Tool integration using OSLC

Ejaz Ul Haq

Supervisor:
Mehrdad Saadatmand

Examiner:
Mikael Sjödin

NOVEMBER, 2013
MASTER’S THESIS

MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN
ABSTRACT

In Software development process various tools are used in different phases of product lifecycle. They are either developed by any vendor or custom in-house solutions. To collaborate in development processes, these tools are required to communicate and transform data values. Different solutions like data object, data transformers and data translators are designed to communicate and transform data between tools. But as with technology evolves (new tools, APIs) it could become less viable that these integrators could address tool integration challenges and facilitate in the development processes. This leads to require a better solution. Open Service Life Cycle (OSLC) appears to overcome these issues. OSLC standardize open resources in the form of URIs to integrate them between different tools. In this thesis work, a careful study of OSLC and comparison with other tool integration mechanisms, practical implementation and the data interoperability is established between ViTAL (Framework for EAST-ADL model-verification using UPPAAL-PORT) and Farkle (Testing environment) tools using OSLC.

This OSLC tool integration solution for Quality Management domain shows that it facile the resource availability and transformations of data for targeted tools.
Table of Contents

1 Introduction ................................................................................................................................. 7
  1.1 Problem Statement .................................................................................................................. 8
  1.2 Overview .............................................................................................................................. 8
  1.3 Limitations ............................................................................................................................ 8

2 Tool Integrations ....................................................................................................................... 10
  2.1 Life Cycle Management ......................................................................................................... 10
    2.1.1 ALM .............................................................................................................................. 10
    2.1.2 PLM ............................................................................................................................. 11
    2.1.3 ALM v/s PLM ................................................................................................................ 11
  2.2 Integration ............................................................................................................................. 12
    2.2.1 Presentation Integration ................................................................................................. 12
    2.2.2 Data Integration ............................................................................................................... 12
    2.2.3 Control Integration ......................................................................................................... 12
    2.2.4 Process Integration ......................................................................................................... 12
    2.2.5 Platform Integration ...................................................................................................... 13
    2.2.6 Parameter passing .......................................................................................................... 13
      2.2.6.1 Pass by Reference .................................................................................................. 13
      2.2.6.2 Pass by copy ........................................................................................................ 14
    2.2.7 IEEE standard for tool integration .................................................................................. 14
      2.2.7.1 IEEE P1175.3 reference model for specifying software behavior ......................... 14
  2.3 History of tool integration .................................................................................................... 15
    2.3.1 Common Meta-model Interoperability ............................................................................ 17
  2.4 Why OSLC ............................................................................................................................ 19

3 OSLC .......................................................................................................................................... 20
  3.1 Introduction ............................................................................................................................ 20
    3.1.1 OSLC organisation .......................................................................................................... 21
    3.1.2 Core specifications .......................................................................................................... 22
  3.2 OSLC Implementations ......................................................................................................... 22
    3.2.1 Linked Data .................................................................................................................... 22
    3.2.2 RDF/XML ...................................................................................................................... 23
    3.2.3 Representational State Transfer (RESTful) .................................................................. 24
      3.2.3.1 Data Elements ....................................................................................................... 24
      3.2.3.2 Connectors ............................................................................................................. 24
3.2.3.3 Components ........................................................................................................ 25
3.3 OSLC Resource shape .................................................................................................. 25
3.4 OSLC Service Provider ................................................................................................ 26
3.5 OSLC SDKs .................................................................................................................. 26
   3.5.1 OSLC4NET ............................................................................................................ 26
   3.5.2 Eclipse Lyo ............................................................................................................ 27
4 Implementation ............................................................................................................... 28
   4.1 Background ............................................................................................................... 28
      4.1.1 VITAL ................................................................................................................. 28
         4.1.1.1 Persistence .................................................................................................... 29
         4.1.1.2 ServiceProviderFactory ............................................................................... 29
         4.1.1.3 ServiceProviderSingleton ............................................................................. 30
         4.1.1.4 ServletListner ............................................................................................... 30
      4.1.2 Farkle Environment ............................................................................................ 30
   4.2 Farkle OSLC consumer .............................................................................................. 31
      4.2.1 Resource Retriever ............................................................................................. 32
   4.3 Results ....................................................................................................................... 33
5 Conclusion and future works ......................................................................................... 34
   5.1 Future work ............................................................................................................... 34
      5.1.1 Other communications protocols ....................................................................... 34
      5.1.2 Source compatibility of formates ..................................................................... 34
References .......................................................................................................................... 35
List of Figures

Figure 2-1: The iterative complexities involved in ALM process [14] ................................................................. 10
Figure 2-2: The suggested approach for tool integration for software architecture [6] ................................. 13
Figure 2-3: The tool-tool mapping of meta-models ................................................................................................... 15
Figure 2-4: The typical node to node message passing in ad-hoc networked software architecture tool integration. [12] ........................................................................................................................................ 16
Figure 2-5: The abstract view of common meta-model [41] ....................................................................................... 18
Figure 2-6: The integration complexity with new tools [11] ..................................................................................... 19
Figure 3-1: OSLC Core specification concept and relationship [1] ........................................................................ 21
Figure 3-2: OSLC Resource shape illustrative example .......................................................................................... 26
Figure 4-1: Overall description of the system ........................................................................................................... 31
List of Tables

Table 1: Methods in Persistence and populate classes.................................................................29
Table 2: method in Service Provider Factory.................................................................................30
Table 3: Methods in ServiceproviderSingleton Class. .................................................................30
Table 4: Methods in ServletListener Class. .................................................................................30
Table 5: OSLC main annotations.................................................................................................33
1 Introduction

Throughout human life history productivity has been significantly increased when human intelligence is supported by powerful tools. Today software application development process normally consists of several different tools networked and utilized in application life cycle. Software engineering systems and tools are usually complex applications produced by different vendors (vendor independent), and designed to support specific functionalities, sometimes to multiple users on one system. Some development environments are vendor dependent that utilizes all tools from single vendor. In general it is believed that tool interconnections within an organization or external organization, users and platform play a key role in software development industries. Also tool integration is an issue of big concern in the software development process for many organizations. These processes are highly complex, dynamic for their requirements, and required to support with extra features such as efficiency, extensibility and reusability. Generally tools of different phases are not designed to cooperate as a built-in function and have no access of common meta-data with each other to assist in development process. For example typically in a project, different tools are used in architectural design process or test case generation process. Integration of these tools that enables data processing from one tool to another could support the user to reduce time, increase productivity and effectiveness. So tools integration is essential for efficient production in application lifecycle process.

