Architectures of Distributed Systems 2011/2012

Component Based Systems

Johan Lukkien

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking
TU/e Informatica, System Architecture and Networking

Goals

- Students have an overview of motivation and concepts of Component-Based Software Engineering
- Student have an understanding of how a CBSE system works in practice

Agenda

- Motivation
- Component models and frameworks
- Composition
- Examples

Design *process* – four elements

- (Domain) analysis
  - increase knowledge, make models
    - use cases, based on stakeholder viewpoints
  - feedback to stakeholders: validation of requirements ("Do we solve the right problem?")

- Apply strategies
  - hierarchical decomposition:
    - top-down (factorization): *specify* advanced building blocks (decompose functional specification, and derive extra-functional properties for the parts)
    - bottom up: *design* advanced building blocks
  - apply patterns, styles
    - pattern, style: *coherent set of design decisions*
  - generate alternatives

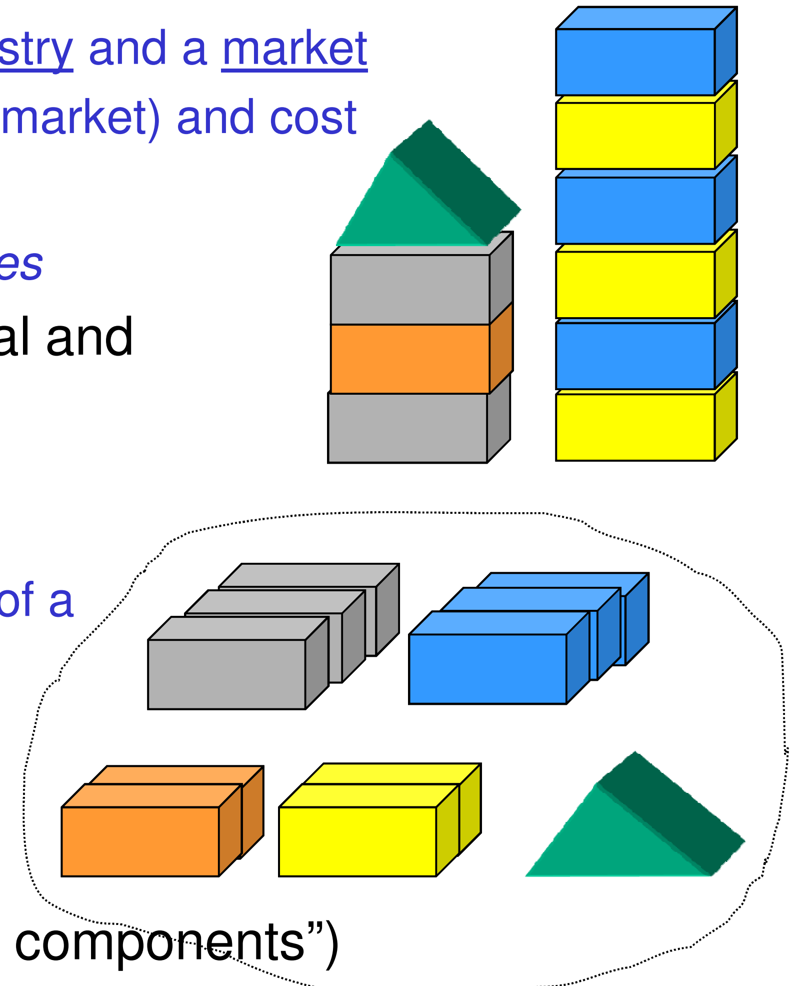
- Synthesis
  - evaluate and choose alternatives, combine partial solutions

- Verification
  - is the system according to specification? ("Did we solve the problem right?")

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

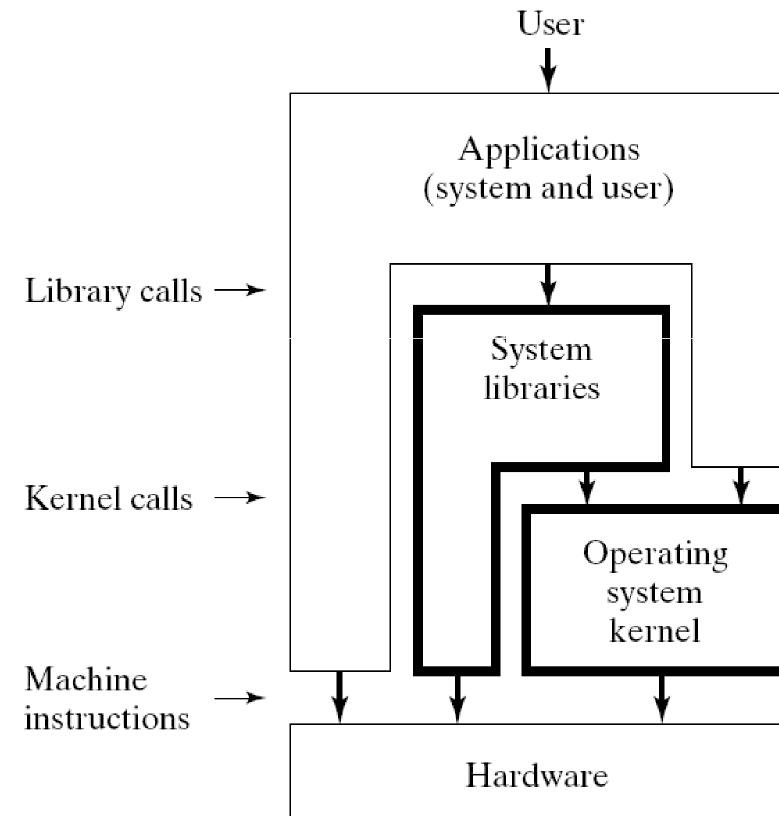
Motivation for Software Components

- Separate *application development* from *component development*
  - manufacturing rather than engineering
  - bring standardization, facilitate a new industry and a market
  - bring re-use: improve productivity (time to market) and cost
- Modularity
  - decomposition, and localizing *dependencies*
- Flexibility – facilitate *change:* easy removal and addition of functionality
  - the methods and processes for this are explicitly defined
  - facilitates *product lines,* different versions of a product
- Similar to other engineering disciplines
  - obtain more *predictable* results
- Early ideas: Douglas McIllroy, NATO conference ’68 (“Mass produced software components”)



Sample system components

- Entire computer systems
  - put together into a distributed system
- Components on a motherboard
- CPU platforms
  - ISA, interfaces to devices, ...
- Operating Systems
  - OS-API, process model, file model, GUI, ...
- Source code libraries
  - standard template library
- Static, compiled libraries
  - Math functions library, file io, concurrency support functions
- Dynamic libraries, DLLs
- Executable programs



