Using modeling to put HCI design patterns to work

Russell Beale *Behzad Bordbar*

School of Computer Science
University of Birmingham
Edgbaston
Birmingham
B15 2TT UK

Abstract

It is recognised that creating effective, usable interactive systems is a highly non-trivial task. One approach to supporting developers and designers is through the use of HCI design patterns - this is now recognised as an effective way to produce usable systems. Design patterns capture the key elements of a design, providing a library of approaches that are known to work, though most design patterns are at best only semi-formal, providing outline structures that are filled in with discursive text and/or images.

In this paper we present a UML/OCL model of design patterns that captures not only the characteristics of the system but also its interface representation, and provide examples of how it can be used. This approach is shown to be flexible and very powerful.

We focus on modelling a design pattern at a high level of abstraction, producing a template, or metamodel representation, from which specific UML instantiations can be refined. This approach immediately captures the relationships between similar designs, showing the connections between related conceptual elements. The approach goes further, however; from a specific UML model, we can derive code that implements the relevant design pattern. The role of OCL is to represent constraints on the model, allowing us to define more tightly the behaviour and representational aspects of the design pattern.

This paper discusses the reasoning behind the approach and our initial results in modeling design patterns using UML.

1 Introduction

Designing effective interactive systems is recognised as a difficult task. Not only is the initial design itself a non-trivial problem, but it also has to be modified and reworked in the light of changing technological or commercial requirements. This means that such system are often component-based, i.e. they are composed of a set of smaller, simpler mechanisms that solve certain issues reliably and effectively. During the process of building a large system, components are assembled together to create a more complex system. However, as technologies change, the pressure to provide revised solutions means we end up hacking the once-clean design. From an HCI perspective, getting the initial design and interaction aspects correct is tricky, sometimes more a matter of craft and iteration than definitive methodology, and these difficulties are worsened over the course of rapid cycles of software production - the user can be forgotten as the technology advances and all too often new features appear in originally well-designed systems that are unnecessary, unwanted, or simply inaccessible. Even when well designed initially, systems can evolve away from users needs.

As an example, consider mobile systems. During the last few years, the massive success of cheaper, smaller and more powerful devices has caused an extensive growth in the mobile industry. Software components of such systems have grown both in size and complexity. In particular, there is a bewildering choice of implementation technologies. Moreover, a single software application typically uses several of these technologies on different platforms. A typical scenario triggered by the advent of a new technology is to create a slight modification of the application to suit the new opportunities. Considering the rapid evolution of new technologies and platforms, a software engineer who produces today's mobile system is a modern day Sisyphus, as his/her product is rendered futile by the birth of new hardware and software technologies.

From an HCI perspective, the constraints on interaction design imposed by the technologies are rapidly changing (different screen sizes and resolutions, ever-changing input devices, new functionalities such as digital cameras, and so on) and providing a consistent, coherent design solution in such a rapidly moving environment is a major challenge.

2 Addressing fundamental issues

There are two issues here

- we need to find ways of reusing the high-level designs of systems in the face of strong demands for rapid development
- we need to ensure that these designs are effective from an interaction perspective, especially in the face of rapid cycles of development, the continuous arrival of new technologies, and the inherent difficulties of doing effective HCI to create usable systems.

These issues have previously been addressed independently, as outlined below.

2.1 Reusing high-level designs: UML and MDA

The issue of rapid cycles of production and new technologies has been addressed in the context of e-business systems design, using UML-related technologies(OMG, 2005c), in particular, Model Driven Architecture (MDA)(Kleppe, Warmer, & Bast, 2003; OMG, 2005a), the new initiative of OMG. The MDA aims to separate the design from its implementation platform. MDA is based on two main features. The first is to promote the role of models, which are captured via widely used industry standards such as Unified Modelling Language (UML) for visualising, storing and exchanging software design artefacts. Using MDA, a Platform Independent Model (PIM) is created that is mapped to various Platform Specific Model (PSM) models, each targeting a specific implementation platform or technology. MDA models are not mere design-on-paper; they are machine-readable models, which are understandable by CASE tools that automatically generate laborious part of implementation in various implementation platforms. As a result, the second key feature of the MDA is its reliance on automation by transformation tools and the benefit that it brings. Promoting the role of the models, which is the underlying idea of the MDA, has emerged as *the* solution for the extending software lifetime.

