OSLC Tool Integration and Systems Engineering - The Relationship Between The Two Worlds

Mehrdad Saadatmand*†, Alessio Bucaioni*
†Alten AB, Sweden, <firstname.lastname>@alten.se
*Mälardalen University, Västerås, Sweden, <firstname.lastname>@mdh.se

***Abstract*—OSLC serves a new standard for the integration of tools used in different phases of software development. It enables to establish relationships among different data artifacts throughout the life cycle of an application. OSLC aims to provide seamless integration of life cycle management tools and it enables to have explicit relationships among data artifacts from the early development phases, i.e., requirements. This helps to gain a better holistic view over the development of software as a system development activity. Systems engineering is in essence an interdisciplinary approach to understand, design, and manage the complexity of different projects and phenomena throughout their life cycle. In this context, to have a holistic view of the system is not a desirable, but a fundamental prerequisite. In this work, we i) investigate how OSLC can strengthen a systemic view in tool integration scenarios and ii) discuss also how systems engineering concepts and principles can be relevant to describe such scenarios. This is done by identifying the relationships among systems engineering and OSLC key concepts. Finally, we show, as a proof of concept, a concrete application of OSLC in building an integrated tool chain.**

***Keywords*—OSLC, systems engineering, systems thinking, tool integration, life cycle.**

I. Introduction

The development of a software product requires different sets of tools for different development phases. Typical examples are requirements management tools, modeling tools, bug tracking tools, test suites, etc. Each of these tools manages a different set of resources, which have relationships with other resources inside or outside the set. Trace links between requirements and test cases are a well-known example of such relationships as they help to create development tool chains. Although the ability to establish and maintain relationships among different development tool artifacts is of great importance, tools, within a chain or integration pattern, can be replaced over time due to cost or to new features availability. Consequently, the ability to cope with changes in tool chains is crucial for avoiding breakages in the chains.

Open Services for Life-cycle Collaboration (OSLC)[1] [1], as a new standard in tool integration, provides a set of specifications for different integration scenarios throughout the life cycle of a software product. Within OSLC-based tool chains, modifications are effortless as long as the tools conform to and implement OSLC specifications. OSLC enables a form of *togetherness* by establishing relationships among development artifacts and by providing semantics for these artifacts and their relationships. Togetherness, is a systems engineering concept in describing complex systems. Therefore, OSLC seems to incorporate, bring along, and apply systems engineering concepts, though implicitly and partially, to tool integration solutions.

In recent years, the International Council on Systems Engineering (INCOSE) has recognized the importance of tool integration: INCOSE has formed the Tool Integration and Interoperability Working Group (TIIWG) for addressing the needs of having integrated tools and environments to foster improved productivity and quality of systems engineering [2].

Our main intention by this work is towards the other direction: we want to bring systems engineering concepts and principles to tool integration scenarios, especially to OSLC-based integration solutions. We believe this is crucial for the research on the interdisciplinary aspects of software engineering [3] as computer systems are becoming key actors in our daily life.

In this paper, we investigate the relationships between the OSLC and systems engineering worlds, identifying the areas where OSLC highlights, applies and strengthens systems engineering concepts and principles. In addition, we point out whether a certain systems engineering concept and method can be applied to tool integration or not. The main contributions of this paper can be summarized in the following points:

- we describe OSLC which is proposed as a new standard for tool integration,
- we show an OSLC application in a research project to solve tool chain instantiation and tool integration challenges,
- we investigate on the relationship between systems engineering and OSLC, highlighting how OSLC can help in applying systems engineering principles and concepts to tool integration scenarios.

The outcome of this study is targeted and expected to be ultimately used for a better design and analysis of tool integration scenarios, by improving their systems engineering aspects as well as their related specifications and standards.

The remainder of the paper is structured as follows: in Section II we provide some background information about OSLC and systems engineering; furthermore we introduce an OSLC case study that has been initiated as part of another work of ours [4]. Section III describes the methodology we

[1]Open Services for Lifecycle Collaboration is an open community building practical specifications for integrating software. In this paper, when talking about OSLC as a standard or approach, we basically mean OSLC specifications and integration methods defined based on them.

have devised to investigate the relationships among OSLC and systems engineering concepts. In Section IV we apply the aforesaid methodology and we perform our investigation going through several key concepts of OSLC and systems engineering to evaluate the existence of any relationship among them. Finally, Section V provides a summary of our findings and draws conclusions along with future directions for this work.

II. Background and Preliminaries

A. Systems engineering

Systems engineering is an interdisciplinary field of engineering devoted to the management of complex systems throughout their whole life cycle. While there exist different definitions for systems engineering, the International Council on Systems Engineering (INCOSE) [2] provides the following definitions for system and for systems engineering, respectively:

System. According to Rechtin [5], a system is a construct or collection of different elements, which produces results not obtainable by the single elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; thus, all the things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behavior and performance.

Systems Engineering. An engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle.

From the [6]'s perspective, the science of complex systems can be seen as the combination of two macro activities streams: *thinking* and *acting* in terms of systems. According to [6], thinking and acting in terms of systems is a "prerequisite to being able to structure and operate organizations and their enterprises so that they can (pro-) actively pursue their purpose, goals and missions". The thinking part can be defined as the set of all those activities which aims to gain a holistic understanding of complex systems by observing the dynamic behavior of the systems in action. The acting part involves the creation and application of engineering structures for the observed systems.

Systems engineering is, in essence, based on systems thinking which is a unique perspective on reality with respect to wholes and parts, and the relationships among the parts of those wholes [7]. Systems thinking has been evolving since 1920 with cross-disciplines contributions, e.g., biology, engineer, computer science, etc., which have strengthened this discipline. Nowadays, systems thinking is considered to be a successful approach for gaining a holistic understanding of a system as well as an effective means for problem solving. According to [6] there are, at least, four fundamental properties which should be considered when analyzing a system:

- *togetherness*: related elements resulting in a new whole,
- *structure*: elements and their static properties,
- *behavior*: effects produced by the elements and their relationships, and
- *emergence*: predictable or not predictable result of a system in action.

