An Evaluation of the DiaSuite Toolset by Professional Developers

Milan Kabáč, Nic Volanschi, Charles Consel

An Evaluation of the DiaSuite Toolset by Professional Developers

Learning Cost and Usability

Milan Kabáč Nic Volanschi
Inria Bordeaux, France
milan.kabac@inria.fr / eugene.volanschi@inria.fr

Charles Consel
Inria Bordeaux & University of Bordeaux, France
charles.consel@inria.fr

Abstract

This paper evaluates a design-driven, tool-based approach, named DiaSuite, dedicated to developing applications involving sensors and actuators. Specifically, we evaluate the usability and the learning cost of DiaSuite as a first step to assess the potential for transferring this technology to the industrial practice of this domain.

We assess the cost of learning DiaSuite by involving four professional programmers in a usability study involving a software engineering task. This experiment brings preliminary evidence that the DiaSuite technology can be used effectively by professional developers after only half a day of training.

We then present qualitative data about the usage and usability of DiaSuite, collected from developers, via questionnaires and interviews. Finally, we discuss lessons learned from this work.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords software engineering tools, domain-specific language, learning cost, usability, sensors and actuators

1. Introduction

This paper is an evaluation of the learning cost and usability of an existing design-driven, tool-based approach, called *DiaSuite* [7], dedicated to developing applications orchestrating sensors and actuators. This evaluation is a first step for assessing the potential for transferring DiaSuite to the industrial practice of this domain.

The evaluation has been carried out in the context of a French collaborative project, called *Objects World*, involving five IoT manufacturers and four research labs [24]. Objects World aims at building a sustainable ecosystem of Internet-of-Things (IoT) stakeholders based upon a nationwide, low-bandwidth radio network, called Sigfox.[1] This network already supports products and services that demonstrate its market potentials. It has been deployed in a number of countries (*e.g.,* France, United Kingdom, Spain, Netherlands) [1, 29]. Sigfox is supported by major players such as Samsung, which integrated the Sigfox network protocol into its ARTIK IoT platform [20].

When the Objects World project was being set up, the companies involved knew from their experience how much software development is a bottleneck to realize the full potential of IoT. In fact, this domain is just emerging from an industrial viewpoint. While extensive work has been devoted to the infrastructure of IoT, such as traffic management and device constraints (*e.g.,* battery life), little effort has been dedicated to the process of developing applications and services [19]. This situation prompted the Objects World consortium to invite a research group specialising in software engineering to participate to the project and bring tools to improve the development process of IoT applications. Hence, our participation to the project.

Our methodology and its supporting toolset, DiaSuite, have been used for developing many successful proof-of-concept applications in several domains: telecommunications [6], building automation [10], avionics [15], software monitoring [12], robotics [13], and assisted living [11]. The variety of domains successfully targeted by DiaSuite demonstrates the broad applicability of the methodology and tools in various kinds of applications involving sensors and actuators. However, most of these applications were developed in a research setting. Although some specifications were fueled by industrial needs, they were implemented by researchers in our group, not by developers from the industry.

As a first step for assessing the potential of industrial transfer of our technology, we evaluated DiaSuite by soliciting professional programmers from the IoT industry in a usability study. In this paper, we present the results of this work. The contributions of this paper can be summarised as follows.

[1] http://www.sigfox.com

- By means of a preliminary usability study, we show that the technology can be efficiently transferred to developers in the IoT industry with only half a day of training.
- Further quantitative and qualitative data are provided, including a usability questionnaire and developer interviews to investigate the perceived utility of the tools.
- We sketch the following steps we are currently taking to consolidate these preliminary results into a more complete evaluation of the DiaSuite toolset, including its impact on programmers' productivity and application quality.

2. DiaSuite

Let us start by introducing DiaSuite and its main concepts. DiaSuite revolves around the Sense/Compute/Control (SCC) architectural pattern [31]. It is dedicated to the design of applications that interact with their physical and software environment using sensors and actuators. This kind of applications covers the IoT domain, but also several other domains such as robotics, avionics, telecommunications, and assisted living. Following the SCC paradigm, applications can be decomposed into three kinds of components:

- *entities* interacting with the environment, either as data sources (for sensing entities) or data sinks (for actuating entities);
- *contexts*: software components refining raw data coming from sensing entities into higher-level "context" information;
- *controllers*: software components fueled by high-level context information and triggering actions on actuating entities.

