

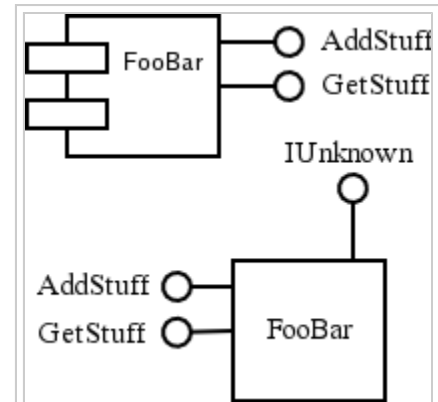
Software componentry

From Wikipedia, the free encyclopedia

Software componentry is a field of study within software engineering. It builds on prior theories of software objects, software architectures, software frameworks and software design patterns, and the extensive theory of object-oriented programming and the object-oriented design of all these. It claims that software components, like the idea of hardware components, used for example in telecommunications, can ultimately be made interchangeable and reliable.

Contents

- 1 Software component
- 2 History
- 3 Differences from object-oriented programming
- 4 Architecture
- 5 Technologies
- 6 References
- 7 See also
- 8 External links



Software component representations: above the representation used in UML, below the representation commonly used by Microsoft's COM objects. The "lollipops" sticking out from the components are their interfaces. Note the characteristic IUnknown interface of the COM component.

Software component

A **software component** is a system element offering a predefined service and able to communicate with other components. Clemens Szyperski and David Messerschmitt give the following five criteria for what a software component shall be to fulfill the definition:

- Multiple-use
- Non-context-specific
- Composable with other components
- Encapsulated i.e., non-investigable through its interfaces
- A unit of independent deployment and versioning

A simpler definition can be: A component is an object written to a specification. It does not matter what the specification is: COM, Java Beans, etc., as long as the object adheres to the specification. It is only by adhering to the specification that the object becomes a component and gains features like reusability and so forth.

Software components often take the form of objects or collections of objects (from object-oriented programming), in some binary or textual form, adhering to some interface description language (IDL) so that the component may exist autonomously from other components in a computer.

When a component is to be accessed or shared across execution contexts or network links, some form of serialization (also known as *marshalling*) is employed to turn the component or one of its interfaces into a

bitstream.

History



Image from Douglas McIlroy's historic lecture on software components at the NATO conference in Garmisch, Germany (1968).

The idea that software should be componentized, built from prefabricated *components*, was first published in Douglas McIlroy's address at the NATO conference on software engineering in Garmisch, Germany, 1968 titled *Mass Produced Software Components*. This conference set out to counter the so-called software crisis. His subsequent inclusion of pipes and filters into the Unix operating system was the first implementation of an infrastructure for this idea.

The modern concept of a software component was largely defined by Brad Cox of Stepstone, who called them *Software*

ICs and set out to create an infrastructure and market for these components by inventing the Objective-C programming language. (He summarizes this view in his book *Object-Oriented Programming - An Evolutionary Approach* 1986.) Cox's attempt did not succeed because of the most obvious, yet fundamental, difference between silicon and software ICs. The former are made of atoms, so it is possible to buy and sell them through scarcity-based economics. The latter are made of bits which don't obey the same laws, which undercuts the incentive to provide them.

IBM lead the path with their System Object Model, SOM in the early 1990s. Some claim that Microsoft paved the way for actual deployment of component software with OLE and COM. Today, several successful software component models exist.

Differences from object-oriented programming

The idea in object-oriented programming (OOP) is that software should be written according to a mental model of the actual or imagined objects it represents. OOP and the related disciplines of object-oriented design and object-oriented analysis focus on modelling real-world interactions and attempting to create 'verbs' and 'nouns' which can be used in intuitive ways, ideally by end users as well as by programmers coding for those end users.

Software componentry, by contrast, makes no such assumptions, and instead states that software should be developed by gluing prefabricated components together much like in the field of electronics or mechanics. It accepts that the definitions of useful components, unlike objects, can be counter-intuitive. In general it discourages anthropomorphism and naming, and is far more pessimistic about the potential for end user programming. Some peers will even talk of software components in terms of a new programming paradigm: *component-oriented programming*.

Some argue that this distinction was made by earlier computer scientists, with Donald Knuth's theory of "literate programming" optimistically assuming there was convergence between intuitive and formal models, and Edsger Dijkstra's theory in the article *The Cruelty of Really Teaching Computer Science*, which stated that programming was simply, and only, a branch of mathematics.

