Software Cinema

Bernd Bruegge, Martin Purvis*, Oliver Creighton and Christian Sandor

Department of Computer Science
Technische Universitaet Muenchen

***Department of Information Systems**
University of Otago, Dunedin

Abstract.
The process for requirements elicitation has traditionally been based on textual descriptions or graphical models using UML. While these may have worked for the design of desktop-based systems, we argue, that these notations are not adequate for a dialog with mobile end users, in particular for end users in "blue collar" application domains. We propose an alternative modelling technique "Software Cinema" based on the use of digital videos. We discuss one particular example of using Software cinema in the design of a user interface for a navigation system of a mobile end user.

Introduction

The advance of electronics and telecommunications technology is not only opening up new opportunities for computer system development, but it is also forcing basic changes in the way we look at software development as such. In particular because of the developments in implementation technology, there is a growing realisation that it is possible to embed situated computational modules into more and more of the real-time activities of human engagement. And this realisation is leading to increasing demands to build such systems – even before we have the procedures and techniques necessary to build them. Thus, although software developers have sometimes been admonished for building systems mostly "for technology's sake" and not letting the immediate needs of the customer drive software development, we now have a situation where the latest technology *is* an important driver: it is pushing the development of a qualitatively new kind of software system, *distributed interactive systems* [Purvis 2002]. In this paper we discuss a new developmental technology, *Software Cinema*, which we believe will be important for the future development of distributed mobile interactive systems (DIS).

DIS should be viewed differently from those computer systems that have conventionally been considered to be more or less encapsulated modules that must interact with the "real world" by means of an input (from the real world) -> process -> output (to the real world) perspective of operation. For DIS, the real world is part of the system and the 'processing' element cannot always be considered in isolation from all the complex processes of the world: the DIS is simply another interactive component in the already complex world-system matrix. Such systems can sometimes be referred to as "blue collar" systems: the users of the systems are people carrying out ordinary activities in everyday life and are typically unfamiliar with computer usage may not even be aware of the existence of computational elements in the system. For a concrete example, consider a new type of system that will be

increasingly important in the coming years: mobile augmented reality systems. Such systems combine augmented reality, computational, and wireless technology to enable a human user to have an enhanced real-time interaction with his or her environment. Because these systems must be location-sensitive, application-dependent, and merged closely with human physiognomy and perception, they must be mostly viewed in the context of a complex, larger real-world system. In this paper we present *Software Cinema* as a new agile development methodology targeting the (still) large gap between customer requirements and software models. We make a case for motion pictures as a semi-formal representation of models and present an example of the issues when designing a user interface for a mobile augmented reality navitation system.

Software Development Technology

Schematically, we can think of software development as generally requiring a mapping from some sort of mental conception of a solution to a real world problem to a computer implementation of the solution in hardware and software (Fig. 1). Note that the "real world" conception shown in Figure 1 may represent a world that does not (yet) exist -- it is a "brave new world" that may be envisioned by people who are thinking about the development of new systems. Of course the mapping from the real world conception to an implementation is acknowledged to be difficult and error-prone, particularly for complex systems, and thus the mapping is not considered to be straightforward and unidirectional. Computer technology has been developed over the years to assist in this mapping procedure, but the development has largely been from the bottom up: additional layers of abstraction have been added to the lowest level so that the mapping from these additional layers down to the hardware/software implementation layer is relatively straightforward (Fig. 2), such as

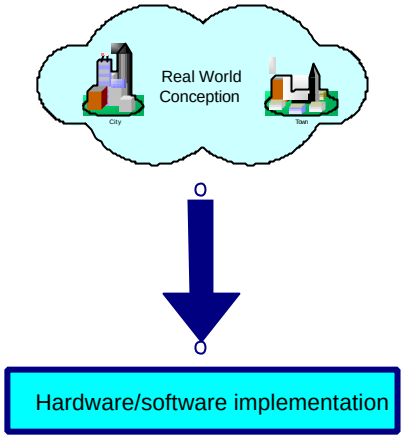

Figure 1. Software development.

- Generic computational architectures
- Interpreters that implement virtual machines for existing machine object code
- Higher-level (more mathematically abstract) languages and compilers
- Modelling representations and case tools supporting functional, system, and object-oriented analysis (including UML, IDL, MOF, MDA,…) [OMG 2003].

