Executable and Translatable UML – How Difficult Can it Be?

Håkan Burden
Computer Science and Engineering
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
burden@chalmers.se

Rogardt Heldal
Computer Science and Engineering
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
heldal@chalmers.se

Toni Siljamäki
Ericsson AB
Stockholm, Sweden
toni.siljamaki@ericsson.com

***Abstract*—Executable and Translatable UML enables Model-Driven Architecture by specifying Platform-Independent Models that can be automatically transformed into Platform-Specific Models through model compilation. Previous research shows that the transformations result in both efficient code and consistency between the models.**

However, there are neither results for the effort of introducing the technology in a new context nor on the level of expertise needed for designing the Platform-Independent Models. We wanted to know if teams of novice software developers could design Executable and Translatable UML models without prior experiences of software modelling.

As part of a new university course we conducted an exploratory case study with two data collections over two years. Bachelor students were given the task to design a hotel reservation system and the necessary test cases for verifying the functionality and structure of the models within 300 hours, using Executable and Translatable UML.

In total, 43 out of 50 teams succeeded in delivering verified and consistent models within the time frame. During the second data collection the students were given limited tool training. This gave a raise in the quality of the models.

Due to the executable feature of the models the students were given constant feedback on their design until the models behaved as expected, with the required level of detail and structure. Our results show that using Executable and Translatable UML does not require more expertise than a bachelor program in computer science. All in all, Executable and Translatable UML could play an important role in future software development.

***Index Terms*—Model-Driven Architecture; Executable and Translatable UML; Platform-Independent Models; Learning Effort; Exploratory Case Study;**

I. Introduction

In Model-Driven Architecture (MDA; [1]), the requirements and responsibilities of the system are given a structure by the use of software models in a Computationally-Independent Model, the CIM, often referred to as the domain model [2]. Features such as specific algorithms and system architecture are defined by the next layer of models, the Platform-Independent Models, the PIM. The PIM has no ties towards the hardware nor the programming languages that will in the end realise the system. Such information is added to the Platform-Specific Model, the PSM. As a result the software models of the CIM and the PIM can describe many different implementations of the same system. The models become reusable assets [3] serve both as a description of the problem domain and a specification for the implementation, bridging the gap between problem and solution.

Executable and Translatable UML (xtUML; [4], [5]) is an extension of UML [6] with models that can be executed and translated into code through model compilers. In MDA terms, the xtUML model is an executable PIM that can be automatically transformed into a PSM. The efficient and consistent transformation from a PIM specified using xtUML to a PSM has been tested and proven in previous work [7], [8]. But it is still an open question how much expertise that is required to use xtUML as an executable modelling language for PIMs.

A. Motivation

For ten years we have given a university course where teams of students go through the different tasks of an MDA process; from analysis to implementation by designing the system using UML models. The process is illustrated in Figure1. The numbers for each activity in the process are specific to the course and state the maximum number of hours for each student.

The analysis phase was used to capture the business rules of the problem domain in models that satisfy the requirements of the system. The focus during the analysis phase was thus on understanding the problem domain by using activity diagrams, use cases and conceptual class diagrams [6], [9].

The second phase of the process, the design phase, was where more detailed UML diagrams were used such as interaction, state chart, class and component diagrams. Even though this phase should be important for the overall process, we found that it contributed little to the overall system for most teams (in Lean terms the models represent waste [10]). The diagrams were incomplete, lacking necessary details in structure and behaviour. The only way of testing the models was through model inspection, making it a matter of opinion when the models are complete [11]. Another problem with the models was that they were inconsistent with each other leading to complications about which model to follow in the transformation to source code. The impact of the problems vary depending on how important the models are in the

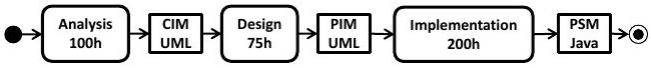

Figure 1: The old software development process

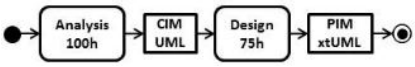

Figure 2: The proposed software development process

development process and when and how the inconsistencies are shown [12]–[15].

The design phase was followed by an implementation phase were the students manually transformed their models into Java. This meant that it took months before the students could test their analysis and design. This is a problem shared with industry [16].

Since the code is manually written with the models as a guide it also means that there is a difference in the interpretation of the problem between the UML model and the hand-written code. By default you reanalyze the problem when you start writing the code, and you often come up with a different solution compared to the modelled solution. Eventually the model and the hand-written code diverge, so the only way to really understand what's going on in the system is to study the hand-written code. The model may then serve as a quick, introductory overview of the system, but it may also be incorrect as soon as you stop updating the model for reflecting the changes made to the hand-written code. This notion of architecture erosion is a well-known problem and is still being reported on [17]–[19].

