Software Architecture

Architectural Styles and Quality Attributes

Alar Raabe

Content

families of systems are related by shared structural and semantic properties

- Summary from Previous Seminar
- Software Architectural Styles
  - Classification of Architectural Styles
  - Examples of Different Software Architectural Styles
- Group Work
- Quality Attributes addressed by Software Architecture
  - Short overview
- Discussion
  - Most relevant architectural styles
  - Quality attributes important for us

What is (Software) Architecture

elements, relationships and principles that correspond to system concerns

- (Software) Architecture is a
  - **fundamental conception** of a (software) system in its
  - **environment** embodied in its
  - **elements**,
  - their **relationships** to each other and to the environment, and
  - **principles** guiding (software) system design and evolution
- (Software) Architecture description is a
  - collection of **models** in **correspondence** (relations),
  - organized into *synthetic* or *projective* **views** (cohesive groups, defined by **viewpoints**) according to te **concerns** addressed
- (Software) System Model
  - **anything** that can be used to answer questions about system

Model

architecture is a model of system and architecture description is a model of architecture

- an interpretation of a theory for which all the axioms of the theory are true
- a semantically closed abstraction of a system or a complete description of a system from a particular perspective
- *anything that can be used to answer questions about system*
  - to an observer B, an object M is a model of an object A to the extent that B can use M to answer questions that interest him about A [*Marvin Minsky*]
  - M is a model of A with respect to question set Q if and only if M may be used to answer questions about A in Q within tolerance T [*Doug Ross*]

Group Work Results: CDI-Hub

- stakeholders
- concerns
- viewpoints
- *views*
- *model correspondence rules*
- *rationale*
- *decisions*

Software Architectural Styles

- What is a Software Architectural Style
- Classification of Architectural Styles
- Benefits of Architectural Style
- Examples of Different Software Architectural Styles
  - Dataflow Systems – Pipes and Filters
  - Data-Centric Systems (Repositories) – Blackboard
  - Independent Components – Service Oriented Architecture
  - Complex Styles

What is a Software Architectural Style

a coherent package of design decisions

- Characterizes a family/class of system architectures that are related by shared structural and semantic properties

- Defines
  - a vocabulary of design elements
  - design rules, or constraints (incl. topology)
  - semantic interpretation
  - analyses that can be performed on systems built in that style

Classes of Architectural Styles

SOA conforms here to Independent Components

- Dataflow Systems
  - Batch sequential, Pipes and filters
- Call-and-return Systems (*explicit calls*)
  - Main program and subroutines, OO systems, Hierarchical layers
- Independent Components (*implicit calls*)
  - Communicating processes, Event Systems
- Virtual Machines
  - Interpreters, Rule-based systems
- Data-Centered Systems (Repositories)
  - Databases, Hypertext system, Blackboards

Classification of Architectural Styles (1)

Boxology – Shaw & Clements

- Constituenty Parts: Components and Connectors
  - Component – unit of software that performs some function at run-time
  - Connector – mechanism that mediates communications
- Control Issues
  - Topology – geometric form of control flow (e.g. linear, tree, acyclic graph, arbitrary)
  - Synchronicity – (in)dependence of components' upon each others' actions
  - Binding Time – when identity of partner for control flow is established

Classification of Architectural Styles (2)

Boxology – Shaw & Clements

- Data Issues
  - Topology – geometric form of data flow
  - Continuity – how continuously is new data generated (e.g. continuously, sporadically (at discrete times))
  - Mode – how data is made available (e.g. passed, shared)
  - Binding Time – when identity of partner for data flow is established
- Control/Data Interaction Issues
  - Shape – isomorphism of the control flow and data flow shapes
  - Directionality – conformance of directions of control and data flow
- Type of Reasoning

