Modeling of Tool Integration Resources with OSLC Support

Weiqing Zhang and Birger Møller-Pedersen
Department of Informatics, University of Oslo, Oslo, Norway

Keywords: Tool Chain Management, Tool Integration, Model, Web Services, OSLC.

Abstract: This paper discusses a class modeling approach for managing tool integration. Model concepts like Artifact and Role are introduced as integration backbones. Artifacts represent real artifacts like model elements that are maintained by tools. Different kinds of tools require different kinds of Artifact classes. The Role classes capture integration scenario-specific properties for Artifacts. As the same Artifact may be involved in different scenarios, and as integration scenarios may come and go, Roles can be dynamically attached to Artifacts. It is also demonstrated the possibility to model with Artifacts and Roles alone, without any real model elements. OSLC Web services (and as part of that, OSLC resources) are generated from these class models, and it is demonstrated that class modeling of Artifacts are superior to plain OSLC specification of resources.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Development processes for software systems involves a large number of different tools that are specialized for certain tasks within their engineering domains, such as requirement analysis, architecture design, model simulation, code generation, software configuration. These tools are usually designed with focus on their specific domains, and they are not designed with integration in mind.

An effective solution is to integrate tools based upon tool adaptors with services that work on commonly defined representatives of the real artifacts (model and model elements), and a modeling approach to this would be to define representative class models for the real artifacts. Experiences from applying this approach to industrial cases have shown that a straight forward representative approach has to be extended in several respects.

With respect to semantics of model elements, the same model element in different engineering domains may be interpreted differently (e.g., a UML Class could represent a hardware component in hardware platform, or represent an application in software design). Different model elements in different languages and tools may also have the same semantic interpretation, e.g. elements of a UML model and the corresponding elements of the corresponding C code.

Model elements in different models in different languages may be derived from a common definition (in another language). For instance, common definitions of data types (e.g. temperature and wind speed classes in a wind turbine project) in a UML class model are used to derive the corresponding definitions in e.g. Simulink and IEC61131 (Rzonca et al., 2007), and placed into Simulink and IEC61131 models, respectively.

The same model element (through its representative) may be involved in multiple integration scenarios, and handled differently in each of these. Different properties of the element are required in addition to the plain representative properties, in order to support the different integration scenarios. E.g. using a UML class as a transformation source based upon a common semantics scenario requires scenario-specific properties that are different from those used in a transformation as part of a common definition scenario.

All of this has led to an approach based upon Artifacts and Roles attached to Artifacts (Zhang et al., 2012a) (Zhang and Moller-Pedersen, 2013b). Based upon the above discussions, in this paper we contribute to further properties of Artifacts and Roles.

In addition we provide some more details on the modeling of Artifacts and Roles. The implementation of the approach is based upon OSLC (Open Services for Lifecycle Collaboration, 2013) Web Services, as OSLC is becoming more and more accepted by tool vendors as a vehicle for tool integration. However, the modeling approach itself is independent of OSLC.

The rest of the paper is organized as follows. Section 2 gives the background, including previous work.

Zhang W. and Møller-pedersen B..
Modeling of Tool Integration Resources with OSLC Support.
DOI: 10.5220/0004713000990110
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 99-110
ISBN: 978-989-758-007-9
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

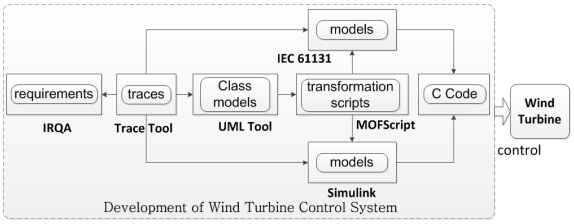

Figure 1: Development Tools in Industrial Cases.

Section 3 describes the model-based approach, with details of Artifacts and Roles. Section 4 presents the extended notions of Artifact and Roles and how to model with these. Section 5 describes how to provide OSLC-based services from the model. Section 6 compares with related work. Section 7 concludes the paper.

2 BACKGROUND

2.1 Industrial Case

We use an wind turbine project (from ABB Norway) throughout this paper. This project is to develop an embedded system to control wind turbines. An overview of involved development tools is illustrated in figure 1.

Sensors are deployed around the wind turbines to collect the environmental data. The control system contains two modules executed on the same microprocessor: a C code module generated from a Simulink model for high speed performance, and a C code module generated from an IEC 61131 model for low speed performance. The control system performs calculations according to the sensor data and sends commands to control the wind turbine. A number of development tools from different engineering domains are involved: A tool for making requirements (IRQA), then tools for designing the IEC 61131 and Simulink models, and a traceability tool for creating traces between these tool elements. A UML tool is used to specify class models that are common to Simulink and IEC 61131 design, and then transform these class models into Simulink and IEC 61131 tools through MOFScript (Object Management Group, 2010).

2.2 Review of Previous Work

In previous work, we defined representatives for real models and model element artifacts, named Artifacts, and these formed the basis for the generation of OSLC resources with adequate properties, and we made tool

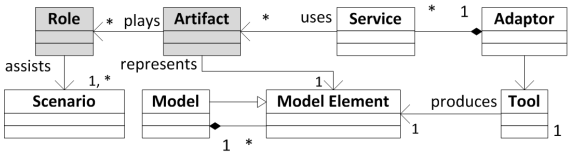

Figure 2: Concepts of the Integration Approach.

integration based upon tool Adaptors with services that work on these representatives. Instead of relying on mappings between proprietary representations of model elements, Artifacts define model element representatives that are common to all tool adaptors and to other kinds of application that use these tool adaptors. As representatives, Artifacts have properties that reflect the actual model elements for the purpose of tool integration. General kinds of tool integration, e.g. tracing between any kinds of model element, are readily supporting by Artifacts, by defining a trace that links between Artifacts. Tool integration that involves transformations between models may need Artifacts with properties like source and target language.