B. Aim and Research Question

As a result from collaboration between industry and academia we came up with the idea to use xtUML to model the component- and class diagrams and the statemachines instead of UML in the design phase. If the change of modelling language is successful we will get rid of the inconsistency problems. With an executable PIM it should be possible for the students to test and validate their design decisions without having to implement them in Java first. And when the models behave as expected and the design phase is complete, all functionality of the system should be captured in the models [4], [11]. In the long run this will mean that a lot of the work that was done in the implementation phase can be replaced by generating the code straight from the models, leading to shortened implementation times and consistency between models and code.

Swapping UML for xtUML is not a one-to-one substitution. If it is a matter of opinion when a UML model is complete, an xtUML model is complete when all test cases return the expected results [11]. So, xtUML is more than the graphical syntax, the models have to be given semantics to be executable. In addition, test cases have to be modelled, executed and evaluated. These additions demand that the xtUML tool is more than a drawing tool.

Will the immediate and constant feedback that is given from executing the test cases compensate for the increase in modelling effort? Or will the added effort for learning xtUML take so much time that there is no left for modelling? This concern is re-phrased into our research question:

"Can teams of four novice software modellers solve a problem that is complex enough to require the full potential of xtUML as a modelling language within a total of 300 hours?"

To answer our research question we scrapped our old course in favour of a new one that follows the process seen in Figure 2. Instead of spending 200 hours implementing the design to be able to test and verify it, testing will now be a part of the design phase. Just as for the process in Figure 1 the number of hours in each step states the maximum for each student. The introduction of the new design phase was done as a case study with the ambition to explore and explain the transition and its implications.

C. Contribution

Earlier contributions has shown how Executable and Translatable UML enables MDA [3], [5], [11], the reusability of the PIM has been reported on in [20] and the efficiency of the transformations from PIM to PSM is illustrated by [7]. Our contribution shows that xtUML as a technology is mature enough to to be used by novices to design executable PIMs.

D. Overview

In the next section we go into more detail of xtUML and how it can be used. In section III we explain how our subjects made use of xtUML to develop and test a hotel reservation system. Our findings and their validity are presented in section IV. In section V we discuss the implications of our results and in section VI we relate our own case study to previous work. This is followed by a conclusion and some ideas about further investigations regarding the usage of xtUML.

II. Executable and Translatable UML

The Executable and Translatable Unified Modeling Language (xtUML; [4], [5], [11]) evolved from merging the Shlaer-Mellor method [21] with the Unified Modeling Language (UML, [6]).

A. The Structure of xtUML

Three kinds of diagrams are used for the graphical modeling together with a textual action language. The diagrams are component diagrams, class diagrams and state-machines. There is a clear hierarchical structure between the different diagrams; state-machines are only found within classes, classes are only found within components. The different diagrams will be further explained below, together with fragmentary examples taken from the problem domain given to the students, a hotel reservation system.

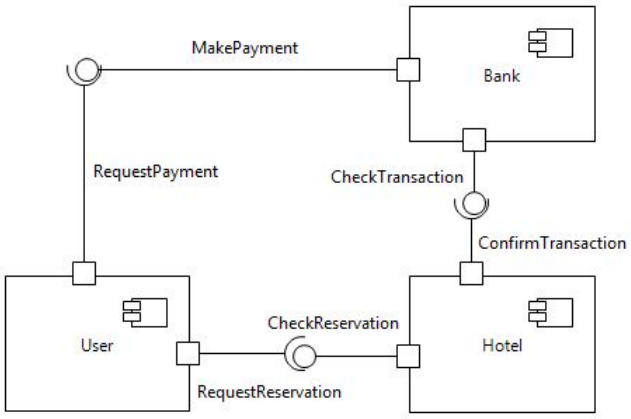

Figure 3: An xtUML component diagram

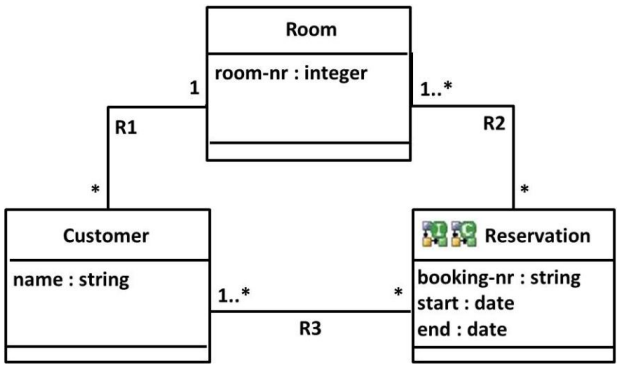

Figure 4: An xtUML class-diagram

1) Component Diagrams: The xtUML component diagram follows the definition given by UML. An example of a component diagram can be found in Figure 3. In this diagram the hotel domain depends on the bank for checking that a transaction has gone through as part of the process of making reservations. The User component represents a users of the system and this is where the test cases are placed.

