UML in Action: Integrating Formal Methods in Industrial Design Education

Jun Hu[1], Philip Ross[1], Loe Feijs[1], and Yuechen Qian[2]

[1] Eindhoven University of Technology, 5600MB Eindhoven, NL
[2] Philips Research, 5656AE Eindhoven, NL

Abstract. When designing product behavior, the designer often needs to communicate to experts in computer software and protocols. In present-day software engineering, formal specification methods such as the Universal Modeling Language have been widely accepted. Teaching design students these formal methods is non-trivial because most of design students often have difficulties in programming the behaviors of complex produces and systems. Instead of programming, this paper presents a technique, namely "acting-out", for design students to master the formal methods. The experience shows that acting-out not only worked out very well as a teaching technique, but also showed the potential for bridging the processes of industrial design and software engineering.

1 Introduction

The industrial design of embedded systems is a contemporary challenge which deserves more attention and better support by methods and tools. For a long time the worlds of embedded systems and industrial design developed separately. The early examples of embedded systems included products such as telephone exchanges, plant control, and military applications. The early examples of industrial design included furniture, radios, cars, and household tools. In other words: the embedded systems where invisible and their users were not given much room for affective or spontaneous behavior. In contrast, the industrial designers had to give high priority to the emotional associations, the styling, the appeal, the usability and even the fun associated with the product. But now most everyday personal objects contain embedded systems and the two worlds are merging. At the methodological level however, much work regarding their integration remains to be done. The present paper is meant as a contribution to this integration.

In Fig. 1 a number of methods are arranged from left to right. Formal methods include Z, Object Z, SDL, VDM, CSP, ISpec, Larch, process algebraic methods etc. They are close to mathematics and they enable very rigorous syntactic and semantic analysis. Structured methods are more flexible; they developed in a pragmatic way to answer the needs of software developers. Several useful notations are integrated in the Universal Modeling Language (UML), which emerged as a synthesis and de facto standard out of a rich variety of notations one or two decades ago (such as Yourdon, SSAD, Objectory, OMT etc.). In the HCI (Human Computer Interaction) and Industrial Design communities, a growing interest for the behavior aspects of products arose, not just the form-giving of screens or the physical forms and materials of products. A new generation of methods is available which we summarize as “engaging methods”. What they have in common is the idea that behaviors are not stand-alone elements that have to be described with great precision. User behaviors arise in social contexts. Moreover, by letting designers engage in the behaviors themselves, their emotional, creative and social skills are activated and thus better integrated into the result as well. We mention for example:

– Personas,
– Focus groups,
– Acting out,
– Interaction relabelling.

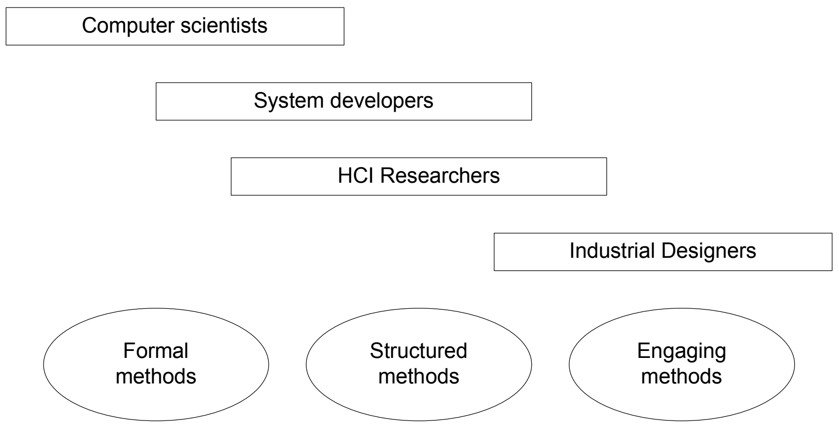


Fig. 1. Methods ranging from mathematical and formal to social and emotional

We agree with the argumentation line of Grudin and Pruitt that these methods exploit *the power of people* [1]. Designers are forced to think about social, political, but also emotional aspects of design that otherwise go unexamined.