2.2 Ensuring interactive system designs are good: HCI patterns

Moves towards user-centered design have addressed some of the HCI issues(Abras, Maloney-Krichmar, & Preece, 2004 (in press); IBM, 2004; Vredenburg, Isensee, & Righi, 2001). Putting key stakeholders at the centre of an iterative process and enabling him/her to influence the development of the system along the way results in the correcting of misconceptions and evaluating intermediate solutions. However, user-centered design is time-consuming and costly, especially in agile development phases with attendant commercial pressures, and is reliant on both specialist expertise and the availability of users and other stakeholders(Abras et al., 2004 (in press)). This limits its usefulness.

A complementary approach has come to the fore, that of the use of design patterns. Derived from Alexander's work (Alexander, 1964, 1965; Alexander et al., 1977; Alexander, Neis, Anninou, & King, 1987)on architectural patterns, and now commonplace in software engineering, they have been embraced by parts of the HCI community as an approach to design(Duyne, Landay, & Hong, 2002; Thomas Erickson; Tidwell) A *pattern* describes a recurring problem that occurs in a given context, and based on a set of guiding principles, suggests a solution. The solution is usually a simple mechanism: a certain style of layout, a particular presentation of information; techniques that work together to resolve the problem identified in the pattern. Patterns are useful because they document simple mechanisms that work; provide a common vocabulary and taxonomy for designers, developers and architects; enable solutions to be described concisely as collections of patterns; enable reuse of architecture, design and implementation decisions. Patterns are useful as they allow us to capture the salient features of a design, and the accompanying issues associated with that choice. They give us a way of sharing concepts, an approach to discussing different options, and a repository of design practices.

At a high level, patterns are therefore highly useful constructs. Design patterns are not perfect, however. There is no commonly accepted pattern language, and those that exist provide a framework for textual descriptions. Design patterns are usually expressed as semi-structured free-form text: they have a regularised layout of name, uses, problems and so on, with the details of the patterns described in natural language(Mahemoff & Johnston). Efforts are ongoing to devise a standard XML expression (e.g. CHI 2003 workshop, 2004 workshop)(T. Erickson), which will provide a framework for effective sharing and exchange of HCI patterns.

In addition, HCI as a design discipline is evolving rapidly, partly as technologies change and partly as we discover more about human reactions to interactive designs and systems, and so the patterns reflect a relatively immature design discipline. There are no completely accepted design patterns for specific situations: indeed, one of the characteristics of breakthrough devices/systems/interactions is that they often flout the accepted practice of the time anyway - it may never be possible to create ideal design patterns. The other major problem is that, being textual, they rely on large quantities of real-world knowledge, are not machine-understandable, and so are hard to apply without a great deal of craft knowledge. As a tool for less experienced designers, if the right pattern can be found then it offers a relatively familiar structure and so can be understood more quickly. But being able to identify the set of candidate patterns, to find related ones, to understand the constraints imposed by one choice over another, is an unsupported, difficult task. Moreover, as patterns are often created by different authors, it becomes hard for any substantial pattern library to identify patterns that are in fact identical, or at least very similar.

The limitations of HCI design patterns have to be combined with the wider impact of new technologies on HCI issues: new technologies offer new functionalities which impact the interactive nature of systems; the emergence of new styles of interaction between users drives new technologies to support these. In addition, acceptance of new metaphors or interaction styles drives changes in the platforms, and the rapid development process causes divergence from original design principles.

3 Merging the approaches

We are working on combining the approaches: we use MDA to integrate both the software architecture and the interactive design aspects. This will be achieved through using UML as a language to capture the core elements as a platform independent model (PIM) and producing transformations to produce implementable systems.

We can model both the systems and the HCI design patterns using UML at an abstract, technology-independent level. The UML has a built-in mechanism that allows the creation of specialised, UML-based languages, called *UML Profiles.* A list of UML profiles adopted by the OMG are available on their web site (OMG). Creating a UML profile for HCI provides support for the specification of the HCI aspects of the system. Moreover, UML profiles are can be used to define transformations that map the platform independent models that are compliant to the profile into implementations on a variety of platforms. Using this approach, we therefore model HCI design patterns at an appropriate level of abstraction and are able to capture their salient aspects in a way that then lends itself to mapping down to an implementation that reflects HCI concerns as much as technical ones. This approach only works as it retains a level of abstraction, separating out the platform independent model from the platform specific one (PSM).