Picture by A.S. Tanenbaum

Component-based software engineering

- According to SEI in CMU/SEI-2000-TR-008:
  - Component-based software engineering is concerned with the rapid assembly of systems from components where
    - components and frameworks have certified properties;
    - and these certified properties provide the basis for predicting the properties of systems built from components
- What is a component?
  - Szyperski: '97: A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only....
    - no dependencies other than through interfaces
  - ...A software component is independently deployable and subject to composition by third parties.
    - no partial deployment
    - can a component be defined at source code level?
- Components have two aspects
  - they implement functionality
  - they represent an abstraction, a style

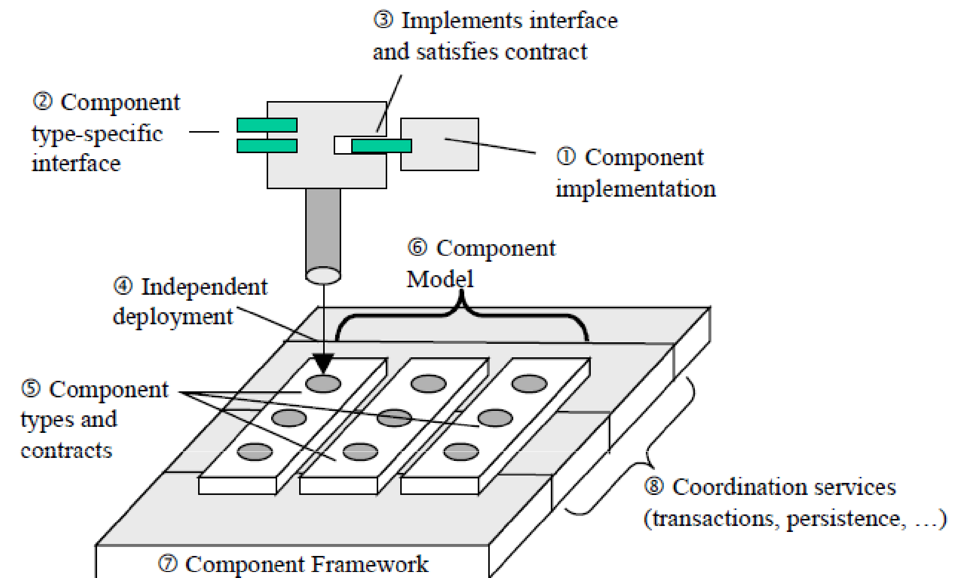
Agenda

- Motivation
- Component models and frameworks
- Composition
- Examples

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

CBSE: Component Model

- Component *model*
  - defines what is, and what is not a component
  - specifies how to use metadata
  - defines a series of concepts
    - coordination, composition
    - quality attributes
    - typing: interfaces
      - set of standard interfaces
    - binding and instantiation
    - interaction style



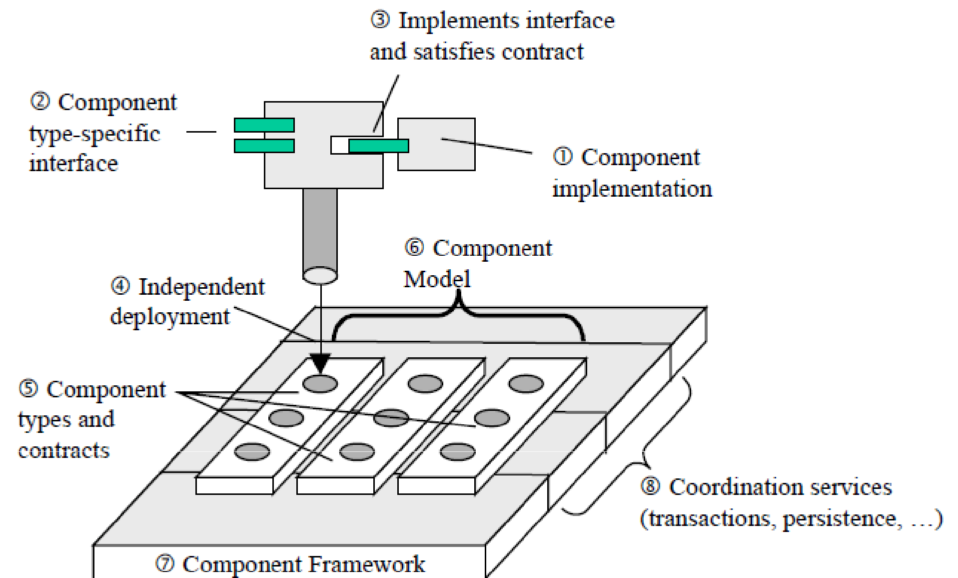
From: *Technical Concepts of Component-Based Software Engineering*, CMU/SEI-2000-TR-008

- A component model specifies the standards and conventions that enable composition of independently developed components

- A component conforms to this model

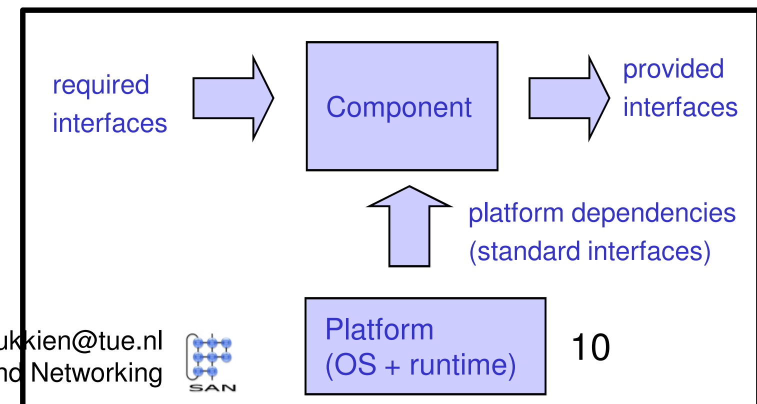
CBSE: Component Framework

- Component *framework*
  - a framework to work with a certain component model, as a style
  - defines *application* life cycle
  - platform, run-time services, component "docking"
  - process model (of the running system)



From: *Technical Concepts of Component-Based Software Engineering*, CMU/SEI-2000-TR-008

- A component has three obvious dependencies:
  - to the platform, using standard interfaces
  - to other components: provided/required
    - both can be managed through the run-time



