A Recipe for Tool Interoperability

Andreas Baumgart and Christian Ellen
OFFIS, Escherweg 2, 26121 Oldenburg, Germany

Keywords: RTP, Tool Interoperability, Tool Integration, Tool Adapter, OSLC, IOS, Meta-Model.

Abstract: Typical engineering and verification workflows for safety-relevant systems are performed with many different tools. For such workflows safety standards like the automotive ISO 26262 require traceability of all V&V-related work products. Therefore, efficient tool-integration, while ensuring all traceability needs for functional safety, is a highly relevant topic for industrial domains. Recent research projects like CESAR have addressed this topic by reusing tools and methods for different workflows in the context of requirements and systems engineering as well as verification and validation. This reuse is done in a Reference Technology Platform (RTP) with common services and a common understanding of exchanged information based on an Interoperability Specification (IOS). Recently, OSLC is discussed for such an IOS. The open question is how tools are connected efficiently and how traceability is ensured. This document provides a guideline on how to use the IOS for engineering workflows performed with different tools based on OSLC. We call it recipe. The recipe was developed in the MBAT project. It considers a systematic definition of semantic concepts for an IOS ensuring traceability and the level of granularity required to perform workflows with a set of tools.

1 INTRODUCTION

Workflows related to different engineering disciplines like requirement definition, system design, analysis, and testing are typically performed with several distributed tools. These workflows have to comply with standards like the automotive ISO 26262 (ISO, 2011) for functional safety which e.g. requires traceability of engineering information. Yet, this engineering information is normally very heterogeneous with regard to the tools and their interfaces. Lately, Open Services for Lifecycle Collaboration (OSLC) has become a promising approach for tool interoperability (OSLC Community, 2013). It can be used as a foundation for an Interoperability Specification (IOS) that enables communication between tools to support engineering lifecycle workflows in a Reference Technology Platform (RTP). Typical engineering disciplines like architecture management or requirements management are addressed by the semantic concepts of OSLC. Yet, they are quite generic and do not reflect workflows that shall actually be implemented in an RTP. Therefore, the semantic concepts of OSLC must be extended when implementing workflows with interoperating tools.

This publication addresses a practical problem of developers responsible for tool integration. They typically have to create or modify tool adapters to support workflows and need to comply with existing processes and standards. Tool integration effort is a highly relevant cost factor for industrial projects. Creation and extension of many point-to-point connections is very expensive and complicates traceability required by standards. A logical consequence is the reduction of interfaces. It is possible to reduce the set of interfaces to one big meta-model covering the data models of all tools. Yet, with this approach commonalities are not shared and traceability is not necessarily enabled. The level of granularity required for the interfaces and a useful modularization of exchanged information elements is therefore crucial. An efficient solution for tool interoperability is tailorable to different workflows, implemented with different tools, and supports traceability.

Our contribution is a development guideline in the form of a recipe. It allows systematic definition of the semantic concepts for an interoperability specification and takes into account the implementation of workflows with different tools. The guideline considers the usage of agreed semantic concepts from existing meta-models and standards. With this approach a homogeneous view on the tools is established in which all information is fully traceable at the right level of granularity required for the workflows.

Baumgart A. and Ellen C..
A Recipe for Tool Interoperability.
DOI: 10.5220/0004702203000308
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 300-308
ISBN: 978-989-758-007-9

This document is structured as follows. Subsequent to this introduction we describe the basic principles needed for the proposed guideline and discuss the related work. In our main section we define the guideline, which we call "recipe". Finally, we evaluate the manual application of the recipe for a typical engineering scenario and finish with our conclusion.

2 BASICS

The guideline for tool interoperability, that we propose as a recipe, considers some basic concepts which will be explained in this section. The first is an Interoperability Specification (IOS) whose semantic concepts are defined with regard to tools and workflows. The other basic concept is a Reference Technology Platform (RTP) with multiple logical layers addressing workflows, tools, and interoperability.