Each layer represents a formal or semi-formal modelling representation at a particular level of abstraction, such that mapping from one layer to the next is made as error-free as possible by means of model transformations. While this technology has been

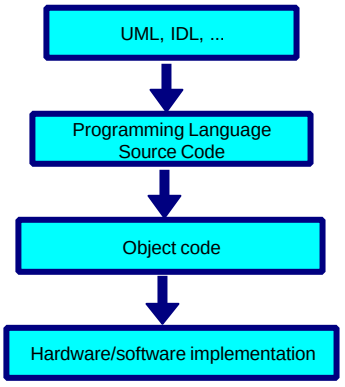

Figure 2. Software development technologies.

enormously useful, there is still a gap that remains that is associated with the problem of mapping from the original real-world conception of the problem down to the highest layer of our modelling hierarchy (Fig. 3). There are some salient points about this remaining mapping gap:

- The nature of the gap is not primarily a matter of abstraction but a matter of *richness.* The real worlds of our experience, even our visions of these real worlds, are far richer and more complex than can be conceptualized and represented in our modelling notations.
- The 'size' of this gap, we claim, represents a large proportion of the 'distance' between conception and implementation, in fact far larger than what is shown in Figure 3.
- Further 'bottom-up' development of abstract modelling layers that can be semi-automatically transformed to lower layers may be difficult to produce, and they will not reduce the size of the gap further. They already throw out the ambiguous and 'illogical' nature of the world of our experience and imagination.

But this very gap is where the greatest difficulty lies and where the most serious development errors occur. It becomes increasingly difficult in connection with the development of complex distributed interactive systems.

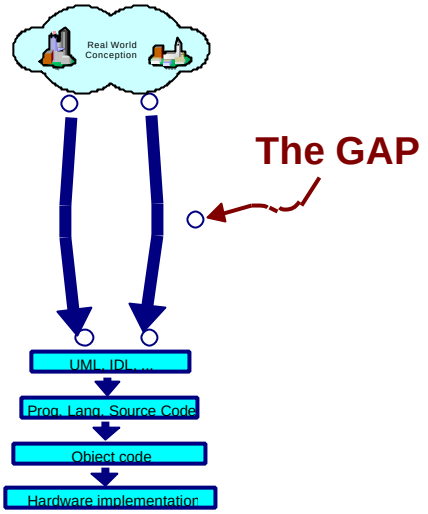

Figure 3. Software development with existing technology.

An awareness of this gap is what underlies the popularity and success of current agile software development methods [Cockburn 2001a; Cockburn 2001b]. Because most software errors occur in the process of developing models across the gap, agile software promoters have advocated deemphasising time spent on formal modelling and other 'lower-level' (with respect to Figure 3) aspects software development. Instead, they place the greatest emphasis on rapid short-term development cycles so that frequent iterations involving consultations with the customers (who implicitly maintain much of the real-world problem conception in their heads) can be carried out. Thus agile software engineering recognises the almost treacherous nature of the gap, but if offers little support for navigating across the gap -- other than stressing that light-weight development processes and frequent testing can facilitate many crossings back and forth across the gap so that errors can be found and eliminated.

The Difference Between Experienced 'Reality' and Abstract Models

While this is not the place to enter into a full-scale debate concerning the possible philosophical interpretations of ontology and epistemology, it is important in our present context to discuss briefly the nature of the terms shown in Figures 1, 2, and 3. The "real world" depicted in the Figures 1 and 3 is the world of our everyday

experience, such as the experience one might have of seeing and smelling a rose[1]. As such, there are really multiple such 'worlds', since each person has his or her own unique experiences. These experiences contain all the richness, complexity, and contradictions that characterise our ordinary experiences.

On the other hand, the modelling layers in the abstract modelling hierarchies shown in Figures 2 and 3 are all "symbol-based": they are representable in terms of 'linguistic' components and usually are or have the potential of being represented by a mathematical or logical formalism [Agre 1997]. It is this abstract, symbolic nature that gives the models their power -- they can be transformed into other representations according to mechanical or semi-mechanical procedures. Yet at the same time this abstract, symbolic nature also lies behind the reasons why such models cannot capture the ultimate richness of experience. No model of a rose can fully capture the richness of the experience of seeing and smelling one[2].