The above industrial case revealed that tool integration is more than adapting tools to a common way of representing model elements in different languages or formats. As more advanced integration scenarios were identified as part of applying it to the industrial case, we found that integration requires more than just Artifacts as representatives of model elements. The industrial case includes integration issues where the handling of model elements depends not only on language/tool-specific properties, but also on properties that are specific for the integration scenarios. Therefore we introduced the Role concept for capturing these integration scenario specific properties.

Figure 2 introduces the main concepts of the approach. Tool integration is performed through tool adaptors. An *Adaptor* is a software unit or plug-in that exposes a subset of tool functionalities to other tools in terms of *Services*, which work on Artifacts. The legal services that can be applied to the Artifact instances are constrained by the operations defined by Artifact classes. *Model Elements* are the real model elements produced and manipulated by tools. Data models, e.g. analysis data about models, are considered as special kinds of models in our approach (Zhang et al., 2012a). *Scenario* provides an overview of how tools collaborate according to certain process and integration requirements. In our approach the exchange of messages in scenarios are specified in terms of Artifacts and Roles.

The overview of this approach is illustrated in figure 3. When using this approach, tool integration modelers create integration models, such as Artifact, Roles, and Choreography models. Because various

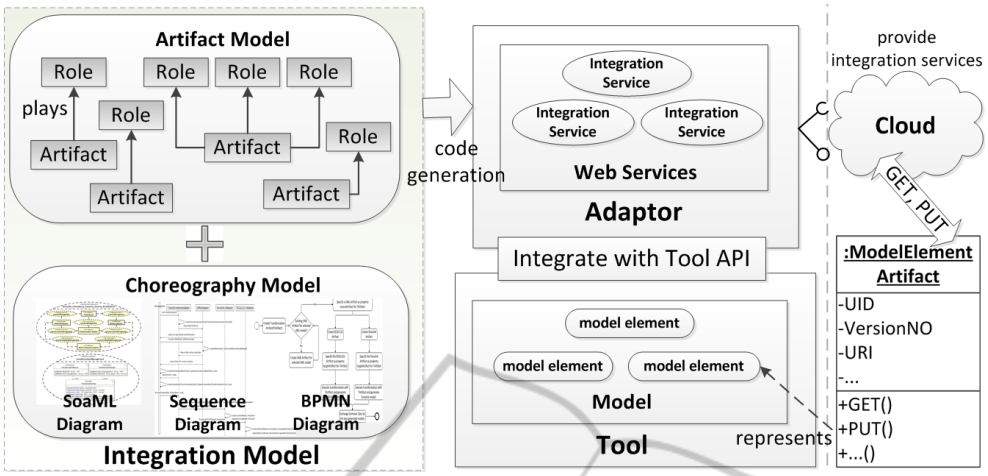

Figure 3: Overview of the Integration Approach.

OSLC integration adaptor servers and clients are operated in a similar way, the server and client code can be specified in a standard way. Therefore, model transformation scripts are prepared for the purpose of generating adaptor server code. The integration models are independent of implementation technology, while Web services are based upon specific technologies like J2EE or .Net. Therefore different transformation scripts are applied to the same integration models to obtain Web Services that runs on different platforms. The adaptors manipulate the tool model elements through tool-specific APIs, and represent them as predefined Artifact objects through integration services.

3 MODEL-BASED INTEGRATION

3.1 Details of the Artifact Concept

One solution to the integration of tools supporting languages with different metamodels is to have a common, exhaustive metamodel that merges all the concepts defined in all tool metamodels. However, experience tells us that the resulting metamodel would be large, complicated and hard to maintain when tools come and go. A common *partial* metamodel may be used for extracting certain aspects of the various models, but it will not solve full tool integration.

Instead of being metamodel-based, our approach is therefore model-based in the sense that it relies on representatives for the real models and model elements as the means for tool integration (Zhang et al., 2012b). These representatives are called Artifacts. A model-based approach to tool integration implies that Artifact is defined as a class of objects, so that each Artifact object represents a real model or model element.

As shown in figure 4, the kinds of tool integrations above are based upon a fixed set of Artifact definitions, with a fixed set of properties and operations that are common for model or model element in specific set of tools. The common properties of general Artifact are software lifecycle management properties that support integration, such as unique identifier (UID), name of Artifact, URI of the represented model elements, description of Artifact, and etc.

The specific Artifacts only require properties that have to do with the represented model elements, e.g. UML metamodel and metaclass of the represented class model element that are required for e.g. a UML model transformation. Artifact like Transformation Artifact contains properties like source and target, which point to the Model Element Artifact instances that represent the source model and the target metamodel, respectively. Model Element Artifacts that represent models or model elements in a given language will, in addition to the common properties, have a property that identifies the metamodel for the language, and a property that identifies the metaclass of the model element, such as the UML Artifact, Simulink Artifact and ReqIF (Object Management Group, 2011) Artifact in figure 4. Thus we can access the metamodels that are required for transformations.

DomainClassArtifact is used to represent model elements when they are involved in different specific semantic domains. In the Domain Class Artifact, the ”domainModel” property identifies the domain (semantics) model while the ”domainClass” property identifies the domain class that the Role links to. This covers the cases where a model from one tool (in one language) must be represented by a model in an-

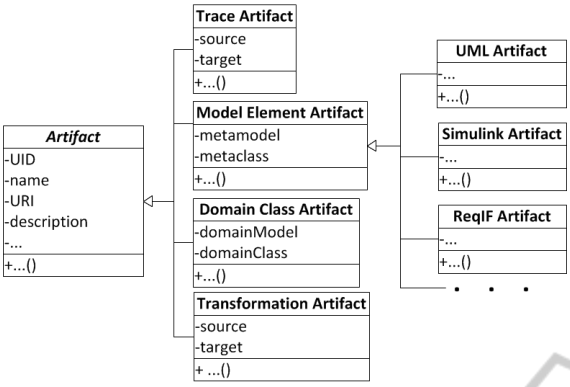

Figure 4: Artifact Overview.

