OSLC-based Support for Integrated Development of Dependable Systems

Alexei Iliasov, Alexander B. Romanovsky, Linas Laibinis, Elena Troubitsynå

OSLC-based Support for Integrated Development of Dependable Systems

Alexei Iliasov, Alexander Romanovsky
Newcastle University
Newcastle Upon Tyne, UK
{alexei.iliasov, alexander.romanovsky}@ncl.ac.uk

Linas Laibinis, Elena Troubitsyna
Åbo Akademi University
Turku, Finland
{linas.laibinis, elena.troubitsyna}@abo.fi

***Abstract*—Engineering of dependable systems is an inherently heterogenous field and involves the use of a wide range of techniques to analyse different aspects of the system behaviour and properties. Various standards typically prescribe a set of techniques to be used and a development process that should be followed to achieve a high degree of dependability and demonstrate it during certification. In this paper, we address the problem of building integrated environments that implement the processes required for engineering dependable systems. We discuss the use of OSLC (Open Services for Lifecycle Collaborations) – a rapidly developing industry-driven standard as a technological platform for such integration and present our ongoing work on building an integrated environment for formal development of dependable systems. Our prototype environment spans over requirements engineering, formal modelling and verification in Event-B as well as safety case construction.**

I. Complementarity and Diversity in Development Process

Development of dependable systems is an engineering field that is heavily regulated by standards [1]. Though standards are often criticised for their rigidness, they nevertheless capture the best practices and define recommendations based on decades of experience in engineering and operation of dependable systems. The majority of modern standards adopt process-oriented approaches to dependability assurance, i.e., they prescribe a set of methods and tools to be used to achieve dependability and demonstrate it during certification.

Let us note that, despite differences in details, the majority of dependability-related standards focus on enforcing complementarity and diversity in system development. Indeed, they aim at ensuring that there are not gaps in the analysis of the system behaviour in both nominal and off-nominal situations, failure analysis is complete, and several diverse layers of protection are built to break the chain of error propagation.

Diversity and complementarity are enforced at both development process and system design levels. For instance, a development process typically complements fault avoidance with fault tolerance and fault removal; diversity is also employed in designing software and hardware to mitigate a risk of common cause failures.

To follow the recommendations prescribed by standards and ensure that diversity and complementarity are properly implemented, the developers should be able to build integrated engineering environments that establish a correct and efficient flow of information within the development process.

The development process generates a large amount of heterogeneous information – requirements, models, designs, tests, documentations, safety cases, etc. Obviously, to efficiently handle it, we need to provide the engineers with an automated support that establishes common information space, supports an efficient dynamic information flow and enables seamless integration of diverse tools. In this paper, we argue that OSLC provides us with a suitable technology to achieve these goals. Next we give a brief overview of OSLC.

II. OSLC

Open Services for Lifecycle Collaboration (OSLC) [2] is an open community, the main goal of which is to create specifications for integrating tools, their data and workflows supporting lifecycle processes. OSLC is organised into domains that address integration scenarios for change, test, requirements and configuration management. Each domain specifies a common vocabulary for the lifecycle artefacts needed to support the integration scenarios.

OSLC specifications focus on how the external resources of a particular tool can be accessed, browsed over, and specific change requests can be made. OSLC is not trying to standardise the behaviour or capability of any tool. Instead, OSLC specifies a minimum amount of protocol and a small number of resource types to allow two different tools to work together relatively seamlessly.

To ensure coherence and integration across different domains, each domain builds on the concepts and rules defined in the OSLC Core specification [3]. OSLC Core consists mostly of standard rules and patterns for using HTTP and RDF (Resource Description Framework) that all the domains must adopt in their specifications. It also defines a small number of resource types that help tools to integrate their activities.

In OSLC, each artefact in the lifecycle – a requirement, a test case or a source code file (or a part of these) – is an HTTP resource that is manipulated using the standard HTTP methods (GET, PUT, POST, DELETE). Each resource has its RDF representation, defining statements about resources (in particular, web resources) in the form of *subject/predicate/object* expressions, i.e., as linked data. OSLC also supports representations in other formats, e.g., JSON or HTML.

The central organising concept of OSLC is *ServiceProvider* that allows the tools to expose resources. In its turn it enables

navigation to all resources as well as creation of the new ones by the service consumers. Two fundamental properties of a *ServiceProvider* are:

1) oslc:creation: the URL of a resource to which you can POST representations to create new resources.
2) oslc:queryBase: the URL of a resource that you can GET to obtain a list of existing resources.

ServiceProviders have a third important property – dialog that describes invocation of HTML web user interface dialogs of one tool by another.

