Supporting the Linked Data Approach to Maintain Coherence Across Rich EMF Models

Jad El-Khoury[1(✉)], Cecilia Ekelin[2], and Christian Ekholm[2]

[1] Department of Machine Design, KTH Royal Institute of Technology, Stockholm, Sweden
jad@kth.se

[2] Advanced Technology and Research, Volvo Group Trucks Technology, Gothenburg, Sweden
{cecilia.ekelin,christian.ekholm}@volvo.com

Abstract. In many development environments, Model-Driven Engineering (MDE) may well be limited to parts of the complete product development process due to the lack of interoperability mechanisms that connect the product data across the model-based engineering tools being used. This is especially the case if the tools are not designed to work tightly together, and/or if they do not share a common technological basis. In this paper, we investigate the use of the OASIS OSLC interoperability standard to facilitate the integration of models from different languages into a single coherent view. We evaluate a fully-automated code generator that provides OSLC interfaces for EMF-based modelling tools, allowing the exposure of modelling elements from any rich modelling language. We argue that such a generator is a critical component for reducing the cost of providing rich and specialized tool interfaces, generally needed when integrating modelling tools. The study is based on a case study that addresses the development process – and the corresponding integrated software engineering environment - at Volvo Trucks used when developing a new electronic architecture including heavy vehicle functions.

Keywords: Linked data · OSLC · Tool integration · Tool interoperability · EMF · Code generation · Model-driven engineering

1 Introduction

Model-driven engineering (MDE) is leading the effort of migrating engineering focus from text-based documentation to a digital representation of product data. Besides a model's ability to facilitate communication between individuals and teams, information conveyed in a model – when made electronically accessible – serves as a basis for analysis and synthesis activities throughout the product development life-cycle.

This MDE promise is currently best achieved with development activities constrained within a certain modelling tool, or a package of tools designed to work together. A first challenge arises with the need to maintain the MDE approach between activities relying on disparate tools. For example, the modelling tool MATLAB/Simulink [1] works well

A. Wąsowski and H. Lönn (Eds.): ECMFA 2016, LNCS 9764, pp. 36–47, 2016.
DOI: 10.1007/978-3-319-42061-5_3

with its own toolboxes, as well as with tightly integrated external products such as TargetLink from dSPACE [2]; while its integration with a UML tool is not so clearly defined. In the best case, this challenge can be handled if the tools happen to share common technologies (such as a modelling framework, storage technologies, etc.), making their integration readily possible. However, considering the variety of modelling technologies encountered during the life-cycle of a typical product development process, it is most likely that the effort needed to maintain the MDE approach across such activities becomes no longer sustainable. As a result, MDE and its benefits are typically constrained to a subset of the development life-cycle.

One approach to expand this subset is to impose the same technological space on tools throughout the development process, leading to the adoption of a more centralized platform (such as PTC Integrity [3] or MSR-Backbone [4]). While this may be feasible at a smaller scale, such centralized platforms cannot scale to handle the complete heterogeneous set of data sources normally found in a large organization. Such platforms may also be less flexible for changes over time, when additional tools need to be introduced.

A second approach is to integrate models across technologies, as advocated by solutions such as ModelBus [5]. In a sense, such solutions also assume a common technological space (that of the integration platform), which all models need to be mapped to, before they can be integrated. Typically, this relies on model transformations, leading to the risk of data duplication, and the challenge of maintaining the data synchronized and consistent across the tool chain.

In this paper, we investigate a third alternative, where one attempts to work in a technology-agnostic way, focusing instead on the model data that need to be integrated, while disregarding how the data is managed within each modelling tool. We apply the Linked Data principles [6], and in particular its manifestation in the OASIS OSLC [7] tool interoperability standard, to enable the cohesion of MDE across modelling tools.

In the next subsection, we give an overview of Linked Data and the OASIS OSLC standard, followed by an argument for our approach in adopting the standard for MDE. In Sect. 2, we present a case study performed at Volvo Trucks to investigate and validate our approach. Section 3 then presents the developed underlying infrastructure that was necessary to carry out the use case. A discussion of related work is presented in Sect. 4, before concluding the paper in Sect. 5.

1.1 Linked Data and the OASIS OSLC Standard

Linked Data is an approach for publishing structured data on the web, such that data from different sources can be connected, resulting in more meaningful and useful information. Linked Data builds upon standard web technologies such as HTTP, URI and the RDF family of standards.

OASIS OSLC is a standard that targets the integration of software tools. It builds upon the Linked Data principles, and its accompanying standards, by defining common rules and patterns to access, manipulate and query resources managed by the different tools in the tool chain.

This Linked Data approach to tool interoperability promotes a distributed architecture, in which each tool autonomously manages its own product data, while providing RESTful services through which other tools can interconnect. This leads to low coupling between tools, by reducing the need for one tool to understand the deep data of another. Moreover – like the web – the approach is technology-agnostic, where tools can differ in the technologies they use to handle their data. That is, both the data as well as the technology is decentralized.

Figure 1 illustrates a typical architecture of an OSLC tool interface, and its relation to the tool it is interfacing. With data exposed as RESTful services, such an interface is necessarily an “OSLC Server”, with the connecting tool defined as an “OSLC Client”. A tool interface can be provided natively by the tool vendor, or through a third-party as an additional adaptor. In either case, a mapping between the internal data and the exposed RDF resources needs to be done. Such mapping needs to deal with the differences in the technologies used. In addition, a mapping between the internal and external vocabulary is needed, since the vocabulary of the resources being exposed is not necessarily the same as the internal schema used to manage the data.

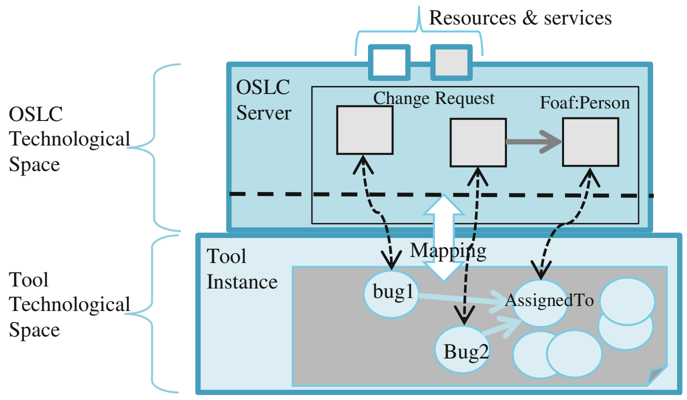


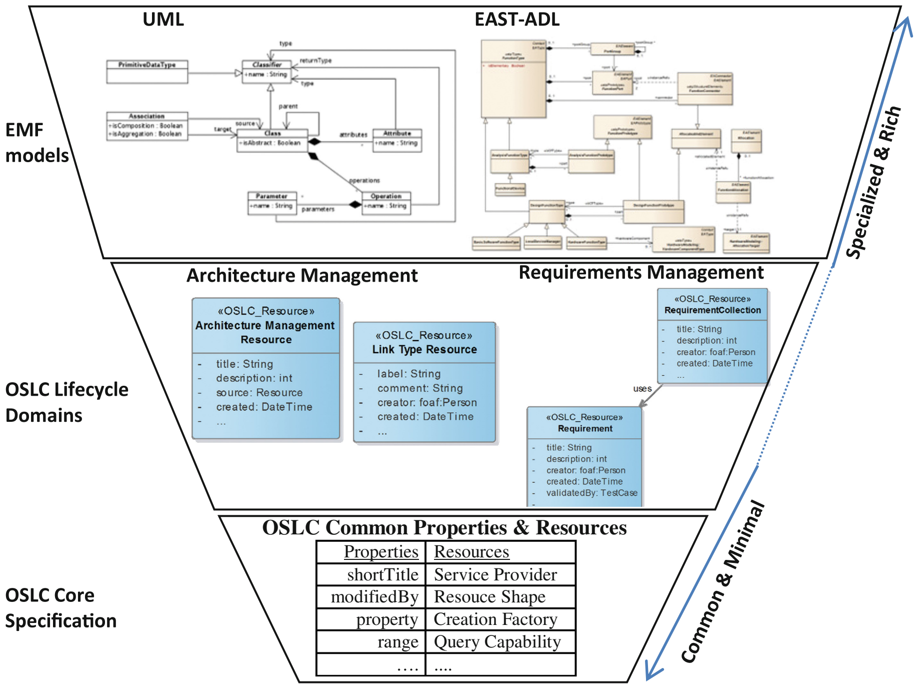
Fig. 1. Typical tool architecture, with an OSLC Server.