In [6], the author summarizes those properties, and their relationship, in the *System-Coupling Diagram* depicted in Figure 1. There is a *situation system*, i.e. a problem or an opportunity to be solved or exploited, for which, by means of the available *system Assets*, a *respondent system* is built. The respondent system, which in turn may be a togetherness of several sub-systems, has its own structure and behavior, which result in its own specific emergence (i.e., emergent properties and behaviors). Depending on the goodness of its emergent behavior, the respondent system can be further modified to better cope the situation system.

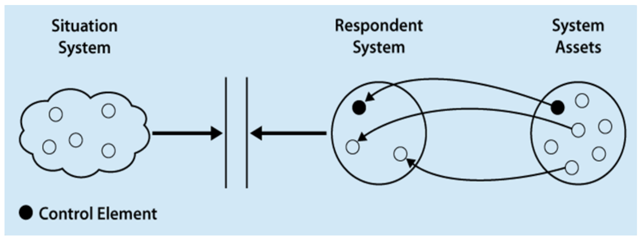


Fig. 1: System-Coupling Diagaram

By tackling the aforesaid activities streams separation, it can be argued that understanding the situation system and selecting the system assets are activities mainly related to the thinking stream, while setting up the system assets and building the respondent systems are activities mainly related to the acting stream.

B. OSLC

Usually, throughout the life cycle of a software product, different tools from different vendors are used, where each of these tools addresses specific activities, e.g., requirement management, test management, bug tracking, etc..

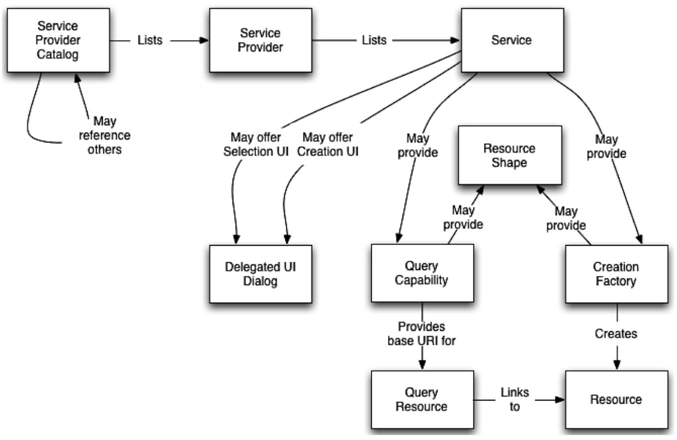


Fig. 2: OSLC Core concepts and relationships [1]

Two of the major challenges when using different tools are the integration of the involved tools as well as the traceability among the different data artifacts used and managed by the tools. The importance of such challenges is even emphasized when considering that companies often replace tools, within their development process, due to cost, better features, support, or because one tool vendor goes bankrupt and out of the market.

OSLC is a new standard, supported by an open community, which aims to alleviate these issues offering specifications for tool integration [1]. OSLC is supported by several companies and organizations such as IBM, Siemens, SHELL, NASA Jet Propulsion Laboratory, and General Motors, to name a few. OSLC community is organized in the form of different work groups focusing on different life cycle integration scenarios.These scenarios, for which specifications are produced, are referred to as *domains* [1], [8], e.g., Requirements Management domain (RM), Architecture Management domain (AM), Change Management domain (CM), Quality Management domain (QM), etc. The specification for each domain is built upon one core specification, namely OSLC Core, which defines the basic concepts and rules and ensures consistency among different domain specifications. Figure 2 shows the core concepts and their relationships as they are defined in the OSLC Core specification.

OSLC tries to be minimal by providing a small set of resources and properties for describing artifacts involved in a specific domain. Figure 3 shows the OSLC core common properties used for the *Test Case* resource definition contained in the QM[2]. Accordingly, one or many testers can be defined as *creators* of a specific test case; each test case is specified by means of its *identifier*, which can be any arbitrary String value; date of creation and date of modification can be specified too using the *created* and *modified* properties. Further properties can be defined if needed, although OSLC deprecates this practice for standardization purposes.

OSLC is based on the concept of *Linked Data* [9], *Resource Description Framework* (RDF) [10], and *HTTP protocol*. Figure 4 shows the application of these concepts and their relationship within OSLC: each artifact is described as an HTTP resource, identified by means of a *Uniform Resource Identifier* (URI), accessed and manipulated with the GET, PUT, POST and DELETE HTTP methods. Considering resource provision and retrieval concepts in an integration scenario, a tool can play two roles, namely *consumer* and the *provider*. That is, for exchanging an artifact, a provider has to encode an artifact's properties in an HTTP resource, and post and provide it under a specific URI; knowing the retrieving URI, a consumer can access the HTTP resource representing the artifact, and simply fetch it from the URI itself. It is important to note that a tool can also be, at the same time, the provider and consumer of different resources.

C. Case study

In our previous work [4] we show how to extend a verification tool for EAST-ADL [11] models, namely ViTAL [12], with automatic test-case generation for functional requirements. The work was developed within the European Combined Model-based Analysis and Testing of Embedded

[2]The complete explanation of the aforesaid properties is outside the scope of the paper.

