Component-Based Design and Software Readymades

Anders I. Mørch[1] and Li Zhu[2]

[1] Department of Educational Research and InterMedia, University of Oslo, Norway
anders.morch@intermedia.uio.no
[2] Department of Computer Science, Università degli Studi di Milano, Milan, Italy
zhu@dico.unimi.it

Abstract. End-user developers need access to tools and techniques that allow them to create, modify, and extend software artifacts without programming. Previous research has shown that visual software components can provide the right level of abstraction. However, component-based design (CBD) will succeed only if there is a good balance of standardization and flexibility (software issues) and a good balance of usefulness and usability (HCI issues). We present a vision for CBD and two approaches toward achieving it: 1) design by composition and 2) design by redesign. We claim that the latter is more user friendly but lacks the flexibility of the former. We propose the notion of "software readymade" as a theoretical concept to integrate them, inspired by the role of the "spectator" in the work of the artist Marcel Duchamp. We propose stand-alone multiperspective tailorable software components to instantiate the concept, and we give two examples (application units and nuggets).

Keywords: Application units, component-based design, nuggets, readymades, software components, tailorable components.

1 Introduction

At the 2011 EUD conference in Brindisi, Italy, Fabio Casati gave a keynote describing failed attempts of end-user application composition environments (e.g., component-based design environments, workflows, service composition, and mashup builders). A reason for the failure is that these technologies expose low-level (software program) features that are difficult for end users to make sense of. However, it is difficult for developers to anticipate end-user developers' needs because the needs are emergent and circumstantial (based on use), and arguably, more network-oriented (rooted in human relations and work organizations) than hierarchical (software organization). This discrepancy can be addressed by software engineering methods that allow applications to be modified during development in response to user requirements, but users should also be actively engaged in the process, drawing on their domain expertise and creativity. We discuss the pros and cons of component-based design (CBD) for end-user development (EUD), addressing both professional and end-user developer needs. We present a vision and two approaches to CBD toward that end: 1) design by composition and 2) design by redesign. Next, we propose a theoretical perspective to integrate the two approaches, and we summarize our efforts at developing tools and techniques based on this perspective (multiperspective, tailorable, autonomous software components).

2 Vision of Component-Based Design

We present an illustrative example for balancing standardization and flexibility (technical issues) and for usefulness and usability (user-oriented issues). Furniture design, and chair design in particular, is a good analogy because chair designers are concerned with many of the same issues that user interface designers and software developers deal with: producing flexible variations of a generic product informed by a design concept. Figure 1 shows a picture of how we envision flexibility incorporated in CBD: enabling end-user developers to create new applications based on an application platform (generic application), without making the same application twice. In this way EUD-enabled applications will also be creativity support tools.

Fig. 1. A vision of component-based design: combining standardization and flexibility within a constrained design space of creative reconfiguration. Reprinted with permission from the artists: Martino Gamper (100 chairs in 100 days) and Angus Mill (photo) [7].

3 Component-Based Design of Software Applications

Our idea of component-based design (CBD) of software applications has been influenced by Fischer's notion of a domain-oriented design environment (DODE) [2]. He proposed that the basic building blocks are domain-oriented components that are connected by meaningful relations as defined by external criteria, ranging from design rules to user preferences. More commonly, component-based design (CBD) means to create new functionality (e.g., applications) by combining existing functionality (e.g., software components). We distinguish two approaches to CBD: 1) design by selecting from a library of basic components and a work area for composing them and 2) design by modifying a generic (tailorable) application to create new applications. We briefly review work in each of the two areas below.

3.1 Design by Composition

The FreeEvolve platform developed by Wulf, Pipek, and Won provides a palette of basic search application functionality for stitching small database applications within the domain [8]. A user study of the system revealed the strength of direct activation of tailoring functionality and the weakness of manually connecting two components. User-assistance techniques such as 3D visualization and organization (part/whole structures) were added to resolve the weaknesses.

Web services are a more recent innovation of software component integration, associated with web applications. A study conducted by Mehandjiev [3], comparing three different web service integration models, found that users preferred one with a logic that abstracted features of programming and more easily aligned with the users' mental model of the task (flowchart model) [3]. However, most application composition environments require developers to follow another logic (dataflow), exposing the various sources and sinks of data required for composition, which prevented end-user developers to participate beyond simple applications.