OSLC defines domain specifications, which include domain vocabularies (or information models) for specific lifecycle domains. The standardized domain specifications are minimalistic, focusing on the most common concepts within a particular domain, while allowing different implementations to extend this common basis. For an example of relevance to modelling tools, the Architecture Management Specification [8] only defines two resources Architecture Management and Link Type, where the former is used to represent any type of modelling elements such as a UML Class, Use Case, or Business Process Diagram.

1.2 Approach

While we agree with the minimalistic principle of the OSLC standard, it becomes apparent from our case study that a more detailed and specialized vocabulary is necessary to deal with the rich semantics generally available in many modelling tools. Modelling languages, such as UML, contain tens and hundreds of modelling artefacts that - depending on the tool usage scenarios - may need to be exposed. Moreover, these artefacts are hierarchically structured and contain relationships among themselves, creating a web structure that may also need to be exposed. As will be explained in the case study in Sect. 2, a typical integration scenario that require the exposure of many fine-grained resources is the linking of information across models at a fine level of details (for example, tracing a requirement to a specific class instead of a complete model or class diagram).

This needs not necessarily conflict with the minimalism of OSLC. As illustrated in Fig. 2, the rich and specialized vocabularies of MDE modelling languages (For a legible representation of EAST-ADL models the reader is referred to [12]), can build on the small but common foundation provided by OSLC domain specifications (such as Architecture Management [8] and Requirements Management [9]), which themselves build on the even more common OSLC Core vocabularies [10]. For ease of adoption and management, a common vocabulary needs to stay minimal; while a more specialized language can afford to be more detailed.



Properties	Resources
shortTitle	Service Provider
modifiedBy	Resouce Shape
property	Creation Factory
range	Query Capability
....	....

Fig. 2. The minimal and common basis of OSLC forms a foundation for the increasingly specialized and rich MDE modelling languages.

However, the development of an OSLC interface for a rich and specialized MDE tool may be potentially costly. First, a richer vocabulary - with the many fine-grained artefacts to be exposed at the interface - requires a larger development effort. Second, a specialized tool has normally few end-users upon which development costs can be shared. On the other hand, in the case when the metamodel of the artefacts is available in a digital format, it ought to be possible to automate the process of creating this OSLC interface. Such an approach is very advantageous since the same automation process can be reapplied to similar tools with similar technological basis, further lowering the threshold needed to adopt OSLC across the tool chain.

In this paper, we propose a fully-automated code generator that provides an OSLC interface for EMF-based modelling tools, allowing the exposure of modelling elements from any rich modelling language. Such a generator forms part of the tool support critical for reducing the cost of providing rich and specialized tool interfaces, generally needed when integrating modelling tools.