Classification of Architectural Styles

Boxology – Shaw & Clements

Style	Constituent Parts		Control Issues		Data Issues		Control/Data Interaction
	Components	Connectors	Topology	Synchronicity	Topology	Continuity	Isomorphic Shapes
Data Flow Architectural Styles							
Batch Sequential	programs	data batches	linear	sequential	linear	sporadic	yes
Data Flow Network	transducers	data streams	arbitrary	asynchronous	arbitrary	continuous	yes
Pipes and Filters	filters	pipes	linear	asynchronous	linear	continuous	yes
Call and Return Architectural Styles							
Main Program / Subroutines	procedures	proc. calls	hierarchical	sequential	arbitrary	sporadic	no
Abstract Data Types	managers	static calls	arbitrary	sequential	arbitrary	sporadic	yes
Objects	managers	dynamic calls	arbitrary	sequential	arbitrary	sporadic	yes
Call-based Client Server	programs	calls or RPC	star	synchronous	star	sporadic	yes
Layered			hierarchical	any	hierarchical	sporadic	often
Independent Components Architectural Styles							
Event Systems	processes	signals	arbitrary	asynchronous	arbitrary	sporadic	yes
Communicating Processes	processes	message protocols	arbitrary	any but sequential	arbitrary	sporadic	possibly
Data Centered Architectural Styles							
Repository	memory, computations	queries	star	asynchronous	star	sporadic	possibly
Black-Board	memory, components	direct access	star	asynchronous	star	sporadic	no

Benefits of Architectural Style

- Design Reuse
  - Well-understood solutions applied to new problems
- Code Reuse
  - Shared implementations of invariant aspects of a style
- Understandability of system organization
  - A phrase such as “client-server” conveys a lot of information
- Interoperability
  - Supported by style standardization
- Style-Specific Analysis
  - Enabled by the constrained design space
- Visualizations
  - Style-specific descriptions matching engineer’s mental models

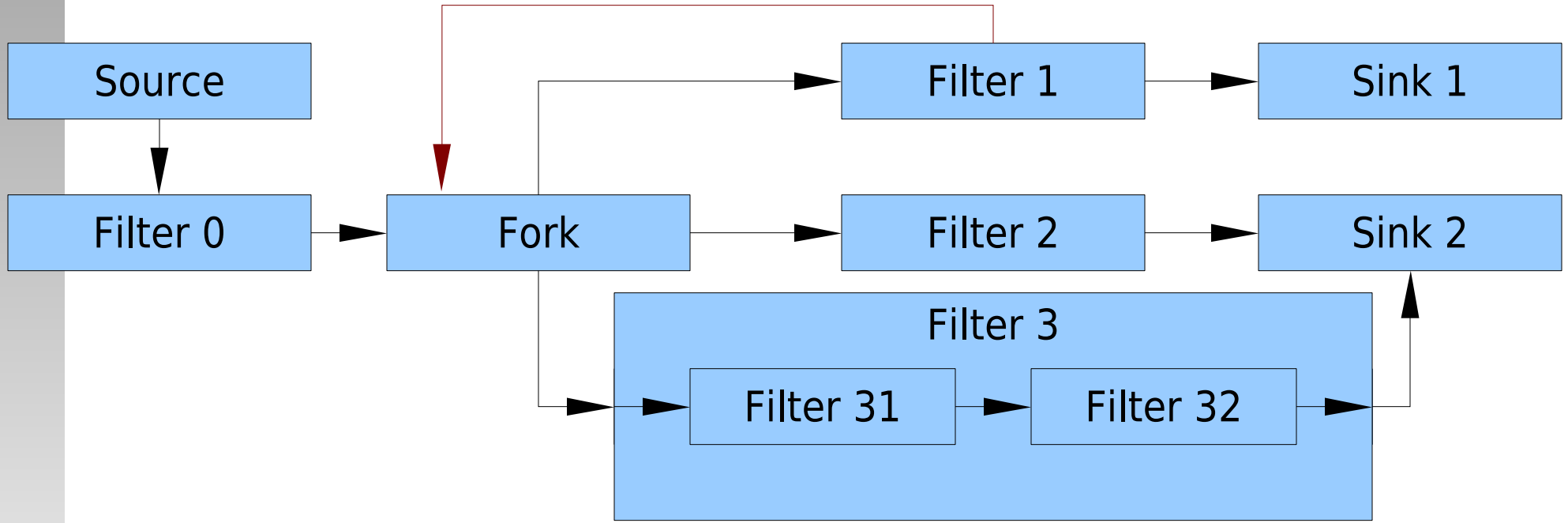
Examples of Different Software Architectural Styles

- Dataflow Systems – Pipes and Filters
- Data-Centric Systems (Repositories) – Blackboard
- Independent Components – Service Oriented Architecture
- Complex (Compound) Styles
  - REpresentational State Transfer
  - Big Ball of Mud (a Null Style?)

Dataflow Systems

shared nothing!

- Dataflow Systems – Pipes and Filters
  - Components (sources, filters, sinks)
  - Connectors (pipes)
  - Constraints (is feedback allowed or not, are pipes buffering, ...)
  - Theory (Queueing Theory (K. Erlang 1909))



Dataflow Systems Advantages (1)

- Modifiability & Reuse (low coupling, encapsulation)
  - filters stand alone and can be treated as black boxes
  - filters interact with other components in limited ways
- Ease of construction
  - systems can be hierarchically composed – higher order filters can be created from any combination of lower order pipes and filters
- Flexibility
  - the construction of the pipe and filter sequence (system configuration) can often be delayed until runtime (late binding)

Dataflow Systems Advantages (2)

- Run-time scalability
  - because the process performed by the filter is isolated from the other components in the system, it is easy to run a pipe-and-filter system on parallel processors
- Understandability/Analyzability
  - system behavior is a succession of component behaviors
  - support certain analyses (throughput, latency, deadlock)

Dataflow Systems Disadvantages

- Difficult to create interactive applications
  - because the problem is decomposed into sequential steps
- Common data representation
  - data has to be represented as the lowest common denominator (typically byte or character streams)
- Parsing overhead
  - if processing must be based on information, every filter may introduce parsing and unparsing of the data stream
- Unknown memory requirements and deadlock possibility
  - if a filter can not produce any output until it has received all of its input, the filter will require a buffer of unlimited size
  - if fixed size buffers are used, the system could deadlock (e.g. sort filter has this problem)
- Data sharing is difficult

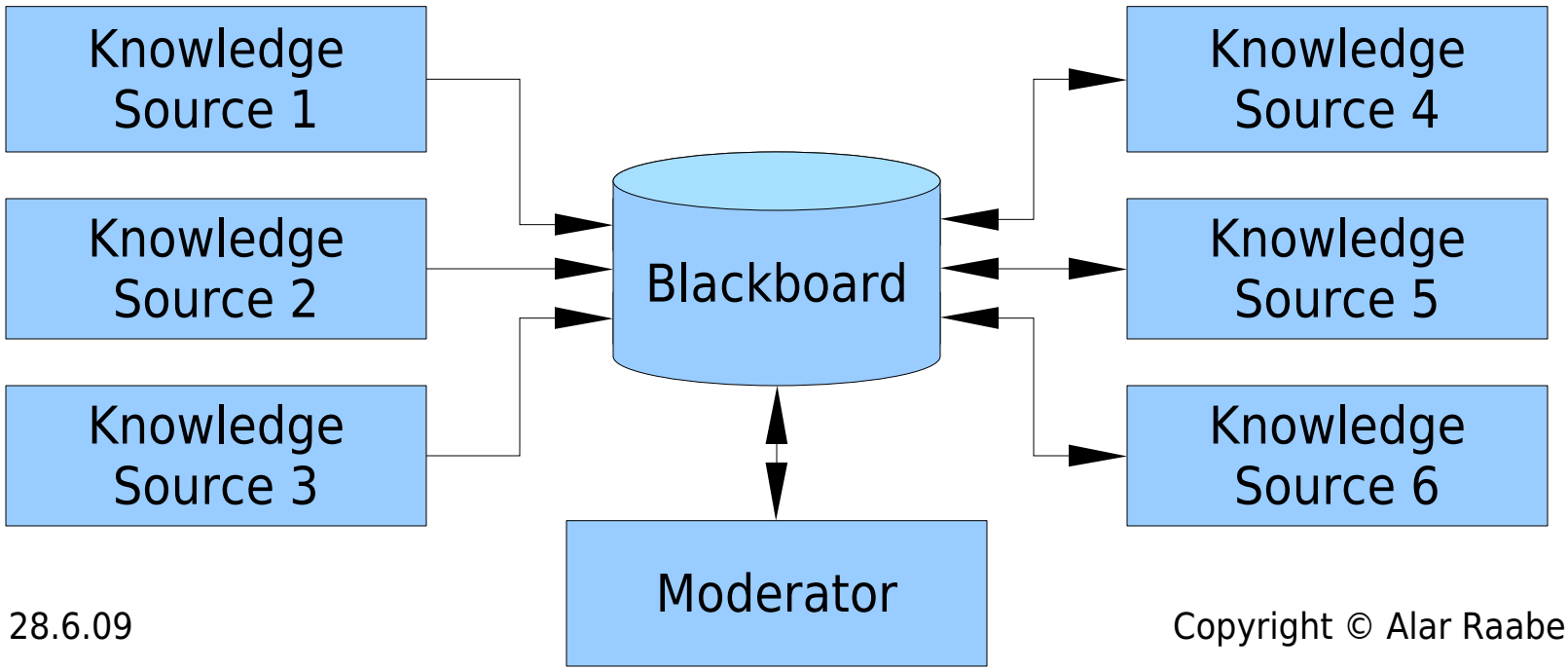
Dataflow Systems Examples

- Batch systems
- Many compilers
- Unix pipelines
- Spreadsheets
- JDPF (Java Data Processing Framework)
- Apache Camel (?)

Data-Centered Systems (Repositories)

shared everything!

- Data-Centered – Blackboard
  - Components (knowledge sources, moderator, blackboard)
  - Connectors (requests to and/or notifications from blackboard)
  - Constraints (transaction consistency)
  - Theories (coalgebras, multi-stream interaction machines (Wegner), coordination theory, transaction theory, ...)



Data-Centered Systems Advantages

- Scalability
  - easy to add more knowledge sources
  - knowledge sources can run in parallel and are synchronized through the central repository
- Separation of concerns (problem partitioning)
  - each knowledge source performs separate function
  - each knowledge source solves part of the problem
- Coupling
  - loose coupling between knowledge sources
- Modifiability
  - knowledge sources can be modified independently

Data-Centered Systems Disadvantages

- Coupling
  - tight coupling between knowledge sources and blackboard
- Scalability
  - blackboard becomes bottleneck with too many knowledge sources
- Difficult to analyze
  - non-deterministic behavior
  - system behavior emerges from the behaviors of knowledge sources

