Model-Based Design and Implementation of Interactive Spaces for Information Interaction

Hans-Christian Jetter, Jens Gerken, Michael Zöllner, and Harald Reiterer

AG Mensch-Computer-Interaktion, Universität Konstanz,
Universitätsstraße 10, 78457 Konstanz, Germany
{hans-christian.jetter,michael.zoellner,
jens.gerken,harald.reiterer}@uni-konstanz.de
http://hci.uni-konstanz.de

Abstract. Interactive spaces with multiple networked devices and interactive surfaces are an effective means to support multi-user collocated collaboration. In these spaces, surfaces like tablet PCs, tabletops, or display walls can be combined to allow users to interact naturally with their personal or shared information, e.g. during presentation, discussion, or annotation. However, designing and implementing such interactive spaces is a challenging task due to the lack of appropriate interaction abstractions and the shortcomings of current user interface toolkits. We believe that these challenges can be addressed by revisiting model-based design techniques for object-oriented user interfaces (OOUI). We discuss the potential of OOUIs for the design of interactive spaces and introduce our own object-oriented design and implementation approach. Furthermore we introduce the ZOIL (Zoomable Object-Oriented Information Landscape) paradigm that we have used as an experimental testbed. While our approach does not provide automated model-driven procedures to create user interfaces without human intervention, we illustrate how it provides efficient support throughout design and implementation. We conclude with the results from a case study in which we collected empirical data on the utility and ease of use of our approach.

Keywords: Interactive Spaces, Information Interaction, Zoomable User Interfaces, Model-based Design.

1 Introduction

Recent work in Human-Computer Interaction (HCI) suggests the use of physical work environments with multiple interactive surfaces (e.g. multi-touch tabletops or walls) for the collocated collaboration of multiple users. These "interactive spaces" are often used to support groups during the collaborative management, presentation, and discussion of information items, e.g. in science, design, and engineering [23,7,17]. Following the Weiserian vision of ubiquitous computing, a fundamental requirement for such interactive spaces is a "natural" style of human-computer interaction where computing interfaces ideally become invisible and unobtrusive. They vanish into the background of our familiar non-digital

R. Bernhaupt et al. (Eds.): HCSE 2010, LNCS 6409, pp. 22–37, 2010.

reality. Therefore the essential operations of our information interaction such as viewing, editing, (re)locating, sharing, and annotating information items should be provided by natural or "reality-based" interfaces. Following Jacob et al.'s notion of reality-based interaction, such interfaces "draw strength by building on users pre-existing knowledge of the everyday, non-digital world to a much greater extent than before." They attempt to make computer interaction more like interacting with the real, non-digital world by employing themes of reality such as users understanding of physical objects or their body and social skills. Fig. 1 shows an example of an interactive space and different reality-based interaction techniques that can provide a more natural and fluid user experience that is ideally not impaired by obtrusive computer user interfaces and technology-induced barriers between them.

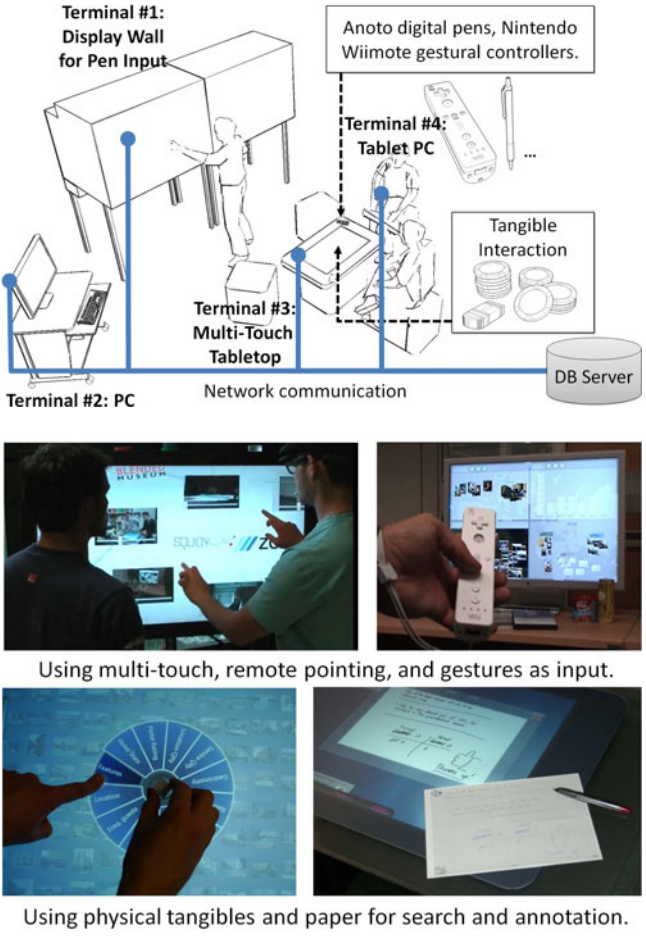


Fig. 1. A ZOIL-based interactive space as realized in our lab (top). Natural interaction styles used in our ZOIL case studies, e.g. tangibles and digital pens (bottom).