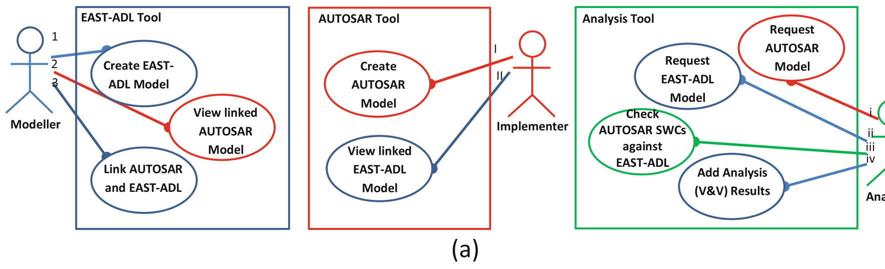
2 Case Study

AUTOSAR [11] and EAST-ADL [12] define two complementary and compatible metamodels for capturing design information for automotive embedded systems. They moreover define XML-based data exchange formats based on the metamodels. Despite the similarities of the two metamodels, no interface yet exists that allows the combination of these metamodels into a single coherent view across tools. That is, previous approaches typically focus on combining the metamodels in a single tool or framework. For example, Papyrus [13] supports an EAST-ADL profile and it would be possible to also define an AUTOSAR profile in order to support views based on the linking capabilities of EAST-ADL. Moreover, tools like EATOP [14] and ARTOP [15], both being based on Eclipse, could potentially offer view functionality by hosting their models in the same framework. It is however unnecessarily restrictive to assume that all EAST-ADL and AUTOSAR models will be based on a single tool or framework. Therefore, as part of the CRYSTAL [16] EU research project, an interface without such restrictions is being developed and assessed for tools dealing with AUTOSAR and EAST-ADL information.

A major delivery of the CRYSTAL project is the so called *interoperability specification (IOS)* [17] that describes common tool interoperability concepts. A foundation for the IOS is the OASIS OSLC standard. It was therefore inherent that also the EAST-ADL and AUTOSAR interfaces would be based on OSLC. This would allow data to be seamlessly linked and/or exchanged in order to form a global view and to maintain consistency in a manner more efficient than offered by file exchange.

In order to address the interface development properly, a case study containing data linking and exchange between EAST-ADL models and AUTOSAR models is defined. The case study addresses the development process – and the corresponding software engineering environment - at Volvo Trucks used when developing a new electronic architecture including heavy vehicle functions. This involves enabling data exchange – with the support of OSLC - for EAST-ADL and AUTOSAR models. A use case diagram for exchanging and linking EAST-ADL and AUTOSAR models is shown in

Fig. 3a. Color coding is used to improve readability of the diagram, where red represents AUTOSAR, blue represents EAST-ADL and green represents analysis combining both EAST-ADL and AUTOSAR. A typical usage scenario is illustrated in Fig. 3b, in which a Modeller and Implementer create EAST-ADL and AUTOSAR models respectively. The Modeller then uses the AUTOSAR OSLC adaptors to query and select a particular AUTOSAR model element (in particular a Software Component) and link it to a particular EAST-ADL element (DesignFunctionPrototype).



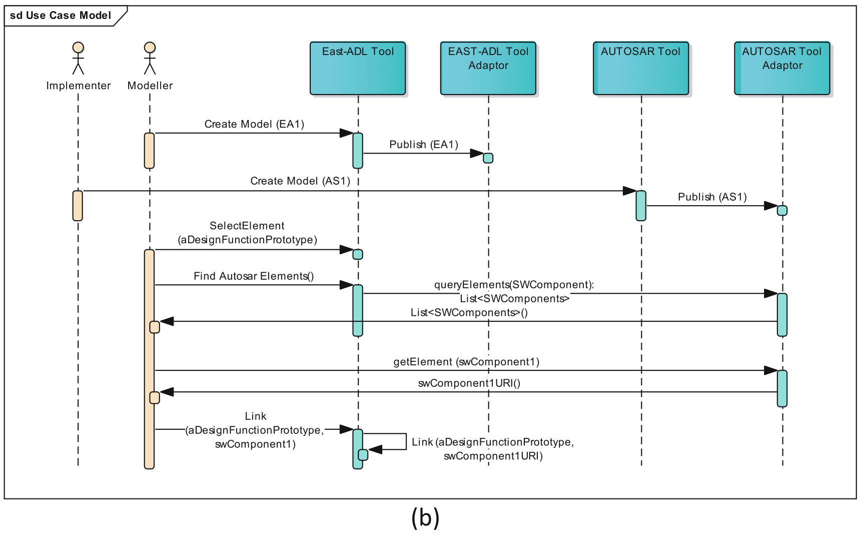


Fig. 3. (a) Use case diagram (b) A typical scenario for exchanging and linking EAST-ADL and AUTOSAR models (Color figure online)

3 Auto Generation of OSLC Interfaces for EMF-Based Models

A major cornerstone in realizing the above scenario is the ability to expose elements of the EAST-ADL and AUTOSAR models, as Linked Data resources and according to the OSLC standard. This requires the development of OSLC adaptors for the EAST-ADL and AUTOSAR tools managing their respective models.