2) Class Diagrams: In Figure 4 we have an example of an xtUML class diagram. It describes how some of the classes found in the Hotel component relate to each other. I.e. a Room can be related to any number of Reservations (shown by an asterisk, *) but a Reservation has to be related to at least one Room (visualized by 1..*).

The xtUML classes and associations are more restricted than in UML. We will only mention those differences that are interesting for our case study. A feature such as visibility constraints on operations and attributes does not exist. They are therefore accessible from anywhere within the same component. In UML the associations between classes can be given a descriptive association name while in xtUML the association names are automatically given names on the form RN where N is a unique natural number, e.g. Room is associated to Reservation over the association R2.

3) State Machines: In the class diagram in Figure 4 the Reservation class has both an instance and a class state machine which is indicated by the small figure in the top-left corner of the class. The instance state machine can be found in Figure 5. This state machine covers the first four states of the Reservation procedure, e.g. from the second state, Get rooms, it is possible to reach the third state, Lock rooms, by requesting the rooms. If there are no available rooms you return to the initial state were you can start a new search. Each instance of Reservation has its own instance statemachine that starts running when the Reservation is created.

A class-based state-machine is shared among all instances of a class and starts running as soon as the system starts, like a static process. For shared resources, such as rooms, a class state-machine can be used to ensure that only one reservation instance can book a room at any time.

4) Action Language: An important difference between standard UML and xtUML is that the latter has a textual programming language that is integrated with the graphical models, sharing the same meta-model [21], [22].

The number of syntactical constructs is deliberately kept small. The reason is that each construction in the Action Language shall be easy to translate to any programming language (such as Java, C or Erlang) enabling the PIM to be reused for different PSMs [20].

There are certain places in the models were Action Language can be inserted, such as in operations, events and states. Over the years a number of different Action Language have been implemented [5] and in 2010 OMG released there own standard [23].

B. Interpretation and Code Generation

Since xtUML models have unambiguous semantics all validation can be performed straight on the xtUML model by an interpreter. During the execution of the test cases an object model is created. The object model includes all class instances with their current attribute values and by which associations they are linked to each other. During execution all changes of the association instances, attribute values and class instance are shown [9] as well as the change of state for classes with statemachines in the object model.

The xtUML models can be translated into Platform-Specific Models by model compilers. Since the Platform-Specific code is generated from the model, it is possible for the code and the models to always be in synchronization with each other since all updates and changes to the system are done at the PIM-level, never by touching the code. The efficiency of the generated code has been reported on by [7]. [8] have used the model compiler to generate test cases for the PSM.

III. Case Study Design

To answer our research question:

"Can teams of four novice software modellers solve a problem that is complex enough to require the full potential of xtUML as a modelling language within a total of 300 hours?"

we have both in 2009 and 2010 let our students use xtUML to design hotel reservation systems. The resulting models have been inspected and compared against our evaluation criteria.

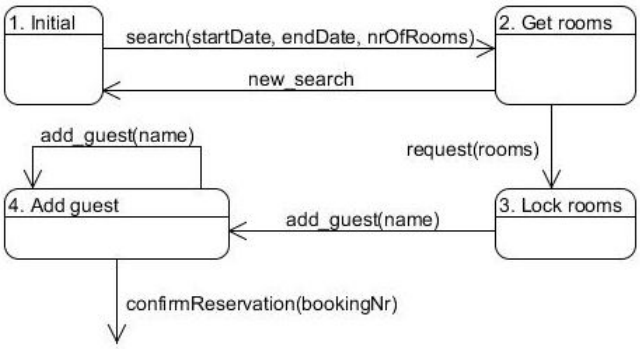

Figure 5: A partial xtUML statemachine

A. Subject and Case Selection

1) The Subjects: Our subjects were students in the final year of their bachelor programs in computer science and software engineering. Their prior knowledge of modelling is limited to class diagrams but they are used to programming in an object-oriented paradigm using Java. In our curriculum the students do two courses in parallel with a working week of 50 hours, so we expect the subjects to work 25 hours a week on our course. A team of four subjects is expected to do a total of 100 hours per week.