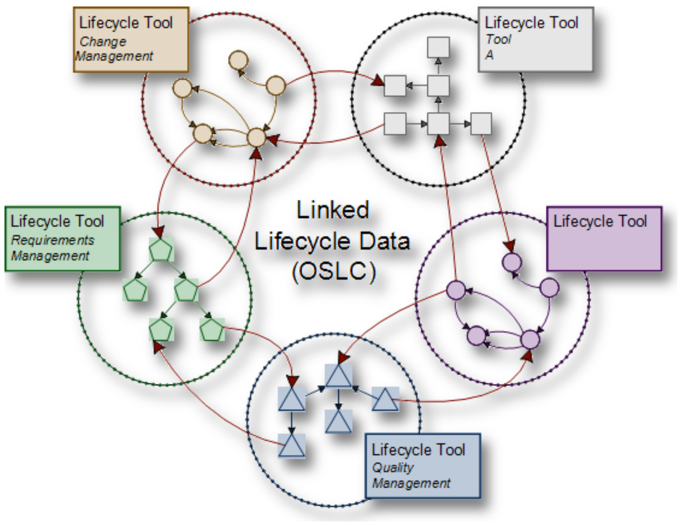


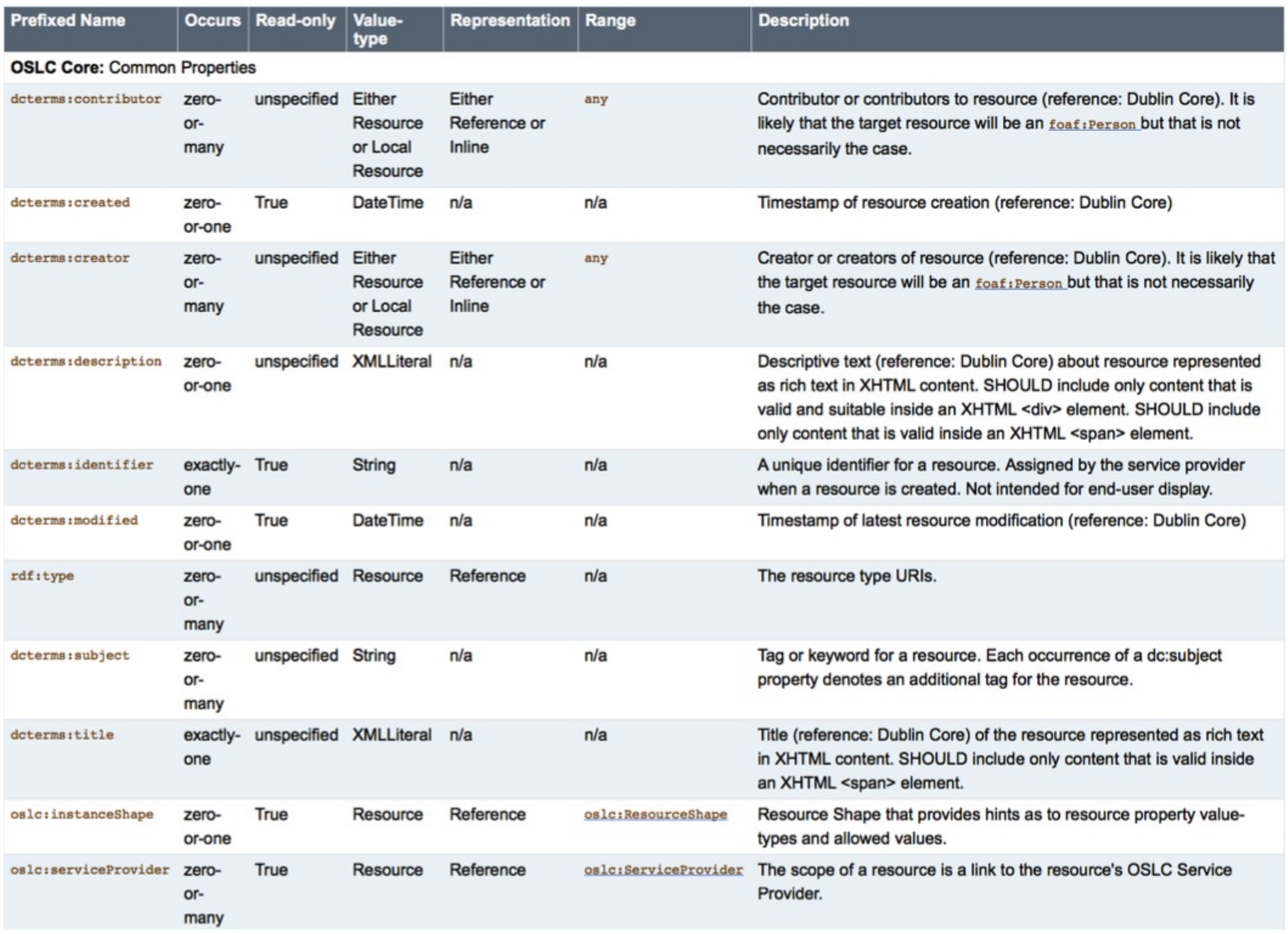
Fig. 4: Linked Data in OSLC [1]

Systems (MBAT) project [13] for showing a new and promising Verification and Validation (V&V) approach, which makes V&V activities more effective and efficient for industrial needs by exploiting the synergy between model-based analysis and (dynamic) testing. To achieve such synergic scenarios, different tools used in the development life cycle need to tightly collaborate, be integrated, and also have a semantic 'agreement' on the artifacts that are exchanged between them. This should also be done in a way that different tool chains can be instantiated and thus the integration is not limited and 'locked' to particular tools. This is where OSLC comes into help.

As a proof of concept for the MBAT methodology, the ViTAL tool which can perform model-based analysis and also generate test-cases is integrated with Farkle [14] test execution framework, as part of a tool chain to: i) derive automatically Python executable test scripts from the ViTAL generated test-cases and ii) execute them using Farkle. The interoperability between ViTAL and Farkle is achieved by means of an OSLC interface. In a similar fashion, ViTAL is also actually integrated with a requirement repository tool using OSLC. The complete methodology is applied to the Brake-by-Wire industrial use-case, provided by Volvo.

1) ViTAL: ViTAL is a verification tool for EAST-ADL models, providing capabilities for simulation, verification and test-case generation. It is an Eclipse-based environment, consisting of the following plug-ins: (i) two editor plug-ins for EAST-ADL system model and timed automata description of the component behavior, respectively, (ii) a plug-in for a model transformation between EAST-ADL models and UPPAAL PORT input models, and (iii) an UPPAAL PORT plug-in for model-checking of EAST-ADL models enriched with timed automata semantic.

2) Farkle Environment: Farkle is a test execution environment, which enables embedded systems to be tested in their own environment. It has been developed for testing embedded systems built using OSE [15] Real-Time Operating System. It can be also adapted to other OSes which support LINX protocol for inter-process communication, e.g. Linux.



Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
OSLC Core: Common Properties						
dcterms:contributor	zero-or-many	unspecified	Either Resource or Local Resource	Either Reference or Inline	any	Contributor or contributors to resource (reference: Dublin Core). It is likely that the target resource will be an foaf:Person but that is not necessarily the case.
dcterms:created	zero-or-one	True	DateTime	n/a	n/a	Timestamp of resource creation (reference: Dublin Core)
dcterms:creator	zero-or-many	unspecified	Either Resource or Local Resource	Either Reference or Inline	any	Creator or creators of resource (reference: Dublin Core). It is likely that the target resource will be an foaf:Person but that is not necessarily the case.
dcterms:description	zero-or-one	unspecified	XMLLiteral	n/a	n/a	Descriptive text (reference: Dublin Core) about resource represented as rich text in XHTML content. SHOULD include only content that is valid and suitable inside an XHTML <div> element. SHOULD include only content that is valid inside an XHTML <span> element.
dcterms:identifier	exactly-one	True	String	n/a	n/a	A unique identifier for a resource. Assigned by the service provider when a resource is created. Not intended for end-user display.
dcterms:modified	zero-or-one	True	DateTime	n/a	n/a	Timestamp of latest resource modification (reference: Dublin Core)
rdf:type	zero-or-many	unspecified	Resource	Reference	n/a	The resource type URIs.
dcterms:subject	zero-or-many	unspecified	String	n/a	n/a	Tag or keyword for a resource. Each occurrence of a dc:subject property denotes an additional tag for the resource.
dcterms:title	exactly-one	unspecified	XMLLiteral	n/a	n/a	Title (reference: Dublin Core) of the resource represented as rich text in XHTML content. SHOULD include only content that is valid inside an XHTML <span> element.
oslc:instanceShape	zero-or-one	True	Resource	Reference	oslc:ResourceShape	Resource Shape that provides hints as to resource property value-types and allowed values.
oslc:serviceProvider	zero-or-many	True	Resource	Reference	oslc:ServiceProvider	The scope of a resource is a link to the resource's OSLC Service Provider.

Fig. 3: OSLC Core common properties for the Test Case resource

In OSE, tasks synchronization and programming are based on the concept of direct and asynchronous message passing. In OSE, the runnable tasks are called *processes* and the messages exchanged among processes are called *signals*. The Python scripts are used to send and receive signals as well as to decide the test-case verdict, i.e., pass or fail.

3) Work-flow and OSLC interface: Figure 5 depicts the work flow used in [4], consisting of seven major steps: i) at first, the system designer creates EAST-ADL and timed automata models, ii) timed automata models are used to describe the internal behavior of system components modeled in EAST-ADL, and are automatically related with EAST-ADL system models by means of a model-to-model transformation; iii) UPPAAL-PORT generates test-cases by applying model checking on timed automata models of EAST-ADL components; iv) the generated test cases are exchanged through the OSLC interface for creating executable test cases, which are v) enriched with some code annotations, vi) converted in test scripts and vii) executed [3].

We hereby focus on the OSLC interface and its realization, corresponding to step 4 of the Figure 5. Within OSLC, integration is reached by i) implementing consumer and provider and by ii) choosing the resource to be exchanged among those as provided within the OSLC domains. Considering the flow of our case study, we have implemented a provider on top of ViTAL and a consumer on top of Farkle. Furthermore, in our particular implementation, we have defined a *Test Case* resource based on the definition of test case resource from the OSLC QM specification, to represent an *Abstract Test Case* which is exchanged between ViTAL and Farkle (other ways of implementation are also possible).

The implementation has been carried out using Eclipse Lyo[4], a software development kit which supports developers in building OSLC-compliant tools. OSLC consumers and providers are composed by two Eclipse projects each: one project contains the implementation of the exchanged OSLC resource, while the other project contains the implementation of the methods realizing the exchanging operations. While the projects containing the OSLC resource implementation have to be identical, i.e., they have to implement the same OSLC resource with the same set of properties, the projects implementing the exchanging operations are different and they implement different methods for consumers or providers.

[3]The detailed explanation of the methodology is outside the scope of this paper; the interested reader can refer to [4].

[4]http://www.eclipse.org/lyo

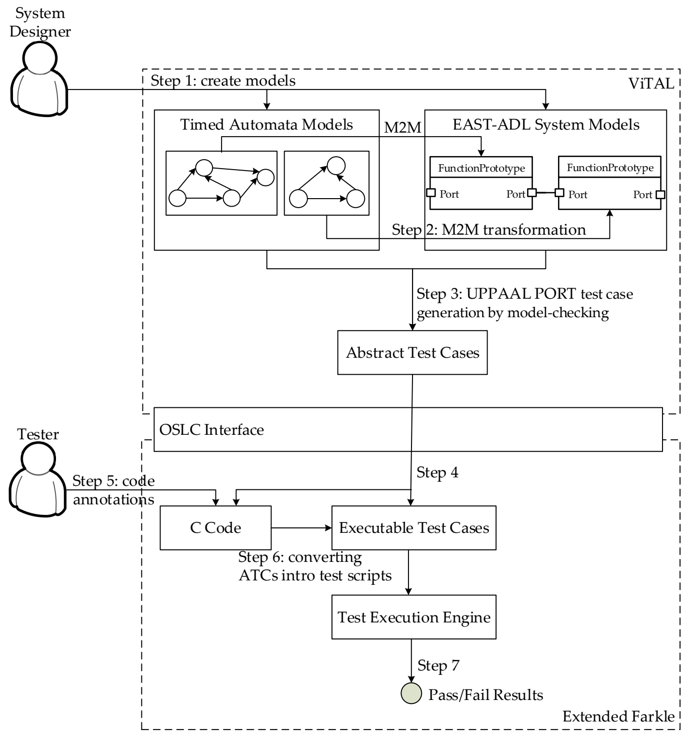


Fig. 5: ViTAL-Farkle integrated tool chain - workflow [4]

```
 1 @OslcResourceShape(title = "Quality Management Resource
       Shape", describes = Constants.TYPE_TEST_CASE)
 2 @OslcNamespace(Constants.MDH_QM_NAMESPACE)
 3 public final class TestCase extends AbstractResource {
 4   private String description, identifier;
 5   @OslcPropertyDefinition(OslcConstants.DCTERMS_NAMESPACE +
         "description")
 6   @OslcTitle("Description")
 7   @OslcValueType(ValueType.XMLLiteral)
 8   public String getDescription() {return description;}
 9   public void setDescription(final String description) {this
         .description = description;}
10 }
```

