Modeling Software Applications and User Interfaces Using Metaphorical Entities

Christian Nill and Vishal Sikka
SAP Advanced Technology Center, SAP Labs LLC
3421 Hillview Avenue
Palo Alto, CA 94304, USA
{christian.nill, vishal.sikka} @sap.com

ABSTRACT
The power of metaphor has long been recognized in user interface design and more broadly in human interaction circles. More recently metaphor also found its way into the software development process. This paper aims to combine occurrences of metaphor in the two fields with ideas from the field of model driven architecture. We suggest that it is possible to create conceptual patterns based on metaphor that allow a high level description of interaction models and user interfaces, and can at the same time serve as structural units for modeling software applications.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *Methodologies*

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces.* D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures, description languages* H.1.2 [**Models and Principles**]: User/Machine Systems – *Human factors* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical User Interfaces, User-Centered Design*

General Terms
Design, Human Factors

Keywords
Metaphor, MDA, Design Patterns, Software Application Development, User-Interface Design, Interaction Design, Tools and Materials Approach

1. INTRODUCTION
Enterprise-centric computing has come a long way already. At least as far as intra-enterprise solutions are concerned, complex integrated systems are being successfully implemented and deployed. That means the encapsulated business logic is largely in place and major enterprise software companies such as SAP are in principle able to deliver suites that can be applied to almost every aspect of a company's operations.

What has however been neglected over a long period of time in enterprise software is what Frankel in [5] calls the *user interaction logic* and consequently also the *user presentation*. Far too often end-users of enterprise software are forced to think in terms of database operations and transactions rather than in terms that their actual tasks would suggest. Additionally, ever more sophisticated user interfaces of packaged consumer software as well as of web applications such as *amazon.com* add to the rising expectations on usability also in enterprise software applications. In this paper we seek to offer building blocks for the model-driven construction of inherently user-friendly enterprise software applications.

Section 2 starts by introducing important principles of metaphor in user interfaces, followed by section 3 where current uses of metaphor in software development and the Tools and Materials approach are briefly introduced. we go on with presenting our ideas on modeling using metaphorically motivated entities in section 4 and comment on the challenges in section 5. Section 6 briefly touches upon current work and section 7 concludes this paper.

2. METAPHORS IN USER INTERFACES
Metaphors appear all over user interfaces. Although the concrete embodiment of some of the most widely known user interface metaphors such as the DESKTOP metaphor are being challenged at times, only rarely does someone call for their total abolition [13]. Taking Lakoff and Johnson's influential findings into account, such a turning-away from metaphors would have to fail altogether anyway, as "metaphor is pervasive in everyday life, not just in language but in thought and action. Our ordinary conceptual system, in terms of which we both think and act, is fundamentally metaphorical in nature" [8, p.3]. Metaphor must not be understood only as a linguistic construct but rather as a fundamental cognitive mechanism that allows us "to use one highly structured and clearly delineated concept to structure another" [8, p. 61].

Take for instance the well-known and widely adopted metaphor DELETING IS THROWING AWAY found in virtually all modern operating systems. The highly technical operation of deallocating a file name is presented to the user as dragging an unwanted item into a trash can. Even more so, for providing a better match between this metaphor's source and the target a file does not get irreversibly deleted but is recoverable until the trash can, or recycle bin, is emptied. Note that one reason why this metaphor is working so well lies in the fact that the metaphor can be visualized quite nicely.

Barr in [3] has examined the semiotic foundations of user interface metaphors following the triadic notion of signs by Charles Sanders Peirce (1839 – 1914). Figure 1 attempts their visualization along the example just introduced. A sign in semiotics is generally "something that stands for something else,

to someone in some capacity" [11], which means that among many other things, metaphors as well as icons can be treated as signs.