An application can be viewed as a directed acyclic graph flowing from sensing entities to actuating entities via context and controller components. For instance, the application in Figure 1 records in a log the events coming from two different sensing entities of a connected door lock: the lock-/unlock events and state information, such as the battery level. The Format context is responsible for unifying information coming from heterogeneous devices within a common data structure called an Event. Events created by this context are forwarded to the Logger controller that is responsible for triggering a Log action on a Logger actuating entity, possibly implemented as a service on top of a file system.

Developing SCC applications is complex because of the number of intertwined aspects to be taken into account by the developer, including heterogeneous devices, communication infrastructure, system APIs, synchronisation, and so on. These aspects force the developer to write a lot of boilerplate code to interact with devices, glue code to interface with various underlying communication and OS technologies, technical code for low-level device coordination, etc. Moreover, this manually-written low-level code tends to introduce platform dependencies in application-level code. DiaSuite [7]

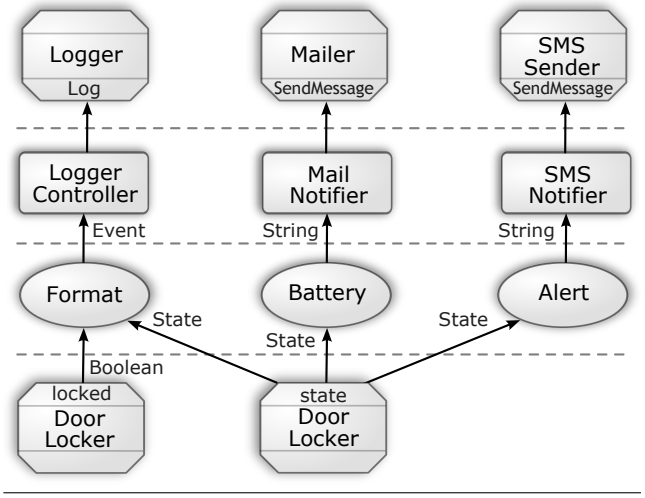

Figure 1: The graphical view of the DoorLocks application.

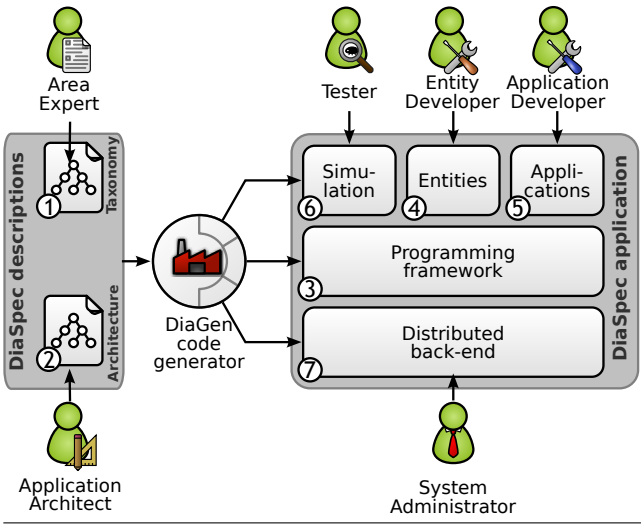

Figure 2: DiaSuite tools support for the SCC application lifecycle.

provides a domain-specific, tool-based development methodology specifically addressing the SCC paradigm; it covers the complete application lifecycle, as shown in Figure 2. DiaSuite provides a declarative design language dedicated to SCC, called DiaSpec, which consists of two layers: the taxonomy layer and the application design layer. Based on high-level declarations, a compiler called DiaGen, generates application-specific support for the subsequent development phases: implementation, testing, deployment, and maintenance. Let us examine the key phases of DiaSuite from design to maintenance.

Taxonomy design. A domain expert uses the taxonomy layer of the DiaSpec language to classify the entities involved in a given application domain (see label (1) in Figure 2). For each entity, either hardware or software, its data sources and actions are declared, as well as attributes such as a name, a

unique identifier or a location. Entities can be grouped into a hierarchy in order to inherit sources, actions, and attributes. This results in a flexible and reusable catalog of entities for a given application domain, abstracting over underlying technologies and implementation details.