To this day, designing and implementing reality-based and tangible user interfaces (UI) for interactive spaces is a challenging task. As discussed by Shaer and Jacob, typical challenges are the lack of appropriate interaction abstractions, the shortcomings of current user interface software tools to address continuous and parallel interactions, as well as the excessive effort required to integrate novel input and output technologies [22]. We believe that these challenges can be addressed by viewing interaction through the lens of object-orientation. We suggest to revisit the user interface modeling and design techniques for object-oriented user interfaces (OOUI) from the 1990's that have widely fallen into oblivion and to apply them on today's novel post-WIMP (post-"*windows icons menus pointer*") technologies and user interface toolkits. In this paper, we make three contributions to this field of research: In chapter 2, we discuss why we believe that this step into the past era of OOUIs has great potential for the design of future computing environments and why this is especially true when considering collaborative interactive spaces for reality-based information interaction. In chapter 3, we introduce the ZOIL (Zoomable Object-Oriented Information Landscape) paradigm that we have used as an experimental testbed for our model-based design and implementation approach. In chapter 4, we illustrate and discuss our approach for modeling OOUIs in detail. While our approach does not provide automated model-driven procedures to create user interfaces without human intervention, we illustrate how it can provide efficient model-based support throughout the design and implementation and we present results from a case study in which we collected empirical data on the utility and ease of use of our OOUI approach from designers and developers.

2 Objects in Collaborative Information Interaction

There is a variety of high-level frameworks in HCI for modeling information interaction, e.g. Blandford and Attfield's "information journey" [4] or the GEMS model from Salminen et al. [15]. Typically these models consider information interaction as a task-oriented series of phases of higher level activities that are separated in time, e.g. *recognizing an information need, acquiring information, interpreting information*, and *using interpretation*. Such generic frameworks can be used as a starting point for interaction design: During a *top-down* design process, these generic high-level activities can be contextualized for the targeted application domain and can be hierarchically decomposed into domain-specific lower level task models (e.g. essential use cases or scenarios). These are used to define the abstract user interface architecture (e.g. the navigation map) and to later flesh out the details of the concrete visual design of individual pages or dialogs. Such a task-oriented top-down design process (e.g. usage-centered design [6]) creates interfaces that resemble virtual pathways to guide users through all the stages, information resources, and interaction contexts that are necessary for completing the tasks from the application domain. These page flows or series of dialogs define the virtual routes that users can take when working with the system. Under the influence of the page-oriented World Wide Web, interaction designers have become very experienced in designing interfaces as such

task-oriented stepwise conversations between a single user and a system that move along predefined paths. They achieve great usability for domains with a finite number of clearly defined tasks or business processes (e.g. in e-commerce). However, we believe that in the post-WIMP era such purely task-oriented thinking during design and implementation cannot leverage the true power of today's novel ways of natural and collaborative interaction.

2.1 Task-Orientation vs. Object-Orientation

In the case of collaborative information interaction in post-WIMP environments like in Fig. 1, designers have to consider interaction not only as a task-oriented sequential process supported by a single interface and its hard-coded functionality. In such settings, information interaction becomes a distributed, concurrent, and sometimes seemingly chaotic activity that does not follow simple task models. Instead, the users' actions are situated in a constantly changing social and technological setting, in which multiple users at multiple points of action simultaneously pick up, use, manipulate, recombine, create, and destroy virtual information objects without following clearly defined processes that terminate at clearly defined goals. Furthermore, such post-WIMP environments with multitouch or tangible user interfaces) also afford more natural interaction styles. Instead of clicking hyperlinks or widgets as an intermediary language to sequentially converse with a system about intended actions, users want to continuously touch, grab, and manipulate physical or virtual objects from the application domain. Ideally the application domain itself becomes directly user-accessible and user tasks are carried out by directly manipulating the objects representing it. Thus the user interface changes its nature from being a task-oriented intermediary language medium based on widgets into a computer-mediated world of cooperating visual and tangible objects that provide users with more means for flexibility, improvisation, and establishing individual working styles.

The challenge of designing and programming interfaces that are entirely based on the direct manipulation of cooperating objects instead of sequential conversations is not new. It is similar to the challenge that designers were facing during the advent of graphical user interfaces and direct manipulation in the 1980s [22]. At that time, Hutchins et al. referred to this new kind of direct manipulation interfaces as "model-world interfaces" as opposed to traditional interfaces which have been designed with a conversation metaphor of human-computer interaction in mind [10]. Model-world interfaces provide a coherent and consistent overall representation of the application domain in which the user can freely navigate and directly act on domain objects using a series of low-level direct manipulations that in sum constitute the intended high-level tasks and activities. Essentially, the design challenges we face now in the design of interactive spaces are the same: How can we break down an application domain and its higher level tasks into cooperating visual and tangible objects inside an interactive space, in which higher level tasks can be carried out in natural ways by lower level direct manipulations of objects?

2.2 Revisiting Object-Oriented User Interfaces (OOUI)

In the 1990s, IBM introduced the term *Object-Oriented User Interfaces* (OOUI) to describe a new kind of direct manipulation model-world interfaces: "An object-oriented user interface focuses the user on objects - the "things" people use to accomplish their work. Users see and manipulate object representations of their information. Each different kind of object supports actions appropriate for the information it represents" [21]. At that time, OOUIs were considered as more usable due to the closer match between the application domain and its virtual counterpart on the screen. Furthermore, unlike application-oriented user interfaces, OOUIs provided greater flexibility and consistency following a "flexible structure-by object" instead of a "rigid structure-by function" [16]. Today, this makes OOUIs particularly interesting for post-WIMP designs that are intended to better support the unpredictable and ill-defined needs and actions of situated users which cannot be anticipated by the task models of the design phase.