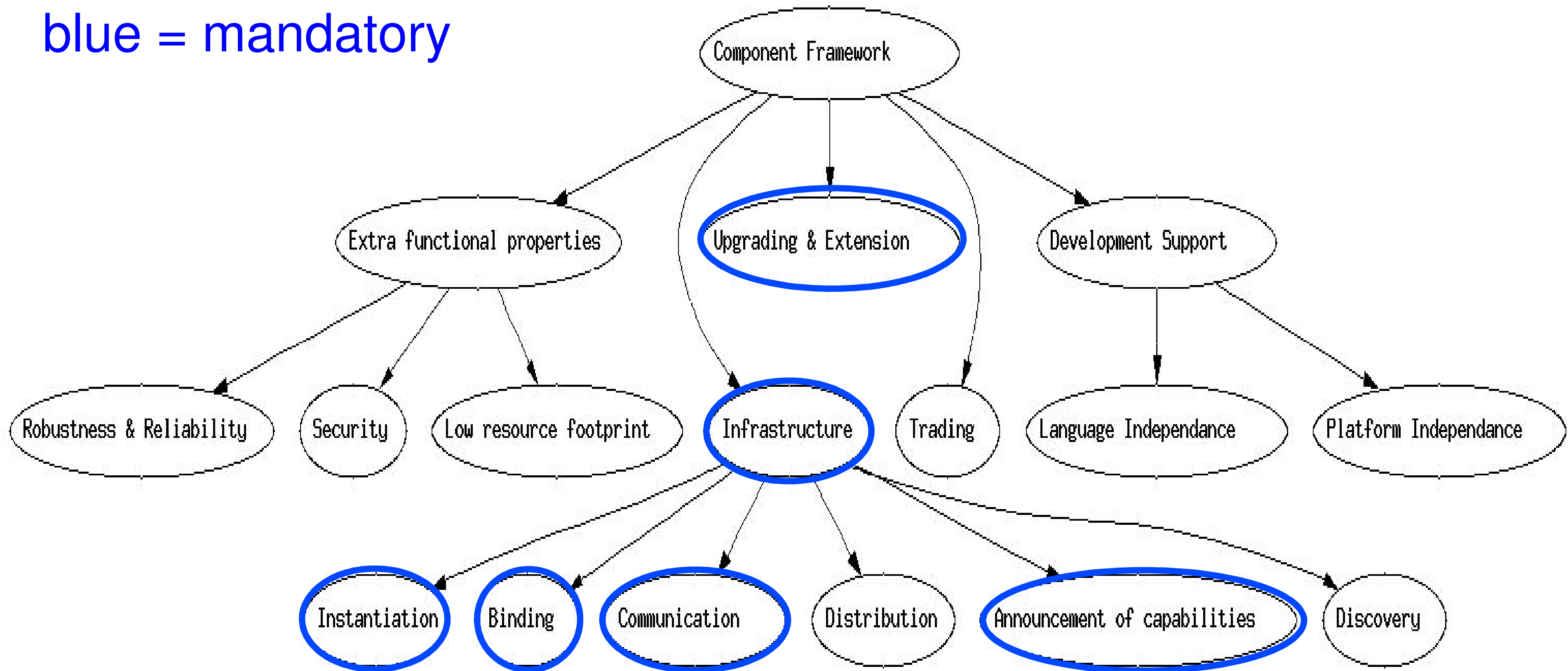
Remember: Framework

- A framework consists of
  - a ‘static’ part
    - programming model, data model
      - libraries
    - life cycle model
    - methods or tooling for development
  - a ‘dynamic’ part
    - a run-time system, or platform
      - entirely separate entity or a library
    - a set of services
      - provided by the platform
      - e.g. binding, installation
    - a process model

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Component framework services

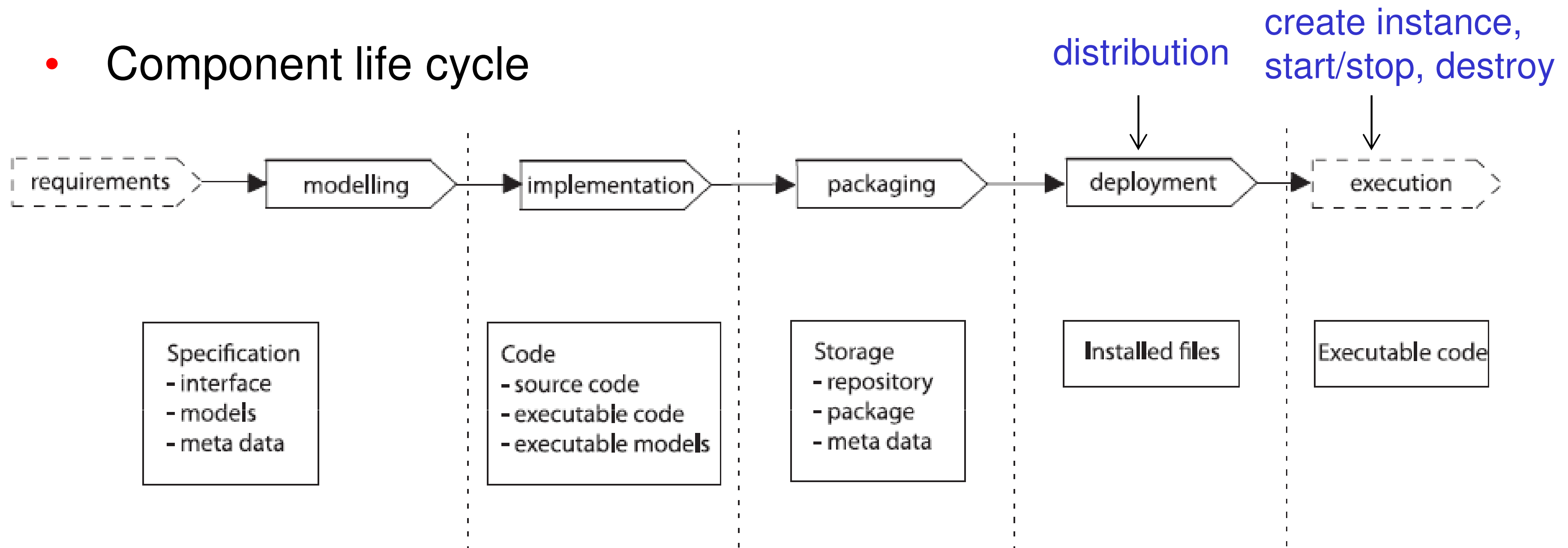
blue = mandatory



Picture by Johan Muskens

Component life cycle

- Component life cycle



A Classification Framework for Software Component Models, Ivica Crnkovic, Severine Sentilles, Aneta Vulgarakis, and Michel R.V. Chaudron IEEE Transactions on Software Engineering, Vol. 37, No. 5, Sept 2011.