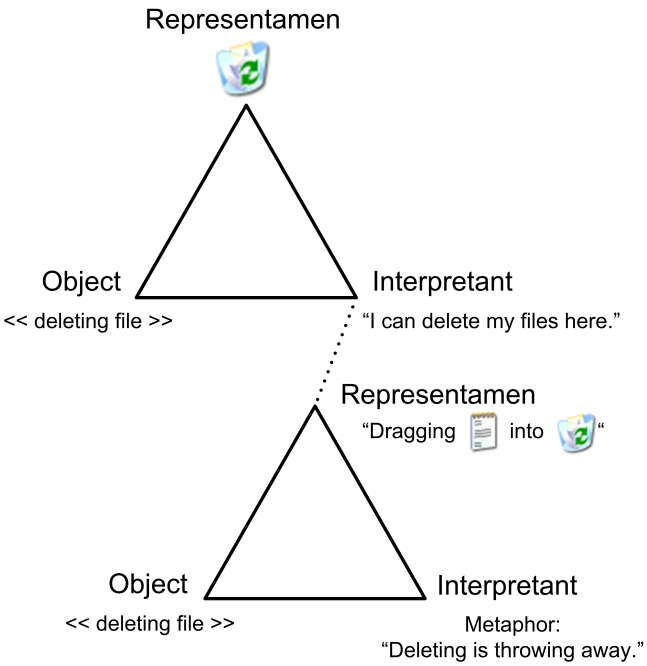

Figure 1: Unlimited Semiosis applied to a "Recycle Bin" icon and the DELETING IS THROWING AWAY metaphor in the semiotic triangle.

In a generative view on the Peircean triad the *object* constitutes the concept to be conveyed by the sign. In the present example that concept is deleting a file. The *representamen* is the directly perceivable portion of the sign, here in form of an indexical icon linking to a recycle bin. What an individual user makes of the representamen is called the *interpretant*. Ideally, object and interpretant show a strong resemblance for the majority of users thus denoting a sign that "works". Important for user interface metaphors is the concept of *unlimited semiosis* meaning that one sign's interpretant can form another signs representamen. In this way a multiply interwoven net of signs can really bring a user interface metaphor to life. Here the fact that one can actually *drag* a "file" *into* the "recycle bin" strongly supports the underlying DELETING IS THROWING AWAY metaphor.

Presenting user interface metaphors in this way might make Marcus' definition more clear who defines them "as the essential concepts in computer-mediated communication that substitute for the underlying code and terminology of operating systems, applications, and data" [10]. However, note in particular from the comments above that for a user interface metaphor to work it needs to be educible. Otherwise it would need to be explained to the user beforehand which is clearly suboptimal.

3. APPLICATION DEVELOPMENT

Quite a lot of work has been published on the use of metaphor in user interfaces and computing in general during the 1980s, but not before a decade later did interest spark in addressing the role of metaphor in the software design process. Rather than only being utilized for explaining functionality that is already in place or being planned for, metaphors are seen as an aid to the construction of software systems. Madsen in [9] reports on the great influence of the selection of originating metaphors on the shape of the resulting system. She then proposes specific guidelines regarding the generation, evaluation and development of metaphors underlying software systems.

The awareness of metaphor as a constructive tool in software development was certainly raised considerably when Kent Beck in [4] listed the *system metaphor* as one of the original 12 principles of extreme programming, although in his work he did not elaborate greatly on its correct use within the XP methodology.

An even more high-level use for a special kind of metaphors in enterprise architecture modeling was proposed by Khoury in [7]. He sees an ideal vehicle for structuring enterprise applications in organizational metaphors. As opposed to concrete metaphors like the RECYCLE BIN mentioned in section 2, organizational metaphors are motivated by societal structures. Examples for sources of these metaphors comprise e.g. auctions, games, or committees.

3.1 The Tools and Materials approach

Largely unique in this area is the *Tools and Materials approach*. Coming from software engineering, Züllighoven and his team proclaimed an application-oriented evolutionary software development process. They define application-orientation as the orientation towards the tasks in a given application domain. They also demand that processes defined within a software system be easily adaptable to any given working context and finally expect such a system to be thoroughly user-friendly [14, p. 102].

3.1.1 Overview

Simply speaking, the T&M approach utilizes a guiding metaphor and three concrete core metaphors or *conceptual entities* along with some others to structure any specific application domain. It sees users being placed at a well equipped EXPERT WORKPLACE working with TOOLS on MATERIALS and having AUTOMATONS which relieve them of repetitive tasks that do not require user interaction.

Züllighoven and his team argue that these metaphors which are partly grounded in anthropology and partly on ergonomics are not only universally understood, but also perfectly apt to describe self-determined human work. The metaphors are deemed sufficiently concrete to let users and developers alike associate useful ideas and at the same time are easily transferable to different application domains [12].

The true beauty of the T&M approach however lies in the fact that metaphors are not only means of observation and communication between all stakeholders in the software development process, but also the cornerstones of the software application's architecture. The approach comprises a complete set of design patterns that allow mapping the conceptual patterns onto an object-oriented software model and constitute a major constructive step toward actual software implementation [14, pp. 185-280]. Along these patterns also the open-source framework JWAM[1] had been created that constitutes a readily implemented infrastructure around the metaphorical concepts mentioned above.

[1] See http://sourceforge.net/projects/jwamenvironment/ for details.