Listing 1: OSLC implementation of the TestCase resource

Listing 1 shows an excerpt of the Java implementation of our abstract test case resource. It is defined as a specialization of the OSLC `AbstractResource` (line 3). The `TestCase` resource definition comprises a set of local variables along with the respective getters and setters. The set of local variables, as well as the Java annotations (e.g., line 6), are standard and specified in the QM domain definitions. The Java annotations realize the mapping between the Java object (representing the resource) and the abstract test case resource properties.

```
 1 @OslcService(Constants.QUALITY_MANAGEMENT_DOMAIN)
 2 @Path("testCases")
 3 public class TestCaseService {
 4   @GET
 5   public TestCase[] getTestCases(@QueryParam("oslc.where")
         final String where) {
 6     final List<TestCase> results = new ArrayList<TestCase>()
         ;
 7     final TestCase[] testCases = Retriever.getTestCases();
 8     for (TestCase testCase :testCases){results.add(testCase)
         ;}
 9     return results.toArray(new TestCase[results.size()]);
10   }
11 }
```

Listing 2: OSLC ATC provider

Listing 2 presents an excerpt of the ViTAL provider implementation. Once the function `Retriever.getTestCases()` has set-up an OSLC resource with the needed ViTAL information, the resource is put online and it can be accessed via HTTP methods. The method annotation `@GET` (line 4,5) specifies the method to be invoked each time the provider receives an HTTP GET from any consumer.

```
1 final OslcRestClient oslcRestClient = new OslcRestClient(
      providers,queryBase, MEDIA_TYPE, timeout);
2 final TestCase[] enoviaSystemDefinitions = oslcRestClient.
      getOslcResources(TestCase[].class);
```

Listing 3: OSLC ATC consumer

Listing 3 shows part of the Farkle consumer implementation. In response to the authentication to the given URI (line 1), the consumer receives back an `OslcRestClient` object, which can be queried (line 2) for retrieving all the test-case resources posted by the provider under the specified URI.

Such a tool integration based on OSLC not only brings more clear and precise semantics on the data resources that are exchanged among the tools, but also makes replacement of tools easier (as long as they still conform to the OSLC specification).

III. Methodology

So far, we have introduced OSLC, together with an example, and discussed how it adds more semantics to the data involved in integration scenarios; and how it contributes to strengthening the traceability among data artifacts. We have also explained how such semantics enables the addition or the replacement of one or more tools in the tool chain: any tool can be added in the tool chain just knowing which type of resources it is expected to consume/provide and representing them using OSLC resource definitions.

In the rest of the paper, we discuss how OSLC integration solutions can be regarded from a systems engineering perspective. In order to address the abstract nature of this work and show how OSLC can highlight and strengthen systems aspects in the development of software products, we perform our investigation following this three-steps methodology.:

1) Collect a list of related keywords and concepts for defining a system and its characteristics.
2) Identify the relationship between the collected keywords and concepts and OSLC.
3) Discuss if and how OSLC contributes to thinking and acting in terms of systems.

The first activity is performed using [6] as the reference literature to identify and extract systems engineering keywords and concepts. The main goal in this step is to collect the concepts that help to define and identify a system and its characteristics, and how a systemic view can be usefully applied in practice. The reason for this choice is that (while considering that here we do not intend to provide a survey on systems engineering concepts) [6] goes beyond just pure systems engineering concepts, offering the dual perspective of thinking and acting in terms of systems. It also discusses methods and tools that are used for applying a systemic view to different problems. Therefore, the collected keywords and concepts in this work will be more than just pure and strict systems engineering concepts.

The second step is carried out by consulting the OSLC specifications and domain definitions. In this step, the relationships between the collected system concepts and OSLC are discussed and assessed. This is done by identifying concepts in the OSLC specification which can contribute to each of the collected system concepts. The degree of the contribution is specified by the following marks:

- **F(Fully):** indicates that either there is (at least) an OSLC concept which fully covers and can be mapped to the system concept, or the system concept/method can be fully applied to and describe an integration scenario.
- **P(Partially):** indicates that either there is (at least) an OSLC concept which partially covers and can be mapped to the system concept to some extent but not fully, or the system concept/method can be partially and to some degree applied to and describe an integration scenario.
- **I(Implicitly):** indicates existence of either a concept in OSLC which does not directly cover the system concept, but can implicitly and indirectly contribute to it, or a system concept/method which may in some cases and with some adaptations be useful in integration scenarios.
- **N(None):** either there is no OSLC concept that can cover the system concept, or the system concept/method is totally inapplicable to tool integration scenarios.

Eventually, based on the coverage results achieved by performing the aforesaid activities, in Step 3 we discuss and evaluate the overall relationship between OSLC and system concepts.

IV. OSLC and Systems concepts

In this section, the collected set of keywords and important concepts in defining a system and its characteristics are discussed along with the OSLC counterpart concepts. The OSLC coverage level for the systems concepts is specified using the terms and notations introduced in the previous section. We start with the four main characteristics of systems that were introduced earlier in Section II: togetherness, structure, behavior, and emergence.

Togetherness: In OSLC, a relationship among data artifacts can be established enabling traceability, through the properties of the resources involved. Although this relationship does not necessarily and explicitly result in a new whole, the so reached integration of tools can result in an integrated tool chain, which in turn may be considered as a new whole. Therefore, the coverage level of the togetherness concept by OSLC can be marked as (***I***).

Structure: In OSLC, each exchanged resource has to adhere to a resource specification. This definition explicitly defines the exchanged data artifacts and their relationships. Therefore, the coverage level of this concept is (***F***).