2.1 Interoperability Specification

The purpose of the IOS is to provide basic technical, syntactical, and semantical principles for interoperability between tools. Together with implementation guidelines, tools can be easily integrated to achieve specific workflows. The proposed recipe builds upon the systematic definition of the semantic concepts as a refinement of OSLC specifications by using meta-models:

OSLC is an open community project aiming to ease integration of tools with other tools. The approach builds upon ubiquitously established internet and linked-data standards like HTTP, RDF/XML, RESTful services, as well as open and extendible minimalistic data specifications. RDF/XML provides common syntax for data resources using a triplet structure that gives them the form subject-predicate-object. Resource type specifications contain definitions of such resources, their semantics, links to other resources, and simple properties. They are defined based on OSLC core principles like resources and properties allowing their usage for OSLC-based service definitions. Common namespaces such as RDFS (W3C, 2004), FOAF (Brickley and Miller, 2010) or DC (Dublin Core Metadata Initiative, 2012) are used for standardized communication of resource properties. OSLC resource types are modular with regard to domains like Requirements Management or Architecture Management that reflect different engineering disciplines. Yet, for typical workflows these resource types need to be extended for which we propose the usage of meta-models.

Meta-Models consist of concepts, relationships, and properties with defined semantics. They allow expressing parameters and results of methods used for workflows. The meta-model concepts are abstractions from domain-specific work products and from tool-specific data elements. As such they are usable to define interfaces for tools and services that support the methods. These interfaces can be reused with other tools that comply with the needed concepts from the meta-model. For instance methods for requirements can be performed with tools that can handle requirements.

The set of meta-model concepts and relationships required in an RTP depends on the workflows and methods that shall be supported. For their support tool-specific data models must be mappable to these meta-model concepts and relationships. Since tools and workflows can vary the meta-model is no static structure but is flexible with regard to the usage scenario. Therefore, defining one big meta-model covering all considerable concepts does not help to specify needed tool interfaces. The meta-model should be minimalistic with regard to the information exposed or required by the tools to support the workflows and allowing traceability of exchanged information. Yet, the usage of widely agreed concepts from existing standards like UML/SysML (OMG, 2010) and EAST-ADL (EAST-ADL Association, 2013) as well as project results like HRC (SPEEDS Project, 2009), the SPES 2020 methodology (Pohl et al., 2012), or the meta-model of the CESAR project (Rajan and Wahl, 2013) for the names of the concepts is highly recommended.

Meta-model concepts and methods are typically related to engineering disciplines like Requirements Management or Architecture Management. Therefore, the meta-model concepts can be associated with respective modules like the resource types in OSLC. Such a modular meta-model providing common concepts with the scope of analysis and testing methods is in development in MBAT (MBAT Project, 2013).

2.2 Reference Technology Platform

A Reference Technology Platform (RTP) is a collection of tools and services that can interact with each other to support specific engineering scenarios and workflows. The RTP includes common principles for tool interoperability and can therefore be regarded at three different logical layers (Vasaiely et al., 2012). These layers are workflow layer, tool layer, and IOS layer. They are depicted in Figure 1 illustrating a small RTP example for which the proposed recipe has been applied.

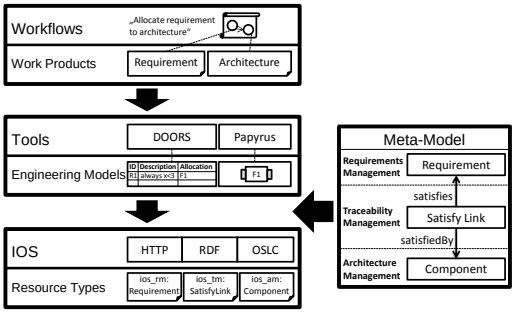

Figure 1: IOS and Meta-Model.