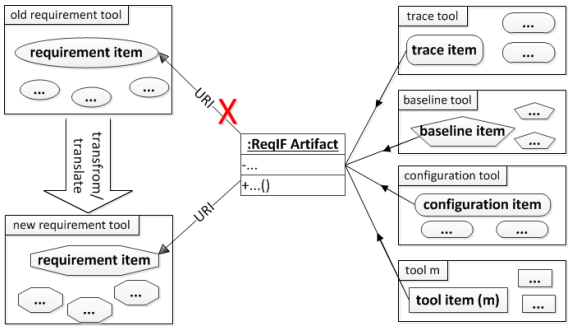

Figure 5: Artifact Benefits.

other tool (in a different language), and where this is based upon an interpretation of the models according to the semantics/domain concepts of some common domain. The best-known characterization (Wasserman, 1989) of tool integrations identifies five views of integration, where language of model is only associated with the data view, and this view does not even consider the semantics-based mapping between languages. DomainClassArtifact will cover this.

The operations of Artifact are the legal operations that other tools can perform to manipulate the model element that is represented by the Artifact. A UML Artifact with GET() and PUT() operations would only allow other tools to GET or PUT its represented UML model element.

Each real tool model element has its corresponding Artifact. The Artifact relates to the real tool model element through its URI. Assume there is a model element (e.g. a requirement item in a requirement tool) that is associated and used by many (M) other development tools (e.g. trace tool, baseline tool, configuration tool). If we want to replace this requirement tool with another similar requirement tool, a common process is that we transform the requirement model elements to the other tool, and then re-establish the associations between these elements and elements of other tools. It takes M times to establish the re-associations. If there are N such elements in the requirement model, then it costs (N * M) times for the re-establishment. Alternatively, if we use Artifact for the above tool replacement scenario, we will have Artifact objects for each model element and all integration between tools are based upon Artifacts. As illustrated in figure 5, as the development tools only know the Artifact, we will only need to simply change the URI of Artifact from the old tool element to the new tool element, then the other development tools are connect to the new tool model elements. We can thus reduce the complexity from N*M times to N. Using Artifact for integration is a flexible solution and will ease tool replacement.

This implies a need for storing Artifacts, which may be kept in a central repository. When tools modify their data like adding or deleting data items, the corresponding updates for Artifact should also be done in this central repository.

In addition to the benefits described above, using Artifact also has the following benefits:

- Standard Interfaces: All integration services work on the same Artifact/ Role objects, so tool-internal elements are handled in a uniform way, facilitating standardization for industrial tool vendors.
- Generic: Tools of the same kind share the same Artifacts, thus integrations is not constrained by any tool internal way of handling their elements.
- Traceable: Artifact has traceability feature by nature, so very little has to be known about the real traced model elements when integration is performed.

Finally, in the Artifact class it is possible to define various tool integration properties, which enhance the tool capabilities to manage different lifecycle tool elements.

3.2 Details of the Role Concept

Roles are designed to cover scenarios-based information that is required for integration, but cannot be covered by the general notion of Artifact. For instance, a UML class, represented by a UML Artifact, may be transformed into different models when different semantics are applied. This semantics information is captured by a Role class.

3.3 Identified Roles

As shown in figure 6, we encountered several kinds of Roles from the industrial integration scenarios. However, the approach is open-ended, in the sense that custom-made Roles (as subclasses of Role) can defined by integration engineers to support their specific integration scenarios.

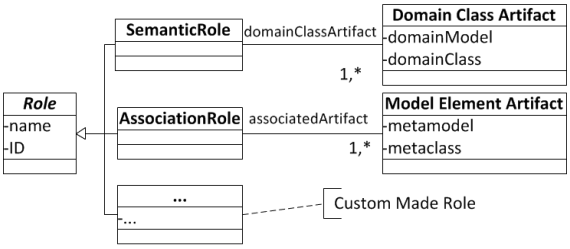

Figure 6: Role Overview.

Semantics Roles. In transformation scenarios it is common that a source model is transformed into different target models governed by different interpretations of the elements of the source models. This is normally captured by different transformations. In the cases we have encountered there is a standard way if transforming most of a source model, while some model elements should be treated differently. In addition, model elements of different models are supposed to be interpreted in the same way, independently of the languages in which these different models are made.

This calls for a mechanism where a model element (and in our case the corresponding Artifact) can play a certain *semantic* Role. It would be possible to do this by means of a simple annotation of the model element, but as we have already Artifacts as representatives of model elements, it is much more flexible to attach the semantic information to the Artifact. For UML models we could have used stereotypes, but the approach should be independent of what kinds of tools are used and their support for stereotyping.

As our approach is model-based it is obvious to require that the semantic information for a given interpretation is defined by some kind of model element. As different interpretations are often due to the interpretation given by different domains, we simply define a semantic Role to have a reference to a domain class in a domain model (see figure 6). A Semantics Role (Zhang and Moller-Pedersen, 2013b) therefore identifies a domain class in a domain model that applies to certain Artifact. As this domain class is a model element that is also represented by an Artifact, this reference (”domainClassArtifact”) is a reference to an Artifact (Domain Class Artifact).

Another benefit of having Roles defined independently of language and tool (i.e. as attached to Artifacts) is that several Artifacts (representing model elements in different models) may have the same Role, thereby capturing that we have *common semantics* across languages/tools. In a transformation scenario, a common Semantics Role for two different Artifacts preserves the semantics consistency of the transformation source and target. As an example, a UML class, a Simulink block and a part of a C code may all play the Role of a being a special kind of part of a system design. Transformation between the different model elements, or updates of the different model elements, must take this common semantics into account.

