

Please don't make me draw!

Andrea Valente

Department of Electronic Systems, Automation & Control,
Aalborg University - Esbjerg Institute of Technology
Esbjerg, Denmark
av@aaue.dk

Emanuela Marchetti

Mads Clausen Institute of Product Innovation,
University of Southern Denmark
Sønderborg, Denmark
e.manum@gmail.com

Abstract—This paper describes the development of a software tool to support knowledge acquisition by means of rich pictures, for Object Oriented Analysis (OOA). Transposition of manual rich picture practise into software proved difficult, therefore we decided to follow a user-centered approach, design and implement a prototype with basic functionalities, then run a usability test with a few students and professionals. The feedback collected in the test validated the design of our prototype, and unexpectedly helped us understand how to support behavioral descriptions (i.e. events), an elusive feature before the test. At a more general level our study suggests the presence of a gap between modern object-oriented analysis practices and programmers values: some techniques presuppose design skills that are alien to our students. To improve and test further our tool, we plan to use it for whole OOA and OOD course, next semester.

Index Terms—rich pictures; knowledge acquisition; object-oriented analysis; qualitative tests; learning

I. INTRODUCTION

Rich pictures [9] are more and more part of object-oriented analysis and design courses (OOA and OOD courses). At our university, bachelor students in Computer Science as well as Engineers are required to perform analysis in small groups (3 to 6 members) and draw rich pictures as part of their projects documentation [7].

Usually rich pictures are created with low-tech support, such as whiteboards or pen and paper; students sometimes adopt some general purpose software, like a painter or a diagram-drawing tool.

Rich pictures represent knowledge about a domain (similarly to Novak's concept maps [1]), and should guide the developers during the definition and construction of the system's early prototypes. However, using a generic tool instead of a specific one has known disadvantages (see [10]). In the case of OOA it means the knowledge acquired is not immediately re-usable, especially for generative purposes. Hence, it is not possible for an analyst using a generic tool to *translate* rich pictures into rough software prototypes of the system under study.

It would of course be possible to use one of the many formal-methods software tools, but they require training from the part of the students, and mostly work with rather complete and detailed knowledge of a system, being therefore typically unusable in the analysis phase or when acquiring knowledge incrementally.

Considering all this, we decided to develop a software tool specific for the creation of rich pictures, to be used in OOA.

This software should be useful both as an e-learning tool for bachelor-level students learning OOA and OOD, as well as for practitioners, working in small teams adopting agile development methodologies.

However, transposing the manual rich picture practise into a software tool proved difficult, so we decided to follow a *user-centered* approach and involve students in a usability test. The feedback collected during the test greatly eased the task of defining the main features of the new tool.

In the following section we present an early version of our tool and discuss our ideas, sources of inspiration and related works. Section III explains how the usability test was constructed and run, and what we discovered observing our students interacting with the tool and later interviewing them. In Section IV we discuss how the feedback from the students is guiding the next iteration of the tool development; some old features needed fixing, while some new are being added. Section V concludes this paper.

II. SOFTWARE SUPPORT FOR RICH PICTURES

According to [9] a rich picture provides "a broad, high-grained view of the problem situation", and it shows *structures, processes* and *concerns* (or *issues*). It is also remarked that there is no best way to construct a rich picture. From this consideration we derive a requirement for our software tool: it should not impose a specific work-flow to its users.

When rich pictures are used for OOA, structures become visual representations of objects or grouping of objects, while processes are understood as events, changing the state of one or more objects instantaneously (as explained in [7]). As for concerns, they are often simply notes written in natural language aside of the different objects in the picture. Our tool should therefore be a drawing program, and it should allow users to create frames (to visually represent objects), eventually nesting them, to group many frames together into one. Furthermore, users should be able to describe events involving many frames, i.e. specify the processes at work in the system model. It should also be possible to write natural language notes, to support concern identification.

We want our software tool to help the user to make explicit the knowledge captured by one or more rich pictures. This will provide support for an automatic generation of (skeletons of) executable prototypes.

