Component-based Software

Introduction and overview

Christoph Kessler

Recommended Reading


Motivation for Component Based Development

- Managing system complexity: Divide-and-conquer (Alexander the Great)
- Well known in other disciplines
  - Mechanical engineering (e.g., German DIN 2221; IEEE standards)
  - Electrical engineering
  - Architecture
  - Computer architecture
- Outsourcing to component producers
- Goal: Reuse of partial solutions
- Easy configurability of the systems
  - Variants, versions, product families

Mass-produced Software Components

- Garmisch 1968, NATO conference on software engineering
- McIlroy:
  - Every ripe industry is based on components, since these allow to manage large systems
  - Components should be produced in masses and composed to systems afterwards

Mass-produced Software Components

- Later McIlroy was with Bell Labs ...
  - ... and invented pipes, diff, join, echo (UNIX).
  - Pipes are still today the most employed component system!
- Where are we today?
Real Component Systems

- Lego
- Square stones
- Building plans
- IC's
- Hardware bus
- How do they differ from software?

Definition of “Component”

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

A software component can be deployed independently and is subject to composition by third parties.”


More Definitions of “Component”

MetaGroup (OpenDoc):
“Software components are defined as prefabricated, pretested, self-contained, reusable software modules - that perform specific functions.”

- Grady Booch

Sametinger:
“Reusable software components are self-contained, clearly identifiable pieces that describe and/or perform specific functions, have clear interfaces, appropriate documentation, and a defined reuse status.”

More Definitions of “Component” (cont.)

Heineman / Councill [Ch.1]:
“A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

A component model defines specific interaction and composition standards.

Composition is the combination of two or more software components yielding a new component behavior at a different level of abstraction ... [which is] determined by the components being combined and the way how they are combined.”

Component as unit of composition

U. Assmann (2003):
- A component is a container with
  - variation points
  - extension points
  - that are adapted during composition
- A component is a reusable unit for composition
- A component underlies a component model
  - abstraction level
  - composition time (static or runtime?)

Are Objects Components??

Szyperski [CS 4.1]: No!
- An object is a unit of instantiation.
  - It has a unique identity.
  - It may have state, and this can be (externally) observed
  - It encapsulates its state and behavior.
- Components are rather prototypes / blueprints / plans from which (stateful) objects can be instantiated
  - e.g., a function definition, type definition, class or set of classes
- No (externally observable) state
  - Only one copy required per context (e.g., process)
  - Unit of independent deployment
  - Unit of third-party composition
**Component Systems** *(Component Platforms)*

- We call a technology in which component-based systems can be produced a component system or component platform.

- A component system has

**Software Composition Systems**

- A composition system has

**Issues in Component/Composition Systems**

- **Component Model**
  - How do components look like?
  - Secrets? (Location, lifetime, language, platform, …)?
  - Binding points, binding time?
  - Interfaces, contracts, substitutability?
  - Parameterizability? Adaptability? Extensibility?
  - Standardization of execution environment, services?

- **Composition Technique**
  - How are components glued together, composed, merged, applied?
  - Composition time (Compile- / Link- / Deployment- / Connection- / Run-time …)

- **Composition Language**
  - How are compositions of large systems described and managed?

**The Ladder of Component and Composition Systems**

**The Essence of the 60s-90s:** LEGO Software

- Procedural systems
- Modular systems
- Object-oriented technology
- Component-based programming with COTS (Components-off-the-shelf) systems
  - CORBA, EJB, DCOM, COM+, .NET
- Software architecture description languages

**Blackbox Composition**

- Component-based applications
Procedural Systems

- Fortran, Algol, Pascal, C, ...
- The procedure is the component
- The activation record the instantiation
- Component model is supported by almost all processors directly
  - JumpSubroutine instruction
  - Return instruction

Procedures as Composition System

Component Model
- Content: binary code with symbols
- Binding points: linker symbols
- Procedures (with parameters) and global variables
- Connection by linking object files
- Program transformation on object files
- Composition time: link-time, static

Composition Technique
- Program transformation on object files
- Composition time: link-time, static

Composition Language
- Program transformation on object files
- Composition time: link-time, static

Modules

- Implementation of a module hidden behind a functional interface
- Static binding of functional interfaces to each other
- Concept has penetrated almost all programming languages (Modula, Ada, Java, C++, Standard ML, C#)

Modules as Composition System

Component Model
- Content: groups of procedures
- Binding points: linker symbols
- Procedures (with parameters) and global variables
- Program transformation on object files
- Composition time: link-time, static

Composition Technique
- Program transformation on object files
- Composition time: link-time, static

Composition Language
- Program transformation on object files
- Composition time: link-time, static

UNIX Filters and Pipes

- UNIX shells style still offers the most used component paradigm:
  - Communication with byte streams via standard I/O ports
  - Parsing and linearizing the objects
  - Extremely flexible, simple
Unix Filters and Pipes as Composition System

Component Model
Content: unknown (due to parsing), externally bytes
Binding points: stdin/output
Secrets: distribution, parallelism

Composition Technique
Adaptation: filter around other components
Filter languages such as sed, awk, perl
Binding time: static

C, shell, tcl/tk, python...
Build management language makefile
Version management with sccs rcs cvs

Object-Oriented Systems

Components: classes
Objects are instances of classes (modules) with unique identity
Objects have runtime state
Late binding of calls by search/dispatch at runtime