Association Roles. In a transformation scenario where various model elements are derived from a common definition model element, the Artifact that represents the common definition model element may have associations to the Artifacts that represent these derived model elements. The common Artifact has an Association Role (Zhang and Moller-Pedersen, 2013b) which links to the Model Element Artifacts that tell where the derived model elements should be placed inside these models. If the common Artifact is changed, the derived Artifacts have to be changed correspondingly.

In the wind turbine model design we transformed UML model elements to Simulink model elements. The transformation engine can only transform the temperature and wind speed UML classes to Simulink model elements; it cannot know where these transformed Simulink model elements should be placed inside Simulink models. An association Role helps to provide this information. The association Role links to the Simulink Artifact in which we can specify the locations of the generated Simulink model elements.

4 EXTENDED NOTIONS OF ARTIFACTS AND ROLES

4.1 Synthesis Artifact

An Artifact model element may represent a synthesis of properties of a number of other Artifact model elements, e.g. a report, analysis result, testing result from applying a set of tests. Whenever some of the involved model elements change, the synthesized Artifact may have to be updated.

There are synthesis links from all the involved Artifacts to the synthesis Artifact, which in turn will know which Artifacts to be involved in the synthesis. E.g. an Artifact that represents a test result in terms of a document may have synthesis links (as ”contain” relation in this case) to many other Artifacts that represent elements like texts and tables that compose this document. The multiplicity of synthesis links between Artifacts is usually one to many.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

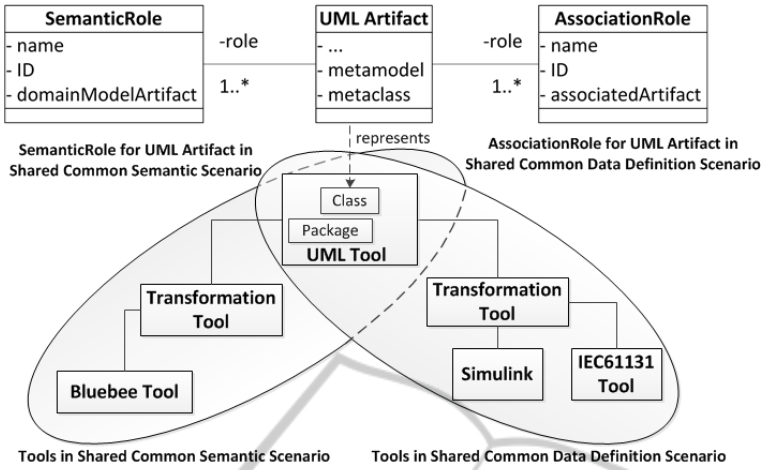

Figure 7: Example of Artifact with Different Roles.

4.2 Dynamicity of Roles

One Artifact may have multiple Roles in different scenarios. For instance, a UML Class defined by a modeler that is involved in a shared common semantics scenario may be used differently when it is involved in a common definition scenario. As figure 7 shows, in the left scenario (described in an ellipse) the Semantics Role of a UML class (represented by a UML Artifact) are used to interpret it according to a common semantics, while in the right scenario the Association Role captures the associated Artifacts that record the placement information when transforming a common definition from a UML model to Simulink and IEC61131 models. The same UML Artifact has different Roles as it requires different context-based information to support the different scenarios.

The same model element (through its representative) may be involved in multiple integration scenarios, and these may change. Different properties of the element are required in addition to the plain representative properties, in order to support the different integration scenarios. E.g. using a UML class as a transformation source in a process based upon a common semantics scenario requires scenario-specific properties that are different from those used in a transformation process as part of a common definition scenario.

The Role of a model element is not fixed in advance and depends on the actual integration. A model refinement transformation would generally produce a more precise model with the properties of its Role being extended. E.g. a UML Class representing a Computing Resource at one level of abstraction might be refined when it later represents a Processor. The Roles of model elements are required to be dynamically assigned, removed, and updated.

When engineers create UML Artifact object, it only represents UML model elements with certain dedicated software lifecycle management properties. However, it is still uncertain what specific information may be required for various integration scenarios. The dymamicity of Roles overcomes this shortcoming. The semantic Role is attached to UML Artifact when this Artifact involved in shared common semantic scenarios. When the scenario is not valid anymore, the UML Artifact can remove this semantic Role and play e.g Association Role in shared common data definition scenario. The dynamicity feature brings flexibility and complementariness for additional integration information for Artifact models.

A usual way of modeling that an object may play different Roles is to model the corresponding class with an *interface* for each Role. However, this has to be determined when the class (in this case the special Artifact class) is defined, and Artifact classes are defined based upon the *kind* of the represented model element, the *language* in which the model element is specified, and perhaps the tool being used.

Another approach to role modeling (Reenskaug, 1997) is that roles really are the things that are specified in order to know what objects do (or shall do) and how they interact, while classes are merely implementations of roles. The role approach of Rolv Bræk is similar (Bræk, 2000) (Floch and Bræk, 2003), although classes of objects are recognized as the main way of modeling objects. Both of these approaches have tried to come up with a more or less automated role synthesis, the idea being that in order to make classes of objects, make first all the role specifications and then synthesize these into classes. The reason that this is important is that when models are supposed to form the basis for implementations in some programming language (and not just be analysis models), then roles models have to be turned into class model, as

most programming languages support classes, but not roles.

All of these approaches to roles have in common that role modeling is part of modeling, which is the outcome of role modeling is to come up with classes of objects that play the roles that have been specified.

For tool integration the situation is that Artifacts may be defined based upon the language used for making model elements that the Artifact objects are to represent, and this does not change once a model element has been made. However, integration scenarios may be made after the model elements (and the corresponding Artifacts) have been made, therefore the scenario-specific properties required for tool integration (defined by Roles) have to be attached to existing Artifacts. While models and their model elements are usually maintained, and some even are used in several projects, integration scenarios may come and go depending on how the models are used in different settings. Therefore Roles may also be removed from Artifacts.