3.1.2 Criticism

Yet the approach can be improved. From an HCI perspective the application-oriented T&M approach still falls a little short when it comes to empathy for the end-user. The reason for this is that it is only concerned about structuring an *application domain* but not about how elements of the resulting software application are being presented to end-users. Only a rather vaguely defined *usage model* is offered, based on the T&M metaphors. We however believe that merely *structuring* the software application the same way as the application model, i.e. using a small set of anthropologically and/or ergonomically motivated metaphors, will not automatically lead to a good user interface for several reasons.

First, in order to work in a graphical user interface any implicit metaphor of the form THAT PIECE OF FUNCTIONALITY IS A TOOL or THIS DATA STRUCTURE IS A MATERIAL has to have a usable metonymy, i.e. one or more parts that stand for the whole of the metaphor. Those parts need to be visually representable, such as the RECYCLE BIN from section 2, to allow users understand possible actions in the user interface. While for TOOLS that seems quite possible (see e.g. the TOOLBOX metaphor in Adobe Photoshop), this will often be difficult when trying to represent rather abstract MATERIALS like a bank account or a workflow template as in [6].

Second, as Johnson and Lakoff have observed, "comprehending one aspect of a concept in terms of another will necessarily hide other aspects of the concept" [8, p. 10]. This means that we usually use vehicles of a number of metaphors greater than one to explain more abstract concepts. Not every aspect of functionality of an application is really best explained using only the TOOL or the MATERIAL metaphor.

3.1.3 Concluding Remark

What remains is the fact that Züllighoven and his team very successfully modeled application domains and software applications alike using the same set of structural metaphors in various major software projects (cf. [14, pp. 9-14]). The T&M approach therefore seems to be a good starting point for further work.

4. MODELING

Software application models, e.g. using the UML notation, are a useful means of communication amongst developers to arrive at a common understanding of a software application's structure and functioning. The perception of models as "needless slideware" shared primarily amongst some followers of agile development practices is likely to be overcome by advances in the fields of Model Driven Architecture and Model Driven Software Development. New disciplines such as Agile Modeling and developments such as the Executable UML are already pointing into that direction.

The kind of models that denote the inner workings of software applications is however in general not a good means of communication when it comes to non-technical stakeholders. Arlow and Neustadt in [1] report on the poor levels of comprehension that domain experts, users, and non-technical managers demonstrate when interpreting UML models of almost any kind. That is why virtually all practical implementations of user-centered design methodologies put an emphasis on often iterating a user feedback cycle based on prototypes [2]. "Low-fidelity" prototypes at an early stage are in the course being substituted by more sophisticated and functionally accurate ones later on. These prototypes form the backbone for end-user involvement once the initial requirements gathering has been concluded. End-users, as well as other non-technical stakeholders, will express their thoughts related to visual elements of these prototypes. In a sense these prototypes act as high-level interaction models that non-technical people utilize for understanding and talking about the software application being developed.

These observations lead to the search for conceptually easy-to-handle entities for modeling interactive software applications on a considerably higher level than objects and classes. Metaphorical concepts such as described in section 3.1 would constitute a natural basis for such modeling entities. These elements could be used in “classical” software application models and at the same time be incorporated into prototypes of all kinds. In this way developers can communicate with end-users using a common vocabulary, greatly reducing impending frictions. However to be adaptable and still remain apt for an automated transformation into Platform Independent Models (PIM) or other forms of less abstract representation, any such interaction element needs to be reasonably well defined. Section 6 touches on a concrete example.

Arlow and Neustadt [1] succeeded in a very similar approach concentrating on the enterprise tier or encapsulated business logic (again using Frankel's nomenclature [5]). They developed so-called *enterprise archetype patterns* for the UML that could be applied to the greatest number of businesses. Their patterns include generic concepts such as *product*, *inventory*, *order*, and so on which can be used as modeling entities in UML diagrams. Every such archetype pattern usually spans a considerable number of classes. The huge advantage of these elements lies in their semantic richness and the hiding of technical details, so that even non-technically inclined stakeholders such as managers and domain experts can more fruitfully participate in the shaping of application models. Yet these patterns can be easily transformed into standard UML diagrams and thus do not deviate from the fundamental ideas behind MDA.

In the same manner we hope to establish a set of *archetype interaction patterns* that encapsulate the sort of metaphorical concepts touched upon in sections 2 and 3. Benefits in terms of communication with stakeholders are only one side. Structuring of interactive software applications along metaphorically motivated entities should also yield the usage model and subsequently a blueprint for the user-interface.

5. CHALLENGES

Without doubt the ideas presented so far are highly ambitious and challenges may seem daunting. However they can be broken up and tackled one by one.