Integration of tools involved in application life cycle has a long history. Previously, tools were less complex and generally operate on single system, so pairwise integration had been the most applicable approach, but it limits for small level of integration, also modification for version updates from the vendor is generally not supported. Another approach for tool integration is to design a tool on the same platform or Integrated Development Environment (IDE) such as in Eclipse IDE [29]. For example Eclipse Modeling Framework (EMF) supports cross-reference or structured meta-data for modeling tools, as defined data structure that can be imported for other solutions. However this approach is limited to a single platform only and does not support for other platform based tool, moreover if operation of a tool changes its environment, the integration does not remain valid.

In this thesis a new approach, “Open Services for Life Cycle (OSLC)” [1] for tool integration is studied and implemented. OSLC provides standard framework for tool integration that is applicable for all integration scenarios. OSLC is an open community, to define a set of specifications to support integration in tools for software development and more broadly Application Lifecycle Management (ALM) [5], Product lifecycle Management (PLM) and services. The intention is to support software or product developers and tools vendors, by tool integration, making it easier for tools to work together. To fully understand its constraints, OSLC is implemented as an integrating solution between ViTAL [35] and Farkle [34] tool. The ViTAL is a research development tool that enables the component functional model in EAST-ADL verified by Timed Automata verification using UPPAAL PORT to generate abstract test cases. Farkle is test case execution tool developed by ENEA [36] that receives abstract test cases using the OSLC tool integration and execute them to verify results. This thesis is carried out at Mälardalen University Västerås.

1- OSLC is now a part of OASIS open standards network.
1.1 Problem Statement

The Industrial software development is facing challenges to integrate different types of tools used in the development process during the application life cycle of a software product; such as tools for the requirement management, architecture management, version and configuration management, deployment and automation. The tools are used to manage different activities and development artefacts in the life cycle of a product are usually from different vendors with no or loose integration. Even in cases where two tools are integrated by, for example, implementing appropriate and specific import and export mechanisms, if a company decides to replace one tool with another (e.g., when a tool provider goes bankrupt and no support is provided anymore, due to licensing issues,) the integration mechanism will not work anymore. OSLC is an answer to such integration issues which provides a standard for tool integration.

This thesis comprises of a careful background study about different tool integration solutions and the OSLC standards with its core specifications. The second part comprises of providing a particular tool chain solution between Farkle and ViTAL tool and study there various behaviors. The motivation behind this tool chain integration is to simplify the validation and verification process of a model with various parameters by executing the corresponding test cases. For this OSLC adapter approach is utilized on the both provider and consumer side. In our case ViTAL generates abstract test cases as provider for OSLC resource and the Farkle that receives and execute test cases is the consumer for this OSLC resource.

1.2 Overview

In Section 2 the Integration approach is discussed that is based on OSLC standards. This section elaborates the background and motivations for tool integrations and discussed about various solutions. The description of current integration solutions, integration concepts and architectural concepts gives a background picture of the thesis work based on.

In section 3 the background of OSLC and the methodologies used in this approach is reported. General functions, methods and solutions are explained in detail. The evolution of OSLC integration and the governing body is also discussed briefly especially different core specifications.

The implementation of OSLC with elaborative example constitutes section 4. The approach used for implementation, tools and functionalities that are supported is discussed in detail. The tools Farkle and ViTAL that are integrated as an example are also briefly introduced.

Section 5 discusses about the results, and also the success points. How OSLC improved the integration approach and the outcomes are discussed. Furthermore the design advantages and its comparisons are also mapped, how the desired results are achieved using this approach.

1.3 Limitations

OSLC provides a broad range of services but we would focus here with Change Management (CM) and Quality Management (QM) domains with specification version 2.0. As the version 3.0 is still evolving so we would remain focus with core specifications version 2.0.

The web security feature OAuth, to verify the authorized access to the online resources is also out of this scope as the implementation is simple and with open source solutions with no security required constraints.
As the research is initial level and the integration scenario is not complex, the UI preview feature (representation of resource using mouse hovering) is also not implemented for the solutions.
The automated resource update is another advance feature of OSLC that is not included in the solution.
2 Tool Integrations

What is tool integration? A simple statement, define integration as,
"Integration is a property of tool interrelationship". [6]

Tool integration is the meta-model based service that allows the accessing and managing information as a set of shared resource. The level of tool interrelation represents the degree of integration. In this chapter integration phenomenon, the role of integration and types of integration between tools is discussed briefly.

2.1 Life Cycle Management

Life Cycle Management is a process, required to manage the involved objects, to attain desired results or solutions. It is a comprehensive standard protocol designed for target oriented projects, to eliminate unnecessary delays.

2.1.1 ALM

Application Lifecycle Management spans to the steps (processes, models or tools) taken from business idea to creation of an application till it get deactivated. ALM is defined by Dr. John Scarpino as,

“ALM is a set of disciplines that together govern the process of turning business ideas into Software”. [7]

There are no specific boundaries for the steps required to be taken and the level of complexities involved in the whole process. As shown in the figure 2-1 there could be numerous and iterations involved in a general ALM process.

![Figure 2-1: The iterative complexities involved in ALM process [14]](image)

ALM could be classified in three aspects, as governance, development and operations. Governance accounts for whole process of ALM, which deals with the business values and the interests of stakeholders. The development rather considered as the construction phase that manages the development process and could be considered as SDLC (Software Development Life Cycle). The operational aspect deals with the resources used in the whole process to make
sure the application could run reliably. ALM defines the set of integrated tools that are required to manage and help in software development project. The following classes of tools are categorized as ALM tools that should be involved for a better and successful ALM design.

- Requirements analysis & management tools
- Modeling and design tools
- Project management tools
- Configuration management tools
- Build management tools
- Software testing
- Release management tools
- Issue management tools[5]

2.1.2 PLM

Product life cycle management is a complete approach for a product development and introduction management from beginning of the product till end. It could be formally defined as

“Product life cycle management is the process of managing product-related design, production and maintenance information. PLM may also serve as the central repository for secondary information, such as vendor application notes, catalogs, customer feedback, marketing plans, archived project schedules, and other information acquired over the product’s life.” [43]

PLM is not only about communication process between product stakeholders but a record for all product related information.

PLM is considered as the formal management related information and the example of creation tools include Excel spread sheets or business assessment tools. PLM could involve marketing strategies, sales bill, broachers or form communications to the board of directors.