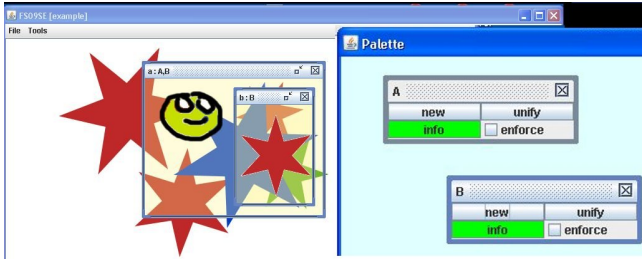


Fig. 1. The GUI of FSSE. The main window (left) contains the elements of the rich picture, while the small window (on the right) is a palette of tags.

We started implementing a software based on these requirements, in JAVA, and we took inspiration from concept maps [1], text graphs [12], and problem spaces modelling [16], as well as from pre-analysis methodological considerations such as the ones in [8].

Concept maps have a very established community, a clear definition and many good software tools. They have been used for many decades in fields like knowledge acquisition, e-learning and knowledge visualization. A concept map is typically a graph structure, constructed from labels containing natural language phrases, and arrows linking labels together. The focus is on the definition of concepts, type-like entities, while rich pictures show more concrete, instantiated examples of a system's state and dynamics.

Text graphs are an interesting attempt at making concept maps meaning more precise. However, they are text-oriented and they offer no clear way to represent different steps in the evolution of a series of concepts. While text graphs are not developed with rich pictures in mind, they suggested a direction of inquiry: what happens when text is replaced with pictures, in a text graphs?

We already explored a possible answer to this question in [16]. In that article we present a typical use case and a list of criteria for conceptual modelling software tools. As discussed in [5], rich pictures are an important tool in the knowledge acquisition process of the Soft Systems Methodology; interestingly the authors write that they

”[...] did not have the time to build up rich pictures with the dedication that one would observe [...]”

and that

”[...] these were not referred to as rich pictures but as diagrams”

thus supporting our claim that lack of proper support is relegating rich pictures to a lesser role than they deserve.

Realizing the relevance of rich pictures in analysis, we decided to make them the focus of our tool (instead of working with concept modelling in general, as in [16]). In fact, the task we proposed to our students in the usability test (see section III) is modelled after our use case in [16], simply refined in terms of rich pictures construction and knowledge acquisition.

A. The tool: FSSE

The new tool is called *Free Sketch for Software Engineering* (FSSE).

The GUI of our tool is visible in Figure 1. It is composed of 2 windows: the largest one is the main drawing area, where users draw their rich picture, and a smaller window called *palette* that contains type-level information about the elements drawn in the rich picture.

The typical work-flow of a user creating a rich picture in FSSE would be:

- Create a new, empty FSSE project.
- Draw an image in the background of the main window (using an external painter program) or alternatively import a scanned hand-drawn image. This background image serves as initial draft of the rich picture.
- Select rectangles out of the background image. Each selection turns into a frame, that the user can move around and clone, to obtain multiple copies of the same frame.
- Each frame can be given a name and a list of tags. Names do not need to be unique, and tags are like types. Tags in FSSE are a clustering device, like tags in blogs.
- More and more frames will be defined, so that the initial background image will be reconstructed by frames. This structuring process starts from a flat image, and converts it into a rich pictures made of objects (i.e. frames).
- Frames can have internal details; to declare that a user simply selects a rectangular area inside a frame, and a new frame will appear, nested in the selected one. It is also possible to insert a frame into another one, via *drag-and-drop*.
- The palette window is automatically populated, and contains at any given time a list of all tags used in every frame in the main windows (without repetition).
- Acting on a tag in the palette, the user can generate new frames, instances of that tag, or she can obtain information about the relationships between that tag and others.

Our tool does not force users to decide in which order to perform their structuring of a rich picture: for example the division of the initial background image into frames can be mixed with the declaration of the internal structure of the frames.

Users can even decide not to assign names or tags to their frames. A frame without name nor tags could be used to group other frames. This means that frames do not correspond exactly to the objects in an OOA: frames are more unstructured than objects, and become representations of objects only when users decide to assign names and tags to them.

To implement frames we drew inspiration from *mobile ambients* [2]. Dynamic tree-like structures with names and types, ambients can easily model objects and proved a good metaphor in the design and construction of FSSE.

In Figure 1 there is a frame named **a**, tagged **A** and **B**, with a face as background. Most of this frame's background is actually transparent, showing the background image below. Moreover, the frame **a** contains another frame named **b**, tagged **B**, with a star as background.