- Notice that a component can be regarded as *a set of models*
  - source code, binary code, performance model, simulation model, ...
  - .....different aspects of what the component is
  - this is the perspective of the *ROBOCOP* component model

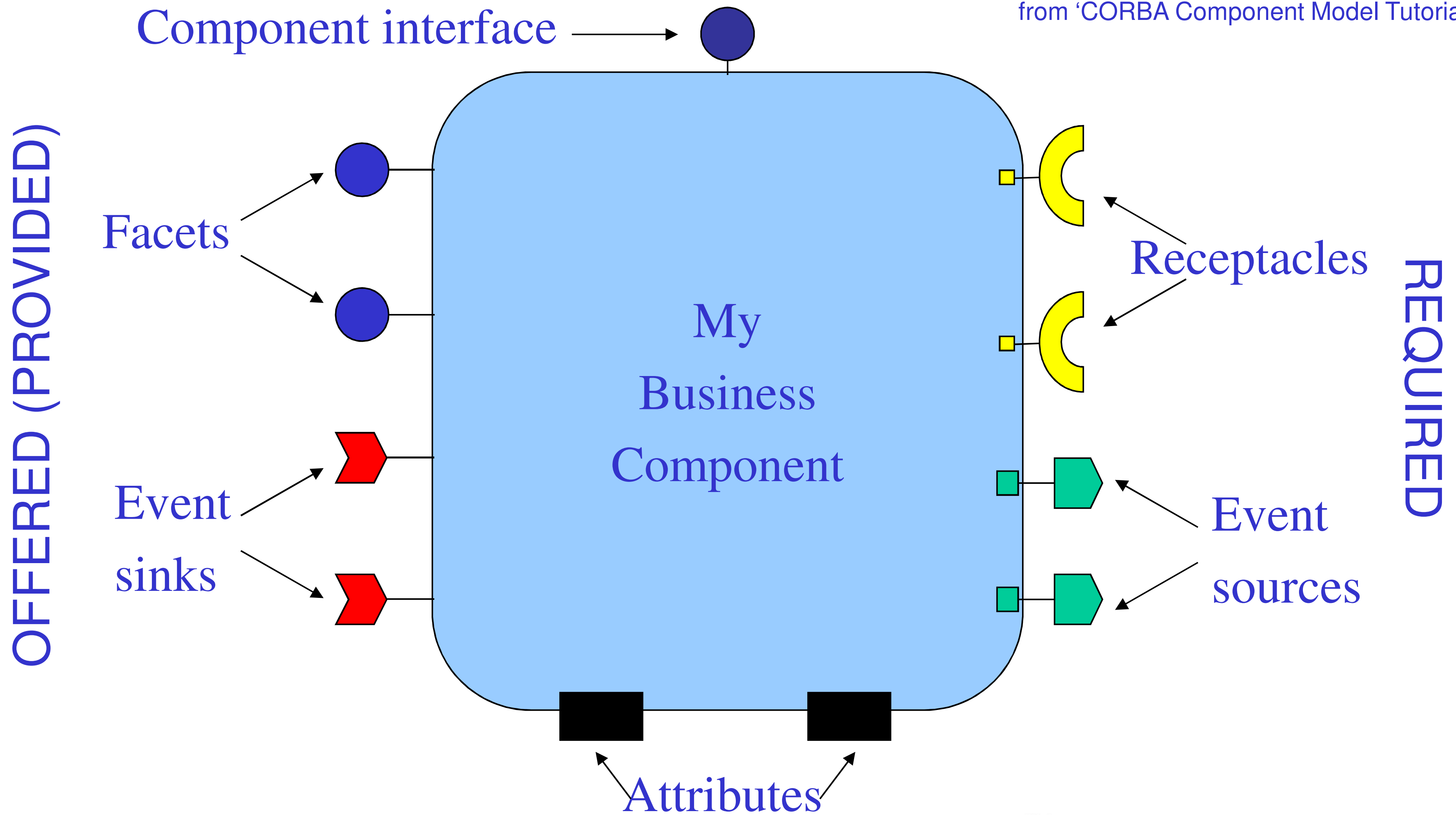
Example: executable

- Component:
  - executable program
- Requirements/modelling:
  - interfaces, and behavior – its effects on:
    - input, output
    - GUI
    - file system, network
- Implementation
  - source, executable
  - possibly: a model that is executable (simulation)
- Packaging
  - (machine readable, loadable) description, e.g. ELF file
- Deployment / distribution
  - put executable file at certain location (e.g. /usr/bin in Linux systems)

Example: Corba Component Model

- Component model:
  - generalization of an object – independent of language, OS, location, protocols
- Requirements/modeling
  - model of requires/provides interfaces (“ports”), and interaction styles (RMI , events)

A CORBA Component



Example: Corba Component Model

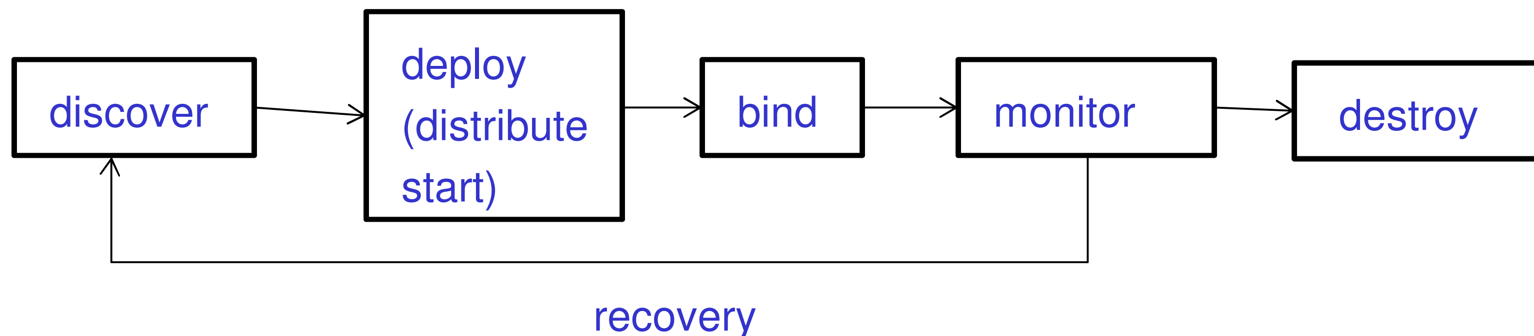
- Component model:
  - generalization of an object – independent of language, OS, location, protocols
- Requirements/modeling
  - model of requires/provides interfaces (“ports”), and interaction styles (RMI , events)
- Implementation
  - using *interface-definition files* and compiler support to implement ‘standard’ interfaces (e.g. to work with a framework) and to specify specific interfaces
  - instances managed at runtime by a *home*
  - a *container* acts as a process context
- Packaging
  - zip, containing code and description
- Distribution / deployment
  - put container on object server / start ‘container’ through its home and register with ORB (object request broker)

Application life cycle

- An application is a set of connected and cooperating components
  - no ‘dangling’ interfaces
- Connecting components is called *composition.*
  - *Vertical :* a composite (‘partial application’) is again a component,
    - *aggregation* (bring internal interfaces outside) and *delegation* (map external interfaces to internal)
  - *Horizontal :* simply connect interfaces without further rules for composite
- Phases in composition:
  - discover (lookup) components, by interface
  - deploy
    - distribute: components to machines
    - instantiate (start) components
  - bind interfaces
    - first party: control resides in either of the bound parties
    - third party: binding control lies outside the bound parties – requirement for CBSE
  - (run-time management: monitoring, (re-)allocation, fault recovery, destroy)