Application design. Based on a taxonomy (1), an application architect can use the application design layer of the DiaSpec language to declare the application components and the possible flows of data connecting them together (2), as has been illustrated in Figure 1. DiaSuite applications are reactive, which means that any computation is initially triggered by data coming from some sensing entity, which activates all the contexts connected to it. To respond to its activation, a context can request extra data from other connected entities or contexts. Thus, both push and pull models of reactive applications are covered. Subsequently, controllers may be activated by the propagations along the reactive chain, and may trigger actions on the actuator entities when needed. Using application-specific code generation, DiaSuite ensures that data only flows along the data flow graph declared during application design. In particular, the implementation of a context or controller cannot retrieve data from components that are not connected to it in the architecture definition; nor can a sensing entity or context push a value to a context or controller that is not connected to it in the architecture.

Implementation support. Given a taxonomy declaration (1) and an application design declaration (2), the DiaGen compiler produces a Java programming framework customised for the application. This framework contains an abstract class for each declared component: entities, contexts, and controllers. Abstract classes contain both concrete methods supporting the development (*e.g.,* entity discovery), and abstract methods that must be implemented by developers in a subclass with device-specific behaviour – when subclassing entities (4) – and with application logic – when subclassing contexts and controllers (5). Thus, the generated framework guides the implementation of all the components, by requiring the developer to implement the abstract methods, and facilitates programming via the generated concrete methods.

Testing support. Given a taxonomy declaration (1), DiaGen also produces simulation support (6) for every entity, in the form of a concrete class implementing a "mock" entity, containing methods to simulate each possible activity of a device, such as pushing data from a sensing entity or making some data available for subsequent pulls. This support allows a complete testing of an SCC application before deploying it on a real infrastructure of sensors and actuators. With no code changes in the application, the mock devices can be gradually substituted by real devices. Similarly, support is generated for testing entity implementations (also called drivers) before any application is developed on top of them.

Deployment platform. Finally, given a taxonomy declaration (1) and an application design declaration (2), the backend of the DiaGen compiler produces a runtime layer dedicated to a given deployment platform (7): it can be local (*e.g.,* OSGi) or distributed (*e.g.,* WebServices). This support allows application developers to abstract over how communications between components are implemented. It also allows applications to be deployed in an already running platform, reusing available entities. Thus, the implementation of entities is typically deployed independently of applications: deploying an application simply consists of deploying its context and controller components, reusing the already installed entities, possibly shared with other applications.

Maintenance support. There is no specific code generated by DiaGen for supporting the maintenance of an application. Rather, maintenance is supported by automatically regenerating all the pieces of code mentioned above, whenever the taxonomy or the application design are modified. The separation of generated code, provided as abstract classes, from developer-supplied code, as separate sub-classes, enables a smooth update of the generated code. Typically, following a design change, the developer-supplied code has to be adapted to conform to the regenerated code API. During this maintenance process, developers are guided by the Java builtin type checker that points at the code locations needing changes. Thanks to the integration of DiaSuite into the Eclipse IDE [32], most of the time, the needed changes are available as suggested editing actions (*e.g.,* adding a method parameter or changing a type).

3. Experimental Evaluation

In this section, we present the evaluation of DiaSuite, which was carried out by means of a usability study, questionnaires and interviews. We give details about the different stages of the user study and the results we obtained. Application code, the used questionnaires and the training material are available on the accompanying Objects World website [24].

3.1 Usability Study Definition

The main goal of the usability study is to evaluate the cost of learning DiaSuite for programmers with various levels of experience. Experimental results are used to validate an upper bound of the learning cost needed for transferring DiaSuite to professional programmers. The study was carried out on a group of four participants with different levels of experience in software development. Participants were assigned a software engineering task, which involved the design, implementation and testing of an IoT application in DiaSuite. Each participant attended a DiaSuite training session prior to the experiment. The perceived usability of DiaSuite was assessed using questionnaires. Finally, an interview was conducted with each participant to acquire further information on the experience in programming with DiaSuite.

3.2 Methodology

Participants. Our study involved four professional software developers with a background in the development of IoT applications and currently working for sensor manufacturing companies. The participants had different level of experience in software development, ranging from 0 to 7 years as depicted in Table 1. The study required the participants to have basic experience in Java programming and the usage of the Eclipse IDE [32]. Three participants matched these criteria. We also recruited one participant who did not match any of the above stated criteria to examine whether a developer with no prior experience in Java programming/Eclipse usage finds the development support provided by DiaSuite useful and easy to use. It is important to note that for this participant, only data collected from questionnaires and the interview have been taken into account in the evaluation process of DiaSuite.

