Towards Integrated Engineering of Adaptive Resilient Systems

Elena Troubitsyna
Åbo Akademi University,
Turku, Finland
e-mail: Elena.Troubitsyna@abo.fi

***Abstract*— Resilience is an ability of a system to deliver trustworthy services despite changes. It is a much sought after property in a wide range of applications. However, currently, development of resilient adaptive systems constitutes a major engineering challenge due to a diversity of methods and tools used in the development and a lack of support for efficient information engineering. In this paper, we discuss the challenges engineering resilient adaptive systems. We propose the Problem-Design-Exploration framework as a model of the adaptive service development process and define the key concepts supporting multi-view engineering. Moreover, we discuss the advantages of the Open Services for Lifecycle Collaboration (OSLC) as a technology enabling integrated information engineering for resilient adaptive systems.**

Keywords-resilience; adaptability; changes; evolution; integrated engineering environment.

I. INTRODUCTION

Resilience is an ability of a system to deliver trustworthy services despite changes [1]. It is a much sought after property in a wide range of applications. Resilience is an evolution of the dependability concept [2] that puts an emphasis on the ability of a system to adapt to changes. However, currently, development of resilient adaptive systems constitutes a major engineering challenge [3]. Firstly, the existing development methods are unable to efficiently and confidently cope with the overwhelming system complexity and deliver required assurance of system trustworthiness. Secondly, they do not provide efficient platform for integrating changes in the system design. Finally, they give a rather limited support for innovation and experimentation, i.e., do not allow the designers to assess the impact of changes on system behavior with high productivity and confidence.

In this paper, we discuss the challenges in creating an efficient environment for engineering resilient adaptive systems. We explore the challenges in adapting to changes of different nature and introduce the Problem-Design-Exploration framework [4] - [8] as a model for the development process of resilient adaptive systems. We propose the fitness criteria that can be used to assess how adaptation to a change impacts different resilience attributes.

Since resilience is a multi-facet characteristic, diverse engineering tools are used for such an assessment [1]. We discuss the problem of tool integration and demonstrate how Open Services for Lifecycle Collaboration framework [9] can facilitate creation of an integrated engineering environment.

We believe that the problems discussed in this paper constitute the important challenges in the area of adaptive resilient systems engineering.

The paper is structured as follows: in Section II, we introduce the concept of resilience. We show its connection with the dependability concept and discuss the role of changes. In Section III, we introduce the Problem-Design-Exploration framework and define the fitness criteria relevant for the design of resilient system. In Section IV, we discuss the tool integration problem. In Section V, we outline the benefit of OSLC as a technological enabler of the integrated development. Finally, in Section VI, we overview the related work and conclude.

II. RESILIENCE AND ADAPTABILITY

Resilience is an ability of the system to persistently deliver its services in a dependable way despite changes [1]. The concept of resilience is an evolution of the concept of dependability – a system property to deliver services that can be justifiably trusted [2]. Dependability is a multi-facet system characteristic that includes the following attributes:

- *availability* is the ability of the system to provide service at any given instance of time;
- *reliability* is the ability of the system to continuously provide correct service over a given period of time;
- *safety* is the ability of a system to deliver service under given conditions without catastrophic consequences to its user(s) or environment;
- *integrity* is the absence of improper system alteration;
- *maintainability* is the ability of a system to be restored to a state, in which it can deliver correct service;
- *confidentiality* is the absence of unauthorized disclosure of information.

Faults of different nature might jeopardizes dependability by propagating to the system or service interface level and, as a result, introduce undesirable deviations in service provisioning.

Engineering of resilient systems relies on four main techniques: *fault prevention*, *fault removal*, *fault forecasting* and *fault tolerance* [1] [2]. *Fault prevention* is a set of techniques aimed at preventing introduction of faults during the development process. It relies on formal and structured techniques aiming at ensuring high quality of the system and spotting problems in the design before the system becomes operational.

Fault removal techniques are used to identify and remove errors in the system. The activities of fault removal process include system verification as well as corrective and preventive maintenance of the system. *Fault forecasting* aims at predicting and evaluating the impact of fault on the system behaviour. It might be performed qualitatively or quantitatively. The qualitative assessment aims at identifying and classifying failures as well as defining combinations of faults that may lead to a system failure. The quantitative analysis is performed to assess the degree of satisfaction of the different attributes of dependability.

Finally, *fault tolerance* techniques aim at ensuring that the system continues to deliver its services or behaves predictably even in presence of faults.

Several decades of research have resulted in creating a solid body of techniques for engineering dependable systems. Majority of these techniques rely on assumption of exhaustive knowledge of system and its environment behavior, i.e., are static by nature. However, currently it is widely recognized that changes are inevitable and hence, the systems should be able to adapt to them while remaining dependable, i.e., be resilient.

The changes with which the system should be able to cope might be external, i.e., in the operating environment of the system or internal ones. In general, the changes can be classified according to their character as follows:

- *nature:* functional, environmental or technological;
- *prospect:* foreseen, foreseeable, unforeseen (or drastic) changes;
- *timing:* short term (e.g., seconds to hours), medium term (e.g., hours to months) and long term changes (e.g., months to years).

The changes cause continuous system evolution. The evolutionary development approach is supported by agile development model, Scrum development approach as well as DevOps. All these models emphasis the need for iterative development and continuous experimentation with the system under construction. Therefore, engineering of adaptive resilient systems provide a powerful support for change management, continuous evolution and experimentation. That requires an integration of current approaches to engineering dependable systems into a highly dynamic engineering environment facilitating modelling, design and assessment of resilient systems as well as supporting a novel iterative model of development process.

In the next section, we propose the Problem-Design-Exploration framework as a model for the development of adaptive resilient systems and demonstrate how to tailor to address various aspects of resilience.

III. PROCESS OF ENGINEERING ADAPTIVE RESILIENT SYSTEMS

The Problem-Design Exploration Model [4] proposes to model design process as two interacting evolutionary domains – problem space *P* and solution space *S*, as shown in Figure 1. The clear distinction between problem and solution spaces is supported in analytical [5], empirical [6] and prescriptive [7] research. The problem space contains mental representations of the developer's interpretation of the requirements" and "the design space" contains mental representations of the developer's specific solutions [8].

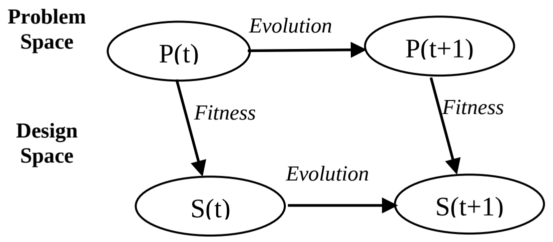

Figure 1. Problem-Design-Exploration Process.

The exploration process shown in Figure 1 has the following characteristics:

1. It is carried in two distinct search spaces: Problem Space and Design Space.
2. These state spaces interact in real time.
3. The horizontal movement represents is an evolutionary process such that
   a. Problem space P(t) evolves to P(t+1), P(t+2) etc.,
   b. Solution space S(t) evolves to S(t+1), S(t+2), etc.
4. The diagonal movement represents a process where goals lead to solution. It exists in two variants: "*Problem leads to Solution"* (downward arrow) or "*Solution refocusses the Problem"* (upward arrow).

The problem space P(t) is the design goal at time *t* and S(t) is the solution space, which defines the current space for the design solutions. The solution space S(t) provides not only a state space where a design solution can be found, but it also prompts new requirements for P(t+1), which were not in the original problem space, P(t). This is represented by the dashed upward arrow from design space S(t) to problem space P(t+1). The upward arrow is opposite: S(t) becomes the goal and a "*search*" is carried out in the problem space, P(t+1), for a "*solution*". This iterative relationship between problem space and design space evolves over time.

The Problem-Design-Evolution framework fits the main requirement for a development model of resilient adaptive systems because it explicitly supports evolution. Indeed, both the problem space and the solution space co-evolve simultaneously as a result of exploration. The basis for co-evolution is to consider the representation and application of the fitness function so that the problem definition can change in response to the current solution space.

To tailor the Problem-Design-Evolution model to engineering of resilient adaptive systems, we need to understand how changes affect the main design objectives. The design objectives are defined by the dependability attributes. Therefore, we should devise the guidelines to be followed while assessing the impact of changes and creating a corresponding artefact in the design space. These guidelines are defined as the fitness criteria.

Table 1 presents the examples of the fitness criteria that might be evaluated while introducing changes of different nature in the design of resilient services.

The Problem-Design-Evolution paradigm provides us with a suitable general model of the process of adaptive system development. However, we also need to “zoom into” the development process and address the problem of integration. Indeed, a variety of methods and tools are used to achieve different design objectives. Therefore, to support the process of designing resilient adaptive services, we need to create an integrated development environment that establishes and information continuum between diverse methods and tools. The problem is traditionally addressed by the Application Lifecycle Management – the concept that we study next.

IV. Integrated Engineering Environment

Application Lifecycle Management (ALM) is a concept that aims at studying “The coordination of development lifecycle activities, including requirements, modelling, development, build and testing, through:

1. enforcement of processes that span these activities;
2. management of relationships between development artefacts used or produced by these activities; and
3. reporting on progress of the development effort as a whole” [10].

The term artifact broadly refers to any item (requirement, code, model, test case) produced during the development of software. ALM often seen as a concept that tries to syncronise all the lifecycle activities instead of focusing on any specific lifecycle activity [10].

The concept of ALM is still rather new and lacks well-established definition. In this paper, we focus on the technological aspects of ALM – the tool integration.

Tool integration is a rapidly growing interdisciplinary research area. It is a cross-road between Software engineering, Systems engineering, Human-Machine interactions and Economics. The tool integration discussion was originated in STONEMAN report [11] where among the other Buxton introduces the notion of integrating tools throughout a software project life-cycle.

The essence of tool integration was defined in the seminal paper by Wasserman [12]. He introduced the following 5 types of tool integration: Control, Data, Platform, Presentation and Process Integration.

Control Integration is the ability of tools to notify each other of events and activate each other under program control.

Data Integration is the ability of tools to share data with each other and manage the relationships among data objects produces by each other.

Platform Integration is a set of system services that provide network and operating systems transparency to tools and tool frameworks.

Presentation Integration refers to the set of services and guidelines that allow tools to achieve a common representation from the user’s perspective.

Process integration defines linkage between tool usage and the software development process. Usually it tries to tie process integration to the definition and integration of process models.

From the ALM point of view, tool integration should therefore produce integrated environments that support the entire software development lifecycle. According to Pederson [13] an integrated environment allows the users easily move from one function to another without having to work with multiple, disconnected tools and manually integrate data between these tools.

It is easy to observe that the evolutionary aspect will result in creating various dynamically changing interdependencies and data. Now, we discuss the technological platform enabling creation and maintenance of such an integrated environment.

We can now define the requirements to integrated environment for engineering resilient adaptive systems as follows:

1. The integrated engineering environment should be non obtrusive and support heterogeneous design space.
2. The environment should allows the designers to continue to use their native tools regardless whether they are open source or proprietary
3. The environment should enable global traceability and querying of information by different engineering teams. The new information, introduced as a result of changes, should be easy to incorporate and link.

In the next section, we introduce OSLC and show that it satisfies the abovementioned criteria.

TABLE I. EXAMPLES OF FITNESS CRITERIA.

	Functional	Environmental	Technological
availability	Can new functionality result in an interruption of a service?	Can the system cope with peak loads?	How new platforms affect performance?
reliability	Does new functionality reduce the level of redundancy?	Can the system maintain reliable operation under the stress conditions?	Does the changed platform increase redundancy?
safety	Does new functionality expand safety kernel?	Which safety mechanisms are affected the by change?	Does new technological platform allow for the use of existing safety mechanisms?
integrity	Does new functionality requires weakening access policy?	Does the new environment introduce different data handling mechanisms/policy?	Does the new platform allows for the same degree of data protection?
maintainability	Can the relationships between the new and existing functions be properly documented and observed?	Which maintainability requirement will be introduced in the new environment?	How the existing maintainability routine will be affected?
confidentiality	Does new functionality increases openness of the system?	How can confidentiality be preserved if new access channels are introduced?	Does new platform introduced any additional vulnerabilities?

V. INTEGRATED INFORMATION ENGINEERING

OSLC [9] is an open community, whose main goal is to create specifications for integrating tools, their data and workflows in support of end-to-end lifecycle processes. OSLC is organised into workgroups that address integration scenarios for individual topics such a change management, test management, requirements management and configuration management. Such topics are called *OSLC domains*. Each workgroup explores integration scenarios for a given domain and specifies a common vocabulary for the lifecycle artefacts needed to support the scenarios. OSLC has received a notable industrial uptake.

Essentially, OSLC specifications focus on defining how the external resources of a particular tool can be accessed, browsed over, and specific change requests can be made.

OSLC does not aim at standardising 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 in a collaborative way.

To ensure coherence and integration across these domains, each workgroup builds on the concepts and rules defined in the OSLC Core specification. The OSLC Core specifies the primary integration techniques for integrating lifecycle tools. This consists mostly of standard rules and patterns for using HTTP and RDF that all the domain workgroups must adopt in their specifications.

OSLC is based on the W3C Linked Data. The four rules of linked data introduced by Berners-Lee [14] are as follows:

- 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.

In OSLC, each artefact in the lifecycle -- for example, a requirement, defect, test case, source file, or development plan and so on -- is an HTTP resource that is manipulated using the standard methods of the HTTP specification (GET, PUT, POST, DELETE).

According to the third rule of linked data, each resource has an RDF representation. OSLC mandates RDF/XML, which is the most widely adopted RDF notation, but can have representations in other formats, like JSON or HTML.

The OSLC Core specification defines a number of simple usage patterns of HTTP and RDF and a small number of resource types that help tools integrate and make the lifecycle work. The OSLC domain workgroups specify additional resource types specific to their lifecycle domain, but do not add new protocol.

OSLC defines the concept of ServiceProvider to allow applications to expose their external resources for integration scenarios. ServiceProviders answer two basic questions, which are:

- To which URLs should one POST to create new resources?
- Where can one GET a list of existing resources?

A ServiceProvider is intended to represent a "container" of resources that is hosted by a tool, not the tool itself. A single instance of a tool will typically host multiple ServiceProviders, for example one for each "project" or "product".

ServiceProvider is the central organising concept of OSLC, enabling tools to expose resources and allowing consumers to navigate to all of the resources, and create new ones.

Two fundamental properties of a ServiceProvider are given below:

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

ServiceProviders have a third important property -- dialog -- that is the foundation of the second major OSLC integration technique based on invocation of HTML web user interface dialogs of one tool by another.

There are three different approaches to implementing an OSLC provider for software:

- Native Support approach is to add OSLC support directly into the application, modifying whatever code is necessary to implement the corresponding OSLC specification.
- Plugin approach is add OSLC support to the application by developing code that plugs-in to the application and uses its add-on API.
- Adapter approach is to create new web application that acts as an OSLC Adapter, runs along-side of the application, provides OSLC support and "under the hood" makes calls to the application web APIs to create, retrieve, update and delete resources.

The Native approach allows tool vendors to add the OSLC support to their own products. The Plugin and Adapter approaches are best suited for adding OSLC support to the tools that have been bought from a tool vendor or obtained from an open source project.

Eclipse Lyo is an SDK to help the Eclipse community adopt OSLC specifications and build OSLC-compliant tools.

Lyo OSLC4J is a Java toolkit for building Open Services for Lifecycle Collaboration providers and consumers. It includes:

- annotations to decorate Java objects with OSLC attributes;
- annotations to assist with resource preview UIs;
- built-in support for service provider and resource shape documents;
- libraries to simplify service provider and consumer development;
- Tests for the sample applications to complement the Lyo OSLC Test Suite.

We argue that OSLC satisfies the criteria defined in Section V for an integrated engineering environment for designing resilient adaptive systems. Firstly, it is non-obtrusive because it does not enforce any standards on the engineering tools. Secondly, it allows the designers to continue to use their native tools and smoothly introduce the facilities for linked data. Thirdly, it support highly dynamic information creation and management via support of linked data.

VI. Related Work and Conclusions

Tool integration facilitates a productive development environment by allowing the user to launch tools and transfer information easily between different tools. Booch and Brown [15] introduced an interesting vision of a ‘frictionless surface’ provided by Collaborative Development Environments. They argue that such environments can remove the points of friction in the daily life of the developer that hinder effective operation. These friction points relate to issues, such as insufficient work product collaboration and problems maintaining effective group communication, including knowledge and experience, project status and project memory.

In his recent paper [16], Ralph introduces a further development of the Problem-Design-Exploration framework -- The Sensemaking-Coevolution-Implementation Theory of software design. In this theory, he aims at blending the boundaries between the problem and design space. It is an interesting theory that fits the novel trends in software development, such as DevOps. We are planning to investigate the use of this theory in the development of resilient adaptive systems in our future work.

The notion of resilience is a subject of active research discussions. Among the most prominent initiatives that contributed to defining the concept of resilience and taxonomy of related terms are the projects ReSIST [17] and Resilinets [18]. In our paper, we rely on the definitions introduced in these projects.

The concept of resilience addresses a wide variety of issues in system design [18]. Therefore, an integration various design methods and tools is especially interesting for resilient systems engineering. The problem of integration has been explored in the context of formal modelling and verification of safety-critical and fault tolerant systems. In particular, [19], [20] and [21] address an integration of safety analysis into formal system model.

In this paper, we discussed the problems in establishing an integrated environment for engineering resilient adaptive services. We introduced the Problem-Design-Exploration framework as the model of engineering for resilience. The model is targeted towards supporting continuous experimentation and introducing changes in the design. We defined the fitness criteria, which serve as guidelines while assessing the impact of changes on resilience and devising the suitable design solution.

Resilience introduces multiple, sometimes conflicting objectives in service design. Since diverse methods and tools are used to achieve them, it is important to provide the designers with a powerful platform for integrated engineering. We discussed the problem of tool integration and identified the need to support dynamic data as the main requirement for the technological support. We argued that OSLC provide us with an adequate technological support for creating and integrated engineering environment and enables non-intrusive integration that supports experimentation.

ACKNOWLEDGMENT

This work is supported by the Finnish National Program Need for Speed http://www.n4s.fi/en/.

REFERENCES

[1] J. C. Laprie, “From Dependability to Resilience,” In 38th IEEE/IFIP Conference On Dependable Systems and Networks, IEEE Computer Press, pp. G8-G9, 2008.

[2] J. C. Laprie, Dependability: Basic Concepts and Terminology. New York, Springer-Verlag, 1991.

[3] Top Challenging Issues for Software Development. [Online]. Available from http://www.iaria.org/conferences2013/ filesICSEA13/. 01.05.2015.

[4] M. Maher, J. Poon, and S. Boulanger, “Formalising design exploration as co-evolution: a combined gene approach”, Preprints of the Second IFIP WG5.2 Workshop on Advances in Formal Design Methods for CAD, 1995, pp. 1–28, doi=10.1.1.56.4459&rep=rep1& type=pdf.

[5] K. Dorst and N. Cross. “Creativity in the design process: co-evolution of problem–solution,” Design Studies, 22(5), pp. 425–437, Elsevier 2001.

[6] J. S. Gero and T. McNeill, “An approach to the analysis of design protocols”, Design Studies. 19 (1), pp. 21–61, Elsevier, 1998.

[7] P. Checkland, Systems Thinking, Systems Practice, Wiley, 1999.

[8] S. Purao, M. Rossi, and A. Bush, “Towards an understanding of problem and design spaces during object-oriented systems development”, Information and Oranisation, 12 (4) , pp. 249-281, Pergamon, 2002.

[9] OSLC: (Open Services for Lifecycle Collaboration.) [Online] Available from http://open-services.net/ 01.05.2015.

[10] C. Schwaber, “The Changing Face of Application Life-Cycle Management”, Forrester Research Inc., White Paper, August 2006. [Online] Available from www.serena.com /docs/repository/alm/changing-face-applic.pdf 01.05.2015.

[11] J. N. Buxton. “STONEMAN, Requirements for Ada Programming Support Environments,” Technical Report. Department of Defense. 1980.

[12] A. I. Wasserman, “Tool Integration in Software Engineering Environments”. In Software Engineering Environments: International Workshop on Environments, Chinon, France, September 1989, ISBN:3-540-53452-0.

[13] J. Pederson, “Creating a tool independent system engineering environment”, IEEE Aerospace Conference, March 2006.

[14] W3C web site. [Online] Available from http://www.w3.org/DesignIssues/LinkedData.html 01.05.2015.

[15] G. Booch and A. Brown, “Collaborative development environments”, Advances in Computers, Vol. 59, Academic Press. 2003. doi=10.1.1.84.3292

[16] P. Ralph, “The Sensemaking-Coevolution-Implementation Theory of software design,” Science of Computer Programming, v.101, pp.21-41, April 2015.

[17] ReSIST project. [Online]. Available from http://www.resist-noe.org/ 01.05.2015.

[18] RESILINETS project [Online]. Available from https://wiki.ittc.ku.edu/resilinets/ Accessed 01.05.2015.

[19] E. Troubitsyna, “Elicitation and Specification of Safety Requirements”, The Third International Conference on Systems (ICONS’08), IEEE Computer Society, April 2008, ISBN978-0-7695-3105-2.

[20] K. Sere, K. and E. Troubitsyna, “Safety Analysis in Formal Specification”, World Congress on Formal Methods 1999, LNCS 1709, pp. 1564-1583, Springer, 1999, ISBN:3-540-66588-9.

[21] I. Lopatkin, A. Iliasov, A. Romanovsky, Y. Prokhorova, and E. Troubitsyna, “Patterns for Representing FMEA in Formal Specification of Control Systems”. Proceedings of the 13th IEEE International High Assurance Systems Engineering Symposium (HASE 2011), pp. 146-151, IEEE, 2011, ISBN 978-1-4673-0107-7.

[22] A. Iliasov, A.Romanovsky, L.Laibinis, E.Troubitsyna, and T.Latvala, “Augmenting Event-B modelling with real-time verification,” FormSERA@ICSE 2012, pp. 51-57, IEEE Computer, 2012.

[23] E. Troubitsyna, “Reliability assessment through probabilistic refinement,” Nordic Journal of Computing 6 (3), 320-342, 1999.