2.1.3 ALM v/s PLM

ALM is the technical development process that includes the development phase of the application.

PLM is the managerial development process that includes the business management phase of the product.

ALM defines the processes for an application that for instance could be the part of any another project or product, but the Product Life Cycle Management spans for the whole process involved in transforming any application or solution into a market product. So the PLM could consist of one or several ALM processes chained up to be a part of PLM. Product Life Cycle Management is the generic term that involves processes from various domains and aspects.

Both ALM and PLM have their specific tools assist in their respective processes.
2.2 Integration

What is a resource? A resource could be any piece of meta-data that could be utilize in a meaningful purpose. It could be a value, control information or any system state. Tool integration is the level of support that a tool could share resources in the form of User- interface conventions, data, model or any special function of a tool. IEEE has created a standard [4] that references basic framework of tool integrations in terms of different platforms, users and organizations. The standard also describes the behavior between users and other tools. The concept of integration is first documented by Wasserman [8] who suggested the dimensions of software tool integration. His work is then preceded by Nejmeh and Thomas [6], beside quoted Wasserman’s approach, they suggested a scheme of classification in the dimensions of software tool integration. Wasserman classifies five types of software integration categories that are discussed in detail on later sections. Wasserman also argues for a process management tool however he believes that there is no consensus on the best way of data representation. He argues about the key issues for tool integration as the capability of determining appropriate level of integration with set of conforming tools that specifies some standards and layered approach with respect to that level. The level of integrations a tool can support could be summarized in five dimensions of tool integrations that are summarized below [8].

2.2.1 Presentation Integration

Presentation integration is the type of integration in which tools and products developed as a similar shape or presentation giving an organizational trademark. This type of integration provides the similar functional organization of the product, helps user to familiarize with different products. The behavioral interaction is designed to improve the effectiveness of the product by having a known user experience from one tool to another.

2.2.2 Data Integration

Data integration is the type of integration in which various tools share their data to keep it consistency during a process. Generally data sharing occur by passing resource to each other or store them as a shared repository to access or modify it accordingly, which is similar to database systems. All tools must agree on some standards of formatting when to modify or save data. Another approach, which is mostly common in the tools of same vendor or its close partners, is to set different standards between the tools and to transfer data as inter process communication.

2.2.3 Control Integration

In control level integration various tools are deployed to update other tools with current systems status and notifications. For example if in a requirement tool, some requirements gets update, these changes must consequently update the corresponding test case in testing tool. So in control integration tools could notify and perform action for corresponding events.

2.2.4 Process Integration

In process integration, different tools are integrated to track and manage development processes. To achieve this, these management tools are integrated with the tools that are contributing in development process with presentation, control or data integration.
2.2.5 Platform Integration

In distributed systems, software tools in a process operate within their corresponding systems. The Systems services that provides framework and support to the tools to integrate with corresponding tools of different environment using network or operated systems determines the platform integration.

![Diagram of Platform Integration]

Figure 2-2: The suggested approach for tool integration for software architecture [6]

2.2.6 Parameter passing

With the tool integration, the parameters passed within various tools during communication occurs in two ways, [13]

- Pass by reference
- Pass by copy

2.2.6.1 Pass by Reference

In this approach network objects represent the values of parameters that are subjected to transfer while the provider tool keeps the original data. The tool which is required to send information is called provider tool while the tool that requires to receive the information is consider as receiving tool. The receiver tool invokes the object to request for accessing data. The data repositories that represent objects in the form of graphs (UML) are required to be networked instead of just local data repository. Also the receiver tool points to the object inside the model to modify data.
2.2.6.2 Pass by copy
In this approach receiver tool receives the copy of the information from provider which can updates its values with a newer version.

2.2.7 IEEE standard for tool integration
The Institute of Electrical and Electronics Engineers (IEEE) is the world largest professional association that sets global standards in the broad range of fields and industries. It published a standard in 1991 that addresses the integration issues, and effective guidelines for software systems tool integration.

The standards provide a framework for tool integration, its adoptability for computer software systems and automated development support. The clauses discussed in the publication are summarized below.

2.2.7.1 IEEE P1175.3 reference model for specifying software behavior
IEEE P1157 standards address the behaviour of software processes specified by common modeling concepts and logical model. It also specifies Semantic Transfer Language (STL) that fascinates in communication of software states within different tools. STL is both human readable and machine processible syntax which allows tool to tool data exchange, user review and can easily be modify using any text editor. IEEE P1157.3 specifies this syntax to optimize the transformation of software behaviour within software engineering environment which is complement to Unified Modelling Language (UML), Extensible Mark-up Language (XML), Meta-data Interchange (XMI) and Meta-Object Facility (MOF).

To standardize the above syntax, IEEE P1175.3 specifies uniform integration model and the textual syntax representing common properties and attribute for the above concepts that used to express the behaviour of the software model. If in an environment two tools are used for system description using their meta-model, at least one tool must map its individual meta-model against the next tool to share information.

IEEE P1175.3 specifies “tool-neutral”, another modelling concept for data sharing between two tools by mapping their internal data into IEEE P1175.3 specified STL syntax and exchange information using STL syntax.

So every tool is not required to know the internal structure of other tool as all tools are able to map their own data against IEEE P1175.3 standards. Which means every tool can exchange information to other because of one concept mapping per tool, which reduces the effort of tool integration as compare to pairwise tool integration mechanism.
Figure 2-3: The tool-tool mapping of meta-models

So it defines the standard to pass UML models or other MOP-compatible models from one tool to another using format specified by XMI. These specifications and standards are not applicable to the methodologies that are not MOF-compliant but the models can exchange data that can be mapped to STL, giving possibility to transfer information between the models that are not compatible to XMI.

2.3 History of tool integration

Brown [15] first discussed the concepts of tool integration by suggesting that the integration strategy must be both flexible and adaptable that could accommodate variety of scenarios and also at the same time efficient and simple. Brown stated, with integration techniques that were mostly focused on data integration as compare to a control strategy based on sharing of data would be more appropriate solution. Brown suggested three strategies of work regarding tool integration, the definition of new standards, integration semantics and the relationship between integration and process. This opens a new approach rather than simple tool to tool integration, a more delicate and extensive characteristics of tool integration that give requires the need to
achieve a mechanism of central repositories with standard data structures. In this way, any software tool or system is required to have a central data management system that could be accessed to other systems. The example problem shown in the figure 2-4 is in which node to node connectivity within network is suggested to replace by a central repository that route the communications. Brown also suggested a research for control integration solution more easier, simpler and adequate, that could typically enacted using a broadcast or multicast message server, or via point-to-point communications facility. In later he also discussed three systems; FIELD, SoftBench and ToolTalk that uses message passing to achieve control integration and then addressed the practical issues that could enhance tool integration.