Training session. The usability study was preceded by a training session to give the participants basics on rapid prototyping of IoT applications in DiaSuite. The training was dedicated to the development of an IoT application revolving around an HVAC (Heating, Ventilation, and Air Conditioning) system. The application development was guided by a DiaSuite expert, member of our group, and addressed all the development phases from design to testing. The training session lasted approximately four hours.

Object of the study. The study was dedicated to the development of the DoorLocks application presented in Figure 1. The DoorLocks application was specified by Axible Technologies [4], a provider of digital products and services for gate access control. The goal of this application is to remotely monitor instances of electronic door locks, connected to the Internet through an onboard Sigfox transmitter. The features specified were: real-time event logging, abnormal event alerts (stuck or forced door), and low battery state notifications. These features can be implemented by integrating in a DiaSuite application the smart door locks, an email web service, a logging device, and mobile phones for SMS notifications.

Figure 1 shows the architecture of the application. Door locks provide two sensing capabilities: the “"locked“" facet detecting when a door has been locked or unlocked, and the “"state“" facet detecting state changes, such as a door becoming stuck or forced and the battery level. The Format context component is triggered by any event coming from a door lock. The two kinds of sensors do not provide the same information, but the Format context builds a uniform Event structure. Event structures are received by the Logger controller component, which forwards them to the Logger device. Thus, the Format context together with the Logger controller completely implement the feature of real-time event logging. A separate data flow, going through the Battery context and the MailNotifier controller implements the low battery notification feature: the Battery context parses the events coming from the state sensor of the door locker, selects those concerning a low battery state, and forwards them as a concise string to the MailNotifier controller. This latter controller reformats these compact messages in more explicit textual form, producing a complete e-mail message for a pre-defined e-mail address — the service ensuring general maintenance of the door locks. Finally, a third data flow going through the Alert context and the SmsNotifier controller implements the critical conditions notification feature: whenever a stuck or forced door is detected, an SMS is sent to a predefined cellphone number, for urgent intervention.

Following the training session, the software engineering task evaluated in the study comprised the design, implementation and testing of this application. All the participants had precisely four hours to complete the task. Participants had access to the training material, the online help of DiaSuite and Eclipse, but no internet access was provided. At first, participants were asked to establish the design of the application. We enforced a limit of 30 minutes for this phase, with the possibility for us to provide a correct design after this period, in case the produced design was too complex or incorrect. This limit was imposed to avoid compromising the measurements of the subsequent phases.

Then, all the functionalities had to be implemented and tested one by one. The implementation of a specific functionality had to be validated via unit tests, which were given to the participants prior to the task assignment. An application functionality was considered complete upon the successful execution of the corresponding tests. Participants who provided a valid application design were allowed to redefine their design if needed during the study.

Questionnaires. Participants filled out a first questionnaire assessing their experience in software development, Java programming and the usage of the Eclipse IDE. A standard System Usability Scale (SUS) [9] questionnaire and a custom DiaSuite evaluation questionnaire were handed out to the participants at the end of the study.

Interviews. The interviews with the participants were conducted to collect further information on the DiaSuite programming experience and to discuss the software development process and support currently used by these companies. We also discussed the possibility of programming IoT applications with DiaSuite.

3.3 Experimental Results

The experimental results of the development task assigned to the participants are given in Table 1. The first two columns give the participants' experience in Java programming and the usage of Eclipse in years. The next four columns give the time it took for each participant to complete the design phase, respectively the coding of each of the three application features, in minutes. The last two columns give the size of the code of the completed application, and the percentage

Developer	Java exp. (years)	Eclipse exp. (years)	Design (min)	Dev. feat1 (min)	Dev. feat2 (min)	Dev. feat3 (min)	Code size (LOC)	Man. part (%)
Dev0	0	0	N/A	N/A	N/A	N/A	N/A	N/A
Dev1	1	1	30	92	30	-	1671	10.4
Dev2	3	3	30*	55	15	5	2141	10.7
Dev3	7	7	27	87	18	9	1724	10.3
Expert	3	3	15	32	14	6	1760	10.2

Table 1: Experimental results for the DoorLocks development task.

of the manually written part. As the participant Dev0 has been included only for the questionnaire and interview, his development times were not measured (he was assisted to complete them, to sample the DiaSuite functionalities). The other participants are generically called Dev1 to Dev3, in order of increasing experience in Java/Eclipse. The last participant is a DiaSuite expert in our research group, used as a baseline for the development times.