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Requirements on model and framework



- The component *framework* must include the services for composition
  - discovery / searching, perhaps a registry, or a *repository*
  - deployment: component allocation and instantiation, further management
    - typically, a framework component called a *dock*
    - that implements *policies* for this part of the lifecycle, e.g. when to create/destroy, security
  - possibly: resource management, monitoring
- The component *model* must include the interfaces to perform the component-related tasks of composition
  - third party binding, perhaps monitoring, start/stop

Example

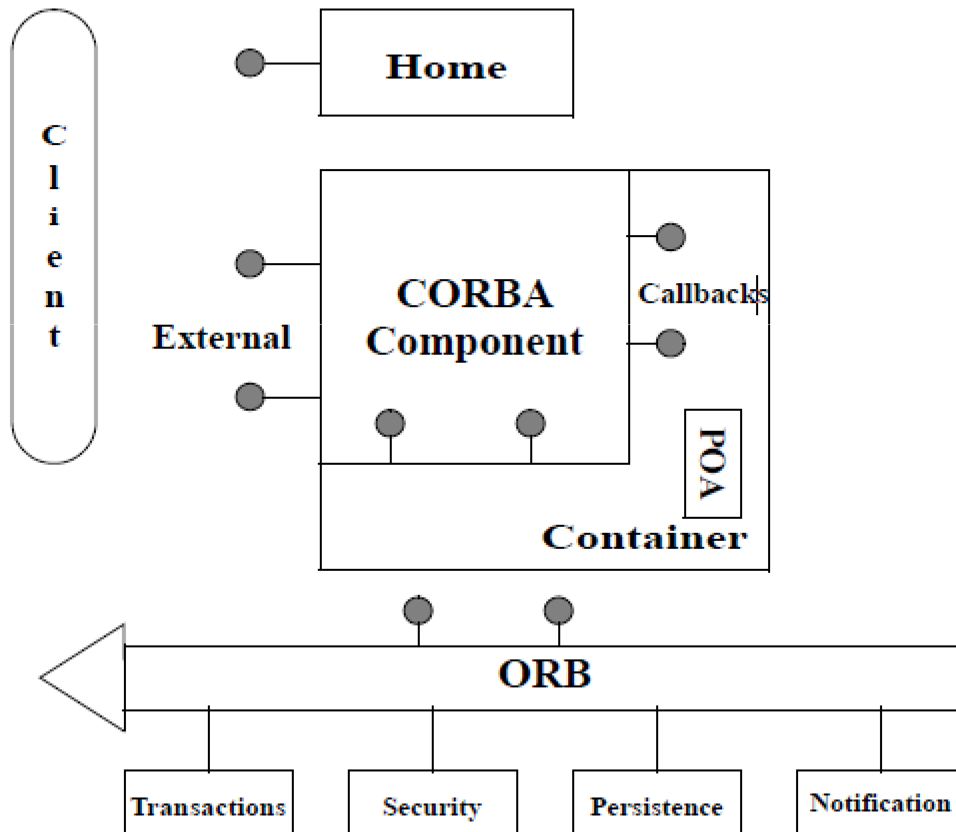
- Component:
  - executable program
- Composition
  - discover: OS finds location of command based on name
  - distribute: depends on whether the OS manages several processors
  - start: regular program start
  - bind: e.g. Unix pipes connect output of one command to input of next
    - cat file | sort | unique | wc
- Note: this was not *designed* to be a component framework

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Example

- Component
  - Corba Component
- Composition
  - register / discover: registration with ORB / search functions in ORB
  - distribute: not in the framework
  - start: control lies with an ‘Assembler’ tool that takes a descriptor and activates home and containers on the relevant nodes (after discovery)
  - bind: through Corba’s ORB mechanism

Runtime



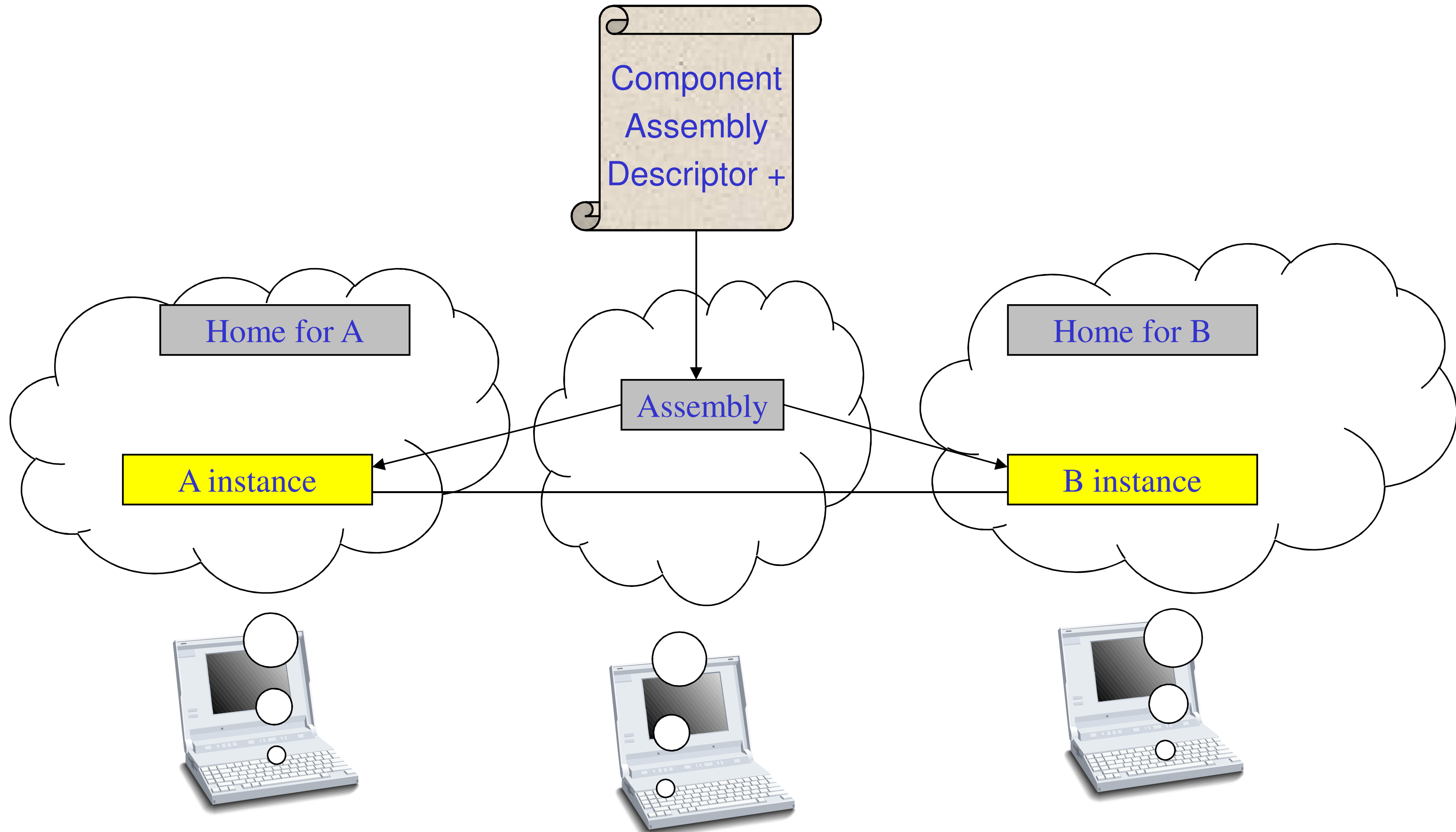
from Corba Component Model Specification 4.0(OMG)