4.3 Modeling with Artifacts and Roles

4.3.1 Choreography Models

In the approach we also model the integration processes that correspond to various required interaction scenarios. While Artifacts form the basis for tool Adaptors with services, the integration processes use these services. The models that are specifying the collaborations between tool Adaptors are called choreography models. Choreography models are used to define, control, and monitor integration processes, and they are using Artifact and Role instances.

From the investigation (Guo and Jones, 2009) we know that widely-used modeling languages such as UML /SysML do not give full support for modeling of tool integration, so one of our future work items is to define a domain specific language for this. For the purpose of this paper, and in order to explore to which extent mechanisms of existing languages may be used, we have experimented with the use of SoaML, UML Sequence Diagrams and BPMN for making choreography models.

In our experiment the integration services are specified by SoaML Service Architecture models and Service Contract models. The requirements from the integration process are viewed as service architecture, thus indicating tool adaptors act as participants in the integration process. The service contracts specify how participants interact, and how their services and requests are orchestrated. Service architecture formally specifies the integration requirements performed by interacting service participants, without addressing any implementation concerns. Service Contracts simply indicate the agreed-upon interactions between participants that play the indicated Roles in the service architecture.

We have illustrated how it would be possible to use SoaML models, UML sequence diagrams, and BPMN models for constructing of the choreography models (Zhang and Moller-Pedersen, 2013a). The orchestrator orchestrates integration scenarios. It makes the individual tools integrated through adaptor services. To simplify the implementation, orchestrators for various scenarios are driven by user inputs, and they are based on the core principles of OSLC and calls OSLC services using basic HTTP commands. However, it is also possible to generate these orchestrators with process models like BPMN.

Choreography models have been specified by Service Architecture diagrams and service contract diagrams from SoaML. Part 1 of figure 8 illustrates a Service Architecture diagram and shows how different adaptor participants enable the tool integration through service contracts in shared common data definition scenario. For instance, the MOFScript adaptor participants provide Transformation Management Services for the Simulink and IEC 61131 adaptors. Part 2 illustrates the Transformation Services details that are provided and consumed in the Transformation Management Service Contracts.

4.3.2 Modeling with just Artifacts and Roles

In (Zhang and Moller-Pedersen, 2013b) the notions of integration models were shortly introduced. There it was assumed that the involved Artifacts with their respective model elements are defined when the integration models were made. However, we also introduced the distinction between *integration* models and the *base* models maintained by the integrated tools. We compared integration models with variability models within software product lines (exemplied by CVL (Object Management Group, 2013)(Haugen et al., 2008)), with both integration models and variability models being emphorthogonal to the (base) models. CVL variability models have *variability elements* that reference real model elements, and these variability elements have properties that express what kind of variability that applies to the referenced model element, like our Artifact and Roles have properties that tell what kind of tool integration the real model element may be subject to.

Variability modeling is used to specify variability within a product line, with a base model, a number of variability models, and for each variability model a number of resolution models. Feature models are

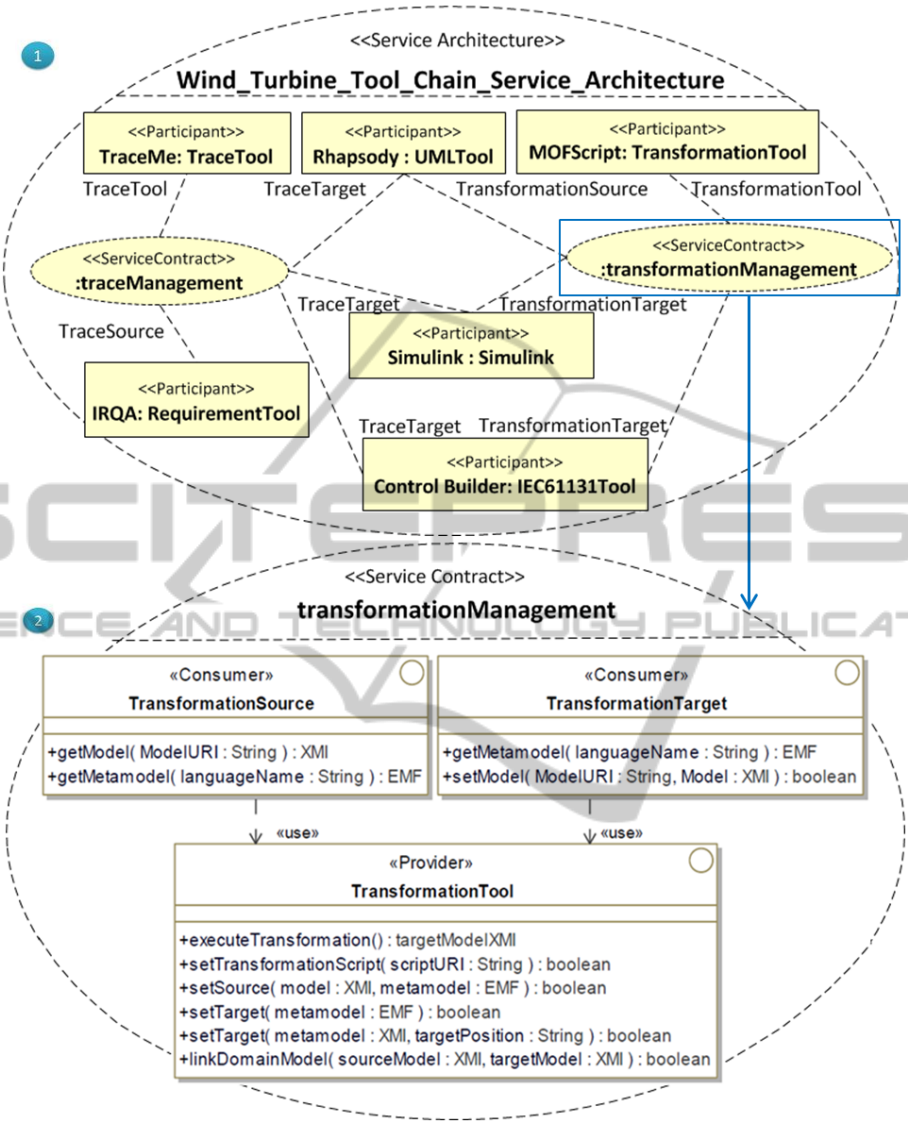

Figure 8: Sample of Service Architecture Diagram and Service Contract Diagram.