The challenge to which we present a contribution in this paper is to better integrate the methods of Figure 1 horizontally. One way to do this is designing another method or language, as done in the DUTCH design method [2]. Rather than proposing a new language, here we address the question how to open up the language of the formal and structured methods to industrial designers. One could try to educate them in software engineering and UML in the traditional way; but in view of the amount of other skills that industrial designers are expected to have, this is just another load to their already overloaded education path.

1.1 UML in Industrial Design Teaching

A product with sensors, actuators and network connections can offer an interesting, useful, or playful behavior to its users and to the other products, systems and services to which it is connected. The master students from the department of industrial design (ID), Eindhoven University of Technology (TU/e) take responsibility for the creation of this behavior. In many cases. if the product isn't stand-alone, neither is the designer. Whenever product behavior is realized through computer software and protocols, the designer often needs to communicate to experts in these matters. In software design, formal methods are often used for this purpose [3–6]. In present-day software engineering, formal specification methods such as the Universal Modeling Language (UML) [7] have been widely accepted. It contains "activity diagrams", "use case diagrams", "class diagrams", "state charts" and "message sequence charts" that are useful for product designers and software engineers to exchange their ideas and concepts to reach a common understanding. A module called "Formal Software Specification in Action" is now taught at ID, TU/e to help the students understand and use such formal methods for this purpose.

The knowledge and skills that students gain by participating in this module helps them to express the structure and behavior of the software components of their design in a way that is understandable to other parties. The students develop an understanding and appreciation of what it means to master complexity through formal methods. The scope is widened from small programs to complex software systems. Although developing and maintaining such systems usually involves computer scientists as well, the ID Master students are expected to be well equipped to use formal methods such as UML and thus specify system structure and desired behavior.

UML as a formal specification tool is widely used in an object-oriented design process in software engineering. Such a process can follow a traditional waterfall model going through object-oriented analysis (OOA), object-oriented design (OOD) and object-oriented programming (OOP) [8], testing and delivery. It can also following a spiral process that includes fast iterations, each of which going through analysis, design, implementation and evaluation. The later is more and more used in software industrial, often being referred as rapid prototyping, or in a more recent buzz word, Agile design [9].

Either method has a phase of implementation or iterations of implementations so that the ideas and concepts under design can be visualized and tested. This often requires the designers or the engineers to have enough programming skills to implement the system (waterfall model) or to prototype the behavior (agile design). In a teaching course of such methods, students with no experience of programming would find it difficult to fully understand the entire process.

This is exactly the difficulty one may expect in teaching industrial design students a formal specification language such as UML. Most of the students in an industrial design education are not trained extensively in programming tasks, neither do they need to. They are not software engineers, they are designers. But as a designer who is expected to design "intelligent products and services", understanding of and communicating about the structure and the behavior of a complex system are certainly unavoidable. To deal with this dilemma, instead of pushing extensive programming courses to the design students, we need to find a method that can be better integrated in an industrial design (not software design) process. Hence a technique called "Acting-out" was proposed and tried out in our module for the master students at ID, TU/e.

1.2 Acting-out as a design technique

In design of interactive, networked products and systems the intended interaction experience often emerges as a main design criterion. "Traditional" design techniques, like storyboards, or on screen simulations seemed unfit to deal with the multifaceted and dynamic character of interaction experience. Several design and design research communities have developed approaches to deal with this complexity in different stages of the design process, based on introducing real, bodily involvement in the design process. Different communities gave different names to these techniques, e.g. "Experience Prototyping" at IDEO [10], Informance Design [11], Designing actions before product [12], Choreography of Interaction [13], Gesturing out [14]. In this paper, we refer to this collection of techniques using "acting-out". These approaches have in common that they allow designers to make aspects of a product or system experiential and vivid by physically acting out (elements of) interaction scenarios. Buchenau et al. [10] describe the following advantages of acting-out techniques: understanding existing interaction experiences and contexts, evaluating design ideas, exploring new design ideas, and communicating designs to an audience.