MDA also provides us with a route to providing effective CASE tool support(Frankel, 2003; Kleppe et al., 2003), which allows us to automate the process of producing specific implementations on specific platforms. This also means we can rapidly prototype, produce and compare a variety of systems and design choices simply and cost-effectively. Other main advantage to this approach is that it allows us to capture good design practice in the form of semi-formal pattern libraries. In addition, the CASE tool should be able to support developers in identifying the set of possible design patterns, at both an abstract and domain specific level. Having a tool that supports rapid prototyping we are more easily able to facilitate user-centered design, since we can involve users in evaluating multiple prototypes that use different patterns (i.e. approaches to solving problems) in order to ascertain which system works best.

If this works effectively, it should allow a wider variety of developers to produce more usable, more effective interactive systems, as they are guided through the process. It should allow the rapid dissemination of good practice, and facilitate the production of agile systems since new technologies do not require a complete rethink of

approaches. Importantly, any tool produced should not constrain the designer to use only those solutions that fit the patterns, but should allow the designer to create their own approaches if they feel that a new style or solution is called for. Since the CASE tool takes care of much of the actual code production, rapid prototyping becomes much easier and more options can be investigated in the same timeframe.

We are pursuing this research agenda for a number of reasons. HCI design patterns are a useful concept for capturing effective aspects of interactive systems, but are not easy to use as a tool for designers. Because both MDA and HCI design patterns exist at a similar level of abstraction, we have good reason to suggest this will lead to interesting results. In addition, an integrated solution is required for effective evaluation. Simply having a CASE tool is not enough - the system should allow us to create more usable software more quickly and be adaptable as technologies alter. We therefore address a number of issues: we extend the lifetime of software, we support the maintenance and speedy upgrade of software on top of rapidly changing technologies, we promote good interactive design practice by providing assistive tool support for patterns, and encourage the spread of consistent effective HCI through a common notation.

3.1 Can design patterns be represented using UML?

The UML proposes a simple mechanism for the modelling of design pattern based on the use of collaboration. Collaborations are generally used to explain how a collection of cooperating instances achieve a joint task or set of tasks. A collaboration describes a structure of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality. Collaborations, although very practical, have some deficiencies(Sunye, Guennec, & Jezequel, 2000) in terms of expressing power. (Sunye et al., 2000) adopt a meta programming approach; applying design patterns by means of successive transformation steps. However, they do not address the issue of interaction and focus on static aspects. (France, Kim, Ghosh, & Song, 2004) and (Kim, France, Ghosh, & Song, 2003)address both static and interaction aspects of the specification of the design patterns. (Mak, Choy, & Lun, 2004) and (Dong & Yang, 2003)both make use of UML class diagrams and OCL statements and suggest extensions of the UML via a profile for the modelling of the patterns, and (Dong, 2003) studies the composition of design patterns.

3.2 Can HCI patterns be represented?

A central question is whether something that is recognisably difficult to quantify, such as HCI issues, can be represented in UML. It is instructive to reflect on the successes in modelling Quality of Service. We plan to address the HCI issues in much the same way as researchers have addressed the design and specification of non-functional aspects of systems such as Quality of Service (QoS) issues. A concept somewhat like 'Good HCI' - we know it when we get it (in fact, it's more that we really know it when we don't get it), defining the constituent parts of a system to deliver a certain QoS is hard. QoS is, like good HCI, an emergent end-to-end property based on the multitude of interactions between different components, not a specific module that you can add in at a later stage: it is an integral, inherent feature of the fundamental design and specific implementation – just as the HCI issues should be.

However, QoS has been successfully modelled via the UML. At the time of the writing, the OMG is reviewing a major submission on "UML Profile for Quality of Service and Fault tolerance"(OMG, 2005b), which is a specification defining a set of UML extensions to represent Quality of Service and Fault-Tolerance concepts. The study of methods of using the UML for the specification and modelling of QoS is a popular area of research. For example, (Bordbar, Derrick, & Waters, 2002; D.H.Akehurst, B.Bordbar, J.Derrick, & A.G.Waters, 2002)present a method of specification of QoS via the UML, which has resulted in the creation of a design support environment for distributed systems, which enables the specification and verification of the QoS. There are also methods of the integration of the QoS into the MDA (Solberg, Husa, Aagedal, & Abrahamsen, 2003)and there are already MDA tools (ISIS, 2004) for modelling the QoS aspects of some systems. Given that this approach works for QoS, we can have some confidence it may work for modelling some of the interactive aspects of the system as well.