During OOUI design it is important to avoid unnecessary realism in interface metaphors or an unintelligible plethora of different object types and behaviors. To achieve this, OOUI designers employ rigid object-oriented mechanisms such as *inheritance*, *generalization*, and *polymorphism* to analyze and model the essential characteristics of the application domain. Thereby they view the domain through the lens of object-orientation from a user's perspective. Using these mechanisms, the user-perceived similarities and differences between domain object types are modeled in common base classes or subclasses. "Interactions should be consistent across objects of the same class; where possible, operations should be polymorphic - applicable to different object types. This reduces the number of interaction behaviors and simplifies the interface" [5]. This way the modeled class hierarchy can integrate very different types of domain objects into a single model while preserving a maximum degree of consistency in interaction. This model is then used to design and implement an interface with consistent behavior, functionality, and appearance. If properly applied users experience a "logical" behavior throughout the entire OOUI. Thus they can more easily apply their previous experiences to infer their strategies for handling novel tasks.

Although OOUIs strongly influenced the design of the "desktop metaphor" in today's operating systems, OOUI design approaches have not been subject of intense scientific research. Most efforts only lasted until the late 1990s (e.g. [1,2,16,5,21]) and after that there has only been some OOUI-related work in the context of Pawson's radical *Naked Objects Pattern* which tries to eliminate the need for specific user interface design by making all code objects and data models directly user accessible [18]. In conclusion, we are aware of only two publications that have proposed entire OOUI design methodologies: IBM's comprehensive description of the OVID methodology in [21] and the brief description of Beck et al.'s TASK methodology in [2].

The OVID methodology *(Object, View, and Interaction Design)* for OOUI design was intended to bridge user interface and software engineering by using the UML notation and modeling techniques of successful code design and combine these with user interface design and usability engineering. At the heart of OVID

is the *designer's model*, a conceptual model that includes "descriptions of the objects users will employ to perform their tasks, the properties of those objects, and the interrelationships between them" [21]. To identify the objects that users have to act on and that should be provided to them on the user interface, textual and formal notations of tasks (e.g. use case diagrams) can be used, so that "task analysis will reveal information about what the users do and which objects they work with". Despite OVID's comprehensive treatment in [21], only high level descriptions of iterative design and prototyping are provided and many of the necessary steps, rules, or tools remain unclear.

Before OVID, Beck et al. introduced the TASK methodology for integrating OO analysis into graphical user interface design for desktop systems [2]. During TASK's analysis activity, a task model and an initial object-oriented *object model* is built, which is then refined to an object-oriented *application specification*. This specification is used as a *conceptual user interface model* during user interface design and the views, dialogs, and the actual screen representations of conceptual objects are derived from it. The successful application of TASK and its supporting tools is mentioned for the design of insurance and production planning systems. However, the detailed tools, rules, and the amount of human intervention for translating the *conceptual user interface model* into concrete user interface design and its implementation are not revealed in detail.

3 Exploring OOUI Approaches Using the ZOIL Paradigm

To explore OOUI methodologies for the design and implementation of post-WIMP collaborative information interaction, we have developed our own model-based approach. Thereby, we have taken the promising parts from the TASK and OVID methodologies and adapted them to the design of present-day multi-user and multi-surface environments (see chapter 4). Three questions have been guiding our work: Can we adapt OOUI analysis and design techniques and notations to efficiently inform the domain-specific design of present-day interactive spaces? Can we define concise translation rules for creating the initial visual and interaction design for the user interface directly from our model in a simple step-by-step process? How well can designers and programmers apply our OOUI approaches and how do they assess their practical value?

As a testbed for our experimental approach, we have chosen our Zoomable Object-Oriented Information Landscape (ZOIL) paradigm. ZOIL provides a reference interface design for interactive spaces, a reference client-server architecture for distributed information interaction, and a software framework facilitating their implementation. Thus ZOIL provided us with the necessary infrastructure to efficiently explore our model-based approach. The ZOIL reference design, architecture, and framework have been used before in different projects to realize domain-specific prototypes for information interaction. For example Jetter et al. have designed a ZOIL-based user interface for basic personal information management for interactive television devices [12] and two interactive spaces for discussion and presentation, e.g. for students of media science or for scientists in the field of nano photonics. Heilig et al. have designed

an interactive wall for a public library [9]. In future, Geyer at al. will be using ZOIL to create collaborative design rooms for interaction design [8].[1]

A ZOIL-based interactive space consists of several interactive surfaces (e.g. tabletop, tablet PC, wall-sized display) that serve as user terminals to access the shared information space (Fig. 1 top). Each of the terminals thereby provides a window into a much larger planar visual workspace that contains all the shared information and functionality of the application domain. This visual workspace resembles a zoomable whiteboard of infinite size and resolution and is called the "information landscape". ZOIL's zoomable information landscape facilitates the navigation in the application domain and its information spaces by "tapping into our natural spatial and geographic ways of thinking" [19]. All domain objects and their relations are organized and visualized in space and scale to foster natural visual-spatial approaches to accessing, sharing, and manipulating information. Regions of the landscape with items, piles, or clusters can represent certain user activities, domain processes, or personal vs. shared information repositories. The landscape is used as a flexible multi-scale medium for visually accessing the application domain and its information spaces and objects. Content and functionality of an individual object can be accessed spatially using panning and "semantic zooming" [19] without the need for opening folders or dedicated applications and the then-necessary management of overlaying or occluding windows (Fig. 2). This zoom navigation is also in line with reality-based interaction: It draws strength from the users' environment awareness and skills, e.g. their familiarity with approaching, touching, moving, and organizing objects in physical space and the simple fact that "all objects in the real world have spatial relationships between them" [11]. Therefore visual objects at different locations and scales (e.g. virtual Post-It notes, project logos) can further augment the landscape with global or relative landmarks that support orientation. Furthermore, all regions of ZOIL's landscape can be visually annotated with ink strokes using stylus, touch, or digital Anoto pens on physical paper. Annotations can also be made directly on objects, e.g. slides (Fig. 2).

