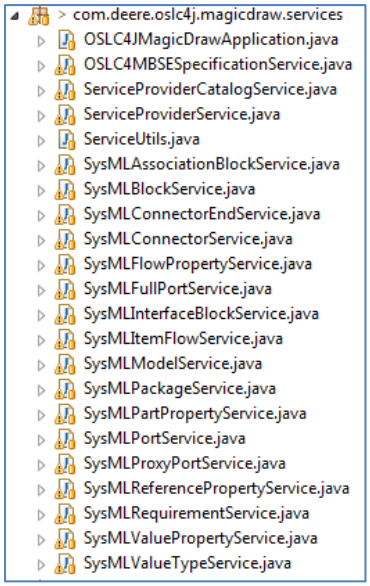


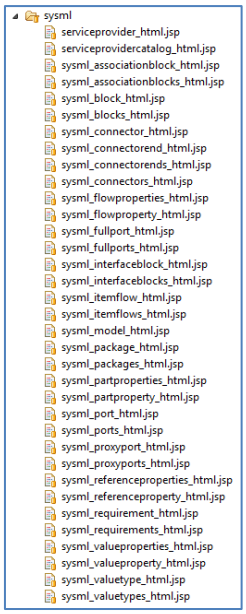
The query methods expose:

- The list of SysML elements of that type in HTML
- The list of SysML elements of that type in RDF/XML
- A specific SysML element in HTML
- A specific SysML element in RDF/XML

The specific element will be identified either by its qualified name if it has a name or by its MagicDraw ID if it has no name.

The *edu.gatech.mbsec.adapter.magicdraw.services* package contains the currently available servlets as shown below left.





The methods which return OSLC resources in HTML will dispatch the request to JSP templates. Two JSP templates need to be implemented:

- JSP to return a single SysML element in HTML
- JSP to return a list of SysML elements of the same type in HTML

The folder under **src/main/webapp/sysml** as shown above right contains all currently implemented JSP templates.

Registering the Servlet as a RESTful web service

Once the servlet is implemented, it has to be registered in the **OSLC4JMagicDrawApplication** class within the *edu.gatech.mbsec.adapter.magicdraw.services* package as indicated below:

```
RESOURCE_CLASSES.add(SysMLBlockService.class);
```

Registering the Services as belonging to a Service Provider

Typically, the OSLC Service Provider refers to all available query and creation factory services. The new query and creation factory services displaying the new SysML concept as OSLC resource need to be registered with the OSLC Service Provider.

This is done in the **MagicDrawServiceProviderFactory** class in the *edu.gatech.mbsec.adapter.magicdraw.serviceproviders* package as shown below through the *RESOURCE_CLASSES* reference.

```
public class MagicDrawServiceProviderFactory {
    private static Class<?>[] RESOURCE_CLASSES = {
            SysMLAssociationBlockService.class, SysMLBlockService.class,
            SysMLModelService.class, SysMLPackageService.class,
            SysMLPartPropertyService.class,
            SysMLReferencePropertyService.class, SysMLRequirementService.class,
            SysMLItemFlowService.class, SysMLValueTypeService.class,
            SysMLConnectorService.class, SysMLConnectorEndService.class,
            SysMLFlowPropertyService.class, SysMLValuePropertyService.class, SysMLPortService.class,
            SysMLFullPortService.class, SysMLProxyPortService.class, SysMLInterfaceBlockService.class,
            SysMLBlockDiagramService.class, SysMLInternalBlockDiagramService.class};
```

Registering the QueryCapability and CreationFactory services

Register if necessary the QueryCapability and CreationFactory services with the **@OslcQueryCapability** and **@OslcCreationFactory** annotations as shown below in the servlet (for example the SysMLBlockService class) implementing the services.

```
@OslcQueryCapability(title = "SysML Block Query Capability", label = "SysML Block Catalog Query",
        + "/" + Constants.PATH_SYSML_BLOCK, resourceTypes = { Constants.TYPE_SYSML_BLOCK }, usage
@GET
@Produces({ OslcMediaType.APPLICATION_RDF_XML,
        OslcMediaType.APPLICATION_XML, OslcMediaType.APPLICATION_JSON })
public List<com.deere.oslc4j.magicdraw.sysml.resources.SysMLBlock> getBlocks(
```

```
@OslcCreationFactory(title = "SysML Block Creation Factory", label = "SysML Block Creation",
        + "/" + Constants.PATH_SYSML_BLOCK }, resourceTypes = { Constants.TYPE_SYSML_BLOCK },
@POST
@Produces({ OslcMediaType.APPLICATION_RDF_XML,
        OslcMediaType.APPLICATION_XML, OslcMediaType.APPLICATION_JSON })
public SysMLBlock addBlock(@PathParam("projectId") final String projectId)
```

Setting the *About* attribute of the new Service

The *About* attribute of a Service is the URL of a service through which it can be retrieved. The *About* attribute of a Service is specified ("hard coded") in the **ServiceProviderService** class in the *edu.gatech.mbsec.adapter.magicdraw.services* package as shown in the code snippet below:

```
public void getHtmlServiceProvider(@PathParam("serviceProviderId") final String servicePr
{
    ServiceProvider serviceProvider = ServiceProviderCatalogSingleton.getServiceProvider(

    Service [] services = serviceProvider.getServices();

    for (Service service : services) {
        if(service.getDomain().toString().contains("model")){
            service.setAbout(URI.create("http://localhost:8080/oslc4jmagicdraw/services/"
        }
        else if(service.getDomain().toString().contains("block") & !service.getDomain().t
            service.setAbout(URI.create("http://localhost:8080/oslc4jmagicdraw/services/"
        }
```

Specifying the *About* attribute of a Service should also take place in the getServiceProvider() method of the **ServiceProviderService** class if web clients want to parse the Service Provider resource in RDF/XML for all available services.

Registering new namespaces/prefixes for representing OSLC resources in RDF/XML

Additional namespaces/prefixes can be defined with OSLC annotations in the **package.info** class within the *edu.gatech.mbsec.adapter.magicdraw.resources* package as shown below.

```
@OslcNamespaceDefinition(prefix = Constants.SYSML_ASSOCIATION_PREFIX, namespaceURI = Constants.SYSML_ASSOCIATION_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_ASSOCIATIONBLOCK_PREFIX, namespaceURI = Constants.SYSML_ASSOCIATIONBLOCK_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_BLOCK_PREFIX, namespaceURI = Constants.SYSML_BLOCK_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_MODEL_PREFIX, namespaceURI = Constants.SYSML_MODEL_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_PACKAGE_PREFIX, namespaceURI = Constants.SYSML_PACKAGE_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_PARTPROPERTY_PREFIX, namespaceURI = Constants.SYSML_PARTPROPERTY_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_REFERENCEPROPERTY_PREFIX, namespaceURI = Constants.SYSML_REFERENCEPROPERTY_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_REQUIREMENT_PREFIX, namespaceURI = Constants.SYSML_REQUIREMENT_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_NAMEDELEMENT_PREFIX, namespaceURI = Constants.SYSML_NAMEDELEMENT_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_MULTIPLICITYELEMENT_PREFIX, namespaceURI = Constants.SYSML_MULTIPLICITYELEMENT_NAMESPACE),
@OslcNamespaceDefinition(prefix = Constants.SYSML_PREFIX, namespaceURI = Constants.SYSML_NAMESPACE)
```