The rich picture in Figure 1 represents system in which there is an object named **a**. This object is of type **A** and at the

same time of type **B**. Object **a** contains another object, **b**, of type **B**. The tags **A** and **B** are repeated in the palette window.

A frame can have multiple tags, which corresponds to an object with multiple types (or classes). We designed FSSE to allow for multiple hierarchy, in this way a rich picture could have rich and/or loose relationships among tags, and the user can decide, at a later time, to clean up her tags into a single inheritance tag system. This kind of alteration of tags relationships (i.e. relationships among classes) reminds of refactoring practises.

As soon as a tag is used for a frame, FSSE automatically adds it to the palette window. Moreover, our program analyzes the relationships among tags, and finds out the *typical structure* of a tag. Consider a tag **T**: some of the frames tagged **T** in the rich picture might have a nested frame tagged **T1**, and some not. If **T** and **T1** were classes in a UML class diagram, **T1** would be associated to **T**, with cardinality **0..1**. The same association is computed automatically by FSSE, and we would call that *typical structure* of the tag **T**.

Events are not yet supported in Free Sketch. It was unclear to us, before running the usability test with our students, how to best add them.

Concerns can easily be expressed by writing comments directly on the background image.

III. THE TEST

A. Theoretical framework

At the current development stage of FSSE, feedback from students was needed in order to complete or even to substantially change the tool. As we wanted to develop our tool according to the students' necessities, we were interested in gaining a detailed account of their working habits, how they would like to work, and eventually be supported by a tool like ours. Hence, we decided for user-centered design, which allows developers to iteratively collect feedback from end users, involving them in the design process. As discussed by Redström in [13], design moved gradually its focus from material objects to users' experience. This change was motivated by the fact that many design products were praised by the design community, but were not as successful according to end users [11]. As a consequence there are nowadays many approaches to collect knowledge about users and inform the design process with their needs. In our study we opted for: ethnographic observations, analysis of video recording, and situated interviews. Qualitative analysis of video recordings is among the most applied technique in representing users' needs to support the design process, as discussed by Cramer and others from Intel [3]. Situated interviews are aimed at collecting detailed knowledge about users and their context of practice [17]. It prescribes to start with open questions and gradually to re-focus attention on details of users' utterances and actions, asking for examples. We preferred interviews to questionnaires to find out what really mattered to the students and to show them that we really cared for their contribution to the design; and this was explicitly appreciated by one of them.

These methods are not concerned with controlled experiments and statistics, but focus on actively involving users in the design process and on making sense of the details of their interaction. This means that there is no minimum number of measurements required [15], and that user-centered methods are well suited for research with small user groups. In our case, the test we conducted was part of the early stage of our design process and it was conceived to involve directly the students in the development, providing them with a ground for expressing their perspective. More tests and workshops will be needed in order to complete and finally validate our software tool.

B. A qualitative usability test: set up and task

Participants to our test were a professional programmer and four engineering students at the 5th semester of their bachelor, who have recently started a course about OOA and OOD. Our aim was to evaluate how users may perceive a tool like FSSE, if it is seen as useful, easy to use, and if it adequately supports work-flow, for individuals and groups. The students were divided into two groups and were invited into a classroom, one group after the other. The students were sitting at a desk, with a laptop running FSSE, and we were in front of them, observing their reactions, taking notes and filming them with a video-camera. The laptop was connected to a projector, so that we could see (and film) their actions on the wall behind them (Figures 2 and 3). The test was articulated into four stages: first we showed the students a 5 minutes video-tutorial, then we introduced them to a task, and we left them free to familiarize with the tool before starting; at this point we started filming.

As shown on the tutorial they had to create one or more rich pictures, identifying objects, classes and events, regarding a pizza restaurant (like in the use case in [16]). A customer can order a pizza from a menu talking to a waiter, the pizzas have to be baked and can be served with wine or other beverages. Finally the customer pays the waiter, and a conflict may emerge between them about the order. The task proposed was designed as a typical modelling problem, of the kind students already faced during their OOA and OOD course, so that they could reflect upon their own experience to evaluate the tool. It was also our interest to observe how FSSE fitted within the team work-flow and how it affected *reflection in action*, intended as a process of critical thinking while performing a skilled practise [14].

After the task completion, we asked them a few open questions about their impressions of the tool. A list of questions was prepared, but it was intended mostly as a reference.