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
OSLC common properties						
All OSCL common properties are to be used as defined in http://open-services.net/bin/view/Main/OSLCCoreSpecAppendixA						
TraceRelationship additional properties						
iftrace:source	exactly-one	False	Resource	Either Reference or Inline	ifcore:ElementDescriptor ifcore:ResourceDescriptor oslc: FileDescriptor oslc: FileVersion	The reference to the source element of the relationship
iftrace:target	one-or-many	False	Resource	Either Reference or Inline	ifcore:ElementDescriptor ifcore:ResourceDescriptor oslc: FileDescriptor oslc: FileVersion	One or more references to the target element(s) of the relationship
iftrace:traceType	one-or-many	False	String	n/a	n/a	Type of the relationship
iftrace:parent	exactly-one	False	Resource	Reference	Trace	The parent of the TraceRelationship

Figure 9: Traceable OSLC Resource.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

special kinds of variability models, where there may not be any base model to which the variability models are associated: they simply specify the desired features, including relations and restrictions on features, and feature dependencies. Later, in order to use feature models in order to make specific models for each selected configuration of features, the feature models are associated with base models.

Similarly, integration models with Artifacts and Roles may be made without any real model elements being made. Up front it may very well be determined that a development of a certain system should result in a number of model element Artifacts (with Roles) and that they have to be integrated in a certain way.

The developers know that wind speed and temperature will be defined in a common language, and from this language they can generate IEC61131 and Simulink models. There may be a number so such common definitions, they may e.g. come from a dynamically generated domain analysis result. Thus we can specify integration model before making the detailed UML definitions of the common concepts, and execute this integration model when some or all the common definitions have been made in a UML model.

The implication of this for Artifact is that the property of an Artifact object that references the real model element has to a property of its own and not just an inherent property of the Artifact, e.g. given by the unique identification of the Artifact object. Integration models may therefore be made before the real model elements are made.

5 PROVIDE OSLC INTEGRATION SERVICES

As the tool integration approach is model-based, it is independent of the underlying implementation technology. In our case we chose OSLC Web Services to implement and validate the approach.

5.1 Models of Traceable OSLC Resource

Our class modeling approach was worked out in comparison with an approach where OSLC specifications were made directly. The following shows one of the main differences between the two approaches.

The difference is illustrated with traces. In this section traces are a little more elaborated than those that we have seen before. For other purposes than tracing, the following Resources had been identified: ElementDescriptor, ResourceDescriptor, FileDescriptor, and FileVersion. E.g. all elements represented by ElementDescriptor should be traceable, based on the ElementDescriptor common properties. When traceability were to be modelled in the pure OSLC approach, the modeller was left with the option of specifying the range of source and target as a list of these resources, see figure 9.

With a modeling approach, one would do as described above, i.e. make all artifacts traceable (and define all kinds of Artifacts as subclasses of Artifact). Alternatively, as illustrated in figure 10, one would define a common superclass for all the traceable artifacts, in case not all of them should be traceable. The obvious benefit is that introducing a new kind of artifact that shall be traceable amounts to define it as a subclass of Traceable, while in the pure OSLC specification style one would have to go through all ranges in other parts of the specification and change these.

5.2 From Models to OSLC Web Service

Tools can be loosely coupled via a set of services, or web services applied on standardized communication protocols. The recent framework provided by the *OSLC* community finds an agreement among the stakeholders on specification for tool integration. The key concepts are a uniform access to shared resources, a common vocabulary/formats, and a loose coupling approach between tools through REST architectures (Fielding, 2000). Using OSLC specifications implies that adaptors work on representatives of the real artifacts of tools in terms of artifact resources, and that these have to be specified in terms of OSLC-tables of properties (OSLC specifications). In addition, adaptors also specify services. A tool is integrated by implementing such adaptor specifications. Our experiment supports the generation of the part of adaptor implementations (server and client code) that adhere to the OSLC specification.

OSLC specifications define a small set of common properties that are common among all resources. In addition there are often some project-specific properties.

In the OSLC experiment we chose UML class model to define Artifact and Role models. In OSLC, Resources are representatives of models and model elements. A *Resource Shape* defines the set of OSLC Properties expected in a Resource for specific operations (i.e. creation, update or query) for each their value types, allowed values, cardinality and optionally. Models are made in languages defined by metamodels - this includes models for which there may be no concrete syntax, as e.g. requirements models.

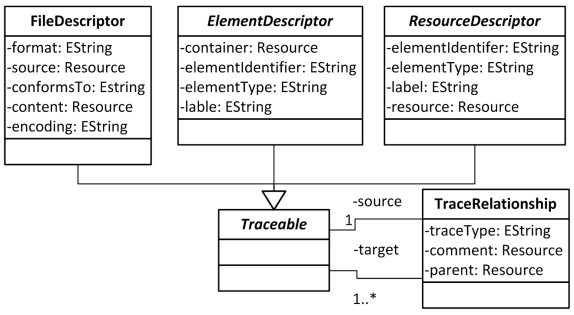

Figure 10: Model the Traceable OSLC Resource

Resources are elements according to models, but not metamodels.