Workflows and Work Products: On the first RTP layer activities and work products of domain-specific but tool-independent workflows and methods are described. The work products are part of pre- and post-conditions of the activities. In Figure 1 the typical engineering workflow "Allocate requirement to architecture" is shown as an example. It is described within all systems engineering standards (e.g. the ISO 26262) and requires traceability of architectures and allocated requirements. The preconditon of the workflow is a defined requirement. The example workflow could consist of activities like "define architecture" and "allocate requirement". Post-condition is that the requirement is allocated to an architecture.

Tools and Engineering Models: On the second layer different tools and services are considered which shall be used to perform the activities of the workflow layer. Work products are mapped to elements native to the data models of the tools and services – we will call them engineering models. In the depicted example the activities are mapped to the UML modeling tool Papyrus with an EAST-ADL profile and to the requirements management tool IBM Rational DOORS. An architecture as work product is represented by an EAST-ADL model in Papyrus. Requirements are stored in DOORS together with the allocation information.

IOS and Resource Types: The third layer addresses the IOS and provides basic technical, syntactical, and semantical principles for the interoperability of tools. For the semantics of communicated data elements the IOS layer includes resource types. These resource types are specifications of concepts, relationships, and properties. They reflect a meta-model with concepts and relationships needed for interoperability between the tools to achieve the workflows.

The meta-model concepts used for the IOS are an abstraction of data elements that shall be communicated between the tools. This abstraction allows a common understanding and traceability of information shared among the tools by establishing a homogeneous view. In this view methods required for workflows can be performed on different tools that relate to the same meta-model concepts. All engineering information in this homogeneous view is traceable with regard to the traceability concepts of the meta-model.

As mentioned in Section 2.1 the meta-model concepts can typically be modularized with regard to specific engineering disciplines like requirements management or architecture management. Such modules reflect the role of the tools and their data elements in a workflow. In the illustrated example the meta-model concepts `Requirement`, `Satisfy Link`, and `Component` are identified from the requirement allocation workflow that is mapped on DOORS and Papyrus. A `Requirement` is a concept from Requirements Management expressing a specific need. A `Component` is a concept from Architecture Management expressing a reusable architectural element that has defined interfaces, which can be decomposed and which can have an implemented behavior. The `Satisfy Link` is a Traceability Management concept having the semantics that the Component shall satisfy the Requirement. It enables traceability required for the allocation of requirements to architectures given by component models. On the IOS layer the corresponding resource types `ios_rm:Requirement`, `ios_tm:SatisfyLink`, and `ios_am:Component` are defined all representing the respective meta-model concepts and modules.

3 RELATED WORK

In this section related work done in research projects is discussed and used to motivate our approach. The topic of tool interoperability has been discussed in many research projects like CESAR or iFEST. Several commercial platforms and freely available frameworks support tool interoperability like Enovia from Dassault Systèmes, ModelBus from Fraunhofer FOKUS, the ATHENA Interoperability Framework, EIF or IADBC. Many recent approaches are related to OSLC. On the commercial side OSLC is supported by the Jazz platform from IBM.

Partners of the iFEST project (Zhang et al., 2012) propose an artifact model and the usage of OSLC-technologies for implementing adapters. They realize interoperability between tools in contrast to our solution directly by accessing tool APIs. The artifact model allows an abstraction of tool-specific meta-model elements as artifacts and considers trace-links

between such artifacts in different tools. Yet, specializations of artifacts are tool-specific. Semantic commonalities for engineering disciplines or different engineering domains are not considered for the model. The approach allows light-weight tool integration and traceability by using the generic artifact concept for information exchange. It does not abstract from tool-specific data to common meta-model concepts that support the implemented workflows.

CESAR partners (Vasaiely et al., 2012) discuss a common interoperability approach for heterogeneous tool environments. The approach considers an open interoperability specification including a service-oriented architecture, a common data format with defined syntax like RDF and semantics as well as common communication protocols like HTTP and REST. These principles are those addressed by OSLC. The approach considers an interoperability specification as part of a layered RTP as described in section 2.2. Each layer has semantic concepts. It is mentioned that those on the IOS layer can be based on OSLC domain specifications but typically require additional concepts and shaping with regard to higher domain layers. Yet, it is neither defined which concepts should belong to this layer nor it is defined how the concepts are chosen to support specific engineering workflows.