The EAST-ADL and AUTOSAR metamodels contain hundreds of unique classes, with a complex hierarchy of multiple inheritances, and many associations between them. Early in the course of the project, it was concluded that a manual development of OSLC interfaces to expose these classes is prohibitive. Instead, the ability to automate the interface development is necessary. Several methods of automatic generation of OSLC adaptors were investigated, finally selecting the Eclipse Lyo Code Generator [18], which generates an OSLC adapter based on a specification model.

3.1 Overall Architecture of the EMF4OSLC Generator

The current Lyo code generator builds upon the Lyo OSLC4J SDK. While OSLC4J targets the implementation phase of adaptor implementation, the generator complements the SDK with a model-based development approach, which allows one to work at a higher level of abstraction, with models used to specify the adaptor design, without needing to deal with all the technical details of the OSLC standard (such as Linked Data, RDF, etc.). The generator can then be used to synthesis the specification model into a running implementation. The software consists of the following components:

- Adaptor meta-model – The adaptor meta-model allows for the graphical and intuitive specification of the OSLC adaptor functionality. It is designed to be loyal to the OSLC standard. It is built based on the EMF Ecore framework [19].
- Lyo code generator – generating the OSLC adaptor based on an instance of the adaptor meta-model.

The code generator produces most – but not all – of the code necessary for a complete ready-to-run adaptor interface, according to the OSLC standard. Only a set of methods that communicate with the source tool to access its internal data remain to be manually implemented (the dotted arrows "mapping" in Fig. 1 above). This communication is reduced to a simple set of methods to (a) get (b) create (c) search and (d) query each serviced resource. So, for a complete generation of the adaptor, the current generator needs to be complemented with two additional features:

- The automatic creation of the adaptor specification.
- The automatic generation of the necessary code to access and manipulate the data in the backend tool.

An additional component – EMF4OSLC - is hence developed that builds on top of this existing generator, in order to provide these two features. Given that the targeted modelling tools are EMF-based, EMF technologies can be readily used to (1) transform EAST-ADL/AUTOSAR metamodels into an adaptor specification model; and (2) provide the necessary Java code to access EMF model data at runtime. Figure 4 illustrates how this component fits with the current generator – as a separate component. Instead of tightly integrating the new component into the code generator, we envisage a pattern where additional components (the top-level dotted components in Fig. 4) can similarly build upon the current general-purpose generator. For example, another model generator can produce instances of the adaptor meta-model for SQL-based tools, and hence allowing for complete adaptor generation for such technologies. This naturally leads to

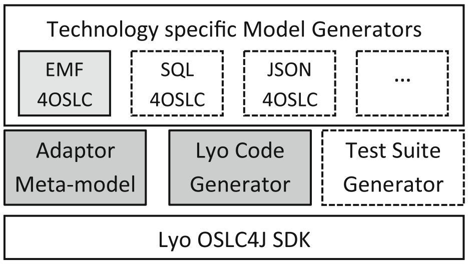


Fig. 4. The layered approach of the Lyo generator, building upon the Lyo OSLC4J SDK, and allowing for the proposed EMF4OSLC model generator.

the better interoperability between EMF-based tools and SQL-based tools, based on the technology-agnostic web services of Linked Data. Relating to the layers of Fig. 2, while the existing generator supports the OSLC core and domain specifications, the EMF4OSLC extends this support to cover the richer layers needed by EMF-based tools.

3.2 Adaptor Implementation

This section presents the implementation details of the EMF4OSLC component. While the process is identical for both EAST-ADL and the AUTOSAR metamodels, we will exemplify using the former. Figure 5 illustrates the resulting EAST-ADL data exchange infrastructure, where the model elements are handled in the tool EATOP. To implement a working OSLC adaptor from an EMF metamodel, as argued in Sect. 3, five steps are carried out, and will be described in the following subsections.