First, an adequate set of interactional archetypes would have to be found. It is likely to turn out that such a set can only be defined for a certain kind of application at a time. The kind of contemplable applications might be expressed quite generally by an underlying *guiding metaphor* such as the EXPERT WORKPLACE proposed by Züllighoven or by any other form of a usage model.

In a second step one would have to think about how the metaphors or differently rooted concepts underlying those

interactional archetypes could be visualized and presented to the user. A set of interrelated UI elements, *user interface patterns*, resulting from this step could e.g. be used as building blocks in a graphical development environment. They would be used to not only produce the user interface but also map out a meaningful skeleton for the software application's implementation.

At the same time also technical implementations for the desired target platforms have to be developed. In an MDA scenario that target platform would be a PIM, while in traditional software development mapping to design patterns as mentioned in section 3.1 presents itself as a natural choice.

Last but not least this approach would only live up to its full potential if adequately supported by a development environment that revolves around the conceptual elements chosen as a basis for modeling. Ideally such a development environment also offers extensive support for visual prototyping and modeling, allowing for generating artifacts helpful in communication with non-technical stakeholders.

6. CURRENT WORK

We are currently utilizing the Tools and Materials metaphors from section 3.1 for modeling a cost planning application that is to be used for gathering budget planning data from a company's cost center managers.

The application is composed from a number of distinct tools that can each be used to work on their specific materials. An important material within the application is e.g. the Planned Cost Spreadsheet which holds planning figures and formulae. The appropriate tool to work on it is a Spreadsheet tool which in turn contains a number of sub-tools that are also being reused elsewhere in the application. One such example of a multiply reused sub-tool would be an Annotation tool that allows adding comments and other unstructured data to materials, in this case to spreadsheets. The advantage of using the metaphorical notion of tools and materials in this modeling process lies in the fact that non-technical stakeholders will find the resulting components easy to understand and their interrelationship easy to comprehend.

At the same time as modeling and describing the application by the means of tools and materials, we are also trying to get to the bottom of how to best compile and present such models to the technically less inclined and also how to best support the transition to more specific design models afterwards.

7. CONCLUSION

Most likely, the production of interactive software applications using metaphorical archetypes as suggested in this paper will not result in the best conceivable interaction and interface design ever. It *will* however allow for a much more intense involvement of non-technical parties in the software development process where such an involvement used to exceed the available resources. At the same time it allows for software product lines that don't only *look* similar, but actually really *feel* belonging together without additional effort. Apart from that the use of interactive archetypes should also speed up the overall software application development process.

8. REFERENCES

[1] Arlow, J. and Neustadt, I. *Enterprise Patterns and MDA - Building Better Software with Archetype Patterns and UML*. Object Technology Series. Addison-Wesley, Boston, MA, 2003.

[2] Ashley, J. and Desmond, K. Oracle. *Interactions*, 9(2):81–86, 2002.

[3] Barr, P., Biddle, R., and Noble, J. A semiotic model of user interface metaphor. In *The 6th International Workshops on Organisational Semiotics*, Reading, UK, 2003.

[4] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, first edition, 1999.

[5] Frankel, D. S. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press. Wiley Publishing, Inc., Indianapolis, IN, 2003.

[6] Gryczan, G., Wulf, M., and Züllighoven, H. Prozeßmuster für die situierte Koordination kooperativer Arbeit. In *Herausforderung Telekooperation (D-CSCW '96)*, pages 89–103. Springer, Berlin, Germany, 1996..

[7] Khoury, G. R. and Simoff, S. J. Enterprise architecture modelling using elastic metaphors. In *Proceedings of the first Asian-Pacific conference on Conceptual modelling*, pages 65–69, Darlinghurst, Australia, 2004.

[8] Lakoff, G. and Johnson, M. *Metaphors We Live By*. The University of Chicago Press, Chicago, IL, 1980.

[9] Madsen, K. H. A guide to metaphorical design. *Communications of the ACM*, 37(12):57–62, 1994.

[10] Marcus, A. Metaphors and user interfaces in the 21st century. *Interactions*, 9(2):7–10, 2002.

[11] Peirce, C. S. *Collected Writings*. Harvard University Press, Cambridge, MA, 1958.

[12] Rose, H., editor. *Objektorientierte Produktionsarbeit*, pages 23–54. Campus Verlag, Frankfurt, Germany, 1996.

[13] Tristram, C. The next computer interface. *Technology Review*, 104(10):52–61, December 2001.

[14] Züllighoven, H. et al. *Object-oriented construction handbook: developing application-oriented software with the tools and materials approach*. Morgan Kaufmann, San Francisco, CA, 2004