Mashup components are the latest trend in application composition, as they are more flexible by allowing user interfaces and data in addition to software functionality to be composed. Muhammad and colleagues [5] found evidence that domain-oriented mashup builders are more usable for end-user developers than generic builders and demonstrate this by developing a domain-oriented builder for scientific publication ratings and comparing it with Yahoo Pipes (a generic builder).

3.2 Design by Redesign

Many domain-independent application environments have turned out to be successful EUD environments. Arguably, the most famous is the spreadsheet application. With the use of a formula language (e.g., Excel macros), numerous applications of the same basic user interface have been created [6]. Several hybrid application/application builders have since been proposed (e.g., MS Office with Visual Basic for Applications), but none of them have achieved the same fame as the spreadsheet when it comes to supporting EUD. A key to success has been a combination of a generic (multipurpose) user interface and a mechanism for producing variation at a scale that is both useful and usable by application users.

Google Maps shares many of the characteristics of a spreadsheet in so far as it provides a combination of multipurpose user interface and a mechanism for generating variation without an excessive amount of programming. An early application (arguable comparable to VisiCalc in fame) is HousingMaps, a "mashup" created by Paul Rademacher in 2005. He integrated a housing-rental and for-sale listing (craigslist) with Google Maps to form a new kind of application (http://www.housingmaps.com/). Numerous Google mashups have since followed.

The variability mechanism provided by Google Maps differs from the spreadsheet formula language. Integrating the Google Map API with data sources and related components in a mashup builder is one way to create applications. Another way is to make custom maps by manually typing in addresses. The latter is more time consuming for large data sets but simpler for users without technical expertise (e.g., the My Places wizard and tutorial in Google Maps).

The tradeoff between flexibility and usability is only partially resolved by each of the two approaches. The success of “design by redesign” depends on “killer apps,” and the success of the “design by composition” depends on access to a sufficient number of interesting components to choose from. We propose a framework for combining them and addressing the vision described in section 2, combining a generic application with a set of components to extend it. This framework is presented below in two parts, first as a theoretical concept (software readymade) that is later operationalized in two prototypes (application units and nuggets). This work is continually evolving and is not yet finished.

4 Software Readymade as a Theoretical Concept

The artist Marcel Duchamp coined the term *readymade* in 1913 when he started to transform everyday manufactured objects into art by a series of operations performed by the artist. He suggested how spectators should perceive, react, and make sense of the artwork. Duchamp’s famous and provocative *Bicycle wheel* and *Fountain* are two examples of readymades, the latter borrowing and appropriating a porcelain urinal, the former using a bicycle wheel mounted on a kitchen stool. Despite the readymades being “locked” from conventional use, Duchamp was an advocate of active and continuous use, and he explained it by the following quote: “After all, the artist alone does not perform the creative act. The spectator brings the work in contact with the external world by deciphering and interpreting its inner qualities and thus adds his contributions to the creative act” [1].

Duchamp’s concept of readymades and how they can be interacted with provides clues for how end users can be involved in composition and redesign. For example, across the two domains of art and end-user development, we find the following common themes: 1) acts of intervention in established practice (manufacturing), 2) opening-up functional objects for introspection, and 3) revealing relations and viewpoints that have previously been hidden. We have developed prototypes for exploring the second and third themes that aim to reveal hidden relations and viewpoints of functional objects (software components) in order to increase end users’ awareness of modification options.

5 Tailorable Autonomous Software Components

The software components we have developed are “small applications” that can be used separately, opened up for viewing, and modified within an application framework. The first author developed “application units” [4], and the second author more recently developed “nuggets” inspired by the former [9]. We present a brief summary of our past and current work.

5.1 Application Units

Application units are components of conventional application, creating cognitive chunks that are easier to comprehend than complete applications. To support “opening up,” each component is organized into views or aspects: 1) user interface, 2) design rationale, and 3) program code. Users can access each aspect by holding down a modifier key (alt, ctrl, or shift) when clicking on the application unit. Each aspect can be modified in a separate editor, and the data is then stored in an initialization file [4]. The goal of application units is to simplify access to the different parts of UI components that have to be modified during end-user tailoring.

Another goal of the application units is to support learning on demand and incremental mastery of computational complexity. We conducted a video-recorded usability test of BasicDraw with twelve users and asked them to perform end-user development modifications to the application, and we found that customization (modifying user interface) and integration (updating design rationale) could be achieved without much instruction or help. The extension (writing program code in method bodies), however, required assistance, or knowledge of programming and basic skills in object-oriented programming [4].