There are some interesting questions to address as to whether this, or any, formalism can really effectively capture all the relevant parts of a pattern (e.g. look and feel, aesthetics), and investigating the depth to which we can capture things forms part of our further research agenda. However, since we do manage to capture aesthetics and other

intangibles in things like interior decorating catalogues, this suggests that we may be able to represent a lot of things effectively.

4 Case Study: The Overview-Detail pattern

The overview-detail pattern is a common interface design pattern, used to provide detailed information selected from a much larger set of summary information. One pattern language definition is given below, quoted from Jenifer Tidwel(Tidwell):

> **Overview Plus Detail**
> **Use when:** You need to present a large amount of content - messages in a mailbox, sections of a website, frames of a video - that is too big, complex, or dynamic to show in a simple linear form. You want the user to see the overall structure of the content; you also want the user to traverse the content at their own pace, in an order of their choosing.
> **Why:** It's an age-old way of dealing with complexity: present a high-level view of what's going on, and let the user "drill down" from that view into the details as they need to, keeping both levels visible for quick iteration. Overview Plus Detail breaks up the content into comprehensible pieces, while simultaneously revealing their interrelationships to the user.
> […] the overview can serve as a "You are here" sign. A user can tell at a glance where they are in the larger context. In the example above, the scrollbar shows that the visible message is near the end of the mailbox.
> **How:** The overview panel serves as a selectable index or map. Put it on one side of the page. When the user selects an element in it, details about that element - text, images, data, controls, etc. - appear on the other side. (Usually the overview panel is at the top or left.)
> […] keeping both halves on the same page or window is key. You could put the details into a separate window, but it's not as effective. You want the user to be able to browse easily and fluidly through the UI, without waiting or messing around with windows. In particular, you don't want the user to jump back and forth between two pages (though it's usually necessary on tiny displays like PDAs; see One-Window Drilldown, which doesn't require the use of two side-by-side panels).
> **J. Tidwell**

Examples of this can be seen in the Windows Explorer and typical email clients, shown below:

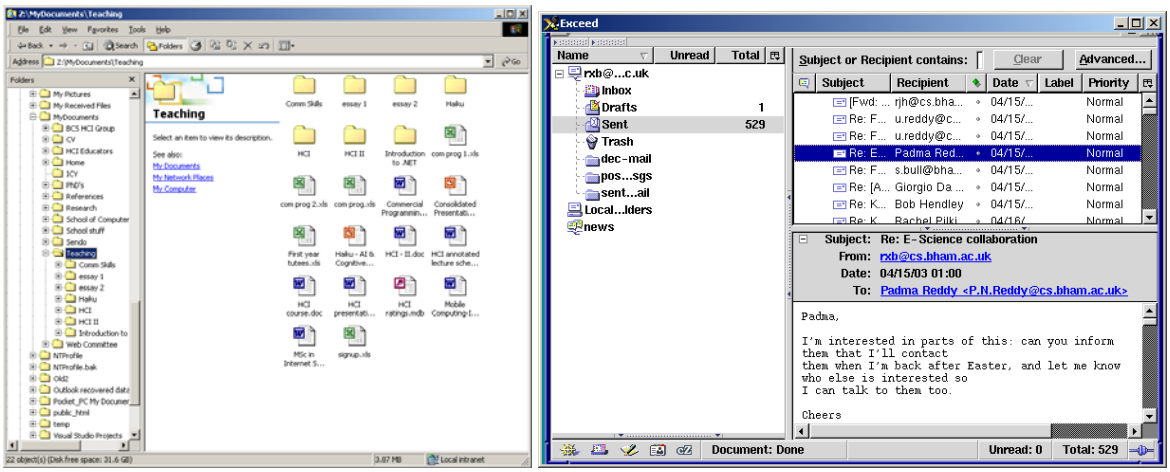

Figure 1: Overview Detail design pattern examples: Explorer and Mozilla.