These last three advantages, i.e., evaluation, exploration and communication in the design process, make acting-out especially valuable in the Industial Design UML module. UML is a highly abstract language, but it is a tool that, in Industrial Design practice, referes to real products and systems that make real life experiences happen. Our assumption in this module is that through acting-out a concept UML diagram, its structures, objects, properties, connections and restraints gain an experiential dimension in an early stage. This experiential dimension could help identify possible inconsistencies and flaws in the UML diagrams and help suggest improvements. Furthermore, it may help communication to other designers or software engineers what the problems or ideas for improvement are.

In the next section, the module is presented, followed by the feedback from the students.

2 Zoo of Tamagotchi animals

In this section we describe the trial based on the principles of Section 1.1. The trial was done as one "education module".

2.1 Participants

The module was a one-week course for 32 master students. Students were divided into five teams (about six students each). Each team was given the same task.

2.2 Procedure

The module was a one-week (5 full days) course, including morning sessions for lectures and afternoon sessions for a project. The lectures gave introductory information about software engineering and formal methods in general, object-oriented design process, selected UML diagrams for the project and acting-out as a design approach. The students were expected to use what they learnt from the lectures in their projects, going through a single process of analysis, design and acting-out. Each team was given the same task: Designing a system called Zoo of Tamagochi Animals.

The module was designed such that the students would not only learn and experience writing specifications, but also to learn and experience reading the specifications made by others. In this way both direct results (the specifications) and indirect feedback results (how the other participants interpret the direct results) could be obtained. This was done as follows:

- The first half of the project was for analyzing the requirements, making and specifying the object-oriented design of the system;
- In the middle of the project, the teams swapped their specifications for acting-out each other's specifications;
- The second half of the project was for acting-out the specifications received.

The students were asked to "implement" the system by acting the specification out to show how the system should work according to the specification. Students could play the objects, showing their behaviors and the communication in between. Stage props could also be used to represent objects, interfaces, and events, etc. Students were asked to use imagination and creativity in acting-out, since we did not have experience in acting-out UML in a design process.

Finally the students were asked to reflect on the process itself, see Section 2.6.

2.3 Materials

The students were allowed to use all the materials that are generally provided to the Master's program, which includes one laptop per student, which includes a lap-top, flip-over's, whiteboards, etc. and several large office-spaces and class rooms. A wide variety of materials, including modelling foam, chairs, pieces of wood were readily available as well and could be used as stage props when desired.

A preliminary description of the Tamagochi Zoo requirements was handed out at the beginning of the module. Students ware asked to design the object-oriented structure of the system - Zoo of Tamagochi Animals.

Each team received the same task description, entitled ”Designing a Zoo of Tamagochi Animals”, described by the following text:

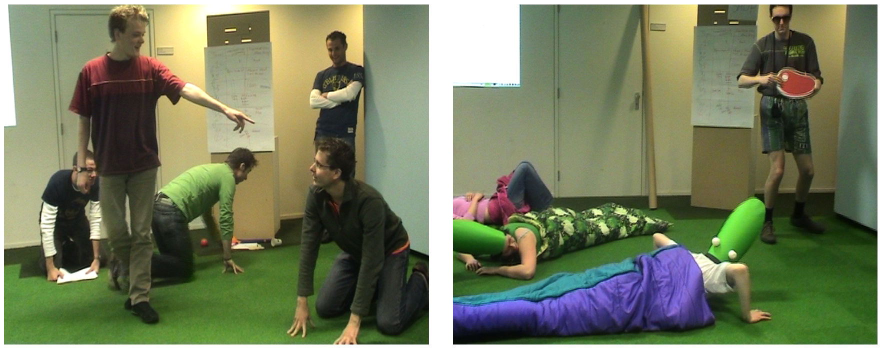
- *All animals live in a zoo.*
- *An animal has got a life after it is born. While the life goes on, the animal moves and sleeps (if it is still alive), eats (if it moves and finds food), grows (if it eats) until one day, it dies because of hunger, illness or age.*
- *Every animal has got a body. Different animals look different because of their different bodies. Every animal has got two eyes and one mouth on its body. When eyes are pinched, animals scream and can be hurt.*
- *There are male and female animals. When grownup males and females meet, they may fall into love and the love may result in baby animals.*
- *Some animals are pets. Pets have names and they wear their name plates on their bodies. People (the users) take care of their pets and feed them with food.*
- *People may play with their pets to keep their pets fit.*
- *Some baby animals will be selected by people and they become pets. The rest are left free in the zoo and they have to strive for food and try to survive by themselves.*
- *When a zoo is created, it is empty, until people get some animals from somewhere else (from shops, for example).*
- *People may exchange their pet animals.*
- *Dogs and cats will be our favorite animals for the time being. Dogs woof and cats meow. Cats are scared of dogs and can be bitten by dogs. When big dogs bark, cats scream and start running away. When big dogs fall into sleep, cats start getting together and partying.*
- *The zoo is open for other animals, including unknown ones.*

2.4 Direct results

An example of the acting-out itself is shown in Fig. 2. An example of a UML state diagram is given in Fig. 3.

2.5 Feedback results

As mentioned before, during the second half of the project the teams swapped their specifications for “acting-out”. They were not allowed to consult the teams that made the specification about the design. The teams were only allowed to read the specifications in order to understand the design. At the end of the project, each team presented (acted out) the specification together as a theater play (Fig. 2). After that students were asked to give feedback on the specifications they got from other teams, their own specifications, and their implementations (acting-out) for other teams. Here an example is included as follows:



(a) Implementing the *Pet* interface on a *Dog* (b) Feeding a *Croc* only if it is a *Pet*

Fig. 2. Acting-out

Feedback of Team 4 on Team 2's specification

...

The state diagram (Fig. 3) has an excellent division between sub- and superstates. It can be mentioned, however, that some things could be improved. For instance, you have to be able to return to the previous state. In this diagram, you could not leave the shop without selecting a pet. Exchanging pets was not very clear too. There should have been more conditions in this diagram. We did make a misinterpretation of having a shop out of the zoo, while it was specified as being in the zoo. As these formal methods are not yet very natural to us, it is apparently hard not to make intuitive decisions, but stick to what is there in the diagram.

...

Team 2's reaction on Team 4's feedback and acting-out

...

In general, the implementation as acted out was helpful, even refreshing, in showing us how our system would work. The comment was that a male as well as a female animal can give birth. This was not our intention, but we agree it was not specified clearly. We should have made an activity diagram involving two animals who are about to mate, containing the condition which animal is female. This animal would then have the method giveBirth() *and the attribute* isPregnant. *Because a male and female animal have different behaviors, we should have also specified this in the class diagram, by introducing the abstract classes Male and Female. Also, apparently the condition for the* Dance() *method (an animal starts dancing if his stomach is full) was not clear to the implementation team.*

...