The Artifact and Role models are the basis for the generation of OSLC resource definitions and server/client code for tool adaptor interfaces in terms of services. Moreover, giving some mapping rules defined between the tools concepts and engineering domain concepts, Roles can be automatically attached or removed from their artifact during the process execution.

As there are different development tools involved, UUID (universally unique identifier) is chosen as standardized UID to identify every single Artifact and Role. The experiment code is based upon the OSLC Eclipse LYO (The Eclipse Foundation, 2013) project. It adopts OSLC specifications and builds OSLC-compliant tool adaptor server and client. The code is based upon RDF and Web Service technology. The tool adaptor server part includes the common definitions used for the adaptor, such as Resource definitions (compliant to Artifact model and Role model) and constants. It mainly includes three parts. The *OSLC Registry* web application is used as an OSLC Catalog for the service providers. When each application starts, it registers its OSLC service provider details with OSLC Registry. The *OSLC Wink* is the framework that builds RESTful Web services. The *OSLC Provider* provides OSLC Resources in RDF or JSON. The adaptor server provides services and receives requests from consumers to manipulate the OSLC Resource through HTTP methods (GET, PUT, DELETE, and POST). The OSLC project uses annotations and JAX-RS methods to simplify the development process, which can also be generated from the models. The client code acts as service consumer to test the services provided by adaptor server.

Beside the implementation code, our experiment shows it is also feasible to generate tool adaptor specifications (e.g. in Microsoft Office Word format) from above class models. The generated specifications mainly include the data description (e.g. OSLC property tables, figure 9) and service description (e.g Artifact operations). Data specification defines a set of specific Resources for delegated tool adaptor, and their properties and relationships to be exposed by the provided adaptor services. Provided Services define functionalities made available to tools, while Required Services defines set functionalities needed by tool type to function as expected.

6 RELATED WORK

Tool integration has been a research area since the 90's, and it has been subject to many characterization attempts. (Brown et al., 1992) splits integration in a *Conceptual* axis that represents the understanding of integration and the *Mechanistic* axis that represents the technical way of realizing it. In (Wasserman, 1989) integration is characterized by five dimensions *control, data, presentation, process* and *platform*. Besides (Brown and McDermid, 1992) classifies integration into interoperability levels where *syntactic* relates to the agreement of tools on common data structures, and *semantics* addresses the meaning of exchanged data.

Several integration patterns have emerged: *point to point* consists of ad-hoc tools connections, *Integration Framework Environments* that integrate tools around a common framework based on standardization architectures (e.g. CORBA (Vinoski, 1997)), common formats (e.g. EIA/CDIF (Flatscher, 2002)), or infrastructures facilitating tool collaboration (e.g. Jazz (Jazz, 2011)). Tools can be loosely coupled via a set of services, or web services applied on standardized communication protocols (Web Service Oriented Architecture).

The above work forms the basis of our discussion. With the increasing complexity of systems, more and more development tasks are realized by models conform to metamodels as common representations for data, and tools have to agree on both the syntax and semantics of models that are to be exchanged (Kapsammer and Reiter, 2006). Tool metamodels are used in different ways. The Fujaba (Henkler et al., 2010) approach provides a generic solution for integrating different tool data through various metamodel design patterns. Due to the nature of MDE, approaches like VMTS (Gergely Mezei, 2006) focuses on the integration of various model-based design tools, but ignore the tools in other software development phases. Thus, lifecycle management aspects are not covered accordingly. Some of the approaches (e.g. MOFLON (Amelunxen et al., 2008), GeneralStore (Reichmann et al., 2004), CDIF (Flatscher, 2002), VMTS (Gergely Mezei, 2006), and Seman-

tic Integration from Vanderbilt (Karsai and Gray, 2000)) are designed for generic integrations without investigating specific tool integration scenarios. WOTIF(Karsai et al., 2006), jETI (Margaria et al., 2005), and ModelBus (Sriplakich et al., 2008), build the integration based upon Web Services.

Recently, ontologies for semantics integration have been used. Ontology is defined as a representation of knowledge of a domain and represents semantics information in a form that enables reasoning about data. Ontology is considered as a domain model and is different from of a metamodel which defines a language for models within a domain of. ModelCVS (Kramler et al., 2006) utilizes semantic technologies based on ontologies to fill the gaps between different tool metamodels.

Tools are used in different contexts and thus data semantics may evolve. The *Role* mechanism (Kristensen and Østerbye, 1996) can make tool chains more dynamic, adapting an object to different needs through attached Roles, each one representing a role played in one particular context (Reenskaug et al., 1996) and (Bäumer et al., 2000). (Steimann, 2000) gives a broad overview of Role implementations, and (Seifert et al., 2010) proposes one example of Role-based metamodelling approach to tool integration.

In (Weiqing Zhang, 2013), we introduced three different approaches to the modeling of OSLC Resource/ Resource Shapes by class modeling. Comparison of the different approaches is illustrated and discussed.

Our paper handles both metamodel and ontologies issues through representative Artifact models, and use the Role concept in order to dynamically assign the required integration information to Artifacts.

7 CONCLUSIONS

This paper has presented extended notions of Artifacts and Roles, to enhance their features for integrating tools. The same Artifacts are involved in multiple scenarios, which extends integration support by using the dynamicity of Roles and attach them to these Artifacts. Integration models may also be specified by just Artifacts and Roles, even before any real model elements (represented by Artifacts) are made. Finally the paper has shown that class modeling of Artifacts and Roles may be used for generating OSLC specifications, and also implementation code for constructing tool adaptors with OSLC Web Services.

ACKNOWLEDGEMENTS

The authors would like to thank for the support given by ARTEMIS Tool Integration project iFEST (iFEST Project, 2013), including our industrial partners.

REFERENCES

Amelunxen, C., Klar, F., Königs, A., Rötschke, T., and Schürr, A. (2008). Metamodel-based tool integration with moflon. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 807–810, New York, NY, USA. ACM.