Figure 2-4: The typical node to node message passing in ad-hoc networked software architecture tool integration. [12]

He proposed that tools must also be able to encapsulate in order to facilitate the other third party tools for integration which requires standard messaging protocol such as used in COBRA. Reiss [16] Later, discussed Desert software development environment which is an extension of FIELD, integrates using broadcast messages. This resolves problems of data integrations using, hypertext link to connect the artefacts between different software systems, which lead to require all tools to acquire common framework. Fromme and Walker [17] discussed plugin typed tool integration concept using SoftBench framework, based on ECMA “toaster” model. This uses a broadcast message server (BMS) built as a top layer of socket interface, acts as a central point of communication. Tools are encapsulated to act as wrapper for an object in a BMS customized environment, providing a degree of flexibility.

Stoeckle [18] addresses the tool integration as an exchange of data with format acceptable to other tools. As an experiment he converted the data of a tool called Pounamu in specific formats other than the tool uses itself, such as GXL for SoftArch and other rendering formats such as SVG and VRML. Bao and Horowitz [19] presented another approach to integrate Commercial-Off-The-Shelf (COTS) products by designing the framework compatible with the tool environment. The authors further describe the framework which consists of four parts, a dynamic integration model binding mechanism which separates the relationships from the tools themselves, nontraditional interfaces for tool integration such as GUI and system interfaces, a tool integration language (TIL) capable to access a tool using its interface, and a tool integration server (TISS). This framework provides users to interact through GUI and data integration is achieved using a repository so it has several advantages as no special interface
required and the modeling relationships between tools decouples the integration policy from the mechanism.

Khare [20] first suggested XML based architecture-centric and reusable integrated environment to utilize in tool integration. In this architectural based approach a tool is supported throughout its lifecycle with repositories and automated processes to reuse the data are assumed to be predefined. The author supported hypertext web to aid in integration for whole product. His work is forwarded by Freude and Konigs, [21] they concentrated on specifying functional dependencies and suggested a framework that provide both data and control integration. XML is used as a tool exchange data that can manage by the tool by consistence in change with respect to the tool. The repository in this way is visualized to the user aiding to understand the structure and its dependencies, and is implemented using an API similar to interface of JAVA metadata. Further enhanced by Margaria and Wubben [22] as they suggested tool integration by internet based approach using electronic tool integration (ETI) platform as a repository. The authors suggested tool integration mechanism using XML-based description language, with the architecture executable using SOAP.

Nutter [23] presented interoperable workflow management system named Open Source Component Architecture Repository (OSCAR) that provides extra workflow system capabilities. These capabilities are provision of namespace, and data control as an artefact. OSCAR provides services and functionalities using third party open source applications and use XML to artefact using meta-data. Smith suggested a traceability layer to a tool named Ophelia that supports users to manage a change process and synchronization with resource artefact. This traceability layer utilizes COBRA objects for representation of the artefacts in tool integrations and uses graph-based mappings to manage the relationship between objects and their metadata.

Other tool integration platforms such as, ModelBus [24] uses meta-modeling for resource representation and provide support for generic building blocks or execution of tool chain.

So far numerous efforts and research have been conducted to classify tool integration standard framework or an appropriate solution that could simplify the integration scenarios but yet no any significant solution is identified in industrial experience. Majority of them tend to solve a specific aspect, also it is evident that no ideal software engineering system is available for every situation and so does the integration solution.

2.3.1 Common Meta-model Interoperability

"Common Meta-model interoperability of tools with a heterogeneous data model" is a project of CESAR European research group [39] targeted to provide integration support between the meta-models of various tools to manage data handlings.

This approach is based on the concept to centralize the common resources as “Common meta-model" to reuse the artefacts that could be shared by other tools for example like model based tools used in the development of embedded systems project. As model verification is playing vital role in efficient design and requirement verifications in embedded systems, this initial research is more focused on model management by enabling exchange and traceability of heterogeneous models of the tools, that supports different processes like, requirement management, structural component design etc. As in model architecture based platform, models are considered as primary artefacts to describe a system and its behavior. So the idea behind
this approach, is to provide a common meta-model that is connected to other meta-models using model-bus. [37] This would provide the support to the new tool using its meta-model by either the central meta-model repository or the instance of other already connected meta-models using traceability and linking meta-models. So in short, it provides support to the new tools with required meta-models. The figure 2-5 shows the abstract view of the Meta-model solution with a general example of three tools having combined common Meta-Model shared on different abstract layer level. This common Meta-model could serve in term of resources to other tools that connects to this whole system. As in the CESAR research it offers the support to formalize the structural model and textual requirements generated by EAST-ADL2 (by the support of Papyrus tool [38]) using,

As various tools associate their respected meta-models, a transformation solution is required to configure the common meta-model to enable it as a resource model for other meta-model. In the CESAR research transformation description was created using QVT (Query/View/Transformation) [40] and the transformation for a common view is performed a service connected to modelbus. A typical meta-model consists of artefacts like component, ports and connectors may have attributes for internal references and could hold the parts of other components that could facilitates reuse of the component, also traceability with other processes. The requirements for a model could range from simple solution to complex approach that may involve several components and processes, as its dependencies, required to link the involved artefacts. These artefacts could be provided by Common Meta-model using the mapping of the defined artifact and their relation as a table.

Figure 2-5: The abstract view of common meta-model [41]
This integration is based on models with shared resources and hence limited to the degree of commonality, within the range of specific platform in the process, as compare to OSLC which is quite flexible and provides wide range of services and resources that supports.

2.4 Why OSLC

The desires and expectations for a tool integration solution are always to provide simplified and maintainable solution. Most of the solutions discussed above do have limitations in one or other way. Many solution approaches does not ensure the credibility of data and APIs that are utilized in integration scenarios. Integration itself is not flexible enough to handle the tool evolution environments. For example updating a Meta-model used by any modeling tool could break the references it made by a requirement tool. Same with the API based solutions that could break a link by modifying the corresponding API. This integration problem enhances with respect to the size of the network of tool integrated hence making it a complex solution. As shown in figure 2-6, integrating a new tool in already node to node connected network is not only complex in design but also in resource sharing.

