Design and implementation of a component architecture for Oberon

Author(s):
Marais, Johannes Leon

Publication Date:
1996

Permanent Link:

https://doi.org/10.3929/ethz-a-001676968 →

Diss. ETH No 11697

Design and Implementation of a Component Architecture for Oberon

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

presented by
Johannes Leon Marais
M.Sc Computer Science, RAU
born July 21, 1967
citizen of Cape Town, South Africa

accepted on the recommendation of
Prof. Dr. J. Gutknecht, examiner
Prof. Dr. N. Wirth, co-examiner

1996

Abstract

The traditional view of computer software is that of off-the-shelf programs with little or no possibility of customization. Engineers create this type of software by combining software components in such a way that the delivered product does not show how it was constructed. Unfortunately, this construction technique does not allow the buyer to adapt components, add new components to or remove components from the program. So, buyers unhappy with the functionality of a product have little choice but to wait until the vendor releases an enhanced version of the program. Consequently, software vendors slowly incorporate new features in successive versions of a program, features that are not necessarily useful to all users. The result is that programs tend to get bigger with each new version, with no significant improvement in functionality.

The obvious solution to this problem is to give the knowledgeable buyer the flexibility of composing custom software from prefabricated parts. This approach has the advantage that only those parts that are actually required need to be bought, and with a variety of parts readily available, the buyer has much more freedom in constructing software. The most end-user accessible software that fall in this category of configurable systems are *document-based* user interfaces. A document-based user interface regards applications and their user interfaces as components that can be composed and configured interactively as if editing a document. The distinction between a document-based user interface and a GUI builder is that a document-based user interface allows the users to modify applications even after they are installed. In the simplest example, this would even allow a user to modify a dialog box in an installed application.

The principal technical difficulty in building a document-based interface is to provide the communication infra-structure that allow independently developed components to be combined interactively. Without a mutual communication protocol, components from different vendors cannot "plug" into each other. Without a dynamic connection mechanism, components would not be able to be added to

or removed from a system while it is running.

This thesis describes the design and implementation of a component architecture and document-based user interface for the Oberon system. The *Gadgets system* uses first-class components called *gadgets* to build documents and connect the documents with functionality. Gadgets ranging from typical user interface elements to high-level building blocks are composed interactively in active documents. An important feature is that independently developed components can be integrated with existing components without special effort. This gives the *composer* the ability to compose applications from a large palette of reusable components, and thus speeds up the development process.

This thesis addresses several issues, including how components communicate, inter-component communication protocols, how components are made persistent, how components can be combined into more complicated components, how custom behavior can be attached to component instances, and how documents constructed in this way are consistently transported. The thesis also discusses issues related to system structuring, dynamic extensibility, component authentication and system security.

Zusammenfassung

Traditionellerweise wird Computer-Software mit dem Begriff Programm-Paket ("off-the-shelf" program) verbunden, wobei typischerweise wenig oder keine Möglichkeit zur kundenspezifischen Anpassung der Software besteht. Solche Programm-Pakete sind in einer Weise konstruiert, die es nicht offensichtlich macht, aus welchen Komponenten sie zusammengesetzt wurden. Unglücklicherweise erlaubt diese Konstruktionsweise dem Käufer nicht, einzelne Komponenten anzupassen, neue Komponenten hinzuzufügen oder Komponenten zu entfernen. Ist ein Kunde unglücklich mit der Funktionalität eines solchen Produktes, so bleibt ihm oder ihr nicht viel anderes übrig als auf eine verbesserte Version des Herstellers zu warten. Konsequenterweise fügen Software-Hersteller langsam immer mehr Funktionen in neuere Versionen eines Programms hinzu, Funktionen die nicht notwendigerweise nützlich für alle Benutzer sind. In der Folge tendieren Programme dazu, mit jeder Version grösser und grösser zu werden, ohne dass dabei die Funktionalität signifikant erhöht wird.

Gibt man dem informierten Kunden die Flexibilität, ein System aus vorfabrizierten Teilen zusammenzusetzen, erhält man eine offensichtliche Lösung zu diesem Problem. Dieser Ansatz hat zusätzlich den Vorteil, dass man nur diejenigen Teile zu kaufen braucht, die man wirklich benötigt. Auch hat der Kunde mehr Freiheit, Software aus einer Vielzahl vorfabrizierter Komponenten zusammenzustellen. Aus der Kategorie solcher konfigurierbarer Systeme sind dokumentenbasierte Benutzeroberflächen am ehesten zugänglich für den Endbenutzer. Eine dokumentenbasierte Benutzeröberfläche betrachtet Applikationen und ihre Benutzer-Schnittstelle als individuelle Komponenten die wie Dokumente frei zusammengesetzt und konfiguriert werden können. Die Möglichkeit die Benutzeroberfläche sogar nach der Installation zu verändern unterscheidet eine dokumenten-basierte Benutzeröberfläche von einem "GUI builder". Im einfachsten Fall erlaubt so eine Benutzeroberfläche sogar das Verändern einer Dialog-Box in einer installierten Anwendung.

Die Kommunikationsinfrastruktur die es ermöglicht, dass unabhängig voneinander entwickelte Komponenten frei kombiniert werden können, stellt die prinzipielle Schwierigkeit in der Konstruktion einer dokumentenbasierten Benutzeroberfläche dar. Komponenten verschiedener Hersteller können nicht "zusammengesteckt" werden ohne ein beidseitig eingehaltenes Kommunikations-Protokoll. Komponenten können auch nicht zur Laufzeit hinzugefügt oder entfernt werden ohne einen dynamischen Verbindungsmechanismus.

Diese Dissertation beschreibt das Design und die Implementierung einer Komponenten-Architektur und einer dokumentenbasierten Benutzeroberfläche für das Oberon-System. Das Gadget-System benutzt sogenannte "Gadgets"-Komponenten um Dokumente zu bilden und sie mit Funktionalität zu versehen. Gadgets reichen von einfachen Benutzeröberflächen-Elementen bis hin zu komplizierten Bausteinen und können interaktiv in aktiven Dokumenten zusammengesetzt werden. Eine wichtige Eigenschaft ist, dass unabhängig voneinander entwickelte Bausteine mit existierenden Komponenten ohne grossen Aufwand zusammengesetzt werden können. Dies ermöglicht dem Konstrukteur, schnell Applikationen aus einer Vielzahl wiederverwendbarer Komponenten zusammenzusetzen und dadurch den Entwicklungsprozess zu verkürzen.

Diese Arbeit beschreibt unter anderem wie Komponenten kommunizieren, wie ihre Kommunikationsprotokolle aussehen können, wie man Komponenten persistent macht, wie sie zu komplizierteren Komponenten zusammengesetzt werden können, wie man ihr Verhalten kundenspezifisch anpassen kann und wie man in solcherweise konstruierte Dokumente konsistent transportieren kann. Daneben werden auch Probleme im Zusammenhang mit System-Strukturierung, dynamischer Erweiterbarkeit, Komponenten-Authentizierung und System-Sicherheit diskutiert.