Multi-user collaboration becomes possible by using ensembles of personal and shared user terminals. All terminals inside the interactive space share the same information landscape. All user-initiated changes to the content of the landscape such as moving, resizing, rotating, or annotating information items are immediately sent to a central server and synchronized with the other terminals in real time (typically within 50-250 ms). However, what region of the landscape is currently visible on each terminal can be individually controlled by the users. For example, users can use a tabletop to interactively zoom into the tiniest details of the landscape at many orders of magnification. At the same time they can display the entire landscape on a peripheral wall-sized screen to provide them with an overview for orientation when needed. The boundaries of the currently visible regions can also be transmitted between terminals. For example, users can instruct the remote wall-sized display to zoom and pan to the region of the

[1] Videos of these prototypes are available at `http://www.vimeo.com/12737554` and `http://hci.uni-konstanz.de/jetter/hcse2010.mp4`

landscape that is currently visible on the tabletop or vice versa. Thus, by using terminals as "cameras", the roles of stationary or mobile terminals can be flexibly adjusted by the users depending on the group's task and preference.

In large information landscapes, users also need efficient ways to find, filter, and analyze single objects or specific clusters. For this reason, ZOIL also integrates physical and virtual "magic lenses" [3] that float above the landscape and through which the underlying content of the landscape can be viewed (Fig. 2). These lenses provide movable filters and visualization tools such as lists, bar charts, scatter plots, or tables to provide an analytical view on the landscape and to facilitate the search and filtering of items using spatial metaphors.

Fig. 2. *Left*: Semantic zooming into objects in ZOIL uses the available screen estate for smooth changes between iconic representations, metadata display, and full content and functionality, e.g. for viewing, editing, or annotating the content. The example shows a slide object (top) and a movie object (bottom) at different zoom levels. *Right*: Physical or virtual magic lenses allow users to view the underlying landscape using different information visualization tools.

To realize ZOIL's distributed multi-user and multi-device ZUI, the reference architecture is based on a client-server architecture that provides and synchronizes the data model of the information landscape for all user terminals or clients within an interactive space (Fig. 1). Inspired by Prante et al.'s i-Land with its COAST framework for object distribution [20], we have implemented a dedicated ZOIL server and a client-side data backend as part of our ZOIL software framework for C#/.NET that is based on the db4o object database and its mechanism of *transparent persistence*[2]. For peer-to-peer communication between clients and for input device connectivity, we have chosen the simple but robust stateless *Open Sound Control (OSC)* protocol that can be used for UDP broadcasting within the subnet of an interactive space and enables developers to easily integrate novel input devices (e.g. Nintendo Wiimote Controllers or Anoto

[2] `http://www.db4o.com/`

digital pens) by connecting to input device middleware such as OpenInterface [14] or Squidy [13]. Equally important for ZOIL's realization is the framework's support for fast client-side rendering of complex rich-media zoomable user interfaces. For ZOIL, we have chosen Microsoft's Windows Presentation Foundation (WPF) technology because of following reasons: First, the technology must support high-performance hardware-accelerated renderings of vector-based user interface components, so that smooth zooming animations over many orders of magnification become possible without pixelation. Second, an initial set of fundamental user interface widgets such as buttons or sliders, but also more complex widgets such as video players, document viewers of web browsers should be available from the start to accelerate implementation. Third, a declarative language for user interface definition should be available that supports a clear separation between business logic and visual presentation. In the following we discuss the central role of WPF's declarative XAML language in our model-based design and implementation approach.

4 Model-Based Design and Implementation with ZOIL

For our model-based design and implementation approach, we have employed an object model similar to the *designer's model* in OVID or the *conceptual user interface model* in TASK as a core artifact. The model uses a UML-like notation to define what kind of information objects are visually exposed to the user and become user manipulatable on the different terminals inside the interactive space. Furthermore it reveals what attributes or metadata these objects carry for the user, and what operations or behaviors these objects share and provide. Fig. 3 shows an example conceptual model for an interactive space in which users can collaboratively explore hotel objects that are contained in ZOIL's zoomable information landscape using semantic zooming. Hotels carry (meta)data such as