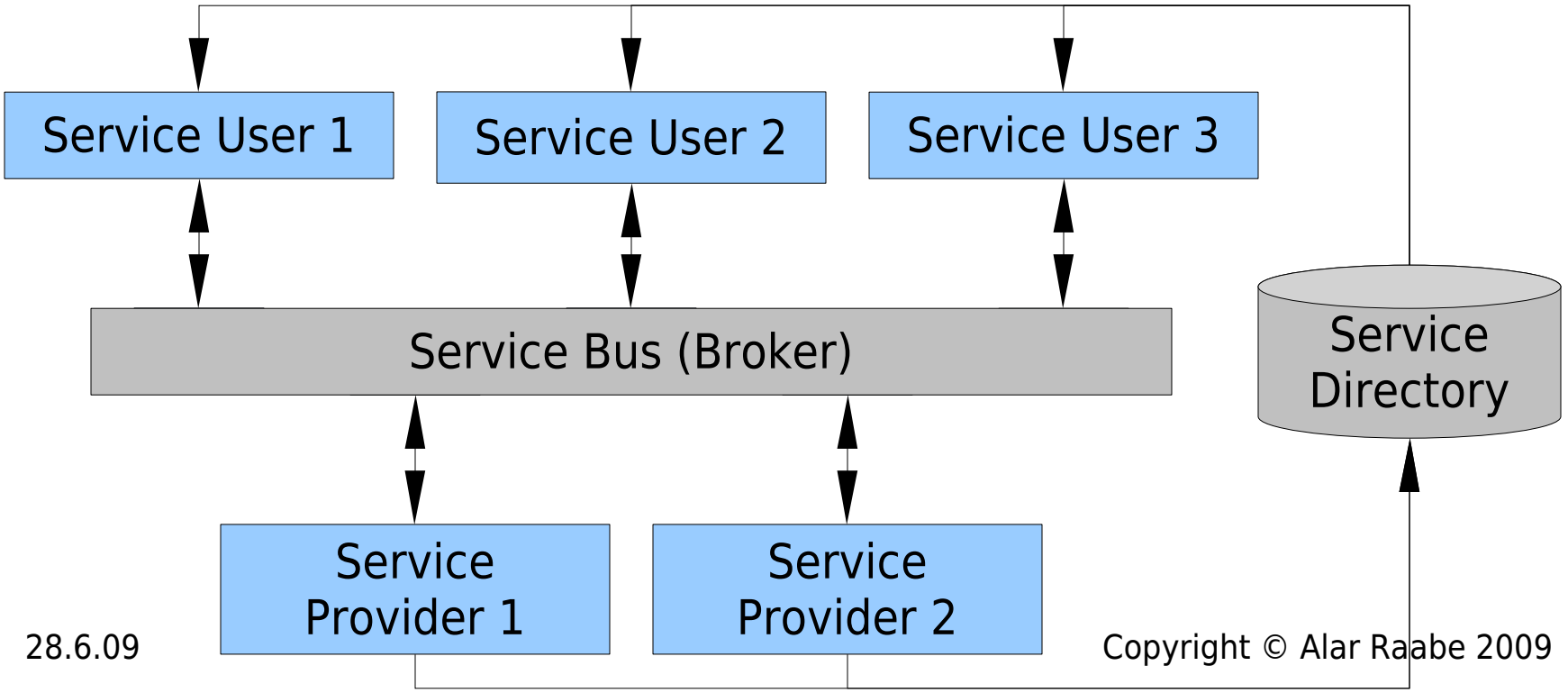
Data-Centered Systems Examples

- Many expert systems
  - Hearsay II (speech recognition system)
- Many language compilers & IDEs
- GBBopen (based on Common Lisp)
- Java Spaces
- Blackboard Event Processor (JVM-based, JavaScript, JRuby)
- Systems with global database

Independent Components (SOA)

bus and directory are optional!

- Independent Components – Service Oriented Architecture
  - Components (Providers, Users/Consumers; opt. *Bus, Directory*)
  - Connectors (synchronous and asynchronous calls, messages)
  - Constraints (call style)
  - Theories (CSP (C.A.R. Hoare), π-calculus (Milner, Parrow), …)