- How did you like the tool? General impressions.
- Given you experience with object-oriented modelling, do you think the tool can facilitates object-oriented analysis and design or no? How and what will you change?
- Do you think that the tool makes object-oriented analysis and design more understandable for users or not? How and what will you change?
- How do you think it will be possible to define events in Free Sketch, within the current user's interface and how

could it work?

- Do you think you would like in future to use a tool like this in your work or not? Why?
- How do you think the tool supported flow of team work? Did it facilitate team work or made it more complex? How could the tool be improved?
- Other comments? What other changes will you suggest to make the tool more effective in supporting object-oriented analysis and design in software development or its understanding from a student's perspective?

During the test in fact we started from the first question and then we adapted to the students' comments, who sometimes covered several issues at one time or even proposed new issues.

For practical reasons we could not meet the programmer in person, we gave him the program and the tutorial, he solved the task in the tutorial and sent us feedback by e-mail.

C. Collected Data

The students responded quite positively to the test and the prototype, it seemed as we were on the right track. They were relaxed with their mates, probably because they were already working together in the same group for the course and the semester project. They sat one aside of the other, one interacted with the computer, the other read from the paper with the task description and often pointed at the screen with one finger, then they talked a lot deciding together on what to do.

We expected the time required for the test to be around half an hour for each group, but in fact it took one hour, as they used extra time to get familiar with the interface. However, they all said that the purpose and the interface of the tool were easy to understand.

Surprisingly for us, drawing appeared as a main concern to all the testers, they felt visibly uncomfortable when they needed to draw new icons, specifically arrows and the menu for the restaurant. The first group expressed their uneasiness exchanging a worried, ironic look, then after several attempts they drew a menu and arrows to connect the pizzas to it (as visible on the back of in Figure 2). A member of the second group said ironically: "Ok, we suck at drawing!", then they modelled the menu as a new frame with the pizzas nested inside, avoiding to draw.

The feedback we received from the programmer was very similar, he wrote that he likes the tool, and he also remarked that he does "not want to play with graphics, it sucks!", when analyzing a system. He then suggested to add a library of free, pre-drawn icons and arrows. In this way he proposed a constructive solution to the same problem that was signaled also by the two groups.

These reactions revealed programmers' perspective on agile methodologies, which include soft skills, such as prototyping and drawing to make rich pictures and storyboards. These skills are taken from the field of design, therefore do not belong to the curriculum of a computer scientist or an engineer, and are not even part of their system of values. Through the interviews we realized that drawing on paper is perceived as an annoying interruption in the process of reflection in

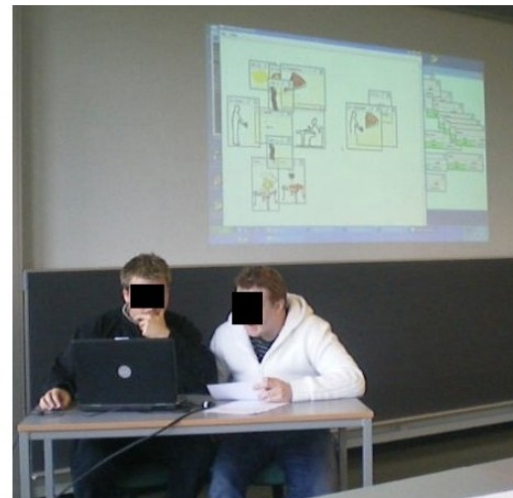


Fig. 2. First group drawing a menu for the pizzas. They decided to represent events by clustering frames and connecting frames with arrows.

action. According to them, it takes time to make a decent icon, approved by the whole group, as they have often "to draw, erase and draw it again", hence "just having a tool would help!". Moreover during the test they were quite precise in selecting icons and spent time erasing the superfluous parts in the external painter, to make them more *readable*. Their quotes and actions show that, despite their dislike for drawing they want nice icons in their rich pictures, but do not want to do them by themselves. In this sense, features like automatic insertion of pre-made icons or creation of icons through selection from background pictures (as currently available in FSSE), do provide a smoother work-flow also from a team work perspective.

It was also proposed, both from students and researchers, the possibility to introduce collaborative user interfaces, to turn the main drawing window of FSSE into a sort of shared, remote desktop.



Fig. 3. Second group defining events: they choose to represent events by frames nested into a new frame.