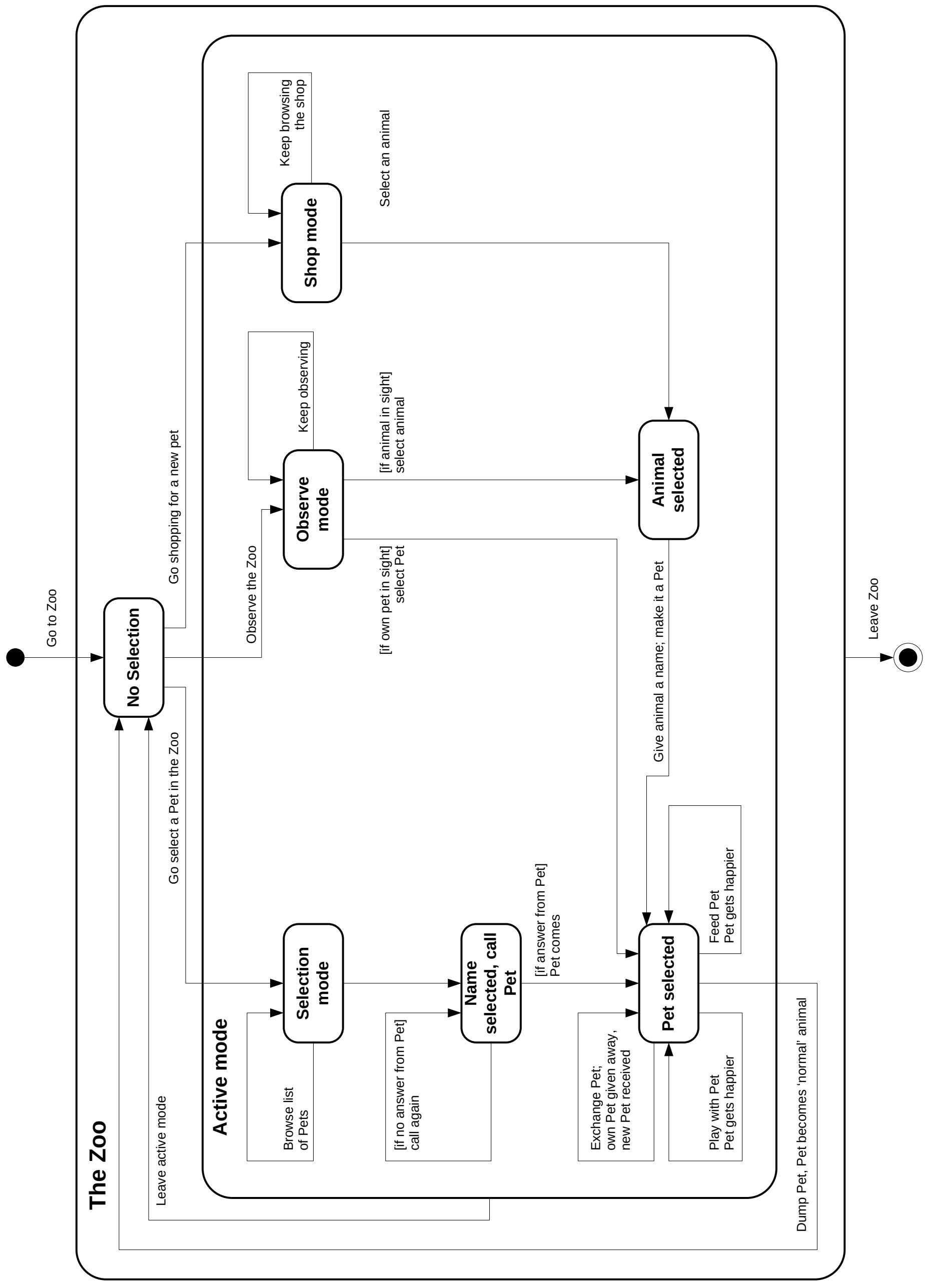

The Zoo
Active mode
Go to Zoo
No Selection
Go shopping for a new pet
Observe the Zoo
Go select a Pet in the Zoo
Leave active mode
Shop mode
Keep browsing the shop
Select an animal
Observe mode
Keep observing
[if animal in sight] select animal
[if own pet in sight] select Pet
Animal selected
Give animal a name; make it a Pet
Selection mode
Browse list of Pets
Name selected, call Pet
[if no answer from Pet] call again
[if answer from Pet] Pet comes
Pet selected
Exchange Pet; own Pet given away, new Pet received
Play with Pet Pet gets happier
Feed Pet Pet gets happier
Dump Pet, Pet becomes 'normal' animal
Leave Zoo

Fig. 3. State diagram of the Zoo, designed by Team 4

2.6 Reflection results

Students evaluated the module and indicated what they felt were the advantages and disadvantages of using acting-out in a design process that incorporates structured or formal methods such as UML. We summarize a few points:

Acting-out as an evaluation technique. Two teams stressed the helpfulness of acting-out in finding bugs in their system design.: '*...designers can better imagine the working[s] of the system 'being' it.*'

Acting-out as a pleasurable learning technique. Two teams remarked that this module was a pleasurable learning experience: '*And, of course, it is nice to do and greatly spices up a design process*', on of the teams writes.