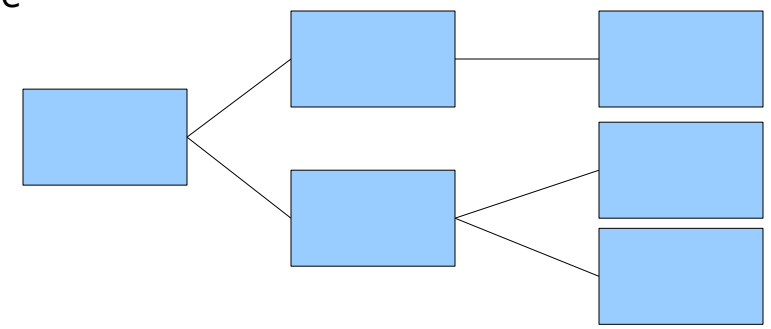


About Topologies

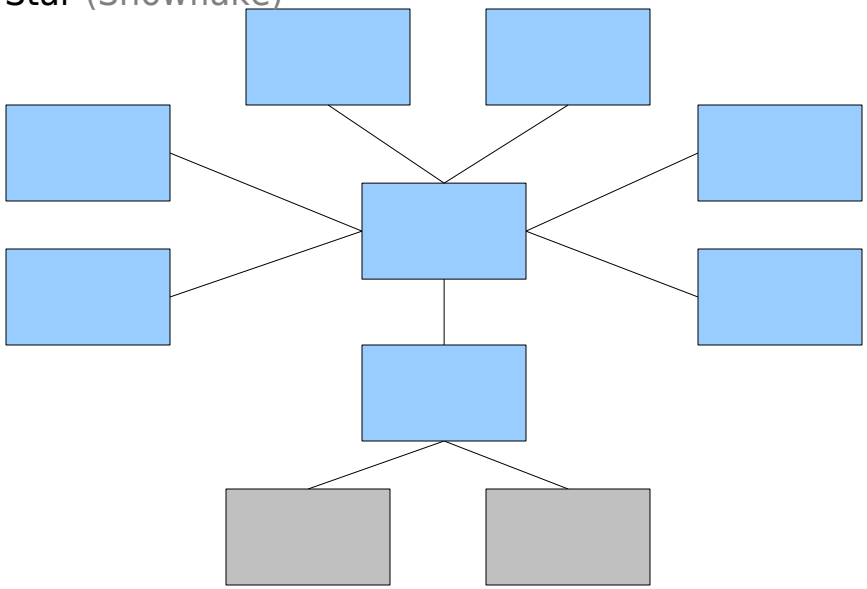
What's the difference between star and bus?

Linear

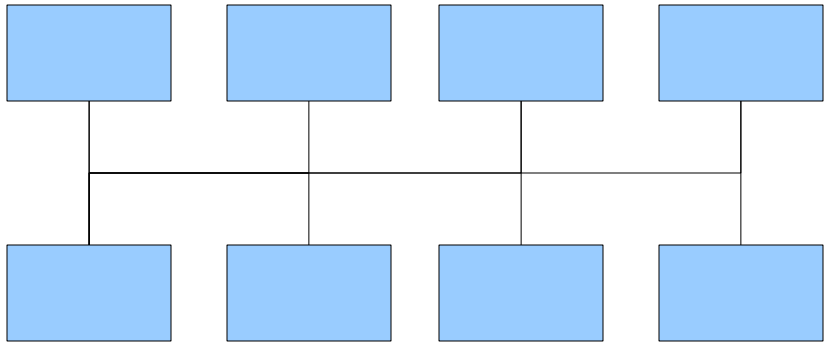
Tree



Star (Snowflake)



Bus



SOA Advantages

- Coupling
  - loose coupling – especially, if asynchronous calls are used
- Interoperability
  - service users can transparently call services implemented in disparate platforms using different languages
- Modifiability
  - loose coupling between service users and service providers
  - services are self-contained and modular
- Extensibility
  - if bus is used, adding new services is easy
- Reliability
  - good fault tolerance, if asynchronous calls are used

SOA Disadvantages

- Performance
  - network overhead
  - overhead of intermediaries (like service directory)
  - message parsing overhead
- Scalability
  - limited scalability, if synchronous calls are used
- Security
  - difficult to achieve end-to-end security (needs message level security mechanisms)
- Testability
  - more complex – difficult to test
- Reliability
  - complex error recovery might be needed

SOA Examples

- CORBA (IIOP)
  - ORB is Bus
  - Naming Service is Directory
- DCOM (RPC)
- JINI (RMI)
- Web Services (SOAP, REST)

Big Ball of Mud

Complexity increases rapidly until the it reaches a level of complexity just beyond that with which we can comfortably cope

Cunningham

- De-Facto Standard!
- Emerges from
  - Throwaway code, Piecemeal growth, Keep it working, Shearing layers, Sweeping it under the rug
- Forces corresponding to emergence
  - time – designing architecture takes time
  - cost – designed architecture costs and is long-term investment
  - experience and skill – designing architecture requires know-how
  - complexity and scale of the problems
  - change – predicting future change requires vision and courage
  - organization – architecture reflects organization (Conway's law)

Advantages of Big Ball of Mud

Mostly business concerns!

- Quick to make – Time-to-Market
- Cheap to make – Cost vs. Benefit
- Does not need governance
- Does not need skills
- ...