ORB: Object Request Broker

POA: Portable Object Adapter

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Deployment Scenario: Component Configuration



Evaluation Existing Component Frameworks

Component Framework	COM	DCOM	EJB	.NET	CORBA	Koala	Robocop	PECOS	Autocomp
Infrastructure									
Instantiation	+	+	+	+	+	+	+	+	+
Binding	+	+	+	+	+	+	+	+	+
Communication	+	+	+	+	+	+	+	+	+
Distribution		+	+	+	+				
Announcement of Cap.	+	+	+	+	+	+	+	+	+
Discovery		+	+	+	+		+		
Extra Functional Properties									
Robust & Reliable operation						+	+		
Security			+/-	+/-					
Low resources	+	+				+	+	+	+
Upgrading and Extension									
Design time	+	+	+	+	+	+	+	+	+
Compile time	+	+	+	+	+	+	+	+	+
Run time	+	+	+	+	+		+		
Development Support									
Language Independence	+	+		+	+/-	+/-	+/-		
Platform independence			+		+				
Analysis techniques							+		+/-
Trading									
Trading							+		

Table via Johan Muskens

not really available

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Component and service

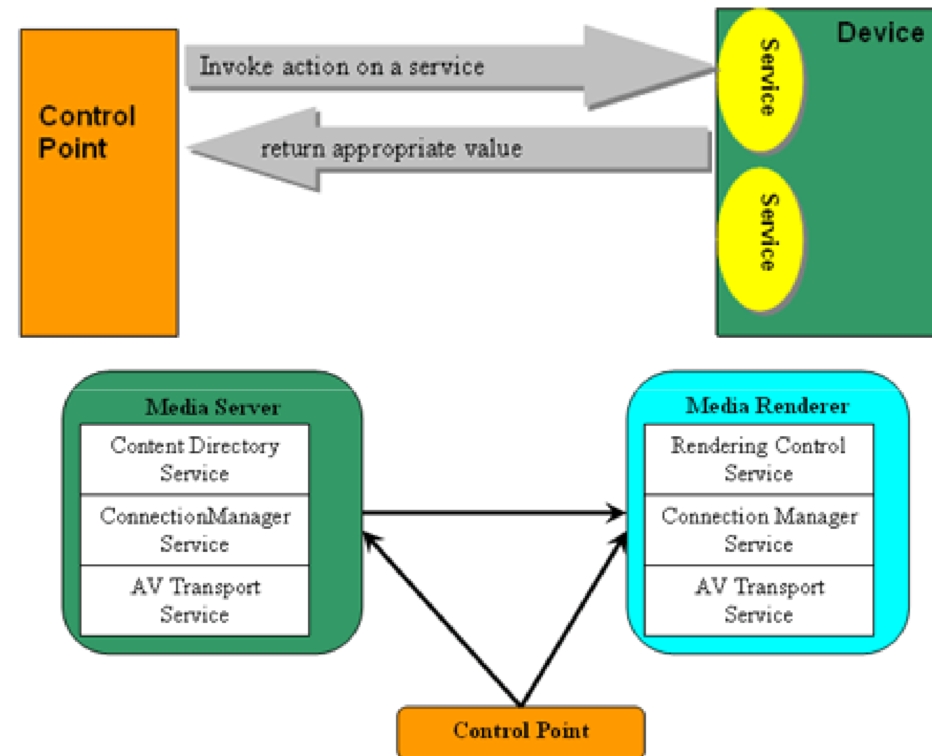
- In networked systems, it is fruitful to discriminate between *component* and *service*
  - separation of functionality, and how it is implemented
- An application is then a set of connected *services*
  - steps remain the same:
    - discover components -> service discovery
    - deploy: allocate and start service
    - bind: connect service interfaces
- This results in the same style for application composition, *without* requiring a component-based realization.
  - it is like the SOA style; however, SOA requires also independence of services

Example: UPnP

- Component model
  - none
- Composition
  - discover: SSDP (immediate, local discovery protocol)
  - allocate: fixed;
    - actually, the service is delivered by the device; this is defined by the manufacturer or owner
  - start: fixed
    - the service is active upon starting the device
  - bind: calling upon the binding interface of the services

Example: UPnP services (recap)

- UPnP services are accessed using a REST-like style
  - although there is some debate on this, see e.g. the paper by *Newmarch*
- Service implementation is entirely hidden, as are the OS and the implementation language



Interactions: control points call actions on services (first party, top) or establish connections (binding) between services (third party, bottom)

Agenda

- Motivation
- Component models and frameworks
- Composition
- Examples

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Certified properties

- Extra-functional properties of a system are ‘emergent’ properties
  - a property of the system as a whole....
  - ....determined by properties of many or all of its parts
- For component based systems:
  - it must be possible to determine the properties of an assembly from the properties of the components
    - (actually, this is the goal, as for other engineering disciplines)
    - then these properties have to be specified as a *model*....
      - as ‘metadata’ of interfaces, or of a component as a whole
      - determined using some ‘testbank’ or modeling method
    - examples:
      - resource model: amount of processing per call, amount of memory
      - information leaking, reliability/availability
  - methods are required to manage trust in such properties
    - certification (by an authority), monitoring (by the platform)