In our prior work (Baumgart et al., 2012) we proposed the usage of a meta-model as semantic model to define common interfaces that support tool interoperability in an RTP. The meta-model was developed in CESAR and addresses different topics of systems engineering like requirements, component-based design or verification and validation. The idea is that the common interfaces establish a homogeneous view on a heterogeneous set of tools that shall interact with each other. Tools can be easily integrated into this view as data-repositories by implementing the interfaces in tool-adapters. Other tools can be integrated as clients and use the common interfaces for engineering scenarios independently from those tools they interact with. Yet, in this prior work we did not mention how meta-model types are derived for the interfaces with regard to engineering workflows that shall be supported.

In contrast to the approaches discussed above our recipe addresses interoperability with OSLC by identifying and adapting semantic concepts based on existing models to enable a specific workflow on different integrated tools.

4 THE RECIPE

Our recipe for tool interoperability targets a systematic implementation of an IOS fitting the needs of workflows and methods that shall be achieved with different interoperating tools. Like a cooking recipe, we describe the ingredients in terms of work products and steps that need to be performed. These steps and work products are depicted in Figure 2. The recipe considers five steps which will be described in detail in the following section together with the prerequisites and the expected outcome.

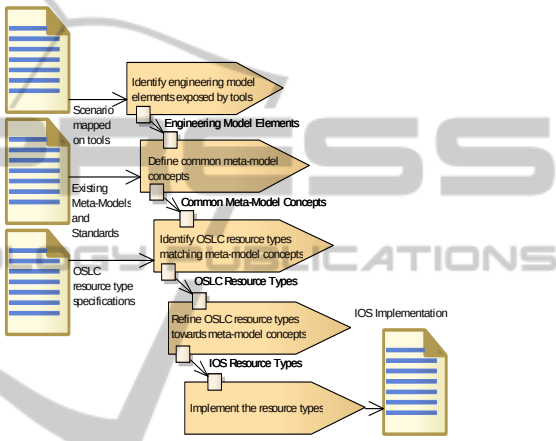

Figure 2: The Recipe with its inputs, outputs, and steps.

4.1 Prerequisites

Before applying the recipe it is assumed that a workflow or method with activities and work products is defined and mapped to involved tools. Therefore, it is defined which activities are performed with which tools. In particular, it is also defined how the work products are represented by the tools and their engineering models.

4.2 Identify Engineering Model Elements exposed by Tools

In a first step all relevant engineering model elements are identified. Relevant elements are those provided or consumed by the tools in order to perform the activities that are mapped to the respective tools. The step is done when all engineering model elements of the tools are identified which represent work products of the scenario activities. It is not necessary to look deeper into further details of the engineering models because all model elements, relationships, and properties needed to support the workflow or methods are

identified by now. The outcome of this step is a minimal set of engineering elements that is sufficient to implement the methods or scenarios with the tools.

4.3 Define Common Meta-Model Concepts

In a second step a meta-model is defined for the identified engineering model elements. The meta-model is an abstraction of these elements and must be rich enough to cover them. It shall on the one hand represent the commonalities of the engineering model elements and on the other hand represent their specifics where needed. Generally the meta-model must cover all traceability needs coming with the workflows to be implemented with the tools and their engineering models.

Granularity depends on the information needed for the implementation of the workflows with the tools. Therefore, the concepts of the meta-model can be defined in a granularity range between the full engineering model and only abstract artifacts and relationships. For example if single components have to be assigned to an implemented behavior in a simple traceability scenario, the meta-model must only provide abstract concepts for components, behavior, and the implementation link between them. Yet, in a more enhanced integration testing scenario the single components have to be more detailed and are implemented and interconnected via their ports with other components to form a component architecture. In this case also the information about the parts of the component architecture, the ports, the implemented interfaces and the interconnections is needed. The required level of abstraction can be even finer and very specific to an engineering model. A typical example is a tool-specific formalism used to define the implemented behavior.