2) The Case: We chose a domain that the subjects could relate to and have some prior knowledge about. The idea is that the subjects shall focus on modelling, not learning a new subject matter. The domain should also have distinct concepts so that an object-oriented solution made sense and have problems where it is natural to use state machines. We also wanted the domain to include problems with algorithmic complexity. Our last requirement was that the domain should represent an open-ended problem so that there is not one right solution. A system for handling hotel reservations seemed to fit all our requirements.

In the hotel domain reservations, customers and rooms are all examples of distinct concepts. The booking process itself has a chain of states that it is natural to control with a state-chart, while finding all the possible matches to a set of search criterias for a reservation is an algorithmic problem. These two together, controlling the order of events and searching for rooms, meant that the domain poses the problem of access and allocation of shared and limited resources.

The new design phase was given three weeks, just as the previous design phase. The work was done in teams of four subjects, with a total workload of 75 hours per subject.

The subjects used BridgePoint [24] from Mentor Graphics [25] to design the xtUML models. There were three 90-minute lectures related to xtUML and BridgePoint. Two of these lectures were given by industry representatives, one from the tool vendor Mentor Graphics and one from Ericsson AB as users of BridgePoint. The subjects were encouraged to study xtUML on their own and we recommended them to read [5] and [26]. Each team had half an hour a week with a researcher to discuss design issues. Besides lecturing and supervising on design issues the researchers played the role of project owners.

In 2010 we added limited tool support for the subjects. A subject from 2009 used a total of 22 hours spread over the three weeks to help the subjects of 2010 with BridgePoint. This meant that each team had access to less than an hour of tool training for the entire design phase.

B. Data Collection Procedures

We have done two data collections, in 2009 and 2010 respectively. In 2009 there were 88 subjects split into 22 teams. In 2010 we had 108 subjects divided into 28 teams, with four (sometimes three) members per team. We used three forms of data collection; model evaluations, informal discussions and a questionnaire. Model evaluations and informal discussions were used both times but the questionnaire was only used in 2010.

1) Evaluating the xtUML Models: The evaluation of the xtUML models was done immediately after the design phase and took a whole week to complete. Each team was given 20 minutes to demonstrate their system and to run their tests. Thereafter there was 20 minutes to discuss issues related to their models. Every model was evaluated by two researchers against the evaluation criteria that are specified below. The subjects of each team were present throughout the evaluation, permitting a discussion on the how the criteria on functionality and structure had been interpreted and implemented in the model. A short description of each model with our comments was taken down in a spread sheet.

2) Informal Discussions: We thought it vital to have an informal opportunity for the subjects to discuss their experiences throughout the design phase. This was an important opportunity for us to get more in-depth information into the problems and discoveries that the subjects had encountered when using BridgePoint to model xtUML. Since we did not know what to expect for outcome in 2009 we wanted the subjects to have the opportunity to drop us an e-mail, come by our offices or use the lectures for addressing those issues they found urgent. This proved to be a valuable source for data collection, so valuable that we kept it in 2010. The drawback is that it is not a procedure that is always possible to document or systemize.

3) Questionnaire: One of the most important things that became evident from the informal discussions in 2009 was that the subjects found the learning threshold stressing under the time constraint. Besides introducing tool support in 2010 to ease the subjects' stress we conducted a questionnaire. The aim of the questionnaire was to get a better view of how much time the subjects spent on getting confident in using BridgePoint.

C. Evaluation Criteria

Before the subjects started to develop their models we gave them evaluation criteria. The reason was to have a clear idea for both the researchers and subjects of what we expected from the teams.

Based on the use cases from the analysis phase the subjects should come up with executable tests. By running the tests it should be possible to validate that the system is behaving as specified by the CIM. This meant that the object model had to show all relevant changes for objects, associations, attributes [9] and states after a test had been run. At least one test case should be in conflict with the business rules of the system.

However, this does not guarantee that the models are well-structured nor readable [11], [26], [27]. Therefore the criteria enforced an object-oriented design.

For the class diagram we did not accept models with a central object representing the system [26]. Due to the lack of visibility constraints in xtUML we stated that the only way to obtain or change the value of an attribute should be through operations. It should only be possible for a class instance to call another class instance if they are linked by associations. This is so that the dependencies between the class instances are explicit in the class diagram. We wanted all the meaningful associations and concepts in the class diagram. We requested names on classes, attributes, operations and variables that were relevant for the domain.

For the state-machines it was necessary to include all the states and transitions relevant for capturing the lifecycle of the class where it resides. The name of events and states should be meaningful. To ensure that the subjects made use of the power of state machines we required that they should be used for modelling the reservation process.

IV. Results

Over the two years that we have used the new design phase we have evaluated 50 xtUML models. Of these 43 have fulfilled the success criteria. The subjects had to overcome a learning threshold before they got confident in using BridgePoint to develop their xtUML models. All in all, we can answer our research question by stating that the teams did manage to use the full power of xtUML within 300 hours.