Disadvantages of Big Ball of Mud

- Maintainability
  - Difficult to maintain
- Modifiability
  - Hard to change
- Testability
  - Difficult to test
- ...

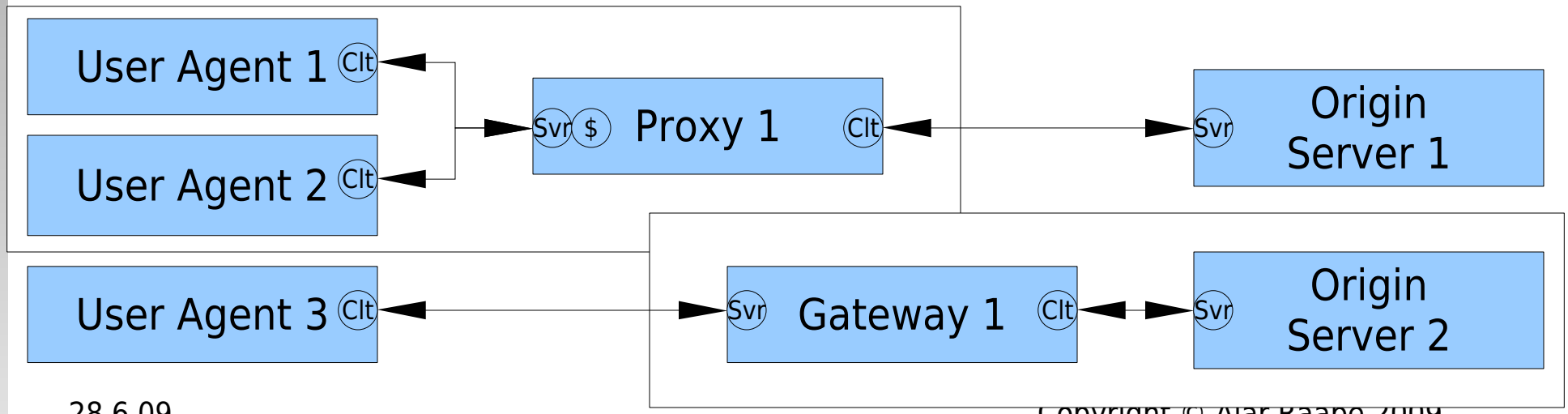
Big Ball of Mud Examples

- … ?

REpresentational State Transfer

architecture of web!

- Compound Style – Representational State Transfer
  - Components
    - Data (resources, resource identifiers, representations, representation metadata, resource metadata, control data)
    - Processing (origin servers, gateways, proxies, user agents)
  - Connectors (clients, servers, caches, resolvers, tunnels)
  - Constraints (data is not encapsulated)
  - Theory: Fielding analysis



Properties of Interest for REST

Fielding

- Performance
  - Network and User-Perceived Performance
  - Network Efficiency
- Scalability
- Simplicity
- Modifiability
  - Evolvability and Extensibility
  - Customizability and Configurability
  - Reusability
- Visibility
- Portability
- Reliability