![Figure 2-6: The integration complexity with new tools](image)

OSLC is relatively a compact solution that eliminates all the limitations that are subjected with each other tool integration solutions and also provides the dynamic property which means if a tool changes its platform, it won’t break its interface. OSLC is an industrial effort that supports the specifications of widely used domains of software engineering solutions such as Requirement Management or Change Management. As compare to the above existing approaches, OSLC supports a better way to handle tool replacement issues due to its standardized adapter structure and corresponding domain specific services. Due to the fact that OSLC specifies the same standards and mechanisms for any tool element to link with other platforms with minimum amount of required protocols, it enhances the capabilities to manage the integrations easier and better within software lifecycle management.
3 OSLC

3.1 Introduction

As business innovation and processes are getting highly dependable on software tools, organizational requirements are getting increased, consequently demanding more software developments and support. In bigger organizations, the clusters of software tools are developed by different vendors, desired to be highly interchangeable so to overcome any barriers in future extensions. Whether any tool is created by commercial vendor or an in-house solution, it’s hard to collaborate it with other related tools during whole application lifecycle. Integration itself is complex to develop and expensive to maintain especially in ever changing tool requirement scenarios. Tool integration platform [26] (TIP) which is supported to tool integration falls as a burden to many tool provider despite most tool providers does not consider it their job, and believe infrastructure provider’s task. These issues urge to have a better tool integration solution approach that is easier to implement and dynamist to its behavior. OSLC tool specifications are striving to provide solution to this. The OSLC tool integration specifications are developed by OSLC organization, aims to provide specifications and standardization to the all aspect of tool integrations.

OSLC specifications are designed to support development and management of systems by providing the support for easy integration using defined rules for managing the subject resources. OSLC organization is classified into corresponding workgroups to establish individual domains or core specifications [31] and guideline to ensure the commonality in integration as according to their functionalities.

When data is intended to integrate from one tool to other, it is required to be represented as a compatible way for that tool and in this process it transforms with respect to its requirement. In OSLC Data is considered as an OSLC resource and made available to the web services using RESTful (Representational state architecture) by provider. As the resources uses transformational syntax in RDF/XML or JASON or other supported formats and the types of the resources (resource shape in OSLC) are defined in their respect domain specifications. As this resources are in a standard format, it eliminates the problem that arises if a new tool is required to interface as the only thing requires is to follow the specified standard.
Figure 3-1: OSLC Core specification concept and relationship [1]

For web security OSLC supports OAuth as resource security feature that enable authentication to resources access.

Figure 3-1 elaborates the overall working of OSLC design, and services provided by OSLC using the modules that performs respected functionalities. The resources are created in creation factory and displayed using service provider catalog. Delegated UI preview enables GUI resource representation that facilitates consumer to create resources and to select resources.

3.1.1 OSLC organisation

The OSLC organization aims to remove barriers and challenges for software lifecycle integrations. The objectives of OSLC steering committee are,

“Open and generally available data instead of proprietary point-to-point integrations,

Scenario-driven specifications instead of exhaustive standards,

Public discussion and debate instead of backroom dealing,

Responding to the needs of the community instead of following the agenda of a single member”
As OSLC facilitates to promote adoption, so all the specifications and the work is provided as an open source material and published under “Creator commons” license.

3.1.2 Core specifications
Open Services For Life Cycle (OSLC) Management has defined standards for core specifications and its various domains specifications for development of integration between software lifecycle tools. Core specifications define the primary standards and rules for RDF and Http that could consider common attributes. The software tools with specific environment and specifications follow their additional corresponding domains, i.e.

- Configuration Management,
- Quality Management,
- Requirement Management,
- Architectural Management.

As the scope of thesis confine to Quality Management and Change Management so would discuss here in detail.

3.2 OSLC Implementations
OSLC basically built on W3C specifications and uses Resource Description Framework (RDF) as resource repositories and communicate using Linked Data and REST protocol. So OSLC Implementation does not require JAVA, JSP or servlets, etc. As it's an open specification, the communication takes place using HTTP protocol [28] so if its .NET, Ruby or PHP etc. required integration, it would work by just little modification in the application. For example if there is a change request from a tool belong to Requirement Management (RM) domain to change management (CM) domain, it would be a simple URI that would be stored as an attribute of “change request”. So for this RM tool would make its resources available by publishing them as REST (GET, PUT) interface [23]. OSLC is does not specify any standardization to the tool process but provide a protocol that facilitates tool working together in an integrated environment. This facilitates in the way as it eliminates the need for separate rules for different tools or any synchronization issues. Similarly it does not require any change in the internal tool architecture so no consideration for integration requires in new versions of the tool.

Basically data integration in OSLC implemented using linked data method that use to represent the resource of the subject tool using RDF artefact, identified by corresponding URI, and manipulate these resources using RESTful (CRUD) protocol. These artifacts are then linked in desired fashion by embedding URI of each resource representation.

3.2.1 Linked Data
Linked Data is a structured data first proposed by Tim Burners-Lee [42], design for publishing the data. Linked Data is considered as flexible, standardized open source base approach that provides support to share and integrate link heterogeneous and disparate tools for data, data silos across and within the organization [30]. This format aids for better interconnectivity within data connections and also transforms data into simple, easy to integrate and structured way.
Linked Data terminology leverages to specify the methods implemented to interconnect data with its resources like Web architectural concepts using http links and URIs to transfer from one location to other. The Linked Data is specified using RDF as its fundamental data model and SPARQL as its query language. The important properties considered was that it provides a source to structure in RESTful services and can be used as metadata for other tools that could be understandable both by machine and humans. The four principles of Linked Data are,

- “Use URIs as names for things
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things”.

3.2.2 RDF/XML

Resource Description Framework (RDF) is a framework that represents linked data implementation in Semantic Web. The RDF/XML [32] is the primary technique used to manipulate data in OSLC. XML is segmented in the term “Semantic Web” a phenomena that enables Machines to process data. XML stands for Extensible Markup Language and is a common standard to represent data in software world, could be used with any language. It enables data portability, could help transfer data in a network or store it into a database. As it is extensively using in IT, the tools to process and manage XML are developing prominently.

XML constitutes of Elements, Attributes and some other details (freely definable tags). As it possesses infinite tag set, XML is considered as very flexible and fully customizable. XML could be easily use by other programs using available libraries and could convert into other form of representation (XML transformation language).