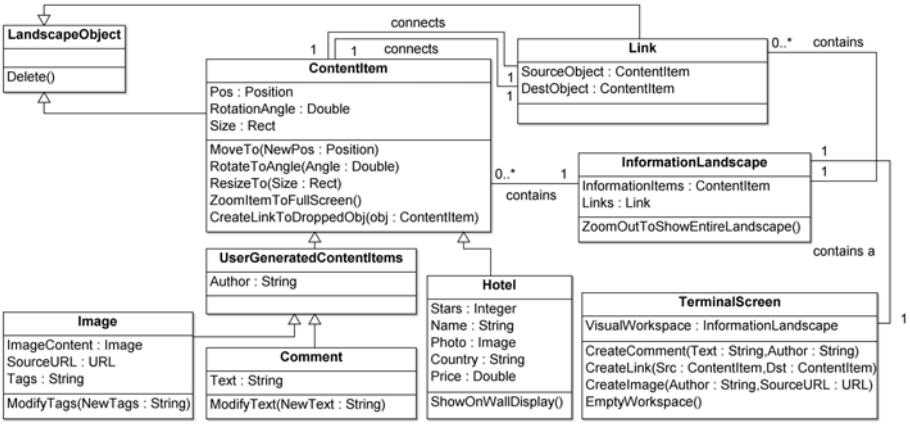


Fig. 3. An object-oriented conceptual model of a ZOIL user interface

the name of the hotel, a photo of the hotel, the country of the hotel, etc. Users can add images from the Web or textual comments as user generated content. Users can explicitly connect all hotels, images, and comments with visual links to structure, annotate, or discuss. It is important to notice that Fig. 3 is not representing the programmer's model of the user interface or its code objects and methods, but that it describes the classes, attributes, and operations of the domain objects and conceptual objects that the user will perceive and act on when interacting with the system's OOUI. "The primary distinction that designers and programmers must keep in mind is that OOUI design concentrates on objects *that are perceived by users*. OO programming focuses on implementation details that often need to be hidden from the user" [21]. Since the conceptual model is used to inform design and implementation based on human intervention, it is not necessary that the notation completely complies with the UML standard and covers all details. It only has to cover the essentials of the UI from a user's perspective using a notation that has been agreed on and is intelligible for all designers and programmers. In our exploration, we have experienced that our UML-like notation used in Fig. 3 has met these requirements.

We have based our example of a conceptual model on typical user tasks during collaborative holiday planning and an OO analysis of the surrounding information space. Task analysis and OO analysis of the information space revealed the objects and their relations, e.g. whether an object of a certain class should contain or refer to one or many objects of a different class. These relations are specific to the application domain and information space, e.g. the landscape in our example contains 0-n objects of the class *ContentItem*, i.e. hotels, comments, or images. Furthermore, all *ContentItem* objects can be linked to other *ContentItem* object via a *Link* object. The OO analysis of the information space also helps to identify the task-relevant metadata or attributes of an class that should be provided to the users, e.g. alphanumeric fields such as *Name* and *Country* of a hotel, or visual images such as the *Photo* of an hotel. In a ZOIL user interface, objects also carry implicit visual properties such as position, size, and rotation angle that are not provided to the user as numeric values but are used to place and render objects. In Fig. 3 all these different attributes are listed in the middle section of each UML class definition.

After having identified the task-relevant classes, relations and attributes of objects, the bottom section of the UML class definition is used to specify the functions or *operations* that objects of this class should expose to the user. Based on the task analysis, basic operations such as creating, editing or deleting an object have to be identified and have to be attached to the object itself or to other user interface objects. For example a virtual Post-It note as *Comment* object should become editable after zooming in to modify its content. Furthermore users should be able to delete outdated comments. Therefore a *Delete()* function should be provided to the user that is attached to the object, e.g. a delete button similar to the close button of a GUI window. However, the functionality to create a new comment *CreateComment(...)* must be attached to the enclosing visual workspace or screen as the create-functionality must be

accessible before the *Comment* object itself exists. Other operations can be modeled for facilitating the zooming navigation, e.g. an object can be assigned a *ZoomItemToFullScreen()* functionality to offer an automated zooming that reveals all attributes, metadata, and operations by a simple tap or click on the object. While modeling the operations of objects, further design decisions have to be made, e.g. whether an object is movable, resizable, or rotatable. Also the functionality that should be executed when using drag-and-drop manipulations can be modeled. For example, the behaviors section of a class can define what should happen as soon as an object of a certain class has been dropped onto it, e.g. creating a link to the dropped object in *CreateLinkToDroppedObj(...)*.

The design of the conceptual model should be accompanied by two continuous activities to ensure its quality: First, choosing appropriate class hierarchies is essential for the OOUI's coherence and consistency. Therefore the model should be continuously checked if all new commonalities in attributes and operations have been modeled in common base classes. Second, during OOUI design the higher level task models have to be decomposed into sequences of lower level direct manipulations of objects and other invocations of their operations. In many cases it is not immediately visible if a model covers all required tasks and therefore this should be frequently verified. This can be achieved by manually simulating a user task and using the conceptual model for a sequential walkthrough that checks if all necessary objects, attributes, and operations for all tasks are available.

4.1 Model-Based Design and Implementation of UI Objects

ZOIL's reference design and architecture provide a generic design and implementation framework in which only the application-specific details of the user interface and interaction design have to be fleshed out. Our model-based approach provides the necessary translation rules in a simple step-by-step process, thereby allowing designers to create initial sketches of visual and interaction design from the conceptual model of the user interface. It furthermore enables designers and programmers to easily turn the resulting sketches into an implementation model for the user interface object based on XAML. This XAML code can then be used to test the design as an interactive prototype. Our model-based translation process can be described as a four phase process and is visually illustrated in Fig. 4 for the example of a *Hotel* object.