These observations may seem all too obvious, but they deserve mention here, because the mental lives of most people (customers and system developers alike) are spent mostly in the world of experience, not in the world of abstract modelling. When we encounter a problem in the course of activities, it is in the 'real world' of experience, and when we imagine a possible way out of that problem, it also in terms of this same world. In fact, our experience of this world is sometimes contradictory and illogical, and our imagined solutions are often even further from logical accountability. Such imagined 'visions' of the world are difficult even to cast in a modelling representation that assumed logical consistency. Yet it is this kind of vision of "impossible things" (at least impossible at present) that can lead to great advances. Thus people in past ages dreamed of conversing over great distances, flying through the air, and travelling to the Moon long before these activities were physically realisable.

Thus, we assert that visions of what can be, what may be built, ways to solve our current difficulties, are envisioned first in the real world of experience, not in the world of abstract models. One does not first envision an alternate UML model that may represent a solution to an existing problem; one first imagines something physical, and perhaps later constructs a UML model to support that vision.

Consequently, we hold that support is needed for understanding the real world experiences and visions that people have about the world. In fact support is needed in several respects:

- Support for understanding someone's (a customer's, say) experiences and 'understanding' of the existing world,
- Support for understanding a visionary representation of the world has not yet come to be, but perhaps could,
- Support for understanding how a newly-constructed system is experienced by the intended target audience or users,

[1] This is not to say anything about the possible existence of an ultimate ("thing-in-itself") reality that may underlie our experiences.

[2] In fact one might imagine an argument claiming that the richness of the experienced world is so great compared to the modelled world, that the 'height' of the modelling hierarchy shown in Figure 3 should be miniscule.

- Support for integrating and merging these above, visionary views of the world with the abstract modelling notations and methodologies that are used in software system construction.

We contend that our approach of 'Software Cinema' can provide important support in all of these areas and help software development teams navigate across 'the gap' of Figure 3.

Software Cinema

Software cinema employs existing digital video (DV) technology to develop film documents that are used in the software development process. Just as commercially made motion pictures do, the software cinema films represent perspectives, or particular points-of-view, of the world as seen by the filmmakers. Using non-linear DV editing techniques, the filmmakers can assemble a collection of scenes that can be traversed in differing sequential orders to examine various possible scenarios of an envisioned system. These film scenarios could represent differing, possibly contradictory, views on the part of various stakeholders. Alternatively, they can represent alternative, branching event sequences based on differing circumstances. We emphasise that the film scenarios could, just as commercial films, sometimes represent a reality that is not yet realisable -- a vision of what is desired.

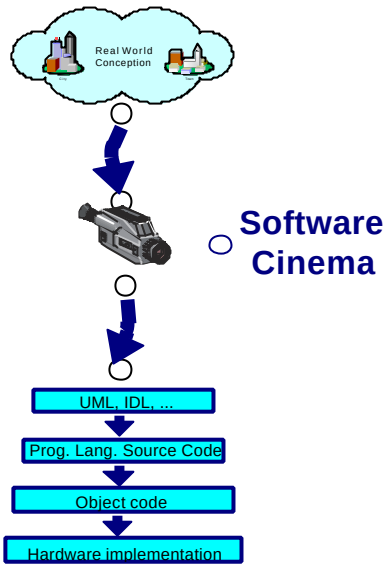

Figure 4. Software Cinema in the Gap.

The film scenarios are stored as system document artifacts that represent a rich perspective on the real world that is envisioned by the system stakeholders (Figure 4). The film segments may display the real-time sequence of events that are pertinent to the system (of course, these real-time segments may be paused or viewed in slow-motion as dictated by development needs). Because the films are stored digitally, clips can be embedded in other software documents, and, alternatively, other software documents, commentaries, and notes can be embedded directly within the film so that pertinent issues and rationale discussions are placed directly in the real-world context with which they are associated.