A. Results from Evaluating the Models

In 2009 all 22 teams came up with an executable model, capturing at least the minimum functionality. 18 of the 22 teams, equivalent to 83% managed to come up with a model within the time frame that met our criteria for a successful model. The models that did not meet the criteria had either a monolithic class that represented the whole system and/or not used the associations to access class instances. Most teams did not use components, but that was not a strict criteria either. This was a shortcoming of the criteria as we had wanted to see the subjects use components, since it would have added a whole new level of abstraction to their models. On the other hand, we were encouraged by the fact that all teams had delivered testable models that fulfilled the criteria for functionality.

Three teams came up with models that went beyond our expectations. They had used components and modelled more functionality than we thought possible. One of the three teams had even made extensive use of design patterns [28].

In 2010 there were 28 teams. 25 of these managed to deliver executable models within the time frame. This is an increase from 82% to 89%, compared to 2009. One of the teams that failed to specify an executable model did so due to unresolvable personal issues within the team. The other two teams misjudged how long time it would take to come up with a model and were not finished in time due to their late start. In 2009 all teams succeeded to come up with executable models compared to 89% in 2010. But this time all teams had components and interfaces with the consequence of using the full power of xtUML. This was not an intended outcome of the added tool training but highly appreciated even so!

B. Outcomes From the Informal Discussions

The subjects are used to programming in Java. In 2009 it took the subjects some time before they started to reason about objects and programming in an xtUML-way. When they encountered a problem many of them expected a solution using detailed Java-specific datastrucutres or libraries. In contrast BridgePoint is mostly used for embedded systems were the companies have their own private libraries.

Particularly state-machines were difficult to use since they had no counterpart in Java. It is not possible for us to say if this is due to that Java is the only language they are used to or if it is due to the more abstract level of reasoning in xtUML.

BridgePoint is a powerful tool; enabling modelling, execution of models and translation to source code. All the functionality makes it a complex tool. BridgePoint is also to a large extent menu-driven — many of the design choices are implemented by choosing from drop-down menus and tool panels. The challenge is to get used to all the different combinations of choices that are needed for elaborating the design. The version we used is a plug-in for Eclipse which in itself has a number of features to get used to.

In reaction to the problems concerned with BridgePoint as a tool we decided to use one of the subjects from 2009 for tool training in 2010. The intention was that this would mean less time spent on understanding the tool and more time to spend on developing the models.

In 2010 the subjects requested a version control system so that they could more easily split the design work between themselves. This was never an issue in 2009 and a sign of more confident subjects. If this is a consequence of the tool training or not is to early to answer.

C. Experienced Learning Threshold

In 2010 we used a questionnaire to get a better idea of how the subjects experienced BridgePoint. The question we asked was ”How many hours did you spend learning BridgePoint before you got confident in using the tool?”

In total 90 of 108 subjects answered the question. Besides the answers given in Figure 6 six subjects answered that they never became confident and one subject had no comment. The number of hours it took to become confident are given on the x-axis, the number of subjects for a given number of hours is displayed along the y-axis. This means that 27 subjects

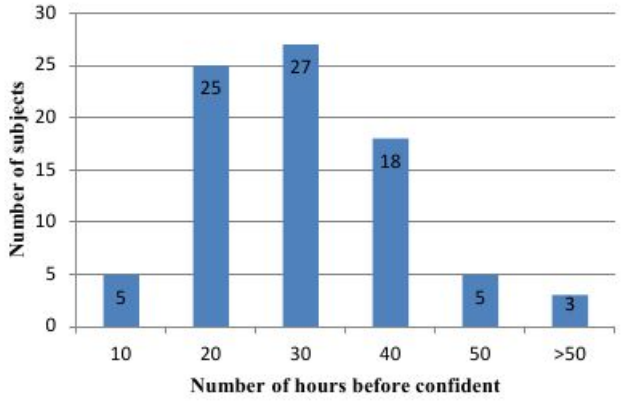

Figure 6: Number of hours that the subjects needed to become confident in using BridgePoint

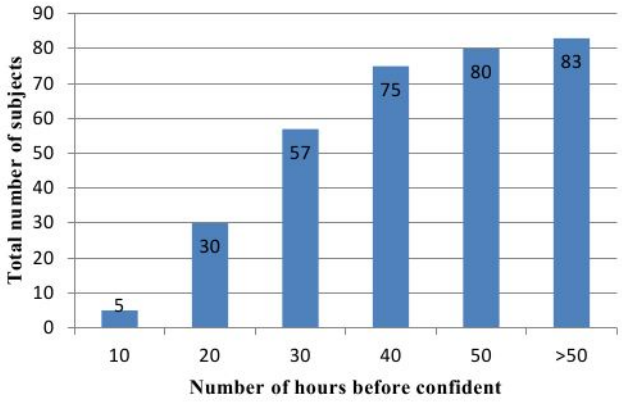

Figure 7: Total number of subjects that are confident in using BridgePoint depending on time

answered that it took 30 hours to become confident in using BridgePoint.