The meta-model concepts should be defined based on already existing concepts. Many concepts, relationships, and properties are provided by standards or project results as described in Section 2.1. Typically, the exact scope of the workflow at hand varies, but concepts can be reused and extended in a way that the workflow is supported. Trace-links between elements should be represented explicitly as elements. This allows bidirectional relationships with additional attributes, direct traceability of proof obligations, and external storage of links for tools that do not support additional relationships.

This step of the recipe is finished when the meta-model is rich enough to cover every identified engineering model element, relationship, and property. In the end, every identified engineering object is mapped to a corresponding concept, relationship, or property in the meta-model. Therefore, it is ensured that the meta-model can be used to express the work products of the workflow and to implement it with the tools.

4.4 Identify Matching OSLC Resource Types

In a third step, all concepts, relationships, and properties of the meta-model are mapped to OSLC resource types. The OSLC project defines several domains for typical engineering disciplines specifying resource types with specific links and properties. Therefore, the meta-model concepts are grouped with regard to engineering domains and then they are mapped to resource types. The relationships and properties of the meta-model concepts are mapped to existing links and properties defined by the OSLC resource types where applicable. Examples of OSLC domains are the Requirements Management (`OSLC_RM`) which defines the basic requirement resource type and the Architecture Management (`OSLC_AM`) which defines abstract architecture resources and links.

If a meta-model concept cannot be mapped to a resource type of an existing OSLC engineering domain then it has to be mapped to a `Resource` as defined by the OSLC Core.

The step is done when all meta-model concepts are mapped to OSLC resource types and when all relationships and properties of the meta-model concepts are processed and mapped to links and properties of the resource types where applicable.

4.5 Refine OSLC Resource Types towards Meta-Model Concepts

In a fourth step the OSLC resource types are refined. The purpose of this refinement is to support the level of granularity required for the workflows on IOS level. This level of granularity is given by the identified meta-model concepts, relationships, and properties that are mapped to the OSLC resource types. Therefore, OSLC resource types must be made fully compliant to the concepts, relationships, and properties of the meta-model. Further type information, properties, and links maybe needed. By adding them to the OSLC resource types, refined IOS resource types are defined that support the level of granularity required for the workflows.

The refinement of the resource types is done in a systematic way. All mappings of meta-model concepts to OSLC resource types are processed. If the identified OSLC resource type is fully representing the mapped meta-model concept with its properties

and relationships then the step is done for that meta-model concept. Typically, the corresponding OSLC resource type is not fully sufficient to represent the meta-model concept. Thus, it has to be extended towards the respective meta-model concept.

The extension of an OSLC resource type towards a meta-model concept is also a systematic procedure. The OSLC resource type is considered as a base type. The meta-model concept must become visible as a refinement. This can be done by defining an own namespace and a URL for the identification of the meta-concept as resource type. This resource type URL is simply added to the definition of the resource type. Typically, the type of an OSLC resource is identified either by the URI of an RDF node or the value provided with the `rdf:type`. Therefore, the additional resource type URL identifying the meta-model concept is either one of a set of RDF types or it is the identifier of the RDF node. This can be formulated as a constraint as part of the description of the resource type.

As an example the meta-model concept of a `Component` is defined related to the Architecture Management engineering domain. Therefore, the concept of a `Component` is mapped to the OSLC Architecture Management resource type `(AM) Resource` as base type. In order to identify `Component` as extended resource type a URI is defined for that type which addresses the concept and its namespace. In this example the namespace is IOS_AM and has a URI `http://ios.artemis.eu/am#` with a defined prefix `ios_am`. Therefore, a `Component` resource can be identified via the URI `http://ios.artemis.eu/am#Component` or via the shorter prefixed URI `ios_am:Component`. The `rdf:type` is constrained in its description that it must contain the URIs `ios_am:Component` and `oslc\am:Resource` if not already addressed by the RDF node identifier. Thus, a `Component` resource is understood as `Component` and as OSLC `(AM) Resource`.