The first step of the translation process is to decide which attributes and operations of an object should appear on which level of semantic zooming. Attributes or operation can either appear globally at all zoom levels or they can be assigned to different zoom levels, so that they only appear or become active after the user has zoomed in. In Fig. 4, the *Delete()* function is global and appears at all levels of detail. This is also true for the manipulation of the object's position, rotation angle, or size (*Move()*, *Rotate()*, *Resize()*) and its functionality to react to objects that have been dropped onto it such as *CreateLinkToDroppedObj()*. The most important attributes that a user frequently needs to recognize or recall an object (e.g. *Photo* or *Name* of an hotel) already appear at small zoom levels in the early stages of zooming. The attributes only necessary for more

in-depth exploration (e.g. *Stars*, *Country*, *Price*) appear after enough screen estate is available, e.g. on zoom level 3. This is also true for advanced functions like *ShowOnWallScreen()* that shows a hotel on a shared wall-sized display.

In the second step, this assignment is used to sketch the global appearance and behavior of the object (Fig. 4 top right). The different operations and their triggering manipulations or widgets are modeled using simple sketches: In our example, the typical multi-touch gestures known from tabletops or smart phones are used for *Move()*, *Rotate()*, and *Resize()*. A zoom-to-full-screen animation is issued by a single tap with the finger on an object (*ZoomItemToFullScreen()*). Another item can be dragged on the object with the finger, activating the *CreateLinkToDroppedObj()* functionality if the item is of the type *ContentItem*.

In the third step, the individual zoom levels are sketched based on the assignments of attributes and operations from step one (Fig. 4 right). These sketch models are created for each zoom level to move from conceptual design to the concrete design of the visual appearance of objects. Since the necessary attributes and operations for each zoom level are known, the complexity of the design task is minimized and can be carried out with standard techniques.

In the final step, the sketch models of the different zoom levels are translated into the implementation model of the user interface object (Fig. 4 bottom). This translation is supported by ZOIL's software framework that extends the declarative XAML user interface description language of WPF with ZOIL-specific elements. By introducing ZOIL's *ZComponent* user interface control, an object's appearance at different semantic zoom levels can be defined entirely using declarative approaches (similar to HTML) without the need for procedural programming. The different zoom levels are managed by ZOIL's *ZComponentFrames* container that selects the appearance of an object depending on the available render size. To avoid harsh visual changes, zoom levels smoothly blend between two appearances using an opacity animation. Furthermore designers and programmers can easily assign predefined ZOIL behaviors to an object using the *attached behavior* software pattern[3]. This pattern helps to encapsulate frequently used ZOIL-specific behaviors (e.g. "object can be manipulated with multi-touch", "object zooms to full-screen after tap", "object is a target for dropping another object") in a central behavior library. Behaviors from the library can be easily attached to classes or individual instances of objects using declarative XAML code without the need to know procedural programming or to fully understand the underlying class hierarchies. We believe that this combination of the *ZComponent* object and the attached behavior pattern introduces a great expressive power to the declarative XAML language and a very natural view of interactive behavior into user interface programming. It greatly facilitates the translation of sketch models with their visual appearance and behavioral properties into implementation models. As illustrated in the implementation model in Fig. 4, the process of translating a sketch model in XAML is thereby a straightforward task that does not rely on advanced programming skills.

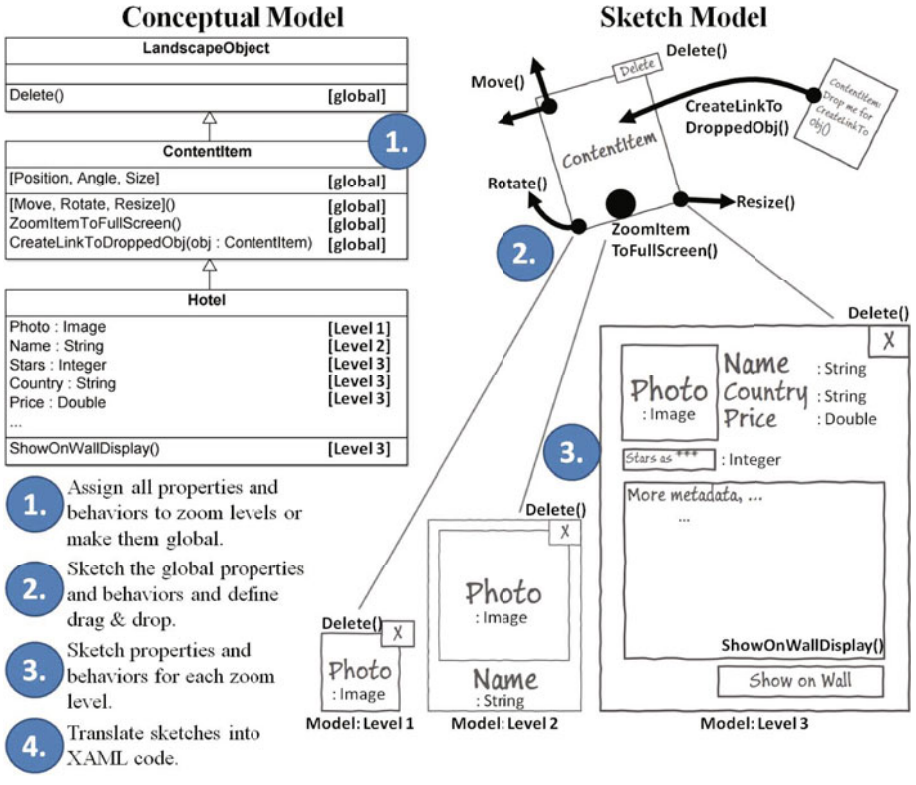
[3] `http://blogs.msdn.com/b/johngossman/archive/2008/05/07/the-attached-behavior-pattern.aspx`