Acting-out as a exploration technique. Three of four teams remarked that acting-out also has potential as a generative technique, when it would be applied earlier in the process.

Acting-out as a communication technique. One team remarked that acting-out was primarily a communication technique for them: '*We discovered most flaws by reasoning, but acting-out was a good way to communicate flaws in a diagram to the audience.*'

3 Discussion and Conclusion

Using acting-out as a prototyping or verification technique, students learned to understand and apply object-oriented design principles and the formal software specification methods, up to a level sufficient for basic communication with experts, in just three days. Without involving the students in heavy programming activities, this module gives more time and space for the students to concentrate on the essential issues and disciplines.

Students found learning and applying formal specification methods in this module a pleasurable learning experience. We think this pleasurability is an important aspect of our module, since it may enhance the learning experience overall.

Students stressed different advantages of using acting-out in a design process incorporating a formal specification method. The evaluation and communication aspect was experienced by the students and they were optimistic about a possible use of acting-out in the exploration phase. We can not say at this point whether acting-out could actually be beneficial as a generative tool in the context of formal methods in industrial design, but it seems worthwhile to explore this aspect of acting-out.

Furthermore, we are looking for methods to integrate industrial design processes with software design processes in designing "intelligent products and services", and we speculate the acting-out design approach may provide a good bridge that helps make the transition from a general concept to an engineering level smoother. However, establishing an acting-out based method for this bridging purpose, requires more research and experiences in intelligent product and systems design practice.

References

1. Grudin, J., Pruitt, J.: Personas, participatory design and product development: An infrastructure for engagement. In: Proceedings PDC. (2002) 144–161
2. van Welie, M., van der Veer, G.: Structured methods and creativity - a happy dutch marriage. In: Co-Designing 200, Coventry, England (2000) 11–13
3. Hu, J., Feijs, L.: IPML: Structuring distributed multimedia presentations in ambient intelligent environments. International Journal of Cognitive Informatics and Natural Intelligence, Special Issue on Ambient Intelligence and Art **(to appear)** (2007)
4. Hu, J.: Design of a Distributed Architecture for Enriching Media Experience in Home Theaters. Technische Universiteit Eindhoven (2006) ISBN:90-386-2678-8.
5. Feijs, L., Hu, J.: Component-wise mapping of media-needs to a distributed presentation environment. In: The 28th Annual International Computer Software and Applications Conference (COMPSAC 2004), Hong Kong, China, IEEE Computer Society (2004) 250–257 ISBN:0-7695-2209-2, DOI:10.1109/CMPSAC.2004.1342840.
6. Feijs, L.M.G., Qian, Y.: Component algebra. Science of Computer Programming **42**(2–3) (2002) 173–228
7. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language for Object-Oriented Development (Version 0.9a Addendum). RATIONAL Software Corporation (1996)
8. Taylor, D.: Object-Oriented Technology: A Manager's Guide. Addison Wesley (1990)
9. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall (2002)
10. Buchenau, M., Fulton Suri, J.: Experience prototyping. In: Designing interactive systems: processes, practices, methods, and techniques., New York, ACM Press (2000) 424–433
11. Burns, C., Dishman, E., W., V., Lassiter, B.: Actors, hairdos & videotape-informance design. In: CHI, New York, ACM Press (1994) 119–120
12. Buur, J., Vedel Jensen, M., Djajadiningrat, T.: Hands-only scenarios and video action walls: novel methods for tangible user interaction design. In: DIS, New York, ACM Press (2004) 185–192
13. Klooster, S., Overbeeke, C.: Designing products as an integral part of choreography of interaction : The product's form as an integral part of movement. In: the 1st European workshop on Design and Semantics of Form and Movement, Newcastle, UK (2005) 23–55
14. Ross, P., Keyson, D.V.: The case of sculpting atmospheres: towards design principles for expressive tangible interaction in control of ambient systems. Personal and Ubiquitous Computing **11**(2) (2007) 69–79