The Overview is shown in the pane on the left in Figure 1: Folders in Explorer, Mail folders in Mozilla, with details about the selected item on the right – files and folders in Explorer's case, an email message header in Mozilla's. In the Mozilla example, there is also a pane below the detail pane; the overview detail pattern is applied again to the

contents of the mailbox, with a set of message headers in the top as the overview and the detail given in the pane below them.

Tidwel's description of the design pattern is typical of many – a flowing, clear, description in natural language that conveys the essence of both the circumstances under which it is appropriate and what it actually means for the interface. The full pattern also identifies other patterns that are related. The problem is that using these patterns requires very good craft knowledge, as there is no tool support or effective way of browsing or searching them. This makes sharing knowledge with up and coming designers more difficult

4.1 Modelling Overview-Detail

A pattern specification can be seen as a metamodel that characterises UML design model of the pattern solution (France et al., 2004; Kim et al., 2003). As a result, we shall start by creating a metamodel representation of the Overview Plus Detail (OPD) pattern.

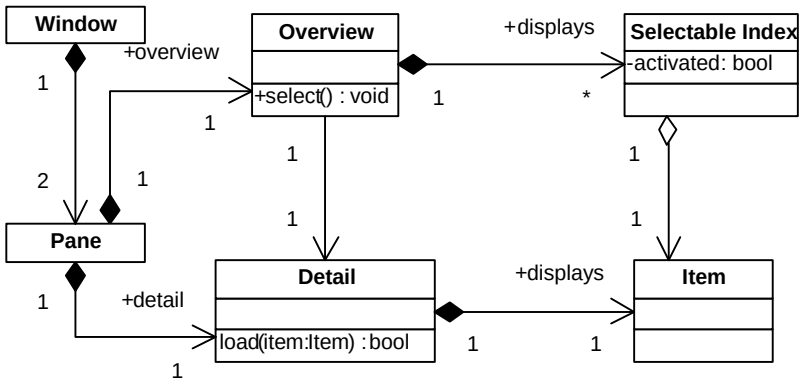

Figure 2: Metamodel specification of OPD

Figure 2 depicts a static view of the OPD comprising of the concepts involved and their relationship. Each *Window* includes two panes; one pane is for the *Overview*, which presents a high-level view of context and the second pane is for *Detail*, which depicts the details related to the high-level view. The Overview is in correspondence with the only one Detail: this is depicted by a unary association connecting the two.

Each Overview meta-element displays a number of *Selectable Index* items. For example, the list of emails depicted in Figure 1 above. Selectable Indexes are linear structures such as tree, list or table (although, it is possible to have other structures; see the signal processing example in (Tidwell). Each Selectable Index is represented by an *Item* depicted in the Detail window. Each Detail window *displays* only one Item; this is depicted by the association from Detail to Item.

To complete the specification of the pattern, we have to specify the dynamic aspect of the pattern by specifying the interaction between the elements. To explain this, consider the mailer example. If the user `select()s` a Selectable Index, e.g. a mail header, its state is changed on the GUI: for example, the email within the Overview window gets highlighted. This results in the change `activated = true`. As a result, the corresponding Item is downloaded to the Detail (invoking `load()`). In case of success in displaying the item `true` is returned, otherwise `false` is returned. As a result, as specified in `load(item:Item):bool`, `load` accepts an object `item` of type `Item` and return Boolean (`bool`).

Such interaction aspects of the system can be represented via a sequence diagram(OMG, 2005c) or an OCL statement. The sequence diagram, which represents a possible interaction of the metamodel elements, is shown in Figure 3