Behavior: OSLC in its nature is static and does not address dynamic behaviors of data artifacts or tools (***N***).

Emergence: Considering that OSLC does not deal with behavioral aspects of tools and systems, this concept is not covered by OSLC (**N**).

System Topology: A system can have a topology defining the structure of its elements and their relationships, e.g., a hierarchy or network topology. While OSLC does not impose or introduce any specific topology, the relationships among data artifacts, created using OSLC, will result in a structured chain of integrated tools. Hence, OSLC indirectly contributes in forming a topology (**I**).

Different Views: A system can usually be considered from multiple views [16]. Although the concept of views does not define a system, it remains as an important characteristic in systems engineering. In OSLC there is no explicit concept for views; however, different domains[5] capture specific integration aspects. From this perspective, the concept of view can be considered as partially covered by OSLC (**P**).

Focus: In systems engineering, different focus levels can be considered in terms of Narrow System-of-Interest (NSOI), Wider System-of-Interest (WSOI), Environment and Wider Environment [6]. On the one hand, OSLC does not help in gaining such focus levels. On the other hand, data artifacts and resources, from different OSLC domains, can be related to each other. From this perspective, for a data artifact within a specific OSLC domain, other data artifacts from other OSLC domains might be considered as an environment or a wider system. Therefore, while OSLC does not have any concept, which can be directly mapped to the focus concept as considered in [6], the contrary might be valid. In other words, different focus levels might be applied to a tool chain and to its different data artifacts, when integrated using OSLC. The concept of focus can not clearly be identified using OSLC concepts, however, OSLC seems to provide some very implicit and weak form of focus (**I**).

System assets: In [6] the author refers to system assets as all the means needed to build or manage a specified system, e.g., tools, knowledge assets. OSLC explicitly covers this concept by targeting and highlighting resources and data artifacts produced, consumed and generally managed in different tools used throughout the life-cycle of a software product (**F**).

System coupling diagram: system coupling diagram is chosen to be checked whether the same concepts that are captured by it have some corresponding concepts in OSLC. This diagram constitutes of three parts: situation system, respondent system, and system assets. Situation system basically captures the elements and relationships that contribute to a situation which thus are described in the form a system [6].

A respondent system is a system, which is created to counteract the situation and whose services are composed of available assets, forming a temporary system asset. Such respondent system cannot be explicitly mapped to any OSLC concepts. OSLC fully covers the asset concept in this three-part

[5]We refer to the OSLC definition of domains

diagram, although some types of assets, such as procedures and processes, may not be directly captured in OSLC. The replacement or the addition of a tool in an OSLC tool chain, might be considered as a situation system. This means that, the system coupling diagram may also be used to describe tool integration scenarios. Summarizing, OSLC can partially cover the concepts necessary for a system coupling diagram (**P**).

System complexity: In systems engineering there are different categories of complexity: Organized Simplicity, Disorganized and People Related [6]. OSLC does not contain any concepts to capture complexity. Nevertheless, the OSLC specification brings, in a way, some concepts to organize the complexity of data artifacts and their relationships in different tools. For this purpose, OSLC can be considered as a means to manage tool integration complexity issues, implying that the tool chain and its integration can be viewed as a system (**F**).

Hard vs Soft systems: Based on the system goals, a system can be categorized as *hard* or *soft* [17]. Based on such categorization, OSLC falls into the definition of a hard system. OSLC, as a tool integration standard, does not deal with these concepts (**N**).

Models and Views: Different views, over a system, are created by using one or more models [18]. OSLC enables to view a system in terms of its data artifacts relationships. Consequently, OSLC contributes to gaining a specific systemic view over the set of tools and data involved in the integration (**F**).

Systemigram: The term systemigram was coined in a pan-European IT project from the contraction of the words *systemic* and *diagram*. The idea behind it was to replace rich-text documents with a more structured diagram, which could summarize concepts and interactions of a system [19]. Systemigrams are often used for describing system of systems. They can even be used for describing tool integration scenarios, by means of relationships among tool resources and data artifacts (**F**).

Integrated OODA & PDCA diagrams: The Plan-Do-Check-Act (PDCA) and Observe-Orient-Decide-Act (OODA) diagrams [20] were originally developed for leveraging the success of military missions and operations. However, [6] shows how systems engineering can benefit from the adoption of these diagram and their utilization in the business process[6]. OODA and PDCA diagrams are fully applicable to tool integration, especially in the integration scenarios described by OSLC, i.e., when tools replacements are easy to happen. In OSLC, there is no concept for integration activities planning; however, an integrated tool chain can be completely described by integrated OODA and PDCA diagrams, as suggested below. PDCA can be used when the set of tools to be integrated is known; then, the integration is planned, done, checked, and eventually refined with further actions. OODA seems suitable when an issue in the integrated tool chain is observed, e.g., a tool vendor goes out of a market, another tool is introduced, a tool gets updated, etc. In such cases, based on the observed issue, some orientation in the tool chain may be done, appropriate decisions may be made and, finally, actions may be performed. Summarizing, there is no one single OSLC concept, which directly maps the concepts presented by these diagrams. However, the flexibility that OSLC provides for integrated tool chains, in terms of replacing tools, can be captured by them. Generally, it seems that these diagrams serve very well to describe tool integration scenarios (**I**).

[6]For the sake of space we avoid the complete explanation of these diagrams. The interested reader can find more details in the provided references.

System life cycle: OSLC's main goal is to address integration scenarios among tools used in different stages of a software product life cycle. This way, OSLC draws further attention to different stages that exist in the life cycle of a software product (**F**).

System of systems: It is usually possible to consider different sub-systems for a system. In other words, systems engineering principles can be well applied to different groups of elements and their relationships that constitute a system. While OSLC does not provide any concept for representing a system and its sub-systems explicitly, the whole integrated tool chain can be considered as a system to which systems engineering concepts can be applied. This way, different tools that can be mapped to different OSLC domains, can be considered as sub-systems providing specific functionality and thus contributing to forming the higher level system, which in this case is the tool chain. Different resources in each tool in the tool chain can also form sub-sub-systems. From this perspective again, systems engineering principles seem to be well appropriate for describing a tool chain. Also, by providing semantics to different tools, their internal resources and their relationships, OSLC can be considered as implicitly contributing to the application of systems engineering concepts in tool integration scenarios (**I**).