5.2 Nuggets

MikWiki is a user-extensible wiki [9] where the components of the system are represented by a set of web pages, referred to as *nuggets*. Nuggets are independent of each other and bundled with EUD tools. As code executes on the client side, users can change the behavior of existing nuggets or create new nuggets from and within the wiki and thus evolve the wiki at use time.

Similar to the “multiple aspects” of application units, each nugget has three perspectives: 1) visualization, 2) format, and 3) data representation. An example is the *Imagenote* nugget that allows end users to create “Post-it-like” image notes, which are synchronized between users. The data page contains the wall data in JSON format, the format page defines how to represent the JSON data in JavaScript, and the visualization page embeds the macro-like code that expands to the visible nugget on the screen.

MikiWiki provides a set of stand-alone components. These are components to support communication, coordination, and localization that track history, enhance awareness, and provide authentication and annotation services.

Empirical studies of MikiWiki extend applications units by demonstrating that EUD can be achieved in naturalistic settings (outside usability laboratory). They are uncomplicated to use once familiar with editing web pages and wiki articles [9].

One shortcoming we have found is that EUD using JavaScript still imposes a steep learning curve for many end-user developers. Additional components, user documentation, tutorials, and templates must be provided to flatten the learning curve.

6 General Discussion and Directions for Further Work

We distinguished between design by redesign and design by composition. A challenge for design by composition is specifying communication protocols between components and resolving unambiguous input-output ports. A challenge of design by redesign is to enable end-user developers to extend generic applications to create something new. We proposed a vision for CBD by the creative reconfiguration of chairs and provided a theoretical account of the vision by the notion of software readymades. Our own efforts to instantiate the concept with multiperspective, tailorable application units and nuggets revealed strengths and weaknesses. The strength is that it can give end users access to multiple aspects of a software component (e.g., user interface, code, and data) in order to simplify end-user tailoring. A weakness is that having access to code may not help, as many end-user developers find programming difficult, and scripting languages is also difficult in this regard.

Directions for further work include exploring how small applications can expose hooks, open points, and reconfiguration options, starting with what manufacturers refer to as sizes and models of standard components, and extending this to modularized generic applications that can be taken apart and recombined to make new applications. Moreover, future work should address what it means for stand-alone components to be integrated without sending or receiving data. For example, integration can be approached from a domain-oriented perspective as in the work of Fischer and colleagues (defined by design rules, perspectives, and preferences). Integration can also be achieved within a social-technical framework, being less about component interfaces and more about new tools for viewing, modifying, and sharing.

References

1. Duchamp, M.: The Creative Act. In: Lebel, R. (ed.) Marcel Duchamp, pp. 77–78. Paragraphic Books, New York (1959)
2. Fischer, G., Girgensohn, A., Nakakoji, K., Redmiles, D.: Supporting Software Designers with Integrated, Domain-Oriented Design Environments. IEEE Trans. on Soft. Eng. 18(6), 511–522 (1992)
3. Mehandjiev, N., Namoune, A., Wajid, U., Macaulay, L., Sutcliffe, A.: End User Service Composition: Perceptions and Requirements. In: Proceedings ECOWS 2010, pp. 139–146. IEEE Computer Society, Washington, DC (2010)
4. Mørch, A.I.: Aspect-Oriented Software Components. In: Patel, N. (ed.) Adaptive Evolutionary Information Systems, pp. 105–122. Idea Group, Hershey (2003)
5. Muhammad, I., Florian, D., Fabio, C., Maurizio, M.: ResEval Mash: A mashup tool that speaks the language of the user. In: Proc. CHI 2012, pp. 1949–1954. ACM, New York (2012)
6. Nardi, B.A., Miller, J.R.: The spreadsheet interface: A basis for end user programming. In: Proceedings INTERACT 1990, Amsterdam, The Netherlands, pp. 977–983 (1990)
7. Thompson, H.: Remake it Home. Thames & Hudson Ltd., London (2009)
8. Wulf, V., Pipek, V., Won, M.: Component-based tailorability: Enabling highly flexible software applications. Int. J. Hum.-Comput. Stud. 66(1), 1–22 (2008)
9. Zhu, L., Vaghi, I.R., Barricelli, B.R.: A Meta-reflective Wiki for Collaborative Design. In: Proceedings WikiSym 2011, pp. 53–62. ACM, New York (2011)