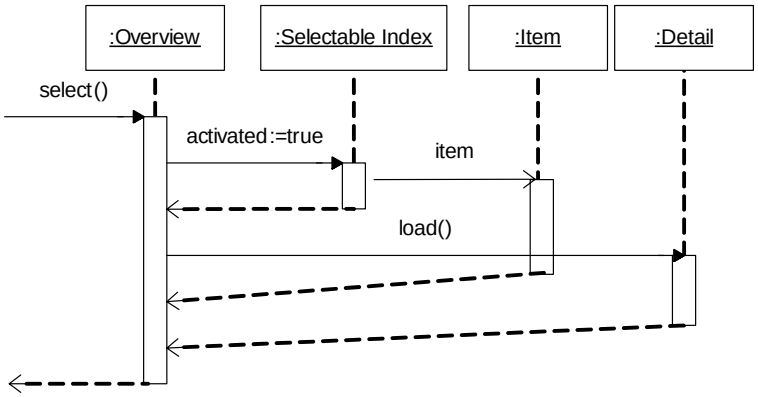

Figure 3: Sequence Diagram representing an interaction in the OWD pattern

We can also use OCL to represent the interaction between the metamodel elements of Figure 2. The OCL representation consists of three main parts, representing the expected behaviour of each method in the context of its related model element. OCL gives us a more precise explanation, which is a logical formalism that can be automatically transformed into code and incorporated into a software tool. We have omitted it here since it doesn't contribute significantly to the current explanation, though it is important to realise that we can programmatically capture the interactive nature of the design pattern.

4.2 Modelling One Window Drilldown

One Window Drilldown (OWD) is an alternative to OPD. It is often used for the user interface of a device with tight space restrictions, such as a handheld device such as a mobile phone. OWD can also be used in building interfaces for applications running on desktops or laptop screens, if complexity is to be avoided. In particular, if the user is not used to computers, they might have little patience with (or understanding of) having many application windows open at once. Users of information kiosks fall into this category; as do novice PC users.

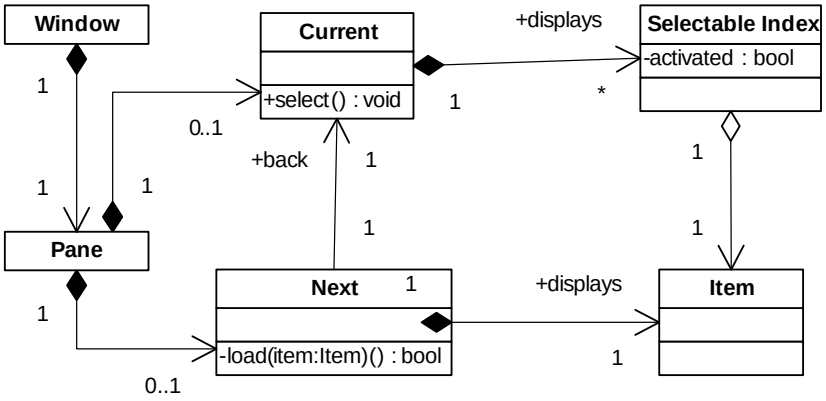

Figure 4: metamodel for specification of OWD

Figure 4 depicts the metamodel for the OWD. There is a single Pane to which a Current and Next data are loaded. On the selection of an item from the Selectable Index, the corresponding Item is loaded as the Next pane. To ensure in the above model there is only one in Current or Next the following OCL constraint is added.

```
context Pane
inv:
  -- There is either a current or a next item (or both)
  -- The if statement takes out the "both" possibility
  self.current -> size() = 1 or self.next->size() = 1 and
        (if self.current -> size() = 1 then
      self.next -> size() = 0
        else
                        self.next->size() = 1
        endif
        )
```

This essentially says that, in the diagram, there could be 0 or 1 Current screen and 0 or 1 Next screen – and we can only display one at a time, hence the need for the constraint.

The behavioural model of the OWD is exactly the same as in Figure 3, which we would expect since the interaction is very much the same. The OCL statement is essentially the same as well (with minor variations, again not presented here).

4.3 Working with formalised patterns

The metamodel specification of overview plus detail, and one-window drilldown, both show that if we have data that has the form of a selectable index that corresponds to more detail, we can use either of the design patterns described here. Since the form of the data is captured in the specification, we expect to extend this work to develop a CASE tool to automatically identify appropriate patterns based on the nature of the data. The interaction in both cases is the same – select an element and see the detail – though the presentation aspects are different. This too is captured in the metamodel – one has two panes in a window, whilst the other only has one. We can use this information to allow us to choose which is most suitable for our application (for a mobile device where screen space is tight, we would choose the single pane model, pushing us to use OWD, for example). It is therefore possible to use the features of the device, or the desired style of the interaction, to allow us to select the set of useful patterns that can be used.

5 Conclusions

We have described an approach to formalising the modelling of HCI design patterns using UML, and presented a case study that models the Overview-Detail pattern and the One-Window Drilldown. The approach allows us to provide automatic identification of when the patterns are potentially applicable, based on the data involved. The models also capture the interactive elements of patterns, and some of their layout/graphical considerations. They also demonstrate the similarities and differences clearly, and suggest that this approach is appropriate for us to automate support for choosing which pattern to apply in a device design situation.