Motion pictures as a semi-formal representation

Another important usability factor of DIS is the individual's experience of the flow of time. In fact, much of the richness of experience stems from the timing and constant (linear) feedback of the real-world objects we interact with. By creating a DIS to support a certain activity, we shape the way people experience the time spent on this activity. This shaping of experience is already well understood in the world of cinematography -- there is a saying that the actual movie is created at the editing

table. Explaining simple, but nevertheless experience-rich, facts about a system, such as concurrency or quasi-simultaneous events, iare extremely difficult in abstract models or linguistic explanations. But these are the cornerstones of interactivity. In fact, one of the criticisms of UML is its focus on the system structure rather than interactive behaviour [Glinz 2000].

Software Cinema intends to make use of the 'blunt tool' of *motion picture language* as a semi-formal representation of the system model. In a regular movie the main intention is entertainment. Using movies for modelling, we aim to create a rich experience with visionary scenarios for a system that doesn't exist yet. We can even go a step further: while scenarios are usually constrained to operate in a single thread in time, the technical possibilities of digital movies, such as simultaneous video and audio streams, allow the user to explore multiple scenarios. By being able to select different parts of the scenario or alternative scenarios, thus effectively allowing the customer to explore a requirements space at design time, we believe we are closer in narrowing the requirements gap mentioned earlier.

Digital editing workstations support multiple video and audio tracks, which can be stacked on one single timeline. This metaphor which can be compared to a musical score: much like a conductor who can grasp the essential complexity of a musical piece by simply looking at its score, digital editors can see the complexity of a film with a glance. This enables editors to experiment quickly with cuts and transitions, to see "what works and what doesn't work" in the film.

In Software Cinema we take this idea one step further. The different perspectives become first-class citizens in an agile model: The decision concerning which perspective is actually shown is deferred to the viewer and possibly even the end user. In film, the filmmaker leads the audience in a certain direction to keep emotional intensity or interest high. In software cinema we aim to make the experience of a visionary scenario through the vehicle of preproduced interactive video. Making this film is now part of the specification process (or replaces the traditional process completely).

We can use existing functionality in video editing programs to support this idea. Take for example, the technique of blue-screens, which enables the composition of a scene from several superimposed parts, such as a video track showing an actor, and a video track showing a flight through clouds. With blue-screen editing, we can easily show the actor how he flies through the clouds. The technique is rather complex when both video tracks are actually filmed. This is because in order to do chroma keying, the action on the front layer has to be filmed in front of a uniformly lit monochromatic screen (which should be a color that is not part of the action at all, as all areas of this particular color will be made transparent). For Software Cinema scenarios involving user interfaces, this is somewhat easier to achieve, as the user interface of the to-be-developed software is usually digital; this can easily be superimposed onto the filmed material.

The real benefit of Software Cinema is that it can produce the richness of the experience, which is comparable to actually using a system, without having to develop a functional prototype. Producing a visionary walk through the system which looks as if the system already fulfills all the requirements, can be done early in the

process, when the requirements are still under discussion. As a result, the interaction of the customer or end user with the system developers can be focused more on the experience than the specification of the system.

Case Study: User Interface Design for Mobile Augmented Reality Systems

To substantiate our ideas, a navigation system for mobile mechanics. called TRAMP, was developed at Technische Universität München in a senior-level software engineering course by 50 students.[3] The system was developed for a real customer Inmedius (http://www.inmedius.com/) using scenario-based design. The scenario includes a navigation sequence, in which a walking mechanic with a wearable computer is guided to a stranded customer with a broken-down car. The navigation information is displayed inside a Head-Mounted Display (HMD worn by the mechanic.

The problem in designing a user interface for such a scenario is that no standard interaction mechanisms are established, yet, for what one could call "blue collar" applications for the mobile worker. In such a case, it is crucial for the success of a project to discuss the possible design alternatives with the end user in a way in which they understand their choices. For truly mobile applications two types of user interface interaction styles must be considered: an egocentric view and an exocentric view based on the World-in-Miniature metaphor [Stoakley 1995]. To explore these two different styles, the user must to be able to switch between them in an intuitive manner. In the following, we describe how the software cinema can help to support the requirements elicitation process under such circumstances.The egocentric view is the primary user interaction mode where navigation information is displayed. Hoellerer et al. [Hoellerer 1999] demonstrated two types of possibilities:

- Displaying an arrow to the user that points into the right direction (see figure 5a). The arrow is updated continuously in the head-up display while the user is moving.
- Augment virtual footsteps onto the floor. Those footsteps mark the best path for the mechanic. (figure 5b)

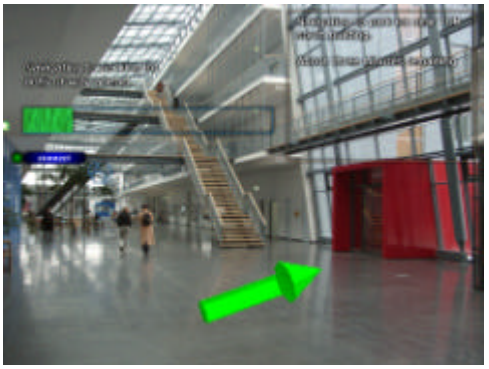

Figure 5a Rotating Arrow

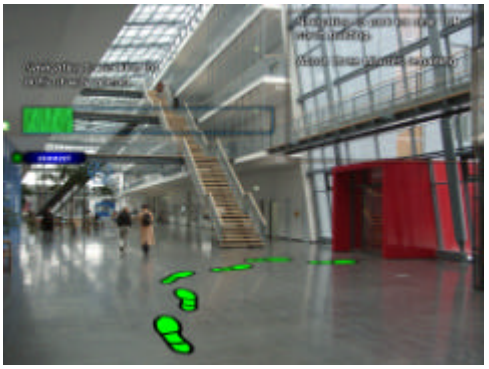

Figure 5b Augmented footsteps

[3] Description of the TRAMP System can be found under http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/ProjectTramp

The exocentric view gives users an overview of their position with respect to the environment. In the navigation task, a map with the user and the highlighted destination can be displayed. Again two alternatives are possible: a "north-up" map (figure 6a) and a "view direction-up" map (figure 6b). According to Darken, the "view direction-up" map is slightly better suited [Darken1999], but there are specific situations, where the north-up map is more appropriate.

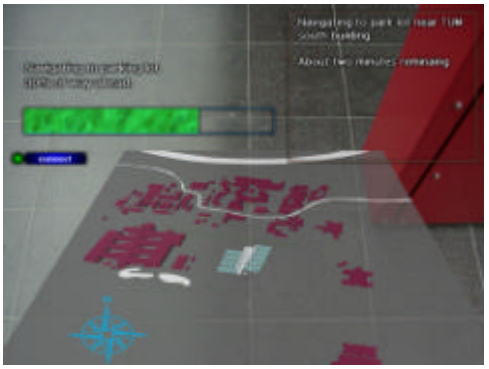

Figure 6a North-up map

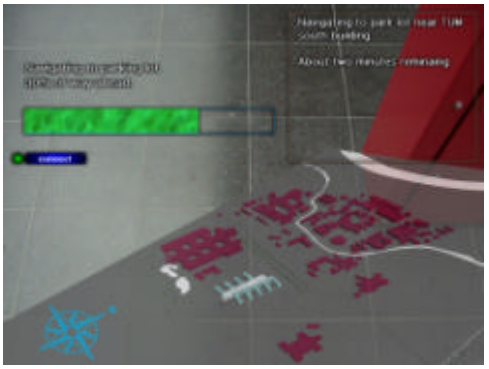

Figure 6b Viewdirection-up map

To let the end user decide which view is better, a software cinema-based prototype needs to support switching between the two types of views. A naïve implementation would use the user interface metaphor from desktop-based systems, which asks the user to press buttons to select the desired view type (see Figure 7). However, a keyboard or mouse can be quite disturbing in a mobile application. The alternative implementation actually used in the prototype is based on a suggestion from Hoellerer [Hoellerer 1999]. By attaching a gyroscope to the user's head, it is possible to determine its rotation around three axes. Whenever the user looks straight ahead, the egocentric view is displayed. When the user looks down towards the feet, the view is switched to the exocentric view (as if the user would look down onto a real map).

The operation and working of this type of user interface exploration cannot be demonstrated with a UML diagram. It also can not be shown on paper, because the switch between the two user interfaces is done with the movements of the head (A movie demonstrating this type user interaction can be found on the TRAMP project homepage). Instead, it requires the kind of multimedia software engineering documentation that we are developing with Software Cinema.