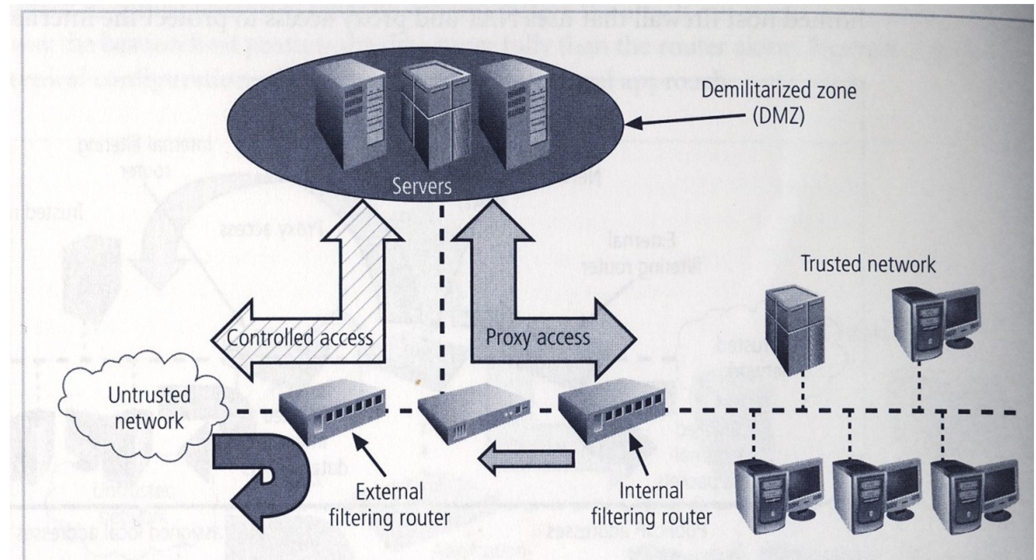
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Reasoning about compositions

- Certainly non-trivial!
- Examples:
  - Performance: response time to an event (say, a keypress) when this processing is executed on a platform in competition with other computations
    - model:
      - set of real-time tasks with deadlines and worst-case computation times
      - scheduling policy, e.g. Earliest Deadline First
      - determine response times from these
  - Security: protection of a website
    - how to decompose into component properties?
    - See example on next slide
  - Availability of a system
    - model
      - assume independence of components, availability is a_i
      - availability is Πa_i

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Example: screened subnet architecture



- Protection is achieved:
  - by combining components, with particular security properties
    - packet filtering firewalls
    - secure communication
  - by having behavior policies
- Network between the two filtering firewalls: a *Demilitarized Zone*
- Policies
  - Servers in the DMZ operate as *reverse proxy* (i.e., proxy at server side)
  - Traffic to the internal network is only allowed from the DMZ
  - Internal servers cannot initiate traffic
  - All traffic is routed (as opposed to flooded)
- Functions of the DMZ servers include access control (e.g. password checking, secure communication) and request verification

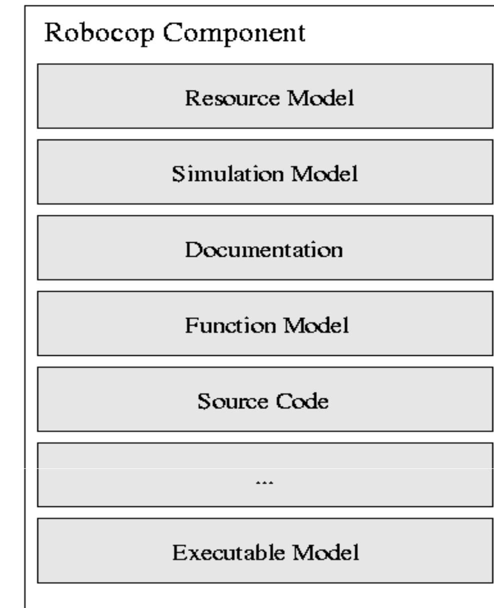
Agenda

- Motivation
- Component models and frameworks
- Composition
- Examples

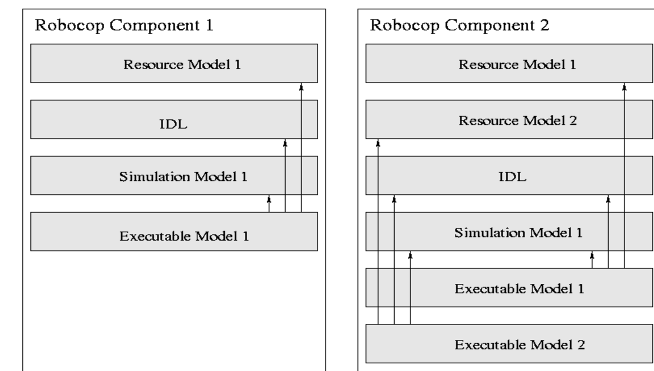
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Example: Robocop component model

- A Robocop component is a set of related models
  - similar to architectural views: different aspects of the same underlying entity
  - models can be left out and added
    - not always all models are required
  - example relations:
    - source code S and executable E are related by compiler C
    - executable E on platform P has resource model R



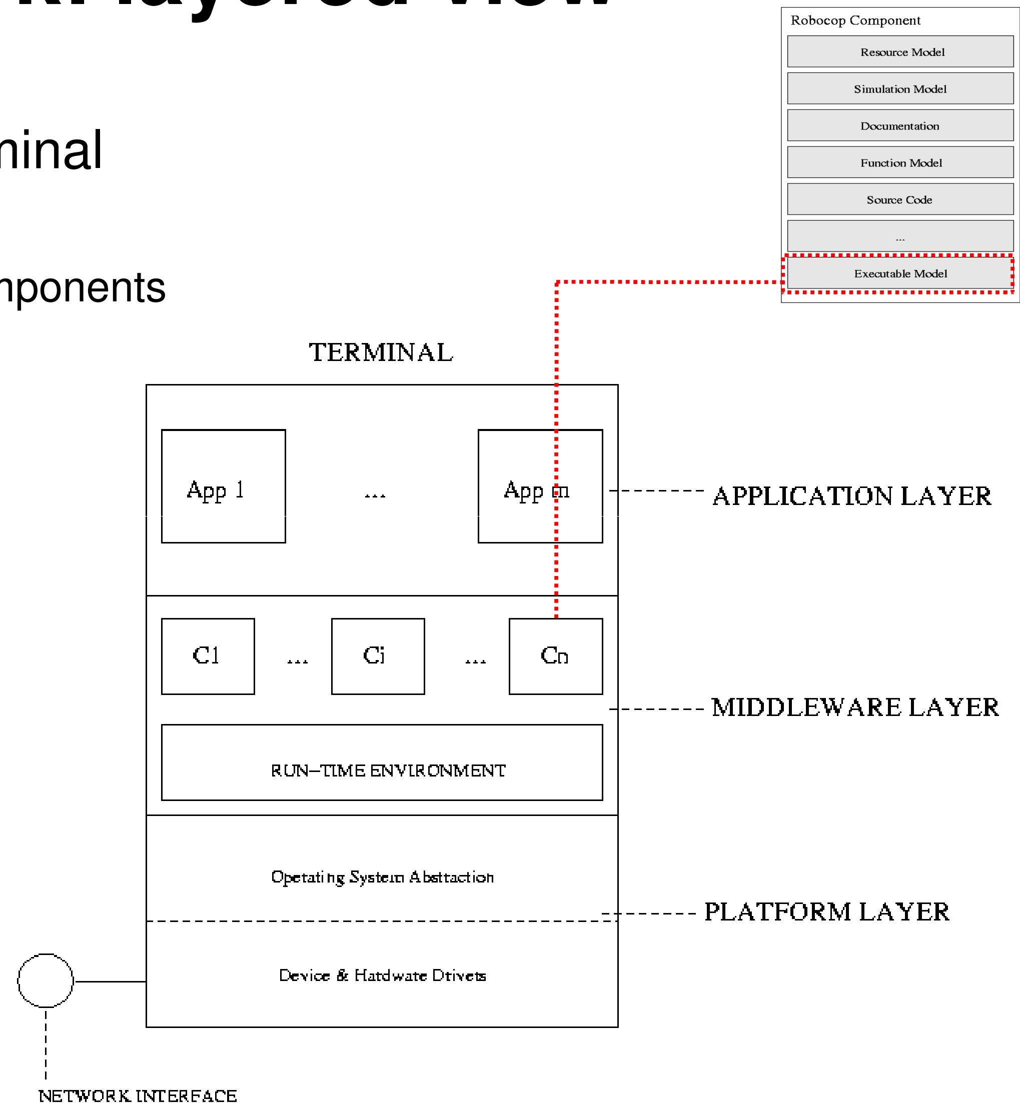
- The models are used:
  - to select components
  - to determine properties of compositions on platforms