From Table 1, we can see that the design phase lasted between 15 and 30 minutes for all the participants. To achieve our 30-minute time limit for this phase, we had to fix the design for only one participant, namely Dev2; his design was 80% correct and the corresponding time in Table 1 is marked by an asterisk.

Globally, the first feature of the DoorLocks application took most of the time to develop. This is due to the fact that participants changed the application design and returned back to the implementation more than once when working on the first feature. Also some helper classes had to be created only once for all the features. One participant did not finish the last feature on time. 89% of the the final application code was generated by the compiler for all the participants.

Overall, data in Table 1 constitutes a preliminary validation of our hypothesis that the initial cost of learning DiaSuite, before starting to code real-world applications, is roughly half a day. Indeed, after such a training, two developers out of three completed the design phase, and the third one completed it at 80%; this represents an average completion rate of 93%. As for the implementation and testing phase, 89% (*i.e.,* 8 out of 9) of the application features were correctly developed and tested. Moreover, DiaSuite appears easy to learn for developers with various levels of experience in Java and Eclipse, ranging from 1 to 7 years.

Questionnaires. The pre-experiment questionnaire revealed that three out of four participants used to define their software architecture in an informal way on a piece of paper and were skeptical about the utility of tools dedicated to software architecture and code generation. Only one participant had a positive opinion on the usefulness of such tools and used to define his software architecture via UML diagrams. Two participants had a prior experience with code generation tools, such as Rational Rose [17], AndroMDA [2] and ArgoUML [3]. The post-experiment SUS questionnaire revealed that experienced developers found DiaSuite more usable than developers with little or no experience in Java/Eclipse programming. Interestingly, Dev2 and Dev3's SUS questionnaires revealed an identical usability score of 67,5 points. Dev0 and Dev1 scores were 62,5 and 57,5 respectively. The overall SUS score of 63,75 indicates a usability level in the middle between OK and GOOD, which makes DiaSuite a candidate for continued improvement to reach high acceptability [5]. The post-experiment DiaSuite questionnaire investigated the perceived level of complexity related to design, implementation and testing in DiaSuite, as well as the usefulness of DiaSuite for coping with challenges in development of IoT applications presented earlier. The questionnaire revealed a unanimity on the application design being easily defined using DiaSuite. Also, three out of four participants found testing applications in DiaSuite easy. Two participants (Dev3 and Dev0) considered the implementation phase easy, while the remaining ones had a neutral opinion. The usefulness of DiaSuite for coping with heterogeneous APIs of smart objects has been confirmed by all participants. Three out of four participants confirmed that DiaSuite is useful for rapid prototyping of IoT applications, thanks to the generated code support, which also greatly facilitates application testing. Half of the participants agreed that DiaSuite can be useful for rapid evolution of applications, although the experiment did not involve this task. Interestingly, the participant having a positive opinion about software architecture tools prior to the experiment, considered DiaSuite useful for the design of IoT applications. In contrast, the rest of the participants kept a neutral opinion. The questionnaire also revealed that DiaSuite-generated code was considered useful by three participants. This is a positive outcome of the experiment because only one participant found code generation tools useful prior to being exposed to DiaSuite. Finally, participants expressed a unanimous appreciation of the integration of DiaSuite with Eclipse.

Interviews. In the following, we present key comments of our participants on various aspects of DiaSuite. We asked the participants about their assessment on the development process in DiaSuite. One participant stated: *"Once the design is established, a big part of the work is already done"*. Another participant compared the design phase in DiaSuite with UML: *"DiaSpec appears to be simpler since it is more abstract. UML is much closer to the programming language"*. When asking participants about how DiaSuite could be used in their companies, one participant stated: *"It could be useful to cross multiple data sources when increasing the number of products. We could provide a solution by rapidly combining the different products we have, even with products from different manufacturers in an heterogeneous environment"*.

We also asked participants what they think about the transfer of DiaSuite to industry. A participant stated: *"It's difficult to move to a new technology when you already master a more "traditional" one"*. Another participant proposed to improve DiaSuite as follows: *"The integration of DiaSuite with the Java EE platform and the Spring framework [25] would be a great asset"*.

3.4 Threats to Validity

In this section we review the threats to the validity of our usability study results and the measures we took for avoiding them as much as possible.