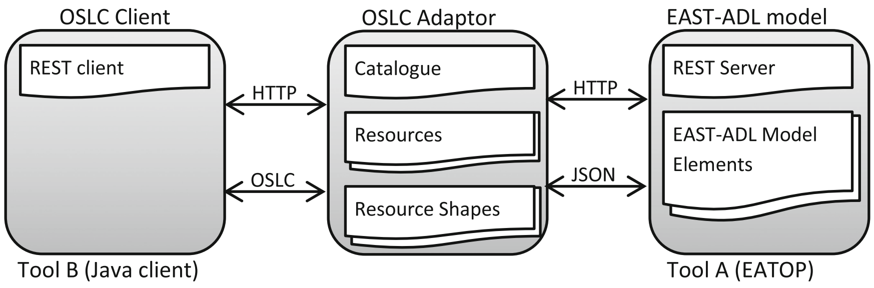


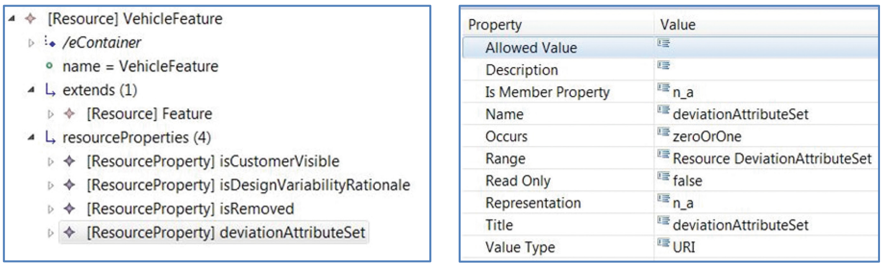
Fig. 5. EAST-ADL data exchange infrastructure

Step 1: Generating OSLC resource definitions.

As a first step, the EMF4OSLC generator imports an ECORE representation of the EAST-ADL metamodel (Version 2.1.12) [12] and traverses its elements (EClasses, EAttributes, etc.) in order to produce the corresponding OSLC Resource definitions according to the following mapping logic:

- Every EClass in the EMF metamodel is mapped to an OSLC Resource.
- Every EAttribute is mapped to an OSLC ResourceProperty, defined to be a Literal (i.e. a Property with oslc:valueType set to a literal). While the most common EAttribute types (such as Boolean, Integer and String) are supported by OSLC, EMF contains a richer set of EAttribute types that have no direct equivalent in OSLC (for example, EChar).
- Every EReference is mapped to an OSLC ResourceProperty, with oslc:valueType set to "Resource". The type of the EReference is mapped to the oslc:range of the corresponding ResourceProperty.
- For both EAttribute and EReference elements, the cardinality defined using the "lower bound" and "upper bound" properties are simply mapped to an equivalent OSLC cardinality description using the oslc:occurs of a ResourceProperty.
- Every EEnum enumeration type is mapped to an OSLC Property with an oslc: allowedValue set to the corresponding enumeration values.

As an example, the EClass VehicleFeature, found in the EAST-ADL metamodel, is translated to the OSLC resource definition VehicleFeature as illustrated in Fig. 6. In this example, the VehicleFeature Eclass inherits from another resource Feature. The resource definition also maps to four ResourceProperties. Each such ResourceProperty is in turn defined through constraints such as its range and cardinality.



```
[Resource] VehicleFeature
  /eContainer
  name = VehicleFeature
  extends (1)
    [Resource] Feature
  resourceProperties (4)
    [ResourceProperty] isCustomerVisible
    [ResourceProperty] isDesignVariabilityRationale
    [ResourceProperty] isRemoved
    [ResourceProperty] deviationAttributeSet
```

Property	Value
Allowed Value	
Description	
Is Member Property	n_a
Name	deviationAttributeSet
Occurs	zeroOrOne
Range	Resource DeviationAttributeSet
Read Only	false
Representation	n_a
Title	deviationAttributeSet
Value Type	URI

(a) (b)

Fig. 6. The corresponding OSLC resource for the VehicleFeature EClass, showing (a) resource properties and inheritance structure (b) constraints defining each property.