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

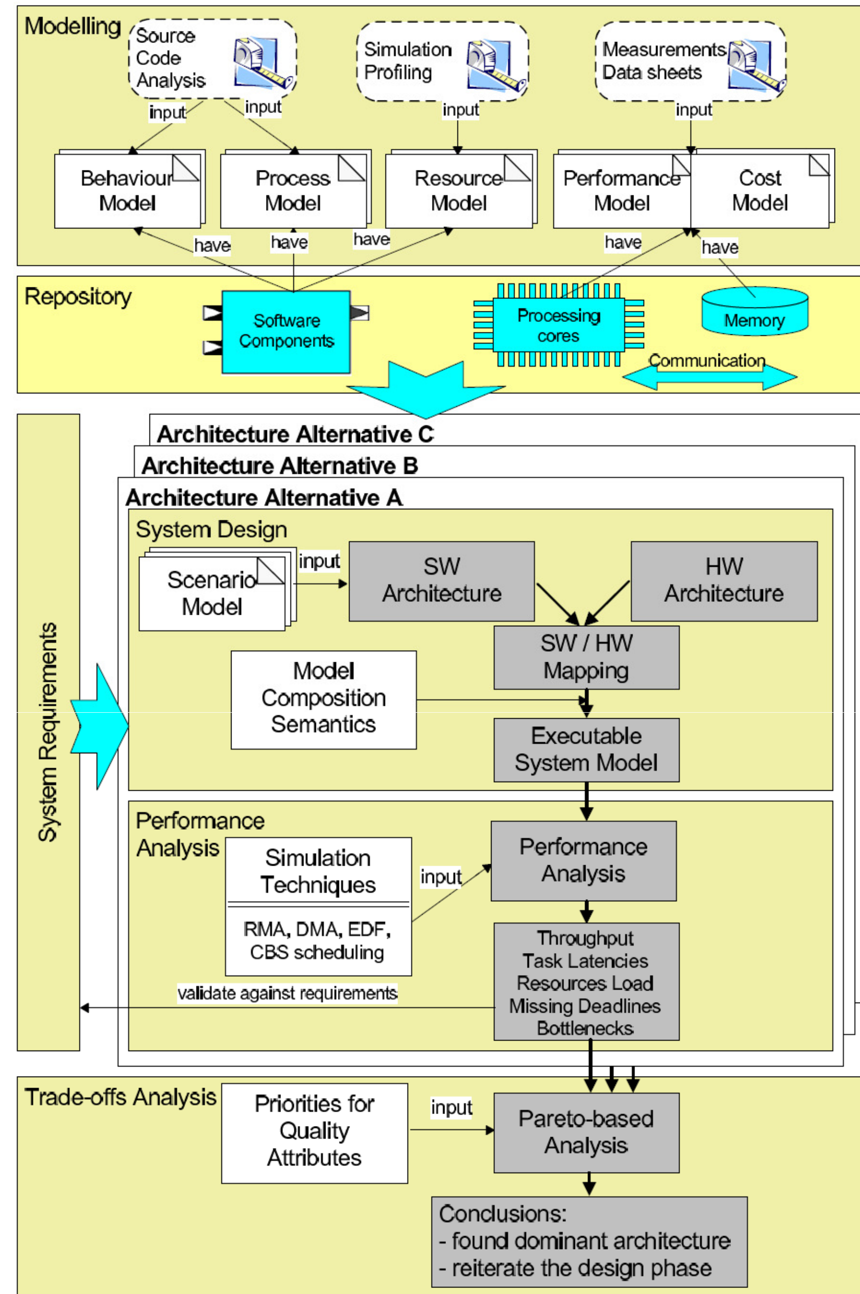
Robocop framework: layered view

- Layered, run-time view of a terminal
  - Application Layer
    - Applications, composed of components
  - Middleware Layer
    - Run Time Environment, providing management services
      - both for applications and components
    - Executable Components
  - Platform Layer
    - OS Abstraction
    - Device & HW drivers



Predictable assembly, exploration

- Tooling, by Bondarev, based on the Robocop model
- Adding besides the *component model* a *platform model*
  - allowing for studying different mappings to different hardware
  - allows tradeoff among several platforms
  - using a simulation techniques based on ‘critical traces’.



Bondarev, Chaudron, de With

Literature

- *Mass produced software components,* Douglas McIllroy, NATO conference '68
- *Technical Concepts of Component-Based Software Engineering*, CMU/SEI-2000-TR-008
- *A Classification Framework for Software Component Models*, Ivica Crnkovic, Severine Sentilles, Aneta Vulgarakis, and Michel R.V. Chaudron, IEEE Transactions on Software Engineering, Vol. 37, No. 5, Sept 2011
- *Component-based Software Engineering, Putting the Pieces together,* G.T. Heineman, W.T. Councill, ISBN 0-201-70485-4, Addison-Wesley 2001
- *Component Software: Beyond Object-Oriented Programming (2nd Edition),* C. Szyperski, ISBN 0201745720, 2002
- *A Component Framework for Consumer Electronics Middleware,* Johan Muskens, Michel R. V. Chaudron and Johan J. Lukkien, Lecture Notes in Computer Science, 2005, Volume 3778/2005, 164
  84, DOI: 10.1007/11591962_9