Construct validity. These threats concern the adequacy of the experimental setup for measuring the desired outcome. The training required participants to develop a minimal application, called HVAC, that regulates the temperature of a room. As a result, one concern is whether this application ressembles the one they have to develop autonomously. If they were similar, the participants could program the DoorLocks application by imitation based on the HVAC example, possibly without really understanding DiaSuite. To address this threat, we intentionally introduced many differences in the DoorLocks application, including greater size (three times more components and three times more features) and several important differences in the interaction contracts, addressing typical pitfalls of a superficial assimilation of DiaSuite. This definitely prevented participants from programming by imitation.

As a downside, these subtleties increased the risk for the participants to get lost during the design phase, and potentially end up with a design more complex than necessary. As the implementation is partially generated from the design, this increased the risk of not finishing the implementation on time. Thus, verifying the design ability could have compromised verifying the implementation ability, while both abilities are complementary indicators of a correct DiaSuite assimilation. To address this second construct threat, we introduced the time bound of 30 minutes for the application design.

Internal validity. These threats concern factors other than the control variables, which may influence the output variables. In our case, one potential perturbation could come from discomfort with the assigned workstation, as compared to the participant's usual computing environment, because of a different operating system, for instance. To avoid this threat, we asked participants to come with their own laptop computer, and installed DiaSuite on their machines.

External validity. These threats are factors that may invalidate the generalisation of our result. The intended scope for our result is the industrial prototyping and development of IoT applications. This is why we selected the participants among professional developers within companies addressing the smart objects market. The first issue is that professional developers are a scarce resource for research experiment. This is why we could only include 4 participants, among which one participated just in the questionnaire and interview parts. The generalisation of our result is thus limited by the representativeness of this very limited population sample. To mitigate this threat, we recruited the participants with various levels of experience, ranging from 0 to 7 years of experience in Java/Eclipse development project. This makes our result more likely to be generalised, but still constitutes only preliminary data, to be confirmed by a larger experiment.

Another external threat concerns the size of the software engineering task under test. Four hours of development is not negligible, but is small in comparison with a real-world development. Nevertheless, application prototyping is a very common task for sensor and smart object manufacturers, as confirmed by our interviews.

Finally, usability studies are frequently externally threatened by the fact that they occur in a closed world, where participants are isolated from the interruptions they handle every day in the real world. To alleviate this threat, we only isolated them during the training, but allowed some amount of interruptions during the development exercise. For example, one participant handled a phone call during 15 minutes, and another participant was preempted for a (real) work meeting in another room during 25 minutes.

4. Related Work

Languages and tools for sensor/actuator applications. Several other approaches than DiaSuite provide languages and tools for developing applications dealing with sensors and actuators, including IoT applications. For instance, several dedicated languages for describing smart object APIs and features have been defined, such as DomoML for the home automation domain [22] and the Puzzle building blocks [14]. Some approaches provide graphical tools for composing applications out of modules, either by developers in Reactive Blocks [18], Dioptase [8], and Compose [26], or by end users in Puzzle [14]. Boilerplate code for integrating reactive application modules is generated for instance in Reactive Blocks from UML component descriptions [18]. Support for testing IoT applications on simulated devices is provided by the Cooja network simulator [16], [23]. Tool support for maintenance of IoT application is provided in tools such as PervML [28].

While these approaches present several similarities with the DiaSuite approach, none of them generates, starting from a device taxonomy and a specific application design, a customised programming framework that both guides and constraints the developer during all the application lifecycle, namely design, implementation, testing, deployment, and maintenance. We refer the reader to an existing paper [7] for a more detailed comparison between DiaSuite and related approaches. On the other hand, we are not aware about empirical studies on these approaches involving professional developers, aimed at assessing learning cost and usability.

Experiments on using tools supporting software development. Vogel-Heusen [33] presents a usability study and seven usability experiments for evaluating the use of UML to increase the efficiency and quality in designing and maintaining software for manufacturing systems. Both generic UML versions such as UML 2.0 and domain-specific dialects such as SysML-AT are considered. A number of interesting aspects about experiment design are discussed. Maeder and Egyed [21] present a controlled experiment to measure the benefits of a source code navigation tool able to trace requirements, when used in software comprehension and software maintenance tasks. Hanenberg *et al.* perform controlled experiments to measure the benefits of using Aspect Oriented Programming (AOP) tools for developing crosscutting concerns. A more general overview of controlled experiments in software engineering can be found in a survey by Sjøberg *et al.* [30]. According to their classification, our software engineering experiment has the following features: the task category is Create, with subcategories Design and Coding; the number of subjects falls in the Small category, professionals only; the task duration falls in the Large category; globally, the experiment size is then considered as Medium.