In Figure 7 the x-axis carries the same information as in Figure 6 while the y-axis now displays the total number of confident subjects for a given time, e.g. after 30 hours a total of 57 subjects felt confident in using BridgePoint. After 40 hours this figure had risen to 75 subjects.

D. Relevance to Industry

We wanted to solve our consistency problems by using xtUML instead of UML. In our view this was successful. But we believe that xtUML can have a larger impact than just solving the problems we had with our old development process 1. From the outcomes of the the evaluation of the models, the subjects' own figures for the confidence threshold for using xtUML and the informal discussions we propose a hypothesis.

We state that xtUML, both as a modelling language and tool, is easy to learn and use without any prior experience of software modelling. It is enough to have the programming experience equivalent to a bachelor student in computer science. Our subjects managed to learn and understand the full expressivity of xtUML within 75 hours, and that includes using asynchronous events in statemachines and sending and receiving signals between components as well as designing the models to an appropriate level of detail. If it is possible for us to manage with the transition from UML to xtUML it should be possible to do so in industry as well.

E. Evaluation of Validity

We have analysed our results using the classification of validity as defined by [29], [30] and [31].

1) Construct Validity: Our solution to the problems we had earlier is set in the same context as the problem was. Our subjects have the same background and experience as previous students, just as before the subjects have the necessary domain knowledge from the analysis phase, they work under the same time constraints and come up with the same kind of models. The only thing that has changed is the modelling language, which is what we want to evaluate.

The subjects were present during the model evaluation. This was done in order to reduce our bias in interpreting their work and how it related to our validity criteria.

In order to assess the quality of the software models we have specified evaluation criteria. In 2009 only three of 22 teams defined two or more components and the interfaces between them. Since the 28 teams in 2010 also had models with several components with defined interfaces we can see that the subjects managed to use the full power of xtUML.

The figures for the number of hours to overcome the learning threshold of BridgePoint were estimations done by the subjects themselves. There might be variations among the subjects of the definition of when the threshold is passed. Our experiences from Ericsson and our own observations correlate with the estimations of the subjects. However, there is a possibility that some subjects have exaggerated or underestimated their figures due to social factors (reactions towards the tool, researchers or team members). The exact nature of the learning threshold for xtUML will be dependent on the background of the subjects and which tool that is used.

2) Internal Validity: The evaluated models are developed on the basis of the CIM from the analysis phase, so that everything the subjects need to know about hotel reservation systems should be found in their CIM. This means that they should not need to spend time during the design phase on anything else than learning and using xtUML.

Even if the evaluation criteria have influenced the nature of the results, by defining what we expected from the subjects' models, the process of getting there was by using xtUML.

In the first run of the experiment we did not know what to expect for results. In the second run we had expectations based on the results from the first run. Therefore we were careful to make sure that we as researchers had the same roles towards the subjects in both runs, which led us to let a subject from 2009 take care of the tool training in 2010. We still cannot

neglect that our changed expectations might have influenced the outcome in 2010, even if we did not get the results we were expecting. On the other hand this is always the case in situations where you want to replicate research that involves human beings. This is also a threat to the reliability of our results.

It is possible that the subjects have been sharing insights and experiences throughout the case study. We knew this could be the case from the start and that was one reason why we wanted an open-ended problem. During the evaluations we have seen 50 unique models which implies that all teams have had their own process to come up with the models.

3) External Validity: Today's students are tomorrow's employees. If our subjects can master xtUML it should also be possible for software developers in industry to do so. This is also claimed by [32] who state that students can be used in research as long as it is the evaluation of the use of a technique by novice users that is intended. We can expect software developers in industry to have at least the same competence as our subjects.

The size of the problem given to the subjects is smaller than most indutrial sized tasks. Our domain has a certain level of complexity and was chosen from our collaboration between industry and academia. By handling the access and allocation of the shared resources within the hotel domain, we made sure that the subjects had to solve a non-trivial task.

4) Reliability: Since the evaluation of the models is subjective there was at least two researchers present at every evaluation in order to reduce the risk of bias and inconsistencies between evaluators. We also used the criteria for a successful model to ensure that the evaluation is less subjective, making the results less dependant on a specific researcher.