Definition of events is central during OOA, but events were missing in the prototype tool that we tested. Nevertheless, the task assigned to the students required to try and represent events. They all expressed their perplexity for the lack of support, but found their own way to solve the problem. Interestingly they all tended to represent events as scenes of a storyboard, but they kept the approach they used to define complex objects. The first students grouped a few frames and connected them with arrows (Figure 2), while the others grouped frames by nesting them into a fresh new frame (Figure 3).

Finally the students seemed to find confusing the distinction between names and tags, so that they discussed with each other how to use the two labels to keep their rich picture coherent. However, it did not take long before they understood that tags work as types and names are just arbitrary identifiers to be assigned to the frames. One of the students showed to be a little frustrated by this ambiguity and said: "if it is a type, why do not call it type!". In FSSE we wanted to use the term *tag*, since tags are supposed to be used with more freedom than types (see section II.A).

Moreover, to facilitate overview of the system created, a student proposed that when a frame is selected, it should be highlighted, together with the other frames sharing its tag.

Furthermore, FSSE was appreciated for its flexibility, enabling users to keep their favorite work-flow and their understanding of rich pictures making. Such flexibility implies that users can start modelling from a chosen level of abstraction, and mix the various activities as they like. This is what is called *middle-out modelling* in [16].

One of the students also commented: "the nice thing is that this tool doesn't impose me a specific way of thinking, it doesn't assume I am stupid!". Hence work-flow flexibility can give a feeling of being more *in control* of a tool, less subject to some prescribed way of thinking and working.

IV. NEXT ITERATIONS

A. Features emerged from patterns of use

During the tests we noticed a particular pattern of use of FSSE. A user would create some frames, give them names and tags, and cluster them in an empty area of the rich picture. This is in fact a kind of spatial reasoning that FSSE supports. Later the user will proceed to create new frames by cloning the ones in the cluster. The cluster itself can be considered as an extension to the FSSE palette: instead of just having tags in the palette, the user could explicitly clone a frame into the palette. That frame can then be referred to as a *prototype* of that tag, i.e. a typical representative of the tag. It is interesting to notice that the students were adopting this *prototype clustering* approach especially when trying to cope with events; in our usability test, this was an impossible task, given that no support existed in FSSE for events at that time.

We decided to promote the prototype clustering pattern of use to a full-fledged feature, and implement it in the next version of FSSE, in the form of an extended palette. This new palette will have an area to store prototypes for each tag.

Our testers also asked to have labelled arrows in FSSE, to connect frames and define relationships (e.g. events) directly on top of specific frames. We have therefore added a simple way to draw labelled arrows in the next iteration.

B. How to support events

Having extended the palette with prototypes and having added labelled arrows, we still lacked events support, so we reflected on the relationships between frames, tags, prototypes and arrows. We realized that the prototype-extended palette would be the perfect place to define events: events are relations between possible states of an instance of a given tag. In our case prototypes are examples of possible instances of a tag, so events could be drawn as a labelled arrow between two prototypes.

In fact the palette contains prototypes, which are special frames the user stored persistently during her exploration of the system concepts and construction of her rich picture. Her palette will eventually contain a representation of all possible objects (grouped by type) that are needed in the current rich picture, and probably in other future rich pictures describing the same system. Therefore, we will use a labelled arrow between two prototypes **p1** and **p2** both tagged **T** to represent an even. Such event, when *activated* on a clone of **p1**, will replace it with a clone of **p2**. This is consistent with our metaphor that frames are objects, prototypes are possible states of objects, and events change the current state of an object into another one of its possible states.

C. Other features

We are also considering to provide a plug-in mechanism to FSSE, to enable users to define their own mapping from rich pictures to external formats or to code.

Given that the new palette will make tags even more similar to classes, more refactoring-like operators will be implemented.

To draw we currently rely on a free external painter (Java Image Editor, by JH Labs). In future releases we would like to have internal painting capabilities, to provide a more uniform environment while drawing rich pictures.

When events are correctly supported and Free Sketch is fully functional, it should be possible to use the tool also to *validate* one's understanding of a system, in analogy to concept maps validation [4].

V. CONCLUSION

This paper described the features and development of Free Sketch SE, a software tool to support knowledge acquisition via rich pictures, for object-oriented analysis. To validate and complete the initial prototype of the tool, we ran a qualitative usability test. Although limited to a small group, the test provided meaningful feedback to be used in a new development iteration.