Object-Orientation as Composition System

Component Model
Content: binary files, objects (code and data)
Binding points: static (monomorphic) and polymorphic (dynamically dispatched) calls

Composition Technique
Adaptation by inheritance or delegation
Extensibility by subclassing

Commercial Component Systems

CORBA / COM / .NET / EJB
Although different on the first sight, turn out to be rather similar

CORBA
- Language independent, location/distribution transparent
- Interface definition language IDL
- Source code or binary

(D)COM, ActiveX
- Microsoft’s model is similar to CORBA. Proprietary
- (D)COM is a binary standard
Java Beans
- Java only: source code / bytecode-based
- Event-based, transparent distribution by remote method invocation (RMI – includes Java Object Serialization)

DOT-NET
- Language independent, distribution transparent
- NO interface definition language IDL (at least for C#)
- source code or bytecode MSIL
- Common Language Runtime CLR

CORBA/DCOM/JavaBeans/...:
Components Off-The-Shelf (COTS)
- Component Model
  - Content: binary components
  - Binding points are standardized
  - Described by IDL, standard interfaces
  - Standard interfaces
  - Described by IDL, standard interfaces
- Composition Technique
  - Adaptation for distributed systems (marshalling) and mixed-language systems (IDL)
  - Dynamic call in CORBA
- Composition Language
  - VisualBasic for COM

Web Services
- Binding procedure is interpreted, not compiled
- More flexible:
  - When interface changes, no recompilation and rebinding
  - Ubiquitous http protocol – independent of a specific ORB

Web Services as Composition System
- Component Model
  - Content: not important
  - Binding points are described by XML
- Composition Technique
  - Adaptation for distributed systems (marshalling) and mixed-language systems
  - Glue: WSDL, SOAP, HTTP
- Composition Language
  - BPEL

Component Model in Software Architecture Systems
- Port
  - abstract interface points
  - (as in Linda)
  - in(data), out(data)
- Components may be nested
- Connectors
  - special communication components
  - Abstract from technology
    - (e.g. component system)
  - Specify connectivity
    - (topology, system architecture)
Software Architecture Systems

- Unicon, ACME, Darwin, ...
  - feature an Architecture Description Language (ADL)
- Split an application into two concerns:
  - Application-specific part (encapsulated in components)
  - Architecture and communication (in connectors defined in architecture description, written in ADL)
→ Better reuse since both dimensions can be varied independently

Architecture / Communication can be varied independently of components

- Reuse of components and architectures is fundamentally improved
- High-level system analysis, verification, testing

ACME Studio

Software Architecture Systems as Composition Systems

- Component Model
- Composition Technique
- Source or binary components
- Binding points: ports
- Adaptation and glue code by connectors
- Scaling by exchange of connectors
- Architectural language (ADL)
- Composition Language

The Essence of Blackbox Composition

- Blackbox composition supports variability and adaptation
  - but not extensibility

The Ladder of Component and Composition Systems
Graybox Component Models
Component integration
- Aspect oriented programming
- View-based composition

Aspects in Architecture

Aspects in Software

Aspects in Software

Aspect Systems
- Aspect languages
  - Every aspect in a separate language
  - Domain specific
  - Weaver must be built (is a compiler, much effort)
- Script-based Weavers
  - The weaver interprets a specific script or aspect program
  - This introduces the aspect into the core

Aspect Weavers Distribute Advice
Components over Core Components
- Aspects are crosscutting
- Hence, aspect functionality must be distributed over the core

Aspect Systems as Composition Systems
Component Model
- Core- and aspect components
- Aspects are relative and crosscutting
- Binding points: join points
Composition Technique
- Adaptation and glue code by weaving
- Weaving is distribution of code snippets
Weaving language
  - e.g. Aspect-J
Composition Language
Composition Systems
with composition operators and expressions

- Hyperspace Programming [Ossher et al., IBM]
- Piccola [Nierstrasz, et al., Berne]
- Metaclass composition [Forman/Danforth, Cointe]
- Invasive software composition [ABmann 2003]
- Formal calculi
  - Lambda-N calculus [Dami]
  - Pi-L calculus [Lumpe]

Composition Systems
with composition operators and expressions

Invasive Composition of Components

- Extension can be used for inheritance (mixins)
  A mixin is a class (i.e., a set of features) by which a superclass can be extended to derive a subclass. The mixin class itself is final, i.e., cannot be subclassed.
- Mixin-based inheritance:
  - copy first superclass
  - extend with fragments of second superclass (mixin)

Composers can be used for inheritance

- Complex composers distribute aspect fragments over core fragments
- Distributors extend the core
- Distributors are more complex operators, defined from basic ones

Composition Languages

- Composition languages describe the structure and build process of the system in-the-large ("programming in the large")
- Composition programs combine the basic composition operations of the composition language
- Composition languages can look quite different:
  - Imperative: e.g., Java+library (in COMPOST)
  - Declarative: e.g., ADL, Aspect-J, Makefiles, C++ templates
- Enables us to describe large systems

Composition program size 1
System size 10
Conclusions for Composition Systems

- Components have a composition interface
  - Composition interface is different from functional interface
    - Marks possible places for code injection in components
  - The composition is running usually before the execution of the system
    - Usually, at/before compile time or deployment time
- System composition becomes a new step in system build

System composition (System generation)
System compilation
System deployment
System execution

Summary: Component-based Systems

- ... are produced by component systems or composition systems
- ... support a component model
- Blackbox composition supports variability and adaptation
- Greybox composition also supports extensibility