BridgePoint was chosen by the authors based on the fact that a team within Ericsson think that this is one of the best MDA solutions today. After we had made our decision on which tool we would prefer to use we contacted Mentor Graphics in order to start a collaboration with them. Mentor Graphics never influenced us on which tool to use. Using another xtUML tool might give other results, especially regarding the threshold, both in figures and what is seen as problematic.

V. Discussion

In our previous MDA process, given in Figure 1, our students manually transformed the PIM into a PSM. From our previous experiences of using xtUML as a code generator [7], [8] we know that this manual transformation can be automated. However, to raise the quality of the generated code the PIM needs to be manually enhanced by a marking model [2], [3], [7]. The generated code will then be sufficient for an embedded system. For systems that interact with human users it will also be necessary to develop the needed user interfaces. All in all the introduction of xtUML should enable a less time-consuming implementation phase compared to our old MDA process.

VI. Related Work

There is a previous experience from using xtUML in the context of computing education reported in [33], [34]. One of their motivations for using xtUML in a modelling course is that they found UML to large, ambiguous and complex. In contrast, xtUML models are unambiguous and easier to understand than UML models. The possibility of verifying the models to see if they meet the requirements is important in order to give the students feedback on their modelling. The authors have used xtUML both for specifying a web application and for 3D drawing software.

By using xtUML in a similar context as ours their work strengthens our claim that xtUML can be used by novice software modellers. However, there work does not report any results from letting undergraduate students use xtUML; there are no clear criteria for what was seen as a successful project and subsequently no reports on how many students that managed to complete the task. It is also unclear how much time the students spend on their models. Another important difference is that we find UML useful for defining the CIM.

Both [5] and [11] describe xtUML in the context of MDA. Starting of from use cases they develop PIMs by using xtUML. While the main focus is on developing an executable PIM both books takes the reader from CIM to PSM. The main differences lie in the choice of tool and in how they choose to describe and explain xtUML.

We have made extensive use of both books as a source of inspiration for how to work with xtUML and as recommended literature for the subjects for obtaining executable PIMs in an MDA context. These are the most cited books on how to use xtUML and they are detailed in how this is accomplished. However, they do not mention the effort for learning xtUML nor the level of expertise needed to use xtUML as a modelling language for PIMs.

VII. Conclusions and Future Work

A. Summary

Even if the subjects spend the first week of the design phase in order to learn BridgePoint the fact that the models have a testable behaviour more than compensates for the increase in effort. Our subjects used the test cases to refine their models until they met the criteria. As a consequence their PIMs had the necessary detail and structure as defined by the CIM. This was possible since xtUML gave them constant and immediate feedback on all their design decisions.

In contrast, UML models are not executable. More or less the only way of checking the quality of a UML model is by performing a model review. This is a powerful method for improving on UML models but it is also time consuming. And it can be hard to catch the mistakes in complex systems. It becomes a matter of opinion when a model can be considered complete. In contrast an xtUML model is complete when it only delivers expected output for all test cases.

Previous work has shown that xtUML enables MDA by the reusability of the PIMs, the efficient transformation from PIM

to PSM and by solving the problems of inconsistencies within the PIM. Our work shows with what little effort and expertise it is possible to develop PIMs, using the full expressivity of xtUML. This implies that Executable and Translatable UML is a technology that is ready to be used within industry.

B. Future Work

We are looking at the possibilities to expand the new course so that it covers the entire MDA-process, from CIM to PSM. Issues we want to investigate is how difficult it is to mark the PIMs for an efficient transformation to PSMs, the effort for deploying the generated code on a platform with the required user interfaces and how much time we can save compared to the old development process, illustrated in Figure 1.

In addition, we want to investigate the reasons behind the socio-technical gap [35] to understand why xtUML is not used more within industry and software development.

ACKNOWLEDGMENT

The authors would like to thank Staffan Kjellberg at Mentor Graphics; Stephen Mellor; Leon Starr at Model Integration; Dag Sjøberg at University of Oslo; Jonas Magazinius, Daniel Arvidsson, Robert Feldt and Carl-Magnus Olsson at Computer Science and Engineering in Gothenburg.

REFERENCES

[1] OMG, "MDA," Accessed January 2011. [Online]. Available: http://www.omg.org/mda/

[2] *MDA Guide Version 1.0.1*, Object Management Group, Framingham, Massachusetts, 2003.