The Resource Description Framework (RDF) is an infrastructure that enables the encoding, exchange and reuse of metadata using World-Wide Web, and is a foundation future of Semantic Web. XML is used as the syntax to represent RDF models into machine-readable files and enable to communicate this among other applications. RDF structure contains statements in the form of “triples” that describe resources in the form of property values. The property types could be uniquely identified using XML namespace mechanism, a method to clearly identify semantics and its particular property-type use.

RDF is a metadata language designed for web applications to comply using Linked Data. It has pretty simple syntax as first term is subject, second predicate and the third is object,

```
Subject + Predicate + Object
```

This basic structure called triples develops to form graph or data model. RDF uses links to connect to the resources as similar to web and could grow by navigating other resource links making it possible to have a unique resource identification inside a network. A typical example as below.

Example: Erik meets Lars.
<http://google.com/people#Erik>
<http://xmlns.com/foaf/spec/#term_meets>
<http://example.com/people#Lars>.

3.2.3 Representational State Transfer (RESTful)

RESTful [32] architecture is developed by W3C Technical Architectural Group (TAG) and is based on HTTP, comprises of Clients and Servers. It provides a standard of resource accession to support web developments and applications using a distributed hypermedia system. The three basic classes of elements used in REST are:

- Processing elements
- Data elements
- Connecting elements

3.2.3.1 Data Elements

Data element is the primary aspect of REST which distinguish data types with respect to its metadata. REST communicates by transforming the data representation into the format of a standard data type based on the data type and recipient and is termed as media type.

Resource is the main abstraction of information, termed in REST as any information or reference that could be named for example any document, image or collection of other resources or objects (e.g. person). The naming authority assigns resource identifier that is used to identify particular resource utilize for component’s interactions. REST connectors are used for these functions that support all the interfaces to access or manipulate the set of resources so to handle any software requests.

REST components operate a resource through a unique resource representation to capture its state. The representation is normally in binary format described by representation metadata which is the form of name-value pairs define value’s structure and semantics. Control data defines the purpose of the data between components could be as a request or a response. A representation can be included in a message and manipulated according to the control data and nature of media type.

3.2.3.2 Connectors

REST connectors works as an interface used to support the activities of accessing and communication of resource representations. The primary types of connectors are client, cache and server. The client generates request command from user and server responds to the request by providing required services. A cache memory is located to the interface of server and client and store cacheable responses to the interaction to enabling reuse of response interaction avoiding repetition of network communication. The parameter used in connector interface consists of resource identifier, control data and resource metadata. The parameters by clients normally contain request control data, resource identifier indicating the target of the request and its optional representation. The client parameters normally comprises of response control data, optional resource metadata and representation. Another form of connectors, “resolver” is used to translate resource identifier address information into network for interconnectivity e.g. DNS.
As a typical example using an internet browser, a URI, “www.anything.com” use DNS (resolver) to acquire its IP address allocated to the resource. The tunnel is a connector use to communicate through boundaries or restrictions such as firewall or any other gateway.

### 3.2.3.3 Components

REST components are the different processing elements dedicated for their particular assignments in overall web connectivity. A “user agent” is a component that supports users to initiate its request and display its response e.g. web browser. Origin server, proxy and gateway are some other components of REST.

### 3.3 OSLC Resource shape

OSLC resource shapes are defined by OSLC specifications [1] to set the properties with their values to standardize the resources of various types. OSLC resources are the type of resources that could be defined in OSLC core specifications (common properties for similar concepts) and share similar characteristics. In OSLC mostly resources are represented in RDF/XML form and defined using RDF properties that support the situation of any resource that has property not define in OSLC core specifications and also support in OSLC tool integration. In simple words, in order to make tools interoperable, they are required to agree for common sets of properties.

If in a case, some properties of a resource are not already defined in OSLC core specifications, it is resolves in OSLC by providing common properties identified by URIs, and define using RDF concepts. The definitions comprises of URIs that describe the structure of the RDF syntax provided by OSLC provider so that consumer could conform to the contracted interface. These RDF graph constraints could serve as a meta-data for different purposes such as query, builder or test case generator depending on the properties and the values of these resources. A resource shape specification [1] lists the properties, values, expected or required occurrence, representations and so on. Shape specifications validate and standardize the given RDF graph with its values, resources and the query, for example change request could be in the form below;

```xml
OSLC Resource Shape changeRequest-shape.ttl

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_cm: <http://open-services.net/ns/cm#> .
@base <http://example.com/shape/oslc-change-request> .
<> a oslc:ResourceShape ;
dcterms:title "Creation shape of OSLC Change Request" ;
oslc:describes oslc_cm:ChangeRequest ;
oslc:property <#dcterms-title>, <#oslc_cm-status> .
```

Page 25 of 37
3.4 OSLC Service Provider

The OSLC service provider is a method that represents the available resources to the consumer platform, and also enables consumer to manipulate these resources or create new. OSLC provides basic service to the resource is serviceProvider, which represents the available resources but also enables consumer to create its own desired resources with OSLC support. Service provider uses two basic properties, OSLC:Creation

Creation is used to generate new OSLC resources.

OSLC:QueryBase

QueryBase is used to retrieve all available resources.

In complex scenarios there could be more than one Creation URI and QueryBase URI properties with their attached resources and respective intended use in a solution.

To utilize the above services, OSLC provides methodologies named CreationFactory and QueryCapability that use the required functions with respect to the services. CreationFactory utilize properties to create a resource associated by creation URI and the same with QueryCapability.

3.5 OSLC SDKs

OSLC organization is continuously developing software development kits (SDK). They are open source projects to support for the development purpose. Different SDKs are developed in their specific domain and provided reference implementations, corresponding functions and libraries. A brief introduction of some SDKs is below,

3.5.1 OSLC4NET

OSLC4NET is an SDK created to facilitate for .Net and C# development to provide annotations to OSLC resources. To represent .Net resources into XML/RDF and parse it back, it provides a library on “dotNetRDF” Package. Similarly ASP.NET MVC 4 library provide supports for REST commands
3.5.2 Eclipse Lyo.

Eclipse Lyo project [29] provides a dedicated environment for OSLC development. The Eclipse foundation is an open source community and has a broad range of software development tools. Eclipse Lyo is a project of Eclipse with a focus on OSLC development requirements. Mylyn and Orion are other SDK projects of eclipse that could support development of OSLC tool integration adapter.