System description and instances: As aforesaid, systems are usually regarded from different viewpoints, where each viewpoint makes use of a specific notation for describing the systems themselves. The artifact, or the set of artifacts, produced by applying these descriptions, are considered to be instances of the system and the system description. OSLC specification can be considered as a system description and each OSLC-based integrated tool chain as an instance of it. However, in the tool chain, only a subset of OSLC concepts is usually used; meaning not having all the characteristics as defined in the systems description. On the other hand, each tool chain may be considered as a system description while an instance of it is a set of specific tools integrated using a set of OSLC concepts. This is possible in OSLC as it allows a replacement of a tool with another one as long as it conforms to the same OSLC concepts. Therefore, OSLC enables to apply the concepts of system description and its instances from systems engineering to tool integration (**F**).

Change management: Change management is usually defined as the set of activities which aims to maximize productivity by minimizing mistakes produced by a poor coordination along the system development [21]. OSLC has a dedicated domain specification for Change Management (CM), which is intended for "the management of product change requests, activities, tasks and relationships among those and related resources such as project, category, release and plan" [1]. This specification includes definition of *ChangeRequest* resource, which is defined to capture different types of change requests such as defect, enhancement, task, bug, and activity, to name a few. Therefore, OSLC explicitly addresses the topic of change management - to a degree relevant for tool integration (**F**).

Life cycle changes, traceability, and versions of systems: Traceability in software engineering is defined as the ability to trace dependencies among several artifacts or versions of the same artifact [22]. OSLC can establish trace links among different resources maintained by various tools in a tool chain. A change with respect to a specific tool (i.e., tool replacement, introduction of a new tool, etc.) can be easily handled by OSLC by ensuring conformance to the OSLC specifications. This is essentially is one of the core goals of OSLC. Different versions of a resource are managed in OSLC using the Change Management (CM) specification as discussed above. Therefore, OSLC is capable to cope with life-cycle changes. But as long as different configurations of a tool chain are concerned (as different versions of a system), it does not capture that. In other words, some meta-concepts capturing the properties of a tool chain as a whole might be necessary for that purpose (**P**).

Execution order in the life cycle: OSLC does not provide any concept for specification of control flow in the tool chain. Therefore, execution order cannot be described using OSLC (**N**).

Life cycle roles (conceiver, developers, producers,...): Ideally, associated to each view there is one or more roles. In OSLC, there is no specific list defining the different roles that can exist in the life cycle of a (software) product. However, using the *dcterms:creator* and *dcterms:contributor* properties of resources, it is possible to include the necessary information. In this case, it is usually the name of persons than their roles (**P**).

System lifetime: Another characteristic of a system is its lifetime. The type of management and leadership needed for systems with short lifetime is different from those which have a long lifetime [6]. In OSLC there is no concept to capture this characteristic for the tool chain, the tools and their internal resources as systems. A tool chain in itself could have been formed in response to a short-time need or a situation with long longevity. Therefore, OSLC does not prevent applying the concept of lifetime for a tool chain, although it does not care about it either (**N**).

Table I provides a summary of the discussions above regarding the relationship between systems engineering principles and OSLC, and how OSLC concepts provide for and cover those principles.

One of the important findings from the above discussion is that system engineering concepts and principles seem very much suitable to describe tool integration scenarios. Figure 6 shows a sample systemigram for the case study introduced in Section II as a very simple example for such application of systems engineering concepts and methods.

V. Summary and Conclusion

In this article, we discussed OSLC, systems engineering and the relationships that can exist between these two worlds. We also presented an OSLC case study. The motivation behind this work is based on the observation that OSLC, as a new specification for tool integration, aims to categorize and capture different core artifacts that exist throughout the life cycle of a software product, as well as their relationships. This is completely aligned with the definition of a system which, in simplest terms, is a meaningful/purposeful collection of elements and the relationships among them. We have investigated how OSLC and systems engineering concepts can be mapped and inter-related, with the ultimate goal of promoting the application of systems engineering concepts and principles to tool integration and their adoption in developing

Systems Concept/Method	OSLC Coverage
Togetherness	I
Structure	F
Behavior	N
Emergence	N
System Topology	I
Different Views	P
Focus	I
System Assets	F
System Coupling Diagram	P
System Complexity	F
Hard vs Soft Systems	N
Models and Views	F
Systemigram	F
Integrated OODA & PDCA diagrams	I
System Life cycle	F
System of Systems	I
System description and instances	F
Change Management	F
Life cycle changes, traceability, and versions	P
Execution order in the life cycle	N
Life cycle roles	P
System Lifetime	N

TABLE I: Summary of Systems concepts/methods and OSLC concepts coverage

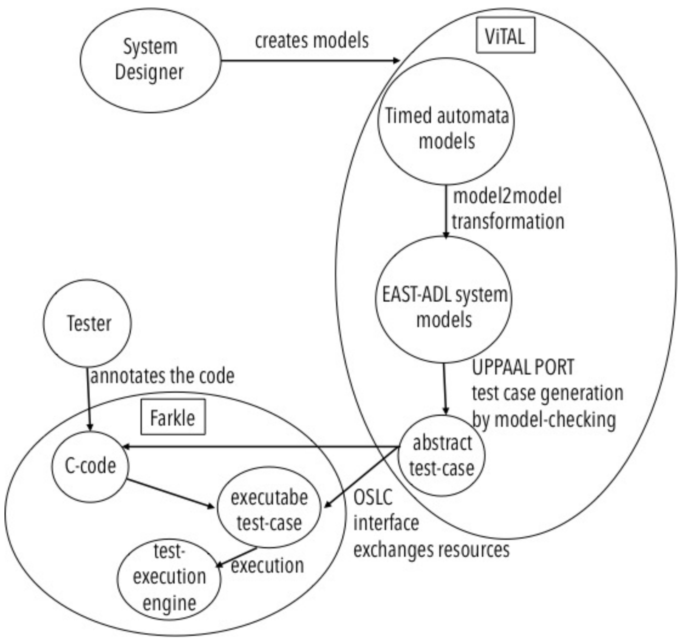


Fig. 6: Systemigram for ViTAL-Farkle integration scenario

tool integration specifications.