In both forms, this notion has led to many academic debates about the pros and cons of the two approaches and possible strategies for uniting the two. Some consider them not really competitors, but only descriptions of the same problem from two different points of view.

It takes significant effort and awareness to write a software component that is effectively reusable. The component needs:

- to be fully documented;
- more thorough testing;
- robust input validity checking;
- to pass back useful error messages as appropriate;
- to be built with an awareness that it *will* be put to unforeseen uses
- a mechanism for compensating developers who invest the (substantial) effort implied above.

Architecture

A computer running several software components is called an application server. Using this combination of application servers and software components is usually called distributed computing. The usual real-world application of this is in financial applications or business software.

Technologies

- Pipes and Filters
 - Unix operating system
- Component-oriented programming
 - Fractal component model[1] (<http://fractal.objectweb.org/>) from ObjectWeb (<http://www.objectweb.org/>)
 - Visual Basic Extensions, OCX/ActiveX/COM and DCOM from Microsoft
 - XPCOM from Mozilla Foundation
 - VCL and CLX from Borland and similar free LCL library.
 - Enterprise Java Beans from Sun Microsystems
 - UNO from the OpenOffice.org office suite
 - Eiffel programming language
 - Oberon programming language and BlackBox
 - Z++ programming language
 - Bundles as defined by the OSGi Service Platform
 - NexWave Software Infrastructure (NSI)
 - The `System.ComponentModel` namespace in Microsoft .NET
- Compound document technologies
 - Bonobo (a part of GNOME)
 - Object linking and embedding (OLE)
 - OpenDoc
 - Fresco

- Business object technologies
 - Newi
- Distributed computing software components
 - 9P distributed protocol developed for Plan 9, and used by Inferno and other systems.
 - CORBA and the CORBA Component Model from the Object Management Group
 - D-BUS from the freedesktop.org organization
 - DCOM and later versions of COM (and COM+) from Microsoft
 - DCOP from KDE
 - DSOM and SOM from IBM (now scrapped)
 - Java EE from Sun
 - .NET Remoting from Microsoft
 - Z++ programming language from ZHMicro (<http://www.zhmicro.com/>)
 - Web Services
 - REST
 - Universal Network Objects (UNO) from OpenOffice.org
- Interface description languages
 - XML-RPC, the predecessor of SOAP
 - SOAP IDL from W3C
 - WDDX
 - Part of both COM and CORBA
 - Open Service Interface Definitions
 - Platform-Independent Component Modeling Language

References

- Brad J. Cox, Andrew J. Novobilski: *Object-Oriented Programming: An Evolutionary Approach*. 2nd ed. Addison-Wesley, Reading 1991 ISBN 0-201-54834-8
- Bertrand Meyer: *Object-Oriented Software Construction*. 2nd ed. Prentice Hall, 1997.
- Clemens Szyperski: *Component Software: Beyond Object-Oriented Programming*. 2nd ed. Addison-Wesley Professional, Boston 2002 ISBN 0-201-74572-0
- George T. Heineman, William T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley Professional, Reading 2001 ISBN 0-201-70485-4

See also

- Business logic
- Web Service
- Third party software component

External links

- *Mass Produced Software Components* by M. Douglas McIlroy (<http://www.cs.dartmouth.edu/~doug/components.txt>)
- NATO Science Committee Software Engineering Conference in Garmisch (<http://homepages.cs.ncl.ac.uk/brian.randell/NATO/>) - reports (PDF)
- *Planning the Software Industrial Revolution* (<http://virtualschool.edu/cox/pub/PSIR>) The history of manufacturing vs software compared.
- *The independence of notion of component-orientation* (<http://distributed-software.blogspot.com/2006/06/independence-of-component-oriented.html>).
- *Cox's feasibility demonstration* (<http://virtualschool.edu/mybank>) of a usage-based mechanism for incentivizing component producers.
- comprehensive list of Component Systems (<http://xplc.sourceforge.net/doc/others.php>)
- Article "Why Software Reuse has Failed and How to Make It Work for You" (<http://www.dre.vanderbilt.edu/~schmidt/reuse-lessons.html>)" by Douglas C. Schmidt

Retrieved from "http://en.wikipedia.org/wiki/Software_componentry"

Categories: Object-oriented programming | Software architecture | Components | Programming paradigms

-
- This page was last modified 12:00, 21 October 2006.
 - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.)
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc.