A Vision on a New Generation of Software Design Environments

(Empirical Theory)

Michel R.V. Chaudron, Rodi Jolak
Joint Department of Computer Science and Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden
{chaudron, jolak}@chalmers.se

***Abstract*— In this paper we explain our vision for a new generation of software design environments. We aim to generalize existing software development tools in several key ways – which include: integration of rigorous and informal notations, and support for multiple modes of interaction. We describe how we can consolidate the environment by integrating it with other software engineering tools. Furthermore, we describe some methods which could permit the environment to provide a flexible collaborative medium and have a practical and inspiring user experience.**

***Index Terms*—software engineering, modeling tools, collaborative design, IDE.**

I. Introduction

Software systems have an important role in the technological evolution which we are witnessing nowadays, and as a consequence, software systems are becoming more and more complex. The increasing complexity of such systems has raised some certain challenges, such as e.g. design uncertainty and run-time changes, making it difficult to meet continuous customer demands for a better software quality [7, 8]. Software modeling plays a pivotal role in software development. Models present an understandable description of complex systems at several levels of abstraction and from a diversity of perspectives. Furthermore, they provide an essential medium matching between problem and software implementation by describing user's needs and prescribing the product to be developed.

Software modeling is a highly complex and demanding activity [21]. Software designers often use software modeling tools to perform a software design. There are two dimensions of these tools that we will challenge in this paper: i) the formality of the notation used, and ii) the modes of interaction supported by the tools. Next, we briefly explain our views on these dimensions.

First, we classify modeling tools into two groups: informal and formal. We mean by informal any tool that supports informal design in the sense that it does not constrain the notation used. Examples of such tools are whiteboards, paper and pencil. Whiteboards are often used to collaboratively sketch software modeling ideas, discover architectural solutions, capture design discussions, etc. [13, 5]. Whiteboards are normally used for sketching when more than two people are involved [2]. Generic diagramming tools such as PowerPoint and Visio are informal in the sense that they do not constrain the notation, but they do provide mature digital editing functionality (move, delete, undo). While on the other hand we mean by formal tools any CASE tool which supports one or more formalized notations. Typical examples are UML CASE tools (like Rational Rose, Enterprise Architect, Visual Paradigm, StarUML etc.). Also for many other modeling languages, tools are often dedicated to a single notation (Archimate for Enterprise Modeling, ARIS-tool for Business Process Modeling, etc.). All CASE tools support mature digital edition functionality.
Table 1 is based on Hammouda [11] and describes some relative advantages of informal and formal modeling tools:

Table 1*:* Relative Advantages of Informal and Formal Modeling Approaches

	Informal	***Formal***
Clarity		High clarity because of strict adherence to syntax
Flexibility	Caters for improvisation of notation.	
Ease of continuous design	In tools based on digital editing, editing (move, resize, delete, undo, etc.) is easier than in sketch-based tools such as whiteboards.	
Ease of learning notation		Formal syntax checking helps in learning the proper syntax.
Intuitiveness of using tool	very simple to use; but limited in functionality	More difficult to learn, but advanced functionalities supported.
Collaboration	Multiple people collabo-rating on a shared design prefer to use informal representations [2].	
Integration	Absence of a formal syntax (and semantics) prohibits exchange of designs.	Formal syntax allows a formal representation of the design that can be exchanged with other tools.

We envision our environment to have the advantages of both formal and informal tools.

The second dimension we challenge is that of the modes of interaction supported by modeling tools. Oviatt and Cohen [19] illustrated the importance of multimodal systems in reshaping daily computing tasks and predicted their future role in shifting the balance of human-computer interaction much closer to the human. Based on that, we want to support multimodal communication interactions by recognizing touch, voice and gesture for a more intuitive software modeling experience.

Summarizing, the following questions are addressed:
Q1: How can we achieve an integrated design environment having the power of both formal and informal tools?
Q2: How can we make modeling tools easier to use and more productive?

- How can tools better support tasks of software developers? Our focus is on tasks related to the design of systems.
- Which sources of knowledge and information can be connected to provide information needed at easy disposal (right information at the right moment, place and format)?

The paper is organized as follows: in section two we describe the related work. Section three illustrates our vision. Finally we conclude and discuss ideas for future work in section four.

II. Related Work

Many empirical studies of formal tools usage have pointed out that software designers consider these tools overly restrictive and this often lead to poor utilization [13, 6]. By doing a HCI study, Plimmer et al. revealed that in early software design phases, the designers prefer to sketch by hand rather than using a keyboard or a mouse [20]. Whiteboards support informal design. They are frequently used by software designers during project meetings to sketch ideas and thoughts about system goals, requirements and design solutions [13, 5].

Electronic interactive whiteboards offer the potential for enhanced support by allowing the manipulation of the content, handling of sketches, and doing collaborative distributed works. Mangano et al. [15] identified some behaviors that occur during informal design. They implemented an interactive whiteboard system to support these behaviors, and identified some ways where interactive whiteboards can enable designers to work more effectively. The main goal of the system that they implemented, called Calico, is to maintain fluidity and flexibility allowing software designers to focus on the content of their sketches rather than the tool used to make it. Mangano et al. revealed a number of weaknesses in Calico ranging from usability issues to challenges inherent to interactive whiteboards. In particular, designers reported that some gestures were not rapidly interpreted, and the large e-whiteboard diminished the quality of their handwriting, forcing them to write slower or larger. We want to support software design not only with interactive whiteboards, but with PCs, touch pads and smart phones.

Baltes and Diehl [2] investigated the use of sketches in software engineering activities by conducting an exploratory study in three different software companies. Their results showed that the majority of the sketches were informal, and the purposes of sketches were related to designing, explaining, or understanding. Baltes and Diehl also showed that the sketches were archived digitally for re-visualization and future use. Like us, they think software design tools should enable informal design sketching.

Wüest et al. [22] stated that software engineers often use paper and pencil to sketch ideas when gathering requirements from stakeholders, but such sketches on paper often need to be modelled again for a further processing. A tool, FlexiSketch, was prototyped by them to combine freeform sketching with the ability to interactively annotate the sketches for an incremental transformation into semi-formal models. The users of FlexiSketch were able to draw UML-like diagrams and introduced their own notation. They were also able to assign types to drawn symbols. Users liked the informality provided by the tool, and stated that they would be willing to adopt it in practice. FlexiSketch runs on tablet computers. It is a single user tool, and does not support collaborative sketching. We think running FlexiSketch on electronic whiteboards could allow for multi-user input and facilitate collaboration. We also think that software design tools should be able to support sketch recognition and its transformation into a kind of formal diagrams as well as allow the exportation of such diagrams to other programs e.g. CASE tools. FlexiSketch Team [23] is an extended version of FlexiSketch, which supports a collaborative sketching via ad-hoc local Wi-Fi network, but it does not allow for a distributed collaboration.

Chen et al. have developed SUMLOW [4]. A sketching-based UML design tool for electronic whiteboard technology. It allows to preserve hand drawn diagrams and supports for manipulation of the diagrams using pen-based actions. UML sketches can be formalized and exported to a 3rd party CASE tool. Their tool does not support design sketching on different platforms like mobiles and tablets. Again as all works previously mentioned, it does not support a collaborative distributed software modeling.

MaramaSketch [10] includes a meta-modeling editor. This editor allows to define a conventional modeling language which is then used to compile a modeling tool for the defined language. However, MaramaSketch needs to create the complete language definition first, and after that, users must strictly follow it. So as a consequence, it prevents any flexible sketching.

Magin and Kopf [14] have created a multi-touch based system allowing users to collaboratively design UML class diagram on touch screens. They have also implemented new algorithms to recognize the gestures drawn by the users and to improve the layout of the diagrams. However, it does not support a remote collaboration, and as they stated, their tool has some usability challenges in creating and editing of sketches, and in the recognition process of hand written text.

In the area of integration of software development tools Open Services for Lifecycle Collaboration “OSLC” [25] is an emerging standard. This standard defines API’s through which development tools can interoperate. OSLC could be a technology that underlies the integration aspects of our vision.

Brosch et al. [3] showed the importance of *model versioning* in enabling efficient team-based development of models. Based on that, we think a version management tool should be integrated within software design environments to track modeling processes and their evolution. There is a fair amount of work ongoing in versioning for software models [1], but none of this has been integrated in mainstream CASE tools yet.

IBM's family of integrated development environments [24] allow for a collaborative software development. In particular, they provide teams with rich capabilities for continuous development, testing, analyzing and optimizing applications. For example, IBM Rational Business Developer is an Eclipse-based environment. It allows complex applications to be modeled graphically. IBM only offers these tools on a commercial basis.

While a comprehensive theory of IDEs does not exist, there are proposals for theoretical models that can serve to support the design of NGDE. Notable approaches are the Cognitive Dimensions approach [9] and the 'Physics of Notations' [16].

III. Our Vision

In this section we present our vision for a more intuitive, inspiring and efficient tool to support exploratory and collaborative software modeling. In particular, we are going to describe some ideas which we consider to be relevant to achieve such a tool. We will refer the next generation software design environment as NGDE.

One first point is that in practice *modeling* and *designing* go hand-in hand. A modeling language provides the notation in which to express a design. As such a modeling language is a part of the toolbox that a designer uses in the creative process of designing a solution. Currently, most case tools are modeling (or even diagramming) tools. Instead, the next generation of tools should take a holistic view on supporting all design activities in which developers are engaged.

Next, we discuss several key aspects in which NGDE can provide better support for the design activities of software developers:

A. Informal versus Formal Notation

Informal tools like whiteboards provide a useful mean for flexible collaborative modeling. In fact, software designers can easily create/extend diagrams, add comments and highlight some parts of their sketches. Even more, they can sketch diagrams of multiple notations without following any restrictive rule imposed by the formality of a one modeling language or syntax. However, re-modeling is a difficult and a time consuming task. Moreover, whiteboards do not support data persistency and transference. Formal tools like CASE tools are restrictive in that they require designers to use some specific notations for modeling. We propose that NGDE tools should support the mix of both formal and informal modeling notations that designers use. Ideally, NGDEs should maintain the characteristics of formal tools in their support of design transfer and persistency. To support informal modeling, tools should allow designers to create different types of diagrams on the same canvas [15]. Furthermore, they should not constrain designers to sketch only some specific notations. For instance, designers should be able to draw and create a variety of sketches e.g. use case diagrams, workflows, arrows, state charts, data models, etc. In general, NGDE should enable designers to add domain specific icons or notations. These kind of notations help to better describe a specific domain problem.

NGDEs should keep from existing editors the abilities to organize diagrams by moving, resizing, grouping and separating sketches, and the ability to modify and evolve sketches.

NGDEs should have the ability to transform sketches into formalized content by providing a recognition unit. This enables designs be formally represented and hence easily exchanged with other software tools. Furthermore, they should have the power of formal tools in maintaining and transferring the designs for further processing tasks.

B. Integration

In their daily work, many software designers work concurrently on different artefacts: changes to a design and followed closely by changes in code and changes in requirements. Unfortunately, with current tools the developer needs to switch to different applications. We propose to design an integrated environment. This does not need to become a 'Swiss army knife' that integrates all functionality in a single tool. Integration of development tools needs to address a shared data model of software development artefacts, but also a shared UI-concept.

The goal behind the idea of integrating other software management tools within a software modeling tool in a 'one stop' environment is to provide effective support for an effective software modeling process.

Software requirements, for instance, evolve over time and they are frequently subject to changes during initial development and later on to delivery. Designers generate many ideas in order to understand a problem and find a solution for it. These ideas are often compared, modified, evaluated and enriched as the modeling process evolves. In order to realize how useful could be having a trace of the requirements within a software modeling tool, let's think about the following scenario: a group of software designers start to document and gather needs of a specific software product, after that, they proceed to create a first design of the product using for example a traditional whiteboard or a CASE tool. Let's also consider that the software needs, written on a simple paper sheet, are given by a client. In both cases, using either a whiteboard or a CASE tool, software designers have to meet again and again whenever new requirements come out or having earlier requirements exposed to changes. This is a time consuming task and especially when designers have to recollect their designs and re-model them according to the new set of requirements.

Here, a NGDE should be able to handle notes written on paper as an artefact. Ideally, this paper is not only stored as a (jpeg) picture, but contents are (partially) recognized and can be transformed into formal concepts.

Modeling involves several stakeholders who conduct the creation of the design in elicitation and formalization phases, and since requirements evolve over time, modeling usually comprises several iterations of elicitation and formalization resulting in an evolving process [12]. Therefore, we think that software modeling tools should be integrated with other software engineering tools to deliver a 'one stop' environment capable of addressing and supporting issues like requirements analysis and management, programming and coding, generation of bug reports, performance and security analysis, testing, versioning, etc. This is in line with the *Twin Peaks* model by Nuseibeh [18]. This model states that in reality requirements and designs develop progressively in concert and mutually influence each other.

Of course, source code is also essential in the development of software. Hence throughout the development process, models and code must be combined – in the sense that developers must be able to view them side by side and jump between editing one and keeping the other synchronized. One challenge is the linking between models and code. It is typical that models are used at various and varying levels of abstraction. Models start out at a high level of abstraction and gradually get refined by adding details.

The integration of code and models also raises the question of *debugging*. While we can always use the IDE's debugging functionality, it usually does not make sense to debug the generated code itself. The developer is more familiar with the model than the code, and if a problem is found, we want to correct it in the model not the code. Modeling tools should support the usage of models in debugging the functionalities at a higher level of abstraction in order to know if the application is doing what it was designed to do.

Modeling tools should support multiple people and teams working on the same design from different locations. In particular, they should provide means to achieve an effective coordination between geographically distributed project members. *Version management*, for example, should be adapted to support collaborative modeling and design. We think modeling tools should provide a repository to keep track of the version history of the models stored in it, as well as provide the ability to observe who is changing what artifacts in the environment. Following the checkout-update-commit interaction paradigm, the repository would offer an interactive model merging tool to resolve conflicts when two users change the same model data. It would increase the potential for parallel and distributed work, improve ability to track and merge changes over time and automate management of revision history. It would also allow multiple designers to work with the same models concurrently, supporting tight collaboration and a fast feedback loop.

Finally, and to provide an effective communication medium for a geographically distributed software modeling teams, we propose integrating *social media* and *chat* tools within the modeling tools. The goal is to make software modeling activity more efficient. For instance when two designers from two different locations want to exchange ideas about a specific design, they could make use of the integrated social media or chatting tools to do such a task, and of course, this reduces the time spent handling emails. In fact, these communicative facilities play an important role in establishing a basis for discussions and negotiations, information exchanging, and sharing data e.g. images, videos, etc.

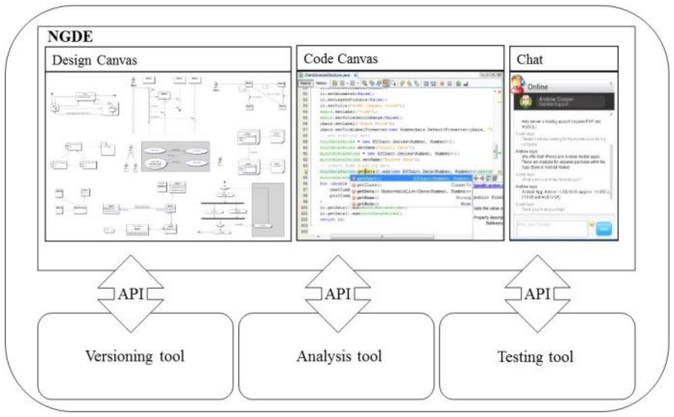

Figure 1. An illustrative view of NGDEs and the integration mechanism.

In general, modeling tools should be "open" providing various integration mechanisms among the different platforms. In particular, they should have programmable interfaces, import/export formats, and enable plug-ins for integration (see Figure 1), thus offering an ideal support for team work, and letting the overall development process becomes easier and faster. On this topic, we will explore to what degree OSLC helps solve this issue.
In summary, integration has several facets:

- Integration of rigorous and informal notation
- Integration of different tools for different activities in the software development lifecycle.
- Integration of (machine and human) knowledge sources.

C. Usability, Interaction and Collaboration

The usability of current CASE tools is a common source of criticism. One key aspect of usability is the manner in which humans interact with the system. Currently this is by using the keyboard and the mouse – essentially we are using the computer as an enriched typewriter. Other modes of interaction have gained popularity because of their intuitive nature and these should also be employed in the area of software design environments.

Touch-based interfaces have become common in tablets and smart phones, and also smart-whiteboards with touch-based interaction have been introduced in class room environments. This introduces the dimension of modality of interaction. While traditional interaction with a computer is via a keyboard (and mouse) currently there are many options available: voice, touch, gesture, eye-focus or laser-focus as pointing. Computers are capable of handling such new types of inputs. This will make interacting with NGDEs much more intuitive.

Whiteboards allow multiple users to draw software models together. In order to emulate this informality in our environment, we propose enabling it to support a collaborative multi-touch modeling. Multi-touch is an interaction technique that's permits the manipulation of graphical entities with several fingers at the same time. Making use of multi-touch screens allows users to design complex diagrams simultaneously by performing simple intuitive touch gestures to draw their part of the diagram.

Such joint drawing sessions typically also trigger a lot of discussion. Such discussion may contain valuable information about a design, such as e.g. its rationale. However, traditional tools do not capture the discussion. NGDE can be equipped with microphones and also record the spoken discussion. New challenges in this area will be to search through this type of recorded spoken text. Inspiring work in this direction is the work by Nakakoji et al. [17]. They describe a system that makes video-recordings of the design discussion in front of the whiteboard. Their system does automated voice recognition and produced a textual transcript of the discussion. Also, their system offers a way of navigating through the discussion using a time-line. The recording of discussions is effortless for (i.e. without any explicit action) the developers. In such a way NGDE can relieve the developer by lowering the cognitive attention needed for inputting relevant information into the system and linking it to related artefacts.

A task that is commonly forgotten is that of design review. Designers frequently review the design progress in order to know what is done and what they have still to do. For that, NGDEs should support design review "on the fly", as well as in detail whenever designers want to add some additional items to their previously sketched design. One approach in this direction is offered by the recent version of the Altova UModel tool. This tool provides a layering-mechanism: here review comments are part of one layer and the software design is part of another layer. The user can then select to see combined layers or layers in isolation.

D. Multiplatform

Rather than tying the development process down to any specific hardware environment, the NGDE should aim to facilitate multiple platforms: smart whiteboards, tablets, smart phones, and traditional desktops and laptops. This increases the accessibility of the environment. Also, in classroom environments, this will open up new opportunities for interactive collaborative design.

IV. Conclusion

Other application domains have gone before software development CASE tools and have shown that rich interaction with computer-based systems is enhancing productivity [19]. Next generation software design tools must keep up with this trend and offer improvements over existing tools in the following dimensions:

- Rich support for multiple modes of interaction (touch, audio, video, gesture).
- Support for mixing formal notations with informal notations (e.g. UML diagrams with additional sketching).
- Higher level of integration of tools: on the one hand integrating tools for different development tasks (requirements, testing, coding) and also analysis tools (performance, security). On the other hand, integration of social/organizational sources of knowledge via ((video) chat).
- Rich support for multiple platforms: work does not only happen behind a PC, there is also discussions at the whiteboard and via tablets. NGDE should offer a seamless environment for this.

The design of software design environments should be driven by studying the needs of actual software developers. We consider it very important that more observation studies are performed about the actual tasks that software developers perform. From this we can learn how to best support them.

This paper describes our vision. It is beyond our own capacity to realize this vision. We therefore call on the community to collaboratively work on the next generation of software design environments.

References

[1] Altmanninger, Kerstin, Martina Seidl, and Manuel Wimmer. "A survey on model versioning approaches." *International Journal of Web Information Systems* 5, no. 3 (2009): 271-304.

[2] Baltes, Sebastian, and Stephan Diehl. "Sketches and diagrams in practice." In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 530-541. ACM, 2014.

[3] Brosch, Petra, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. "An introduction to model versioning." In *Formal Methods for Model-Driven Engineering*, pp. 336-398. Springer Berlin Heidelberg, 2012.

[4] Chen, Qi, John Grundy, and John Hosking. "An e-whiteboard application to support early design-stage sketching of UML diagrams." In *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, pp. 219-226. IEEE, 2003.

[5] Cherubini, Mauro, Gina Venolia, Rob DeLine, and Andrew J. Ko. "Let's go to the whiteboard: how and why software developers use drawings." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 557-566. ACM, 2007.

[6] Damm, Christian Heide, Klaus Marius Hansen, and Michael Thomsen. "Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard." In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 518-525. ACM, 2000.

[7] Fiadeiro, José Luiz. "The many faces of complexity in software design." In *Conquering Complexity*, pp. 3-47. Springer London, 2012.

[8] Filieri, Antonio, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. "Conquering complexity via seamless integration of design-time and run-time verification." In *Conquering Complexity*, pp. 253-275. Springer London, 2012.

[9] Green, Thomas R. G., and Marian Petre. "Usability analysis of visual programming environments: a 'cognitive dimensions' framework." *Journal of Visual Languages & Computing* 7, no. 2 (1996): 131-174.

[10] Grundy, John, and John Hosking. "Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool." In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 282-291. IEEE, 2007.

[11] Hammouda, Imed, Håkan Burden, Rogardt Heldal, Michel R.V. Chaudron, "CASE tools versus pencil and paper: A student's perspective on modeling software design." *EduSymp@MoDELS*. 2014: 21-30.

[12] Hoppenbrouwers, S. J. B. A., H. A. Proper, and Tvd Weide. "Formal modelling as a *grounded* conversation." In *Proceedings of the 10th International Working Conference on the Language Action Perspective on Communication Modelling (LAP05), Kiruna, Sweden. Linköpings Universitet and Högskolan i Borås, Linköping and Borås*, pp. 139-155. 2005.

[13] Lank, Edward, J. Thorley, Sean Chen, and Dorothea Blostein. "On-line recognition of UML diagrams." In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pp. 356-360. IEEE, 2001.

[14] Magin, Michael, and Stephan Kopf. "A Collaborative Multi-Touch UML Design Tool." *Technical reports* 13 (2013).

[15] Mangano, Nicolas, Thomas D. LaToza, Marian Petre, and André van der Hoek. "Supporting informal design with interactive whiteboards." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 331-340. ACM, 2014.

[16] Moody, Daniel L. "The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering." *Software Engineering, IEEE Transactions on* 35, no. 6 (2009): 756-779.

[17] Nakakoji, Kumiyo, Yasuhiro Yamamoto, Nobuto Matsubara, and Yoshinari Shirai. "Toward Unweaving Streams of Thought for Reflection in Professional Software Design." *Software, IEEE* 29, no. 1 (2012): 34-38.

[18] Nuseibeh, Bashar. "Weaving together requirements and architectures." *Computer* 34, no. 3 (2001): 115-119.

[19] Oviatt, Sharon and Philip Cohen. "Multimodal interfaces that process what comes naturally." *Communications of the ACM. Volume 43 Issue 3,* March 2000 Pages 45-53.

[20] Plimmer, Beryl, and Mark Apperley. "Computer-aided sketching to capture preliminary design." In *Australian Computer Science Communications*, vol. 24, no. 4, pp. 9-12. Australian Computer Society, Inc., 2002.

[21] Tang, Antony, Aldeida Aleti, Janet Burge, and Hans van Vliet. "What makes software design effective?" *Design Studies* 31, no. 6 (2010): 614-640.

[22] Wüest, Dustin, Norbert Seyff, and Martin Glinz. "Flexisketch: A mobile sketching tool for software modeling." In *Mobile Computing, Applications, and Services*, pp. 225-244. Springer Berlin Heidelberg, 2013.

[23] Wüest, Dustin, Norbert Seyff, and Martin Glinz. "FLEXISKETCH TEAM: Collaborative Sketching and Notation Creation on the Fly." In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2, pp. 685-688. IEEE, 2015.

[24] IBM Rational Development Family, http://www-03.ibm.com/software/products/en/developer, last visited Sep. 03, 2015.

[25] Open Services for Lifecycle Collaboration "OSLC", an open community building practical specifications for integrating software, http://open-services.net/ , last visited Jul. 16, 2015.