[3] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise, *MDA Distilled*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[4] L. Starr, *Executable UML: How to Build Class Models*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[5] S. J. Mellor and M. Balcer, *Executable UML: A Foundation for Model-Driven Architectures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] OMG, "OMG Unified Modeling Language (OMG UML) Infrastructure Version 2.3," accessed 11th September 2010. [Online]. Available: http://www.omg.org/spec/UML/2.3/Infrastructure/PDF

[7] T. Siljamäki and S. Andersson, "Performance benchmarking of real time critical function using BridgePoint xtUML," NW-MoDE'08: Nordic Workshop on Model Driven Engineering. Reykjavik, Iceland, August 2008.

[8] F. Ciccozzi, A. Cicchetti, T. Siljamäki, and J. Kavadiya, "Automating test cases generation: From xtUML system models to QML test models," MOMPES: Model-based Methodologies for Pervasive and Embedded Software. Antwerpen, Belgium, September 2010.

[9] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[10] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[11] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie, *Model Driven Architecture with Executable UML™*. New York, NY, USA: Cambridge University Press, 2004.

[12] C. F. J. Lange and M. R. V. Chaudron, "Effects of defects in uml models: an experimental investigation," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 401–411.

[13] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Information and Software Technology*, vol. 51, no. 12, pp. 1631 – 1645, 2009.

[14] C. F. J. Lange, "Improving the quality of UML models in practice," in *ICSE*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds. ACM, 2006, pp. 993–996.

[15] R. Van Der Straeten, "Description of UML Model Inconsistencies," Software Languages Lab, Vrije Universiteit Brussel, Tech. Rep., 2011.

[16] N. Mellegård and M. Staron, "Characterizing model usage in embedded software engineering: a case study," in *ECSA Companion Volume*, ser. ACM International Conference Proceeding Series, I. Gorton, C. E. Cuesta, and M. A. Babar, Eds. ACM, 2010, pp. 245–252.

[17] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, pp. 40–52, October 1992. [Online]. Available: http://doi.acm.org/10.1145/141874.141884

[18] N. Mellegård and M. Staron, "Methodology for requirements engineering in model-based projects for reactive automotive software," in *European Conference on Object-oriented Programming (ECOOP)*, Paphos, Cyprus, 2008.

[19] J. Bosch, "Architecture in the age of compositionality," in *Software Architecture*, ser. Lecture Notes in Computer Science, M. Babar and I. Gorton, Eds. Springer Berlin, Heidelberg, 2010, vol. 6285, pp. 1–4.

[20] S. Andersson and T. Siljamäki, "Proof of concept - reuse of PIM, experience report," in *SPLST'09 & NW-MODE'09: Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, Tampere, Finland, August 2009.

[21] S. Shlaer and S. J. Mellor, *Object lifecycles: modeling the world in states*. Upper Saddle River, NJ, USA: Yourdon Press, 1992.

[22] M. L. Crane and J. Dingel, "Towards a formal account of a foundational subset for executable uml models," in *MoDELS*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds., vol. 5301. Springer, 2008, pp. 675–689.

[23] OMG, "Concrete Syntax for UML Action Language (Action Language for Foundational UML - ALF)," accessed 30th April 2011. [Online]. Available: http://www.omg.org/spec/ALF/

[24] "BridgePoint," accessed 8th March 2011. [Online]. Available: http://www.mentor.com/products/sm/model_development/bridgepoint/

[25] "Mentor Graphics," accessed 24th August 2010. [Online]. Available: http://www.mentor.com/

[26] L. Starr, "How to build articulate UML class models," accessed 24th November 2009. [Online]. Available: http://knol.google.com/k/leon-starr/how-to-build-articulate-uml-class-models/2hnjef6cmm97l/4

[27] C. F. J. Lange, B. D. Bois, M. R. V. Chaudron, and S. Demeyer, "An experimental investigation of UML modeling conventions," in *MoDELS*, ser. Lecture Notes in Computer Science, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., vol. 4199. Springer, 2006, pp. 27–41.

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley Professional, January 1995.

[29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers Norwell, MA, USA, 2000.

[30] R. K. Yin, *Case Study Research: Design and Methods*, 4th ed. California: SAGE Publications, 2009.

[31] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[32] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, pp. 721–734, August 2002.

[33] S. Flint, H. Gardner, and C. Boughton, "Executable/Translatable UML in computing education," in *ACE'04: Proceedings of the sixth conference on Australasian computing education*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004, pp. 69–75.

[34] S. Flint and C. Boughton, "Executable/translatable uml and systems engineering," in *Systems Engineering and Test Evaluation Conference (SETE 2003)*, A. McLucas, Ed., Canberra, Australia, 2003.

[35] M. S. Ackerman, "The intellectual challenge of cscw: The gap between social requirements and technical feasibility," *Human-Computer Interaction*, vol. 15, pp. 179–203, 2000.