For the extension of an OSLC resource type towards a meta-model concept also the additional references and properties of the meta-model concept have to be made available. Therefore, for each reference or property an OSLC link or property is defined extending the specification of the resource type. It is highly recommended to reuse links and properties from existing namespaces in order to improve compatibility and common understanding of relationships and properties communicated via IOS. A typical namespace used by OSLC is DC (Dublin Core Metadata Initiative, 2012) which contains many common properties and links like `dcterms:title`, `dcterms:description`, or `dcterms:identifier`.

If existing links and properties cannot be reused for a reference or property of a meta-model concept then a new link or property has to be defined according to the OSLC core specification guideline "Defining OSLC Properties": For the definition the link or property requires a URI which typically consists of a namespace part and an appended identifier for the property or link. Furthermore, a set of properties needs to be defined, e.g. the number of occurring values for a property ("Occurs"), the value-type or the range of resource types allowed for linked resources.

As an example the meta-model concept `Component` has a port reference. This port reference cannot be mapped to an existing link or property. Therefore the new link `ios_am:port` is specified for the namespace `ios_am` and added to the specification of the resource type `ios_am:Component`. The value of the "Occurs" property is zero-or-many – a component may have an arbitrary number of ports. The value-type is `Resource` and the value of the Range property is `ios_am:Port`. Therefore, the value of this link may only consist of `ios_am:Port` resources. The "Representation" property is set to "Either" so a port may either be contained or referenced. The "Read-only" property is considered `false` because ports may be added. The link also has a description.

This step of the recipe is done when all resource types are defined in a way that they are fully sufficient to represent the corresponding meta-model concepts. Result of this step is a set of refined resource types which we call IOS resource types.

4.6 Implement IOS Resource Types

In the final step of the recipe the IOS resource types are implemented. For that purpose technological principles defined by OSLC (OSLC Community, 2013) for the IOS are applied (e.g. RDF and HTTP). Different ways are possible to implement resource types together with OSLC-compliant communication. We consider two possible implementations. One implementation is based on existing Java technologies like the Eclipse Modeling Framework (EMF), the servlet-engine, and HTTP-server Jetty from the Eclipse Foundation, as well as the RDF-library and the serializer Jena from the Apache Software Foundation. The other implementation is based on the reference implementation Lyo respectively OSLC4J which is also hosted by the Eclipse Foundation.