Finally, let us contrast this work with other works evaluating the usability of IDEs for developers, for instance those using various visual languages [27]. First, DiaSpec is a textual language, although the application architecture can be visualised as a graph (see Figure 1). More importantly, our study investigates the usability of DiaSuite's approach, not of its IDE in particular. Indeed, DiaSuite is integrated most naturally within the Eclipse IDE, and it does not aim at changing the usability of Eclipse.

5. Conclusion

This section discusses some important lessons we learned during our experiments.

Learning cost. The usability study constitutes promising preliminary evidence that the DiaSuite tools can be efficiently transferred to professional developers in the IoT domain. Indeed, with a minimal learning cost of half a day, developers are able to rapidly develop a typical prototype application.

Need for further experiments. A larger study is needed to confirm these findings on a statistically representative population, and to cover all the development phases addressed by DiaSuite, including the deployment and maintenance phases. Additionally, the impact of DiaSuite on programmers' productivity and application quality also needs to be evaluated. For this purpose, within the ObjectsWorld project, we are currently developing 10 small but representative industrial applications with DiaSuite, each of them deployed on thousands of sensors and actuators. As it is difficult to precisely measure programmers' productivity and application quality, we identified 6 key challenges in developing applications in the IoT domain. Using these challenges, we will provide a qualitative evaluation by rating the impact of DiaSuite on each of these challenges in terms of development effort and result quality, for each particular application. These results will hopefully be reported in an article in the coming months.

Another, complementary, experiment we are currently preparing consists in assigning to groups of students small projects of sensors/actuators applications, to be developed twice, once with DiaSuite and another time using direct coding in Java.This controlled experiment will aim at estimating in a quantitative way the impact of DiaSuite on development cost and application quality.

Simple contexts help with the implementation, but may complicate testing. By comparing the different solutions of the developers to the assigned development task, we found that cutting application functionalities into several specialised contexts greatly simplifies the subsequent implementation phase. It also allows developers to anticipate many design decisions and uncover architecture defects earlier in the development process, before coding has even started. At the other end of the spectrum is an application architecture with a unique context, connected to all the data sources, and one controller connected to all actuator actions. This degenerated design leads to complex code, where all application features are intertwined. This architecture style postpones the identification of defects, increasing the cost of fixes. A much lighter version of this design error appeared in the solution of developer dev2, who defined a single controller for two devices which mixed two completely independent application features. The manual and training should therefore be improved by adding some explicit design counter-examples to be avoided, clearly mentioning their drawbacks.

However, we found a limit to the benefits of modularisation: when the application architecture graph becomes deeper, the test cases are more difficult to produce. Here, we identified a lack of support in DiaSuite for simulating the output of context modules. This support should be added in a future version of the tools, to further encourage modular application structure.

References

[1] Sigfox Network Operator. Online, accessed 14/8/2015, `http://sigfox.com/en/#!/connected-world/sigfox-network-operator`, 2014.

[2] AndroMDA. Generate components quickly with AndroMDA. Online, accessed 10/5/2015, `http://www.andromda.org`, 2014.

[3] ArgoUML. Welcome to ArgoUML. Online, accessed 10/5/2015, `http://argouml.tigris.org`, 2014.

[4] Axible Technologies. Online, accessed 10/5/2015, `http://www.axible-connects-for-you.com`, 2014.

[5] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[6] B. Bertran, C. Consel, P. Kadionik, and B. Lamer. A SIP-Based Home Automation Platform: An Experimental Study. In *13th International Conference on Intelligence in Next Generation Networks*, pages 1–6, Bordeaux, France, Oct. 2009.

[7] B. Bertran, J. Bruneau, D. Cassou, N. Loriant, E. Balland, and C. Consel. DiaSuite: a Tool Suite To Develop Sense/Compute/Control Applications. *Science of Computer Programming, Fourth special issue on Experimental Software and Toolkits*, 2012.

[8] B. Billet and V. Issarny. Dioptase: Data Streaming Middleware for the Internet of Things. *ERCIM News*, 101:23–24, 2015.