One of the important findings of this work is recognizing that systems engineering concepts and principles can be well applied to tool integration scenarios especially when considering a chain of integrated tools as a system. This is even much stronger in case of OSLC: it adds semantics and provides flexibility for tools replacement; hence, it covers more explicitly principles such as change management, version management, etc.

To cope with the abstract nature of this work, meaning that it mainly concerns concepts and principles, we have defined a methodology for performing our investigation. As part of this methodology we have selected some key concepts and keywords for defining and identifying a system and its characteristics. It might be argued that the selected set of keywords might, somehow, affect the results; that is: if another set of keywords had been selected, the result would have been different. While this can be a valid concern, the important finding, in our opinion, is that a different set of keywords would reveal the inter-relationship between OSLC and systems engineering from a different perspective, not violating the set of keywords we have discussed in this work. Therefore, it can even be considered as a future work to investigate further systems engineering concepts.

This work was done mainly considering the OSLC specification available at writing time of the article. Since OSLC specification is constantly evolving, future versions might contain more systemic aspects. OSLC also has its specific scope and limitations. For instance, the current version of OSLC specification is mainly data-centric and does not support control flow integration [23]. Similarly, as investigated in [23], transaction support and the concepts of atomicity, data integrity and roll-back are not in the scope of OSLC (at least in its current version) which might be of importance in industrial solutions based on OSLC to avoid data inconsistency. Moreover, while OSLC mainly offers a data-centric software integration solution and does not handle hardware (of course, custom extensions in OSLC are possible), other aspects beyond just software, such as hardware as well as human activities are also of concern in systems engineering. Concluding, based on our findings, there is a synergy between these two fields and we can strongly recommend to the OSLC community to consider systems engineering concepts and principles when developing OSLC specifications for tool integration scenarios.

VI. Acknowledgments

This work has been supported by the MBAT European Project and Alten AB through the ITS-EASY industrial research school [24]. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no 269335 (see Article II.9. of the JU Grant Agreement) and from the Swedish Governmental Agency for Innovation Systems (VINNOVA). We would like to thank Prof. Harold "Bud" Lawson for his generous advice on systems engineering concepts.

References

[1] Open Services for Lifecycle Collaboration, http://www.http://open-services.net//, Accessed: February 2014.

[2] The International Council on Systems Engineering (INCOSE), http://www.incose.org/, Accessed: February 2014.

[3] J. D. Herbsleb, "Beyond computer science," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 23–27.

[4] R. Marinescu, M. Saadatmand, C. Seceleanu, P. Pettersson, and A. Bucaioni, "EAST-ADL Tailored Testing: From System Models to Executable Test Cases," Tech. Rep. ISSN 1404-3041 ISRN MDH-MRTC-278/2013-1-SE, September 2013. [Online]. Available: http://www.es.mdh.se/publications/3014-

[5] E. Rechtin and M. W. Maier, *The Art of Systems Architecting*, 2nd ed., ser. Systems Engineering. CRC Press, 2000.

[6] H. W. Lawson, *A Journey Through the Systems Landscape*. College Publications, 2010.

[7] The International Council on Systems Engineering (INCOSE), "Systems engineering handbook: A guide for system life cycle processes and activities v3.2," January 2010.

[8] OSLC Primer - Online book, http://open-services.net/uploads/resources/OSLC_Primer_-_Learning_the_concepts_of_OSLC.pdf, Accessed: February 2014.

[9] Linked Data, http://www.linkeddata.org/, Accessed: February 2014.

[10] Resource Description Framework (RDF), http://www.w3.org/RDF/, Accessed: February 2014.

[11] T. A. ATESST2 Consortium, "EAST-ADL Profile Specification, 2.1 RC3 (Release Candidate)." www.atesst.org, 2010, pp. 10–75. [Online]. Available: www.atesst.org/home/liblocal/docs/

[12] E. P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson, "ViTAL: A Verification Tool for EAST-ADL Models Using UPPAAL PORT," in *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*. IEEE, 2012, pp. 328–337.

[13] MBAT Project: Combined Model-based Analysis and Testing of Embedded Systems, http://www.mbat-artemis.eu/home/, Accessed: February 2014.

[14] Alten - Stockholm, "Farkle Test Execution Framework," http://www.alten.se, Last Accessed: February 2014.

[15] Enea, "The Architectural Advantages of Enea OSE in Telecom Applications," http://www.enea.com/software/solutions/rtos/, Last Accessed: May 2013.

[16] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting software architectures: views and beyond*. Pearson Education, 2002.

[17] P. Checkland, *Systems Thinking, Systems Practice: Includes a 30-Year Retrospective*. John Wiley & Sons, 1999.

[18] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.

[19] B. Sauser, M. Mansouri, and M. Omer, "Using systemigrams in problem definition: A case study in maritime resilience for homeland security," *Journal of Homeland Security and Emergency Management*, vol. 8, no. 1, 2011.

[20] C. Richards, *Certain to win: The strategy of John Boyd, applied to business*. Xlibris Corporation, 2004.

[21] R. S. Pressman and W. S. Jawadekar, "Software engineering," *New York 1992*, 1987.

[22] M. Lindvall and I. Rus, "Knowledge management in software engineering," *IEEE software*, vol. 19, no. 3, pp. 0026–38, 2002.

[23] industrial Framework for Embedded Systems Tools (iFEST) European project, "iFEST proposal towards an adoption of OSLC," http://www.artemis-ifest.eu/node/45, Accessed: May 2014.

[24] ITS-EASY post graduate industrial research school for embedded software and systems, http://www.mrtc.mdh.se/projects/itseasy/, Accessed: February 2014.