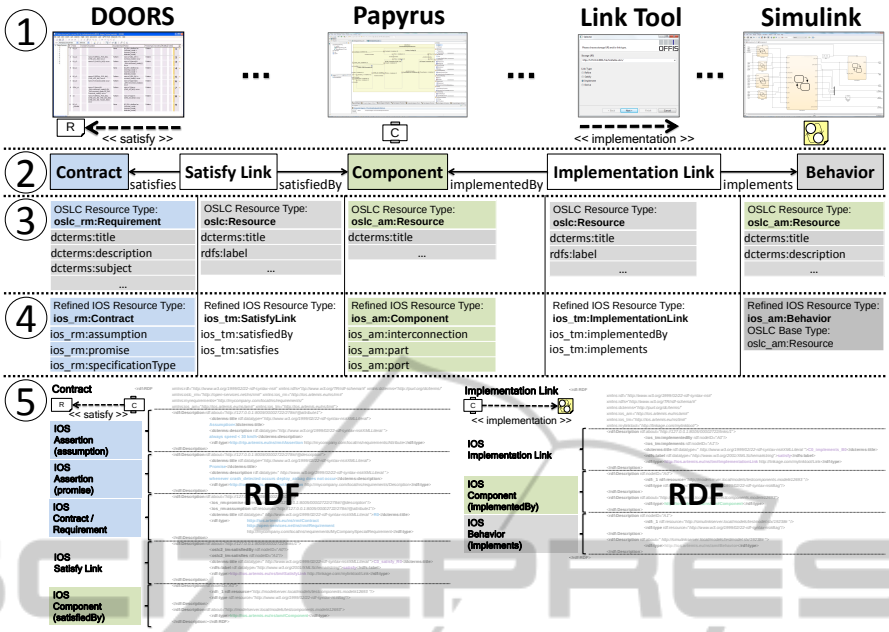

Figure 3: Application of the different steps of the recipe to a black-box testing scenario using DOORS (requirements), Papyrus (component model), and Matlab Simulink (implementation model).

5 EVALUATION OF THE RECIPE

In this section we present our evaluation of the recipe on an extended scenario based on the requirement allocation workflow described in Section 2.2. For the scenario we wanted to create an RTP which facilitates interoperability between different industrial relevant tools hosting data. The idea is to not only support linking of requirements to components (satisfy link), but also support linking of implementation models to components (implementation link). This enables the RTP to perform black-box testing of the implementation of a component against its requirements. Similar to the original scenario (see Section 2.2), the components are modeled in Papyrus by using the EAST-ADL profile and function types. The requirements are managed within DOORS. In addition, we extended DOORS to support the contract-based specification of requirements, which allows to specify assumptions on the environment in which a requirement must hold and the guaranteed/required behavior. For the implementation of a component we decided to use Matlab Simulink / Stateflow models and for the creation of the link elements we used an inhouse RTP link tool.

The first step was the identification of the elements of the internal engineering models which are part of the scenario (Figure 3, step 1). From DOORS we need to be able to access the tables / modules of the database which store the individual requirements and their additional information in different columns. In addition, DOORS supports tracing of requirements and therefore the satisfy relationship between requirements and components is the second model element. EAST-ADL is a very rich model. Therefore, within the scenario it is important to limit the scope of exposed elements. Only the function types defining a component hierarchy are relevant for the scenario to be able to link requirements and implementations to individual (sub-)components. Since the implementation shall be linked as basically a black box object, the work item is only the full model itself without the detailed behavior description. The implementation relation between components and implementation models is the last work item needed for the scenario. In contrast to the satisfy links, the management of these links is not supported by any of the tools and had to be realized in a separate link repository.

The next step was to map the engineering model elements to concepts of a common meta-model (Figure 3, step 2). As suggested by the recipe, we derived our meta-model concepts from existing meta-models which were in this case HRC and the concepts from the SPES 2020 methodology. The DOORS table entries are assigned to the concept of a `Contract` which explicitly names its assumptions and its guarantees. The EAST-ADL function types are mapped to the `Component` concept and the Matlab implementation to the `Behavior` concept. Finally, both traceability links are mapped to the concepts of `Implementation Link` respectively `Satisfy Link`. After this step, a

first high level view connecting the different engineering models had been established.

Within the third step we allocated the meta-model concepts to basic elements of the different OSLC domains (Figure 3, step 3). A `Contract` is a specialization of the generic concept of a `oslc_rm:Requirement` and the `Component` is mapped to the basic concept of a `oslc_am:Resource`. For the `Behavior` we decided to also use the `oslc_am:Resource` as a basis because it is an architectural related artifact. For both trace-link concepts there is no direct counterpart in the OSLC domains and they could only be mapped to the basic OSLC core `Resource`.

Within the fourth step we refined the OSLC concepts by adding additional properties to the original OSLC concepts. In this example we created the new domain identifiers IOS_RM, IOS_AM, and IOS_TM to denote these extensions (Figure 3, step 4). Especially the basic OSLC core `Resource` type does not define any properties and had to be extended for the representation of the trace-link concepts.