There are several different approaches to implementing an OSLC provider for software. In our work, we rely on the *Adapter* approach. It proposes to create a new web application that acts as an OSLC Adapter, runs along-side of the target application, provides OSLC support and "under the hood" makes calls to the application web APIs to create, retrieve, update and delete external resources.

In our work, we used Eclipse Lyo – an SDK helping the Eclipse community to adopt OSLC specifications and build OSLC-compliant tools. Next we will discuss our prototype implementation of OSLC-based integration between a custom-built requirements management tool, the Rodin platform – an automated tool for formal modelling and verification in Event-B, and a safety case generation tool.

III. Prototype of Integrated Engineering Environment

Event-B is a top-down state-based framework for formal development of dependable systems [4]. It relies on a top-down correct-by-construction development approach to specify the system behaviour. Correctness of models and their refinements in Event-B is verified by proofs. An integrated extendable framework – the Rodin platform [5] – provides an automated support for formal development in Event-B. Event-B has been used in industrial setting, especially in the railway domain.

The Rodin [6] and Deploy [7] projects have significantly advanced Event-B and the Rodin platform as well as highlighted the issues in incorporating formal methods into the industrial setting. The practitioners have observed benefits of using formal modelling especially at the requirements engineering stage. They have also pointed out that an automated support is required to communicate the results of formal modelling to different domain experts involved in the system development.

In our prototype integrated engineering environment we aim at addressing this issue. We focus on building an environment that integrates requirements engineering, formal modelling and verification as well as safety case construction. Requirements engineering and safety assurance are known to benefit the most from using formal modelling and verification.

In our prototype engineering environment we strive to retain both flexibility and notation that is native for each domain. Requirements are defined in the natural language. To maintain the link between the dynamically changing requirements and the associated formal models, we have created a prototype tool, Requirements-Rodin adapter [8]. Formal modelling is done using Event-B – the core language of the Rodin platform. Safety cases are generated in the goal-structuring notation [9].

Our requirements tool uses a generic principle of organising requirements into a tree with further optional cross-links between them and their classifications (by taxonomy, component, developer, etc). The tool provides a simple form-based user interface. It embeds a web-service providing OSLC-compliant RDF descriptions of requirements. Every requirement may be referred to by the project name and requirement id.

The second part of the prototype environment achieves a similar goal for the Rodin platform. We have developed a Rodin plug-in that exposes the Event-B model database and proofs as externally referable OSLC resources. Once again, each model element (variable, invariant, refinement) has a unique global identifier that can be used to cross-link it with other OSLC and RDF resources.

The third part of the environment facilitates generation of safety case. It maps relevant elements of requirements and models into the corresponding parts of a safety case, i.e., allows to reuse the results of formal modelling and verification to construct a safety argument.

IV. Discussion

In this paper, we have discussed our ongoing work on building a prototype of integrated environment for engineering dependable systems. We have focused on building a tool chain that links requirements engineering, formal modelling and safety assurance. Our prototype tool chain aims at enforcing principles of complementarity and diversity in the system development. Indeed, an automated live link between requirements and formal modelling and verification provides a continuous cross-check of requirements correctness and adequacy of the associated formal models. Moreover, via Rodin plug-ins we can augment proof-based verification with model checking, which might be seen as a complementary verification technique. Finally, the link with safety cases allows the designers to spot deficiencies in the system design, e.g., if a safety argument cannot be provided. Moreover, it allows the engineers to efficiently reuse the results of formal modelling for safety case generation.

OSLC is gaining a momentum. We believe that our prototype implementation has demonstrated that OSLC offers an advantageous technology supporting creation of integrated environments for engineering dependable systems.

References

[1] Storey, N.: Safety-Critical Computer Systems. Addison-Wesley, Harlow, UK (1996)
[2] OSLC: (Open Services for Lifecycle Collaboration.) "Online at http://open-services.net/".
[3] OSLC-Core: (Open Services for Lifecycle Collaboration. OSLC Core specification.) "Online at http://iliasov.org/oslc/".
[4] Abrial, J.R.: Modeling in Event-B. Cambridge University Press (2010)
[5] RODIN: Event-B Platform. http://www.event-b.org/ (2009)
[6] Rigorous Open Development Environment for Complex Systems (RODIN): (IST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/)
[7] (EU-project DEPLOY) online at http://www.deploy-project.eu/.
[8] Rodin OSLC Adapter: (Using Instructions) "Online at http://open-services.net/bin/view/Main/OslcCoreSpecification".
[9] Kelly, T.: Arguing safety a systematic approach to managing safety cases. Doctoral thesis, University of York, UK (1998)