Bäumer, D., Riehle, D., Siberski, W., and Wulf, M. (2000). *Role Object*, pages 15–32. Addison-Wesley, Massachusetts.

Bræk, R., editor (2000). *Using Roles with Types and Objects for Service Development.*

Brown, A. W., Feiler, P. H., and Wallnau, K. C. (1992). Past and future models of CASE integration. In *[1992] Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, pages 36–45. IEEE Comput. Soc. Press.

Brown, A. W. and McDermid, J. A. (1992). Learning from ipse's mistakes. *IEEE Softw.*, 9:23–28.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California.

Flatscher, R. G. (2002). Metamodeling in eia/cdif—meta-metamodel and metamodels. *ACM Trans. Model. Comput. Simul.*, 12:322–342.

Floch, J. and Bræk, R. (2003). Using sdl for modeling behavior composition. In *In: Proc. of the 11th Int. SDL Forum*. Springer.

Gergely Mezei, Sandor Juhasz, T. L. (2006). Integrating model transformation systems and asynchronous cluster tools. In *7th International Symposium of Hungarian Researchers on Computational Intelligence*.

Guo, Y. and Jones, R. (2009). A study of approaches for model based development of an automotive driver information system. In *Systems Conference, 2009 3rd Annual IEEE*, pages 267 –272.

Haugen, O., Moller-Pedersen, B., Oldevik, J., Olsen, G., and Svendsen, A. (2008). Adding standardized variability to domain specific languages. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 139–148.

Henkler, S., Meyer, J., Schäfer, W., von Detten, M., and Nickel, U. (2010). Legacy component integration by the fujaba real-time tool suite. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 267–270, New York, NY, USA. ACM.

iFEST Project (2010- 2013). iFEST - industrial Framework for Embedded Systems Tools. ARTEMIS-2009-1-100203, .

Jazz (2011). Jazz. http://jazz.net/.

Kapsammer, E. and Reiter, T. (2006). Model-based tool integration- state of the art and future perspectives 1.

Karsai, G. and Gray, J. (2000). Component generation technology for semantic tool integration. *2000 IEEE Aerospace Conference Proceedings Cat No00TH8484*, pages 491–499.

Karsai, G., Ledeczi, A., Neema, S., and Sztipanovits, J. (2006). The model-integrated computing toolsuite: Metaprogrammable tools for embedded control system design. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 50 –55.

Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., and Schwinger, W. (2006). Towards a semantic infrastructure supporting model-based tool integration. In *Proceedings of the 2006 international workshop on Global integrated model management*, GaMMa '06, pages 43–46, New York, NY, USA. ACM.

Kristensen, B. B. and Østerbye, K. (1996). Roles: conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3):143–160.

Margaria, T., Nagel, R., and Steffen, B. (2005). jETI: A Tool for Remote Tool Integration Tools and Algorithms for the Construction and Analysis of Systems. volume 3440 of *Lecture Notes in Computer Science*, chapter 38, pages 557–562. Springer Berlin / Heidelberg, Berlin, Heidelberg.

Object Management Group (2010). MOF Model to Text Transformation. OMG Document ad/05-05-04.pdf .

Object Management Group (2011). Requirements Interchange Format (ReqIF), Version 1.0.1. http://www.omg.org/spec/ReqIF/.

Object Management Group (2013). Common Variability Language Wiki. http://www.omgwiki.org/variability/doku.php.

Open Services for Lifecycle Collaboration (2013). OSLC - Open Services for Lifecycle Collaboration Core Specification Version 2.0 .

Reenskaug, T. (1997). Working with objects: A three-model architecture for the analysis of information systems. *JOOP*, 10(2):22–29, 40.

Reenskaug, T., Wold, P., and Lehne, O. A. (1996). *Working with objects: the Ooram software engineering method*. Manning Publications, Greenwich, CT.

Reichmann, C., Kiihl, M., Graf, P., and Muller-Glaser, K. (2004). Generalstore - a case-tool integration platform enabling model level coupling of heterogeneous designs for embedded electronic systems. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, pages 225 – 232.

Rzonca, D., Sadolewski, J., and Trybus, B. (2007). Prototype environment for controller programming in the iec 61131-3 st language. *Comput. Sci. Inf. Syst.*, 4(2):133–148.

Seifert, M., Wende, C., and Aßmann, U. (2010). Anticipating Unanticipated Tool Interoperability using Role Models. pages 52–60.

Sriplakich, P., Blanc, X., and Gervais, M.-P. (2008). Collaborative software engineering on large-scale models: requirements and experience in modelbus. In Wainwright, R. L. and Haddad, H., editors, *SAC*, pages 674–681. ACM.

Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modelling. *Data Knowledge Engineering*, 35(1):83–106.

The Eclipse Foundation (2013). Eclipse Lyo Project. http://www.eclipse.org/lyo/, 2012.

Vinoski, S. (1997). Corba: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55.

Wasserman, A. I. (1989). Tool integration in software engineering environments. In *SEE*, pages 137–149.

Weiqing Zhang (2013). Class Modeling of OSLC Resources. Technical Report, University of Oslo.

Zhang, W., Leilde, V., Moller-Pedersen, B., Champeau, J., and Guychard, C. (2012a). Towards tool integration through artifacts and roles. In *The 19th Asia-Pacific Software Engineering Conference*.

Zhang, W. and Moller-Pedersen, B. (2013a). Establishing tool chains above the service cloud with integration models. In *IEEE 20th International Conference on Web Services*.

Zhang, W. and Moller-Pedersen, B. (2013b). Tool integration models. In *The 20th Asia-Pacific Software Engineering Conference*.

Zhang, W., Moller-Pedersen, B., and Biehl, M. (2012b). A light-weight tool integration approach– from a tool integration model to oslc integration services. In *7th International Conference on Software Paradigm Trends*.