Step 2: Manual configuration of services.

Once the resource definitions are created, it is necessary to define the OSLC Services (such as query capabilities, creation factories, and the RESTful operations) that are needed for the relevant resources. Not all resources would require such services, and the identification of the necessary services would depend on the usage scenario of the interfaces. For this reason, it remains a manual step to define the required services, through the graphical modelling support already provided by the base Lyo code generator.

Step 3: Definition of backend connection code.

When dealing with a big meta-model that contains hundreds of resources, writing the code connecting the OSLC adaptor with the backend tool (in this case EATOP) would be a laborious task. In this step, the EMF4OSLC component produces code templates for each of the Get/Create/Query methods of each serviced resource. The Lyo Generator in turn uses these templates to produce the appropriate final java methods. Figure 7 shows an example template for the Get-method of a resource.

```
backendCodeTemplate_getResource =
  "String url =
"http://ea_server.volvo.com/element/[ResourceClassName/]?get=app
lication/json&id=[Parameter2/]";
  aResource = new Gson().fromJson( sendGet(url), [
  ResourceClassName].class);"
```

Fig. 7. Resource get-method template code, where template parameters are marked in bold

Step 4: Automatic generation of the OSLC adaptor.

Once the resources are defined, and their corresponding services are configured, the OSLC adaptor is generated through the standard existing Lyo code generator functionality.

Step 5: Building and running the OSLC adaptor.

Steps 1-4 result in a fully functioning Eclipse project that is ready to run as an embedded OSLC web server. Such server can then be reached by an OSLC client using any of the OSLC defined discovery capabilities.

4 Related Work

The prototype proposed in [21] maps between EMF objects and RDF Resources, based on a mapping language between the EMF domain model and the corresponding RDF ontology. More generally, [20, 22] are representatives of a flora of solutions that attempt to translate between the RDF data model and the more traditionally encountered technologies. In particular, [20] provides a library for accessing RDF data from the dynamic object-oriented Ruby programs, while [22] provides a solution to publish relational databases as RDF graphs. While such approaches are crucial in defining the basic mapping between RDF and EMF, our work is dedicated towards the more specific OSLC standard, which is of most relevance to tool interoperability and MDE.

The work presented here is similar to the solution in [23], where an OSLC-based integration of EMF models is proposed in the Rational Software Architect Design Manager (DM) tool. DM allows its EMF-based models to be exposed as OSLC resources. This however assumes that the EMF models are defined and used within the DM tool itself. Instead, our approach provides a generic infrastructure that supports models from different EMF-based tools, and without necessarily knowing the internal mechanisms of the tools. This facilitates the integration of models across tools (EMF-based and/or otherwise). To ensure sustainability, we also desire an approach that builds on an established code generator for OSLC technologies, and which can be further extended to support other technologies as discussed in Sect. 3.1.

5 Conclusion

In this paper, we identified the need for efficient support in the development of tool interfaces in order to expose the rich and specialized semantics normally found in modern MDE tools. It is argued that such support can best be made possible by adopting the technology-agnostic Linked Data approach (and OSLC standard) to tool interoperability. In addition, one can take advantage of the digital access in model-based tools to automate the process of producing the needed interfaces.

A core component of such tool support is the proposed fully-automated code generator that provides an OSLC interface for EMF-based modelling tools, allowing the exposure of fine-grained elements from any rich modelling language. This approach was validated in an automotive case study at Volvo Trucks, using the EAST-ADL and AUTOSAR metamodels. Such metamodels contain hundreds of unique classes - with a complex hierarchy of multiple inheritances – that can now be exposed as OSLC resources for other tools to integrate with. Naturally, such mapping between the different paradigms (the classical object-oriented model of EMF and the RDF data model of Linked Data) is necessarily accompanied with complexities and compromises that one needs to deal with. The work presented here does not attempt to provide such a formal mapping, taking instead an initial hands-on approach. A discussion of the substantial differences in the underlying data models can be found in [20]. A practical issue from our own case study highlighted that while EAttributes and EReferences are defined within the context of their EClass, properties in Linked Data are stand-alone first-class entities that can be reused across many resources. In the case of rich modelling languages, this leads to an explosion in the number of properties that have no natural structuring entity. In addition, while the proposed generator allows for the exposure of the complete complex structure of an EMF model, the capability to configure and limit the hierarchy of artefacts being exposed would be desired.