4.2 Case Study

In order to investigate the utility and applicability of our OOUI approach in practice, we conducted a case study with 11 participants (9 graduate-level and 2 undergraduate students of computer science). The question guiding our study was how well participants can apply our approach and how they assess its practical value during a small-scale project. We divided the participants into five teams (4 teams with 2 members, 1 team with 3 members). In a first one-hour session we presented our modeling approach to all teams: We created and explained a conceptual model of a ZOIL user interface for accessing a fictitious image database. The teams were then given the assignment to create an own conceptual model for a different ZOIL user interface until the next session in two weeks. The user interface to model should allow users to explore and discuss hotels as described in the example in the previous sections. We provided the teams with the same input for their modeling and design activity that we used ourselves to create the example model in Fig. 3, i.e. all teams were handed 8 informal functional requirements (e.g. "user must be able to add a textual comment to the workspace") and a list of 22 required object properties (e.g. "each *Comment* has an *Author*", "each *Image* carries *Tags*").

Two weeks later, we carried out individual one-hour team sessions during which each team completed three tasks. First, each team presented and explained their prepared conceptual model. Then we asked the team to check if their model really supports the 8 functional requirements by carrying out a walkthrough. We then presented the team our alternative model (Fig. 3) and asked them to validate this unknown model by another walkthrough. After this, each team member filled out a questionnaire to rate the difficulty of the three tasks. At the end of the sessions, the teams were instructed to design and implement a user interface with the ZOIL framework based on Fig. 3 until the next sessions in the following week. In these last sessions, each team individually presented the resulting interactive prototype and each team member filled out a further questionnaire to rate the overall usefulness of the modeling approach and the difficulty to apply it on user interface design and implementation.

During the case study, all teams presented conceptual models that were formally correct and supported the 8 functional requirements. All teams were able to carry out a walkthrough to validate their own and unknown models. Furthermore, the presented interactive prototypes covered the requested functionality. However, during the first and second session participants reported initial problems regarding the unfamiliar use of UML class diagrams to model user interfaces. Repeatedly participants mentioned that they sometimes had fallen back into the familiar modeling of code objects and lost track of their original intention to model the user interface from a user's perspective. However, the participants reported that they got increasingly used to the approach and found it useful to support the design and implementation. Fig. 5 shows the results from the questionnaires: the creation of a model (mean=3.45, sd=0.93) and checking the own or someone else's model with a walkthrough (mean=3.1, sd=2.9 and mean=2.9, sd=1.14) was not considered as particularly difficult nor very easy.



Implementation Model

```
<ZComponent x:Class="Hotel"
   ZObjectDragDropBehavior.IsDraggable="True"                    Move()
   ZObjectResizeBehavior.IsResizable="True"                      Resize()
   ZObjectRotateBehavior.IsRotatable="True"                      Rotate()
   ZInformationLandscape.ZoomTarget="True"> …                    ZoomItemToFullScreen()
   <ZObjectDropBehavior Type="{x:Type ContentItem}"
      Command="{Binding CreateLinkToDroppedObj}"/> …             CreateLinkToDroppedObj()
   <ZComponentFrames>
      <ZComponentFrame WidthNeeded="0"> … Level 1 … </ZComponentFrame>     Model: Level 1
      <ZComponentFrame WidthNeeded="100"> … Level 2 … </ZComponentFrame>   Model: Level 2
      <ZComponentFrame WidthNeeded="500"> …                                Model: Level 3
         <Image Source="{Binding Photo}" Grid.Row="0" Grid.Column="0"/> …   Photo : Image
         <TextBlock Text="{Binding Name}"/> …                               Name : String
         <TextBlock Text="{Binding Country}"/> …                            Country : String
         <TextBlock Text="{Binding Price}"/> …                              Price : Double
         <StarRating CurrentValue="{Binding Stars}"/> …                     Stars : Integer
         <Button Content="Show on Wall" Command="{Binding ShowOnWallDisplay}" …
      </ZComponentFrame>                                                    ShowOnWallDisplay()
   </ZComponentFrames> …
   <Button Command="{Binding Delete}" HorizontalAlignment="Right"           Delete()
    VerticalAlignment="Top"> …
</ZComponent>
```

Fig. 4. ZOIL's translation process and rules to translate the object model to user interface design and implementation

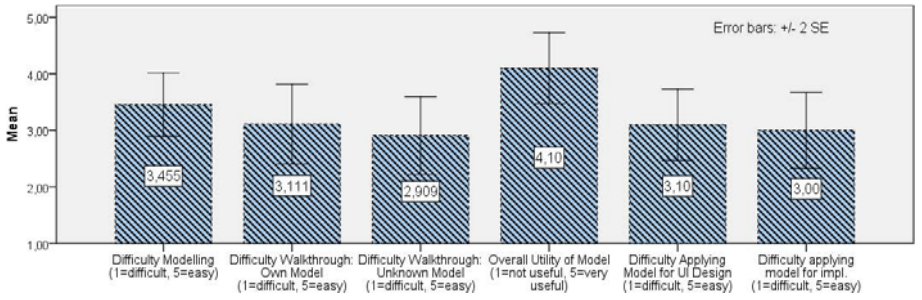


Fig. 5. Collected feedback from the questionnaires of the case study

This is rather encouraging, as the students were given only a very brief introduction to the approach without any proper training phase. Furthermore, the overall utility of the modeling technique was considered as useful (mean=4.1, sd=0.99) by the participants. Regarding the early stage of our approach and the unfamiliar use of object-oriented modeling and design for user interfaces, we consider these results as a promising first evidence that OOUI approaches can be indeed useful for designing interactive spaces.