OSLC4J by eclipse is a java tool kit that supports both OSLC consumer and provider aspect. OSLC4J supports the representation of java object in to OSLC resource and parse it back, also support UI previews. It comprises of “Lyo OSLC Test Suite” a sample application for tests, also Apache Jena Provider, Apache Wink Json4J Provider and Apache Wink JAX-RS [33] provider. Apache Jena Provider and Apache Wink Json4J provider are the libraries that support RDF, XML and JSON transformations. Apache Wink JAX-RS provider is the utility class to support OSLC RESTful functionality for transforming of data.
4 Implementation

The complete understanding for any systems can only be achieved through its implementations. In this thesis work the consumer part of the OSLC implemented that provides the integration functionality between Farkle and ViTAL (the tool developed during this research). The ViTAL generates abstract test cases that are enabled as interoperable resource to Farkle through OSLC tool integration. ViTAL is the tool that utilizes EAST-ADL models and UPPAAL plug-in (port) for validating functional requirements and generates abstract test cases that executes into Farkle test environment. The scope of the thesis is to retrieve the abstract test cases capable to execute in Farkle tool implementation. OSLC tool consumer is utilized for this purpose that extracts the OSLC resource provided by OSLC provider and parses the abstract test cases capable to execute on Farkle.

4.1 Background

The quality of software architecture plays a significant role that must be take account as a functional requirement for development. For this validation, a tool ViTAL is developed and adopted that validates the architectural model developed as component-aware model-checking in EAST-ADL environment, and validates the time automata model using UPPAAL port. The results are then trace tailored to functional block execution and executes using Farkle test suite. The interoperable between Farkle and UPPAAL is done using OSLC tool integration.

The design architectural model developments proved to play a vital role in efficient development environment and to validate system verification.

4.1.1 ViTAL

Verification Tool for EAST-ADL Models using UPPAAL PORT (ViTAL) is a tool that developed to integrate simulate system model using EAST-ADL and UPPAAL-PORT tool.

ViTAL is developed in Eclipse environment and has plug-ins that extract automated transformation model from EAST-ADL model and make it formal design verification using Timed Automata UPPAAL to verify its timing constraints.

The EAST-ADL is an industrially accepted tool that validates models through its functional components using its \( f_p \) functional prototype. On the other hand UPPAAL is a tool used to generate a process defined model and verify it with its functional behaviors to check deadlocks and variable values, also validates TA function of this mode. The semantic mapping is used to transform process from EAST-ADL model to UPPAAL model that enables verification for various requirements like Timed Computational Tree Logic (TCTL). After validation abstract test cases are generated that are covered in our work. The test cases are used as a resource to OSLC provider.

The OSLC ViTAL provider adapter comprises of the defined properties and namespaces that links to the ATC resource. The resource that is shared using ServiceProvider is the abstract test cases shared using RDF framework. The resource use RESTful services through Java API and POST the resource for consumer.
The important functions performing in OSLC provider services are declared in respected modules to support respected functionalities.

### 4.1.1.1 Persistence

Persistence is the important class that along with assistance of populates class, stores the resource values supported by the provider. The resources could be create, delete or accessed to previously dumped resources using this class and also converts into OSLC resources by other classes. So the resources stored in this class are used by *creationfactory* and *providersingleton* classes to create OSLC web resource using *ServletListener* class.

Some functions in respected classes are tabulated below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>private Persistence()</td>
<td>Persistence class along with Populate serves to get resources from the tool.</td>
</tr>
<tr>
<td>public static longgetNextIdentifier()</td>
<td>Every resource has stored with an id by which it could access as its address.</td>
</tr>
<tr>
<td>public static Resource[]getResources()</td>
<td>This class returns all the resources stored in the class.</td>
</tr>
<tr>
<td>public static voidaddResource()</td>
<td>This function adds resources in to the class.</td>
</tr>
<tr>
<td>private static ResourcecreateResource()</td>
<td>This function creates a new resource that could be requested by the provider tool.</td>
</tr>
<tr>
<td>public static voidremoveResource()</td>
<td>This function removes resources from the class.</td>
</tr>
<tr>
<td>public static ResourceupdateResource()</td>
<td>The resources could be update using this function. The parameters it uses are the resource ID and the new resource.</td>
</tr>
<tr>
<td>public static ResourcepostResource()</td>
<td>The created resource are posted as an OSLC resource using this function. It uses servlets to post this resource as web services.</td>
</tr>
</tbody>
</table>

**Table 1:** Methods in Persistence and populate classes.

### 4.1.1.2 ServiceProviderFactory

This class creates resource services for OSLC. It utilizes the methods from *oslc4j.core.model* project.

The function in this class is tabulated below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static ServiceProvider</td>
<td>This function initializes the service provider for the resource</td>
</tr>
</tbody>
</table>
createServiceProvider() services.

Table 2: method in Service Provider Factory

4.1.1.3 ServiceProviderSingleton
This Java class creates service provider catalog that lists all the available resources. It utilizes methods available in OSLC4J.Test.Servlet project.

Some functions in respected classes are tabulated below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static synchronized ServiceProvider getServiceProvider()</td>
<td>This returns the service provider objects.</td>
</tr>
<tr>
<td>public static synchronized URI getServiceProviderURI()</td>
<td>This function returns the URI for the service provider catalog.</td>
</tr>
</tbody>
</table>

Table 3: Methods in ServiceproviderSingleton Class.

4.1.1.4 ServletListner
The web services are controlled using this class. This class set URL (http://localhost:8080/mdhtestcase/ in our case) to service provider catalog and all the resources available. Since in our case the only resource available is ATC as a first id, but this could be extended for other resources using their corresponding id, which is set as the order of their arrival. This class utilizes the methods in ServletContextListener class available in OSLC4J core servlet project.

Some functions in respected classes are tabulated below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void contextInitialized()</td>
<td>This method initializes the Javax servlet and set URI of service provider.</td>
</tr>
<tr>
<td>private static String generateBasePath()</td>
<td>This method generates URI for the initialized service provider.</td>
</tr>
</tbody>
</table>

Table 4: Methods in ServletListener Class.

4.1.2 Farkle Environment
Farkle is a test case execution tool that enables verification of test results for the various systems in OSE Real Time Operating System (RTOS) and Linux. Farkle provides certain inputs to the target using OSE signal and compute its consequential output using scripts written in Python. These test script could use Farkle libraries and function to support proxy execution. After execution of test cases Farkle notifies about the fail/pass result that might invoke the process again. Figure 4-1 represents the overall working of the system architecture, designed and implemented in the research. The designer creates an architectural model that is verified...
using UPAAL PORT and EAST-ADL to generate the Abstract Test Cases in ViTAL tool. These ATC are passed to Farkle using OSLC tool interoperability, which executes these test cases.