6 References

Abras, C., Maloney-Krichmar, D., & Preece, J. (2004 (in press)). User-Centered Design. In W. Bainbridge (Ed.), *Encyclopedia of Human-Computer Interaction*: Thousand Oaks: Sage Publications.

Alexander, C. (1964). *Notes on the Synthesis of Form*: Cambridge, Massachusetts: Harvard University Press.

Alexander, C. (1965). A city is not tree. *Architectural Forum, 122*(No. 1 pages 58-61, No. 2 pages 28-62.).

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language*. New York: Oxford University Press.

Alexander, C., Neis, H., Anninou, A., & King, I. (1987). *A New Theory of Urban Design*. New York: Oxford University Press.

Bordbar, B., Derrick, J., & Waters, A. G. (2002). Using UML to specify QoS Constraints in ODP. *Computer Network and ISDN systems, 40*, 279-304.

D.H.Akehurst, B.Bordbar, J.Derrick, & A.G.Waters. (2002, October). *Design Support Environment for Distributed Systems.* Paper presented at the Procedings of the 7th Cabernet Radicals Workshop.

Dong, J. (2003, 2003///). *Representing the applications and compositions of design patterns in UML.* Paper presented at the SAC '03: Proceedings of the 2003 ACM symposium on Applied computing.

Dong, J., & Yang, S. (2003). *Extending UML To Visualize Design Patterns In Class Diagrams.* Paper presented at the Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco Bay, California, USA.

Duyne, D. K. V., Landay, J. A., & Hong, J. I. (2002). *The Design of Sites: Patterns: Principles and Processes for crafting a Customer-Centered Web experience*: Addison Wesley.

Erickson, T.*HCI patterns*, from http://media.informatik.rwth-aachen.de/patterns/tiki/hcipatterns.org.html

Erickson, T.*Interaction Design Patterns Page*, from www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html

France, R., Kim, D.-K., Ghosh, S., & Song, E. (2004). A UML-Based Pattern Specification Technique. *IEEE Trans. Softw. Eng. PU - IEEE Press*, *30*(3), 193-206.

Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing*: OMG Press.

IBM. (2004). *User Centered Design, IBM Ease of Use*, from www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/570

ISIS. (2004). *CoSMIC (Component Synthesis with Model Integrated Computing)*, from www.dre.vanderbilt.edu/cosmic/

Kim, D., France, R., Ghosh, S., & Song, E. (2003). *A UML-Based Metamodeling Language to Specify Design Patterns*, from http://www.cs.colostate.edu/~georg/aspectsPub/WISME03-dkk.pdf

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained. The Model Driven Architecture: Practice and Promise*, : Addison-Wesley.

Mahemoff, M., & Johnston, L. J. Usability Pattern Languages: the "Language" Aspect.

Mak, J., Choy, C., & Lun, D. (2004, 2004///). *Precise Modeling of Design Patterns in UML.* Paper presented at the ICSE '04: Proceedings of the 26th International Conference on Software Engineering.

OMG.*Object Management Group (OMG)*, from www.omg.org/

OMG. (2005a). *Model Driven Architecture*, from www.omg.org/mda/

OMG. (2005b). *UML Profile for QoS and Fault Tolerance*, from www.omg.org/technology/documents/modeling_spec_catalog.htm

OMG. (2005c). *Unified Modelling Language (UML)*, from www.uml.org

Solberg, A., Husa, K. E., Aagedal, J., & Abrahamsen, E. (2003). *QoS-aware MDA.* Paper presented at the Implementation and Validation of Object-oriented Embedded Systems (SIVOES-MDA'2003) in conjunction with UML'2003.

Sunye, G., Guennec, A., & Jezequel, J.-M. (2000, 2000///). *Design Patterns Application in UML.* Paper presented at the ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming.

Tidwell, J.*COMMON GROUND: A Pattern Language for Human-Computer Interface Design*, from www.mit.edu/~jtidwell/interaction_patterns.html

Vredenburg, K., Isensee, S., & Righi, C. (2001). *User-Centered Design: An Integrated Approach*: Prentice Hall.