5 Conclusion and Future Work

We have discussed why we believe that revisiting OOUIs has a great potential for the design of future post-WIMP environments, particularly for collaborative information interaction. We have introduced our ZOIL paradigm that we have used as an experimental testbed for creating and evaluating our OOUI approach. We have illustrated and discussed our approach in detail and have shown how it can efficiently inform the design and implementation of user interface objects following simple translation rules. Furthermore, we have presented promising results from a first case study on the practical utility of our approach. At the current stage, we consider our approach as a successful first step. However, ZOIL-based interactive spaces offer a great design space and currently only small parts of it have been covered by our approach. For example, the design of ZOIL's magic lenses, visualization tools, or the integration of physical objects or paper is not covered yet. Therefore we will investigate how new and extended modeling notations and translation rules can be used to cover these aspects in future.

References

1. Common User Access Guide to User Interface Design. IBM Corporation (1991)
2. Beck, A., Janssen, C., Weisbecker, A., Ziegler, J.: Integrating object-oriented analysis and graphical user interface design. In: Taylor, R.N., Coutaz, J. (eds.) ICSE-WS 1994 and SE-HCI 1994. LNCS, vol. 896, pp. 127–140. Springer, Heidelberg (1995)

3. Bier, E.A., Stone, M.C., Pier, K., et al.: Toolglass and magic lenses: the see-through interface. In: Proc. SIGGRAPH 1993, pp. 73–80. ACM, New York (1993)
4. Blandford, A., Attfield, S.: Interacting with information. In: Carroll, J.M. (ed.) Synthesis Lectures on Human-Centered Informatics. Morgan & Claypool (2010)
5. Collins, D.: Designing object-oriented user interfaces. Benjamin Cummings, Redwood City (1995)
6. Constantine, L.L., Lockwood, L.A.D.: Software for use. ACM Press/Addison-Wesley Publishing Co., New York (1999)
7. Fitzmaurice, G.W., Khan, A., Buxton, W., Kurtenbach, G., Balakrishnan, R.: Sentient data access via a diverse society of devices. Queue 1(8), 52–62 (2003)
8. Geyer, F., Reiterer, H.: A cross-device spatial workspace supporting artifact-mediated collaboration in interaction design. In: Proc. CHI EA 2010, pp. 3787–3792. ACM, New York (2010)
9. Heilig, M., Demarmels, M., Rexhausen, S., Huber, S., Runge, O.: Search, explore and navigate - designing a next generation knowledge media workbench. In: Proc. SIDeR 2009, pp. 40–43. Eindhoven University of Technology, Eindhoven (2009)
10. Hutchins, E.L., Hollan, J.D., Norman, D.A.: Direct manipulation interfaces. Hum. Comput. Interact. 1(4), 311–338 (1985)
11. Jacob, R.J., Girouard, A., Hirshfield, L.M., Horn, M.S., Shaer, O., Solovey, E.T., Zigelbaum, J.: Reality-based interaction: a framework for post-wimp interfaces. In: Proc. CHI 2008, pp. 201–210. ACM, New York (2008)
12. Jetter, H.C., Engl, A., Schubert, S., Reiterer, H.: Zooming not zapping: Demonstrating the zoil user interface paradigm for itv applications. In: Adjunct Proceedings of EuroITV 2008. Springer, Heidelberg (2008)
13. König, W.A., Rädle, R., Reiterer, H.: Interactive design of multimodal user interfaces. Journal on Multimodal User Interfaces 3(3), 197–213 (2010)
14. Lawson, J.Y.L., Al-Akkad, A.A., Vanderdonckt, J., Macq, B.: An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In: Proc. EICS 2009. ACM, New York (2009)
15. Lehikoinen, J., Aaltonen, A., Huuskonen, P., Salminen, I.: Personal Content Experience: Managing Digital Life in the Mobile Age. Wiley, Chichester (2007)
16. Mandel, T.: The GUI-OOUI War, Windows vs. OS/2: the designer's guide to human-computer interfaces. Van Nostrand Reinhold, New York (1994)
17. Memmel, T., Reiterer, H.: Model-based and prototyping-driven user interface specification to support collaboration and creativity. J.UCS 14(19), 3217–3235 (2009)
18. Pawson, R., Matthews, R.: Naked objects: a technique for designing more expressive systems. SIGPLAN Not. 36(12), 61–67 (2001)
19. Perlin, K., Fox, D.: Pad: an alternative approach to the computer interface. In: Proc. SIGGRAPH 1993, pp. 57–64. ACM, New York (1993)
20. Prante, T., Streitz, N., Tandler, P.: Roomware: Computers disappear and interaction evolves. Computer 37(12), 47–54 (2004)
21. Roberts, D., Berry, D., Isensee, S., Mullaly, J.: Designing for the User with OVID. Macmillan Technical Publishing, Basingstoke (1998)
22. Shaer, O., Jacob, R.J.: A specification paradigm for the design and implementation of tangible user interfaces. ACM Trans. Comput. Hum. Interact. 16(4), 1–39 (2009)
23. Wigdor, D., Jiang, H., Forlines, C., et al.: Wespace: the design development and deployment of a walk-up and share multi-surface visual collaboration system. In: Proc. CHI 2009, pp. 1237–1246. ACM, New York (2009)