[9] J. Brooke. SUS: a "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and A. L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, 1996. URL `http://www.usabilitynet.org/trump/documents/Suschapt.doc`.

[10] J. Bruneau, W. Jouve, and C. Consel. DiaSim: A Parameterized Simulator for Pervasive Computing Applications. In *6th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous'09)*, Toronto, Canada, July 2009.

[11] L. Caroux, C. Consel, L. Dupuy, and H. Sauzéon. Verification of Daily Activities of Older Adults: A Simple, Non-Intrusive, Low-Cost Approach. In *ASSETS - The 16th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 43–50, Rochester, NY, USA, Oct. 2014.

[12] D. Cassou, E. Balland, C. Consel, and J. Lawall. Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications. In *ICSE'11: Proceedings of the 33rd International Conference on Software Engineering*, pages 431–440, Honolulu, USA, May 2011.

[13] D. Cassou, S. Stinckwich, and P. Koch. Using the DiaSpec design language and compiler to develop robotics systems. In *2nd International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob-11)*, Sept. 2011.

[14] J. Danado and F. Paternò. A Mobile End-User Development Environment for IoT Applications Exploiting the Puzzle Metaphor. *ERCIM News*, 101:26–27, 2015.

[15] Q. Enard, S. Gatti, J. Bruneau, Y.-J. Moon, E. Balland, and C. Consel. Design-driven Development of Dependable Applications: A Case Study in Avionics. In *PECCS - 3rd International Conference on Pervasive and Embedded Computing and Communication Systems*, Barcelona, Spain, Feb. 2013.

[16] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 27:1–27:7, 2009.

[17] IBM. Rational Rose family. Online, accessed 10/5/2015, `http://www-03.ibm.com/software/products/en/ratirosefami`, 2014.

[18] F. A. Kraemer and P. Herrmann. Creating Internet of Things Applications from Building Blocks. *ERCIM News*, 101:19–20, 2015.

[19] Links, Cees. The Internet of Things will Change our World. *ERCIM News*, 101:3, 2015.

[20] M2M World News. Samsung Adds SIGFOX Internet of Things Protocol to Its New Samsung ARTIK Platform and Invests in IoT Pioneer. Online, accessed 14/8/2015, `http://m2mworldnews.com/2015/06/15/77001-samsung-adds-sigfox-internet-of-things-protocol-to-its-new-samsung-artik-platform-and-invests-in-iot-pioneer`, 2015.

[21] P. Mäder and A. Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015.

[22] V. Miori and D. Russo. Home Automation Devices Belong to the IoT World. *ERCIM News*, 101:22–23, 2015.

[23] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *31st IEEE Conference on Local Computer Networks*, pages 641–648, Nov 2006.

[24] Phoenix team. Objects World project. Online, accessed 10/5/2015, `http://phoenix.inria.fr/research-projects/objects-world`, 2012.

[25] Pivotal Software. Spring framework. Online, accessed 10/5/2015, `http://projects.spring.io/spring-framework`, 2014.

[26] D. Raggett. COMPOSE: An Open Source Cloud-Based Scalable IoT Services Platform. *ERCIM News*, 101:30–31, 2015.

[27] J. M. Rouly, J. D. Orbeck, and E. Syriani. Usability and suitability survey of features in visual ides for non-programmers. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU '14, pages 31–42, New York, NY, USA, 2014. ACM.

[28] E. Serral, P. Valderas, and V. Pelechano. Towards the Model Driven Development of Context-aware Pervasive Systems. *Pervasive Mob. Comput.*, 6(2):254–280, Apr. 2010.

[29] Sigfox. Sigfox continues to expand network coverage worldwide. Online, accessed 14/8/2015, `http://www.sigfox.com/en/#!/news/sigfox-continues-to-expand-network-coverage-worldwide-6`, 2015.

[30] D. I. K. Sjoberg, J. E. Hannay, O. Hansen, V. By Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A Survey of Controlled Experiments in Software Engineering. *IEEE Trans. Softw. Eng.*, 31(9):733–753, Sept. 2005.

[31] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, New York, NY, USA, 2009.

[32] The Eclipse Foundation. Eclipse. Online, accessed 10/5/2015, `https://eclipse.org`, 2014.

[33] B. Vogel-Heuser. Usability Experiments to Evaluate UML/SysML-Based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation. *Journal of Softw. Eng. and Applications*, 7:943–973, 2014.