While we have demonstrated our solution for the EMF technology, automation support is necessary for other technologies in order to ensure interoperability between a wider range of modelling tools. On-going work is under way to support SQL-based tools. By building upon the Lyo code generator and the architecture of Fig. 4, this further reduces the threshold of integrating modelling tools in a tool chain.

Acknowledgement. The research leading to these results has received partial funding from the European Union's Seventh Framework Program (FP7/2007-2013) for CRYSTAL – Critical System Engineering Acceleration Joint Undertaking under grant agreement No. 332830 and from Vinnova under DIARIENR 2012-04304.

References

1. MathWorks MATLAB/Simulink, April 2016. http://se.mathworks.com/products/simulink/
2. dSPACE TargetLink, April 2016. https://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm
3. PTC Integrity, April 2016. http://www.ptc.com/application-lifecycle-management/integrity

4. Weichel, B., Herrmann, M.: A backbone in automotive software development based on XML and ASAM/MSR. In: SAE Technical Papers (2004). doi:10.4271/2004-01-0295
5. Hein, C., Ritter, T., Wagner, M.: Model-driven tool integration with modelbus. In: Workshop Future Trends of Model-Driven Development (2009)
6. Berners-Lee, T.: Linked data design issues, April 2016. http://www.w3.org/DesignIssues/LinkedData.html
7. OASIS OSLC, April 2016. http://www.oasis-oslc.org/
8. OSLC architecture management specification version 2.0, 2011, April 2016. http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-Version-2.0/
9. OSLC requirements management specification version 2.0, 2012, April 2016. http://open-services.net/bin/view/Main/RmSpecificationV2
10. OSLC core specification version v2.0, 2013, April 2016. http://open-services.net/bin/view/Main/OslcCoreSpecification
11. AUTOSAR: automotive open system architecture, April 2016. http://www.autosar.org
12. EAST-ADL: electronic automotive systems architecture description language, April 2016. http://www.east-adl.info/
13. Papyrus - modeling environment, April 2016. https://eclipse.org/papyrus
14. EATOP EAST-ADL tool platform, April 2016. http://www.eclipse.org/eatop/
15. AUTOSAR tool platform (Artop), April 2016. https://www.artop.org/
16. CRYSTAL - critical system engineering acceleration - an artemis project, April 2016. http://www.crystal-artemis.eu/
17. Loiret, F., et al.: Draft proposal on EAST-ADL/AUTOSAR for IOS, in Interoperability Specification (IOS) - V2, CRYSTAL deliverable D601.022, 2015, pp. 183–188. http://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_601_022_v1.0.pdf
18. Eclipse Lyo code generator, April 2016. https://wiki.eclipse.org/Lyo/AdaptorCodeGeneratorWorkshop
19. Eclipse modeling framework project (EMF), April 2016. https://eclipse.org/modeling/emf/
20. Oren, E., Heitmann, B., Decker, S.: ActiveRDF: embedding semantic web data into object oriented languages. In: Web Semantics: Science, Services and Agents on the World Wide Web (2008)
21. Hillairet, G., Bertrand, F., Lafaye, J.Y.: Bridging EMF applications and RDF data sources. In: 4th International Workshop on Semantic Web Enabled Software Engineering (2008)
22. Bizer, C., Cyganiak, R.: D2R server: publishing relational databases on the SemanticWeb. In: Proceedings of the International SemanticWeb Conference (2003)
23. Elaasar, M., Neal, A.: Integrating modeling tools in the development lifecycle with OSLC: a case study. In: Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems, MODELS, pp. 154–169 (2013)