User Interface Development by Proxy

Jacob Winther Jespersen
User Interface Development by Proxy

Jacob Winther Jespersen
IT University of Copenhagen
Glentevej 67, DK-2400 NV, Denmark
jwj@itu.dk

ABSTRACT
A branch of software development is characterized by the reliance on Partners to adapt a Vendor’s domain specific software to fit Customers. As part of a project that researches this software value chain, we investigate specifically the requirements and techniques to develop user interfaces in a suitable fashion.

In the Vendor-Partner-Customer arrangement, a Vendor is responsible for his platform’s points of variability, i.e. the ways in which a Partner may utilize the Vendor’s software components. With the proper variation points Partners can efficiently construct finished user interfaces to honor particular requirements for task support and visual appearance.

This paper outlines an architecture for the design and construction of user interfaces in the scenario. We point to the relevance of model-based user interface development to the context, present the notion of a user interface family, and discuss preliminary results.

Author Keywords
HCI, partner development, model-based user interface development, user interface family.

ACM Classification Keywords
H.5.m Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
In commercial information systems development, the Vendor-Partner-Customer business model currently seems to be successful. The Partner is often a third-party but occasionally a part of the Vendor or Customer organization.

The key to appreciate the setup is the specificity of the domains that the systems address. Because each Vendor targets a single domain, e.g. accounting or financing, he produces domain-specific tools and components. For example, if a Vendor seeks to facilitate his Partners to build individual accounting systems, he concentrates on providing a solid and competitive accounting core functionality, i.e. the functionality needed in every accounting system. Thus, he need not consider exactly what differentiates a car-dealer from other kinds of dealingships in the accounting sense. Knowing about such differences is left to Partners who are supposed to have specialized knowledge of the business domain they work in.

Partners rely on the variability within a Vendor’s tools and components to do systems that honor particular requirements at their respective customers. Having fewer variation points and semantically richer constructs than a general-purpose programming language potentially make a proficient Partner able to efficiently develop systems to a particular domain, without compromising the quality of the user interface.

MODELING THE USER INTERFACE
Even if the Vendor-Partner-Customer arrangement promotes a development methodology that by conception is not designed to foster user interfaces of a high quality, it is not clear that it prevents them. However, finding a suitable level of abstraction for the Partner’s user interface development has proven to be a challenge [1]. At one end of the spectrum, specifying windows, layout, and widgets does not leverage the domain knowledge that is present; at the other end, mapping pure domain constructs to user interface elements sacrifice flexibility in visual appearance and task support.

Thus, given this development arrangement, the research seeks to clarify: What kind of user interface modeling constitutes a suitable specification of the executable user interface code in a final system?

Being a good Partner
From an HCI perspective, one can argue that the Partner – as portrayed above – plays a superfluous role. He is put in place to facilitate re-use of software components and thus to speed system development, which is not a virtue on its own in the HCI agenda. Also, customers sometimes have difficulties differentiating between Partner and Vendor, and may expect Partners practically to assume the traditional
Vendor role. In many respects, a Partner is able to fill that expectation; one can say that a Partner acts on behalf of a Vendor, i.e. the Partner is a proxy to the Vendor. But, ultimately, Partners do not have control of the variation points in the Vendor’s system components.

Being a good Partner means developing systems within the technical boundaries set by the Vendor, i.e. making do with the functionality in the Vendor’s tools and components. When a Partner resorts to self-made functionality outside these boundaries, he steps down from his privileged position on the Vendor’s platform.

**System families**
On the System Engineering side, there are techniques in place to support the Vendor’s development activity. Concepts such as product families a.k.a. system families [2] have already informed ambitions to reduce the cost of developing individual systems with common properties. (“A product family is a group of products that can be built from the same assets.” [3])

In HCI, there is no direct equivalent to the family scope; rather – to use a complementary term – HCI practice is usually concerned with product lines, i.e. “... a group of products sharing a managed set of features that satisfy the specific needs of a selected market.” [3] For example, HCI practitioners could consider it important to establish coherence in the user interaction within an office application product line (word processing, spreadsheet, presentation creator). However, we are not pursuing product line issues in the current project.

**CONCLUDING REMARKS**
Where the kinship is merely use of windows and widgets only little is gained. Standard window toolkits of most operating systems already have assets at that abstraction level. Kinship of a higher order (e.g. specialized windows with a certain visual appearance and behavior) reuses more valuable assets such as the code to support the complex human-computer interaction involved in controlling non-trivial business objects.

We have designed and implemented a first generation of the architecture in which the abstract model only allows us to represent generic entities with attributes; basically, it is a data browser with editing capabilities. As such, it shares the fate of much related work in model-based user interface development [4]. It does not provide for the necessary sophistication of the user interface construction, primarily because variation points are aligned solely with data structures e.g. as in [5], and not according to a domain’s established style of visual appearance and task support.

The hypothesis we go by, as we continue the research, is that model-based user interface development can provide for user interface families and is well suited to the proxy arrangement. We also believe that a high degree of domain-dependence in the user interface modeling is a necessity to achieve satisfactory results from this kind of development.

**REFERENCES**