The last step was the implementation of the newly refined IOS concepts. For this we implemented server application using the Jena and Jetty libraries (as suggested in Section 4.6). Figure 3 illustrates in step 5 how the final RDF for data interchange looks like.

To further evaluate the benefits of the recipe approach, we modified the scenario by replacing one of the integrated tools. We replaced the requirement and link storage in DOORS with a solution based on MS Excel sheets. In this case, we only had to identify and map the relevant structures of the Excel sheets to the meta-model artifacts (step 1 and step 2) and could reuse the rest of our implementation. Within the MBAT project we extended the presented scenario iteratively to support definitions of faults, failures, verification / validation activities, as well as their results. We realized the extended scenario in an RTP by a further application of the recipe and by implementing OSLC-compliant consumers and providers.

6 CONCLUSIONS

In this document we described a recipe on how to systematically connect tools with an OSLC-based interoperability specification to support engineering workflows. The recipe is a guideline for developers for which it eases tool integration and enables traceability required by standards. With the recipe a common meta-model is constructed which establishes a homogeneous view on the semantic concepts of the tools and therefore allows common understanding of exchanged engineering information at the level of granularity required for the workflows. The evaluation showed an easy integration of tools for a typical engineering workflow. It revealed reuse capabilities coming with the separation of tool-specific engineering models and meta-model concepts that support such a workflow. Tools can be replaced with limited effort by other tools whose engineering models can be mapped to the same meta-model concepts. It is also possible to extend a workflow by adding meta-model concepts which only leads to minor modifications of existing tool adapters. Since our evaluation was purely manual, future work could address automatization of the recipe's steps e.g. derivation of needed meta-model concepts based on ontologies.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the ARTEMIS Joint Undertaking within the European project MBAT under grant agreement n°269335 and from the German Federal Ministry of Education and Research (BMBF) under grant number 01IS11003L. The responsibility for the content of this publication lies with the authors.

REFERENCES

Baumgart, A., Ellen, C., Oertel, M., Rehkop, P., Farfeleder, S., and Schulz, S. (2012). A reference technology platform with common interfaces for distributed heterogeneous data. In *Proceedings of the of the Embedded World 2012 Exhibition and Conference*.

Brickley, D. and Miller, L. (2010). FOAF Vocabulary Specification. http://xmlns.com/foaf/spec/. Version 0.98.

Dublin Core Metadata Initiative (2012). DCMI Metadata Terms. http://dublincore.org.

EAST-ADL Association (2013). *EAST-ADL Domain Model Specification*. Version V2.1.11.

ISO (2011). *Road Vehicles - Functional Safety*. International Standard Organization. ISO 26262.

MBAT Project (2013). Combined Model-based Analysis and Testing of Embedded Systems . http://www.mbat-artemis.eu/. ARTEMIS Call 2010 269335, 2011-2014.

OMG (2010). *Systems Modeling Language (OMG SysML TM)*. Object Management Group. Version 1.3.

OSLC Community (2013). Open Services for Lifecycle Collaboration. http://open-services.net/.

Pohl, K., Hönninger, H., Achatz, R., and Broy, M. (2012). *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer. ISBN 978-3642346132.

Rajan, A. and Wahl, T., editors (2013). *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*. Springer. ISBN 978-3709113868.

SPEEDS Project (2009). *SPEEDS L-1 Meta-Model*. SPEEDS WP.2.1 Deliverable D.2.1.5, Revision 1.0.1.

Vasaiely, P., Keis, A., and Ersch, R. (2012). Towards an European Reference Technology Platform for the development of safety relevant embedded systems: The CESAR Interoperability Specification. In *Proceedings of the Embedded World 2012 Exhibition and Conference*.

W3C (2004). RDF Vocabulary Description Language 1.0: RDF Schema. http://w3.org/TR/rdf-schema/. W3C Recommendation.

Zhang, W., Møller-Pedersen, B., and Biehl, M. (2012). A light-weight tool integration approach : From a tool integration model to oslc integration services. In *ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends*, pages 137–146.