Figure 4-1: Overall description of the system.

4.2 Farkle OSLC consumer
The Farkle OSLC consumer is designed to retrieve abstract test cases provided by the provider using RDF/XML parser. The tool integration is developed in Eclipse Lyo environment using
libraries such as OSLC4J, OSLC CM and OSLC lyo core. The getter and setter methods used to retrieve the resources made available by the provider, the underlying logic of PUT,GET and other RESTful implications has been done using apache wink webservice. This shows the advantages in this technique that lowers the framework restrictions for new connections. The resource shape and sharing has been done by OSLC QM specifications and customized mdh quality management specification to provide the standardization in the process.

The resource parses back as a java object using OSLC library of Jena Service provider and execute in to Farkle environment, where it run to generate python codes to execute test case.

This approach provides a degree of flexibility with tool selections, does not involve hard dependencies for a new tool to fulfill. The Farkle consumer is divided into two parts,

- Resource Retriever
- ATC converter

### 4.2.1 Resource Retriever

The exchangeable artefact ATC is provided to the consumer side using a URI that directs REST client to the OSLC resource from provider. The consumer is static implementations which means that changes request in ATC resource does not invoked consumer if the change in artefact happens through service provider factory. The change request process from provider (which could be change in resource links or just property updates) must be acknowledged from the user to enable the update retrieval for the artefacts.

The service provider catalog provides information about the available resources, in our case the abstract test cases that are available for the Farkle consumer could be retrieve and execute by the user. The resource shape of the ATC is in RDF/XML format could parse back use services provided by Jena.

The consumer.java class has been developed in Eclipse lyo environment that implements the above functionalities, creating objects and functions to hold the parsed data. The proxy.java class use to establish connections with the required URIs by setting up Http connection with the servers and handles the Http response. It passes the output response to the respected class for parsing and manipulating resources.

The major OSLC class annotations used as

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@OslcNamespace</td>
<td>Defines the name space of the class to use, could be custom defined name spaces for specific properties and value types.</td>
</tr>
<tr>
<td>@OslcResourceShape</td>
<td>This declares the type of resource shape by using the title of domain specification, in our case Quality management.</td>
</tr>
<tr>
<td>@PropertyDefinitions</td>
<td>This provides the properties mentioned in the corresponding namespace.</td>
</tr>
<tr>
<td>@OslcDescription</td>
<td>This specifies the unique description of addressed resource, Abstract Test Cases is used in our case.</td>
</tr>
</tbody>
</table>
@OslcReadonly | This specifies the attribute with limited read only permissions.
---|---
@OslcValueType | Specifies the type of the values for the attribute, OSLC4J is used as default type and for our case.
@path | This specifies the path of specific attribute.
@OslcOccur | Specific values could be mentioned.

| Table 5: OSLC main annotations |

Here is some methods used for implementations,

- Dojo.xhrGet() method is used to request the required URL using proxy service.
- ParsePreview() method from Jena model is used to retrieve back the resource from RDF/XML notation.

A java implementation is used to convert the retrieved resources and properties into a format acceptable to the Farkle. The resources are stored into an array of java object using Java notations that requires retrieving into desired format using java class. A method is designed that maps the values with text output

4.3 Results

The motivation behind the implementation of OSLC tool integration is to support development process that is handled by different tools. In our case OSLC tool adapter enables the availability of ATC as an OSLC resource using web services to the Farkle tool which execute them for verifications. This facilitates the overall system, since the provider provides an accessible resource which is available at any time using jetty web server, the Farkle consumer requires REST client with URIs to retrieve theses resources. The advantage here with this tool adapter is that it supports the scenarios if Farkle is changed by any other tool, the integration remains effective and would not discrete by tool up gradations.
5 Conclusion and future works

Tool integration phenomenon has a long history and covered lots of milestones to get developed with more solution oriented. Tool integration plays a vital role in Application Lifecycle Management and remained in research by many scientists with various suggested solutions. The standards defined in OSLC, provides flexible loose couplings and evolving every single day. OSLC has been developed after years of research and experience that correlate from industrial requirements and getting accepted as a standard for tool integration approach.

In the thesis work the process of tool integration is carefully studied, the evolution history and various technique that were implemented. In the next phase the OSLC tool integration standard is understood and applied to fulfill tool integration requirement between Farkle and ViTAL tools. Its interoperability facilitated with extensions of increased functionality and software development processes. As in our case we could have better results and quality production.

The core specifications are expanding with newer versions and expected to complete with more standards of property types and resource shape. In future works the UI preview could be adopted with functionality to invoke the method to fetch the resource by hovering over the mouse on the link.

OSLC has been emerging as a supportive and easy to implement solution that supports backend implementations and resource shapes properties provided by OSLC steering committee and various other bodies and in future could provide services in more domains and ranges like in automotive and Hardware support with its extensions. The libraries for general property types and services could enhance to support more functionality.

The OSLC tool integration is as simple as communicating resources between tools through web and at the same time quite effective in the way it supports complex tool integration scenarios.

5.1 Future work

Some suggestions or future possible solutions to OSLC are briefly discussed below.

5.1.1 Other communications protocols

The development solution with OSLC is web based solution that could be extended to other networking protocols for resource sharing. For instance if any tool is required to communicate using data or control signal, It could currently Post the resource on web as it is supported by OSLC but the media in this case could be any other protocol. So In future work OSLC standards with resource properties could be implemented on other communication protocols. This could lead to possible support for control and automation solutions.

5.1.2 Source compatibility of formats

In complex development environment it is common that tools generate outputs into specific formats that sometimes not recognizable for other tools. In future OSLC could additionally convert the formats of the resource into standard OSLC format.
References


[37] ModelBus (www.modelbus.org/modelbus/)

[38] Papyrus UML (papyrus.org/)

[39] CESAR (Cost-efficient methods and processes for safety relevant embedded systems): European funded project (http://www.cesarproject.eu)


[41] Andreas Baumgart “A common meta-model for the interoperation of tools with heterogeneous data models”

[42] Linked Data - Tim Berners-LEE http://www.w3.org/DesignIssues/linkeddata


[44] Marinescu, Mehrdad Saadatmand, EAST-ADL Tailored Testing: From System Models to Executable Test Cases