Reflecting upon emergent patterns of use, we could progress in the definition of the tool's features, and improve on existing ones. For instance before the test, it was unclear to us how to

support events definition. The feedback received forced us to re-define the relationship between objects and types, and that in turn suggested us how to include support for events.

Moreover, we realized how modern object-oriented development methodologies, such as agile methods, are informed by design, and sometimes assume design skills that programmers do not have or do not value; this *mismatch* is interesting and should be investigate further, hopefully in cooperation with a larger number of students and professional programmers.

On the long run, we plan to improve FSSE, test it further, and propose it as the main tool for a whole OOA and OOD course.

REFERENCES

- [1] A. J. Cañas, R. Carff, G. Hill, M. Carvalho, M. Arguedas, T. C. Eskridge, J. Lott, R. Carvajal, *Concept Maps: Integrating Knowledge and Information Visualization Export*, Knowledge and Information Visualization journal, pp. 205-219, 2005.
- [2] L. Cardelli and A. D. Gordon, *Mobile ambients*, Theoretical Computer Science. Volume 240, Issue 1, pp. 177-213, 6 June 2000.
- [3] M. Cramer, M. Sharma, T. Salvador, and R. Beaugard, *Video utterances: expressing and sustaining ethnographic meaning through the product development process*, in Proceedings of EPIC 2008. Ethnographic Praxis In Industry Conference, Copenhagen, Denmark, pp. 116-127.
- [4] A. Dietrich and C. M. Steiner, *Representing Domain Knowledge by Concept Maps: How to Validate Them?*, In proceedings of the 2nd Joint Workshop of Cognition and Learning Through Media-Communication for Advanced e-Learning (JWCL), Tokyo, Japan 2005.
- [5] K. Kotiadis and S. Robinson, *Conceptual modelling: knowledge acquisition and model abstraction*, in Proceedings of the 40th Conference on Winter Simulation (WSC '08), ISBN: 978-1-4244-2708-6, pp. 951-958, Miami, Florida, 2008.
- [6] J. Löwgren and E. Stolterman, *Toughtful Interaction Design. A design perspective on information technology*, MIT Press, USA, 2005.
- [7] L. Mathiassen, A. Munk-Madsen, P. A. Nielsen, and J. Stage, *Object-Oriented Analysis & Design*, Marko Publishing, ISBN: 87-7751-150-6, 1st edition 2000.
- [8] H. C. Mayr and C. Kop, *Conceptual Predesign - Bridging the Gap between Requirements and Conceptual Design.*, In proceedings of the 3rd international Conference on Requirements Engineering: Putting Requirements Engineering To Practice. ICRE. IEEE Computer Society, pp. 00-90, Washington DC, April 06 - 10, 1998.
- [9] A. Monk and S. Howard, *Methods & tools: the rich picture: a tool for reasoning about work context*. Interactions, vol. 5, num. 2, pp. 21-30, Mar. 1998.
- [10] B. A. Nardi and J. A. Johnson, *User Preferences for Task Specific vs. Generic Application Software*, Conference on Human Factors in Computing Systems CHI 1994, pp. 392-398, Boston, Massachusetts, USA, 1994.
- [11] D. A. Norman, *The Design of Everyday Things*, Basic Books, NY, 2002, ISBN: 978-0465067107.
- [12] E. Nuutila and S. Torma, *Text Graphs: Accurate Concept Mapping with Well-Defined Meaning*, In Proceedings of the First International Conference on Concept Mapping. Pamplona, Spain, Sep. 14-17, 2004. Volume 1, pages 477-485.
- [13] J. Redström, *Towards user design? On the shift from object to user as the subject of design*, Design Studies Volume 27, Issue 2, March 2006, pp. 123-139.
- [14] D. Schön, *The reflective practitioner. How professionals think in action*, Ashgate, London, UK, 1991.
- [15] D. Silverman, *Doing Qualitative Research: Second Edition*, Sage Publications Ltd, 2004, ISBN: 978-1412901970.
- [16] A. Valente, *Visual Middle-Out Modeling of Problem Spaces*, International Conference on Information, Process, and Knowledge Management, pp. 43-48, February 01-07 2009.
- [17] S. Yliriksu and J. Buur, *Designing with video*, Springer, 2007.