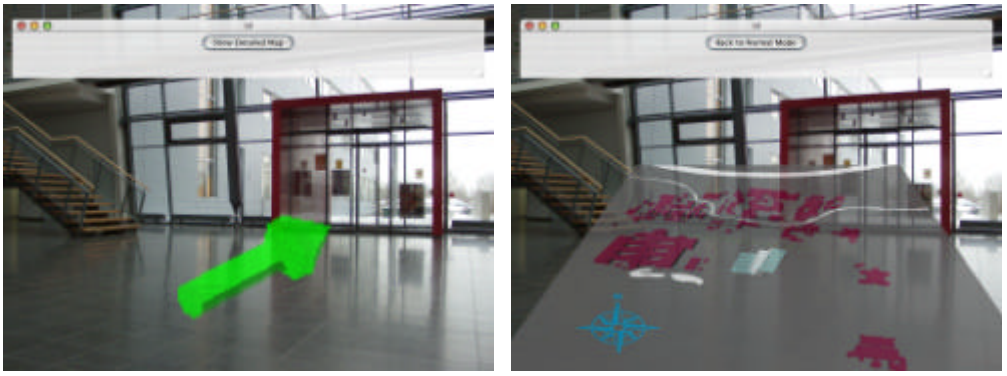

Figure 7 Desktop-based User Interface: Selection of desired view with buttons

Conclusion

In this paper we have argued, that the traditional software engineering modelling notations are not sufficient to model the emerging requirements of an end user. This holds especially true for applications with mobile "blue collar" workers.
We propose to use a modelling technique based on motion pictures to visualize visionary scenarios and a complicated design space involving many alternatives. Discussing possible designs with movies is much more natural for many end users than an abstract specification based on a textual or two-dimensional notation such as UML. With the advance of digital video and powerful editing tools, these movies can be now made in a very short time, allowing multiple iterations with the end customer. The accepted movies could then be use for the generation of the more traditional models, in particular scenarios and use case diagrams. We also speculate, that they can be used as a basis for the development of the system.

In addition, we believe that Software Cinema can play a useful role with software engineering development teams that are distributed over large areas. The movies that are developed in this process can serve as scenario "anchors", providing a common visualisation that will be available for reference on the part of various development team members.

References

[Agre 1997]: P. Agre, *Computation and Human Experience*, Cambridge University Press, Cambridge, 1997.

[Cockburn 2001a]: A. Cockburn and J. Highsmith. Agile Software Development: the Business Factor. *IEEE Computer*, 34:9, 2001.

[Cockburn 2001b] A. Cockburn and J. Highsmith. Agile Software Development: the Business Factor. *IEEE Computer*, 34:11, 2001

[Hoellerer 1999]: T. Hoellerer, S. Feiner, T. Terauchi, G. Rashid, and D. Hallaway.

Exploring Mars: Developing Indoor and Outdoor User Interfaces to a Mobile Augmented Reality System. *Computers and Graphics*, 23:779–785, 1999.

[Darken 1999] : R. Darken and H. Cevik. Map usage in virtual environments: Orientation issues. In *Proceedings of IEEE VR '99*, pages 133–140, 1999.

[OMG 2003]: OMG Specifiations. Object Management Group. http://www.omg.org/gettingstarted/overview.htm#OMGspecs, 2003

[Stoakley 1995] : R. Stoakley, M. Conway, and R. Pausch. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings ofHuman Factors in Computing Systems (CHI '95)*, pages 265–272, May 7–11 1995.

[Glinz 2000]: M. Glinz. Problems and Deficiencies of UML as a Requirements Specification Language. In *Proceedings of IEEE Tenth International Workshop on Software Specification and Design*, pages 11–22, 2000.

[Purvis 2002]: M. K. Purvis, S. J. S. Cranefield, M. Nowostawski, and M. A. Purvis. Multi-Agent System Interaction Protocols in a Dynamically Changing Environment. Information Science Discussion Paper Series, Number 2002/04, ISSN 1172-6024, University of Otago, Dunedin, New Zealand (2002).