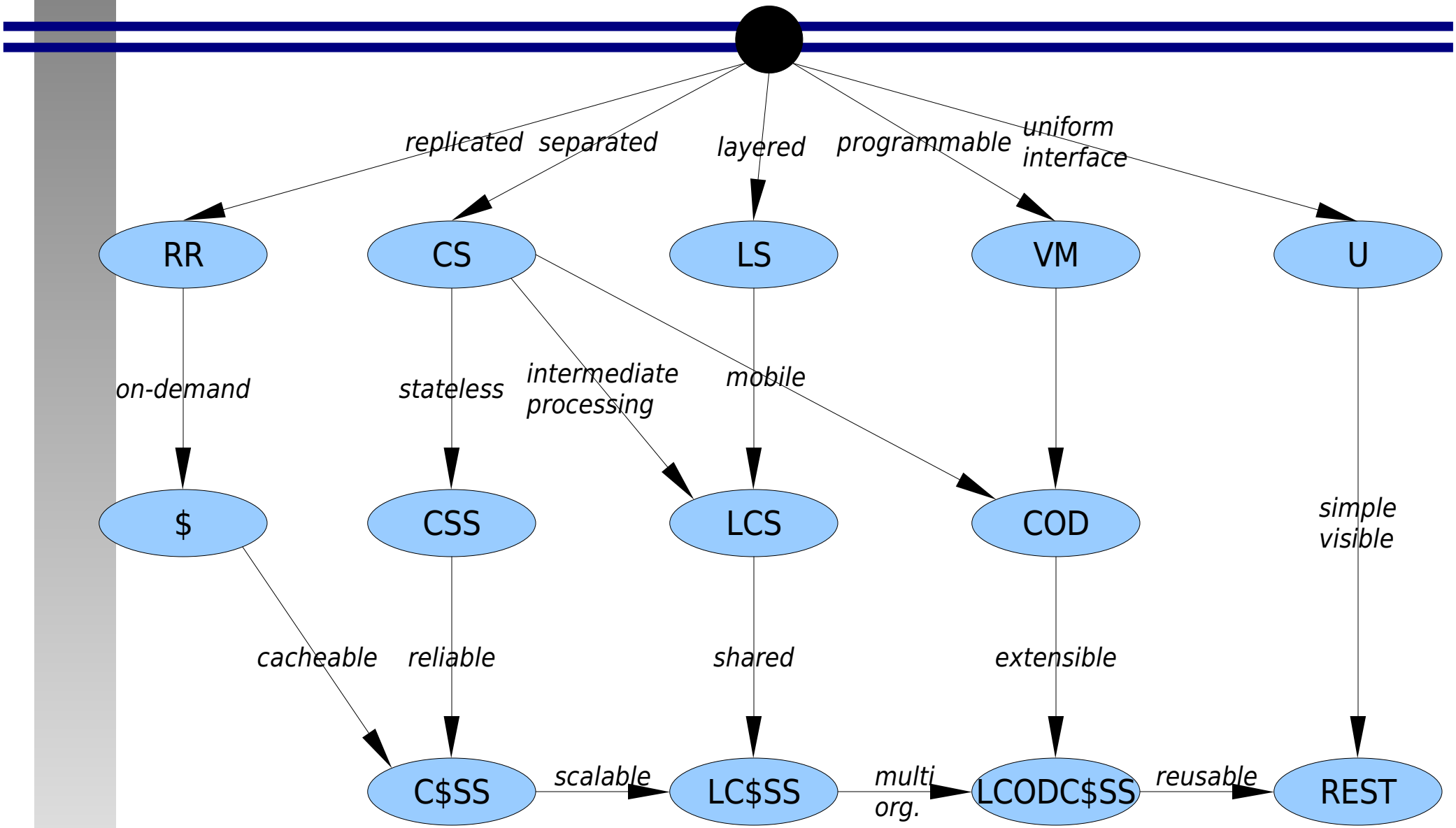
REST Constituents

Fielding

- Null Style – an empty set of constraints
- Client Server Style (CS)
  - separation of concerns, independent evolution
- Stateless Communication
  - session state in client – visibility, reliability, scalability
- Cache ($)
  - network efficiency
- Uniform Interface (U)
  - A constrained set of well defined operations and content types
- Layered System Style (LS)
  - hierarchical decomposition, managing complexity
- Code-on-demand (COD) *{optional}*
  - extensibility, simplified clients, but lower visibility

system with no distinguished boundaries – BBoM?

REST Derivation by Style Constraints



REST Advantages

- Simplicity
  - Less client code is required for thin client development
  - No need for explicit resource discovery mechanism due to hyperlinking
- Scalability
  - Compared with architectures that require stateful servers
- Efficiency
  - Caching promotes network efficiency and fast response times
- Evolvability
  - Support of document type evolution (such as HTML and XML) without impacting backward or forward compatibility
- Extensibility
  - Allows support for new content types without impacting existing and legacy content types

REST Disadvantages

- Limited Functionality
  - Selected uniform interface (HTTP) presents technical challenges for real time asynchronous events to a thin client or browser based application
- Scalability
  - Managing URI Namespace can be cumbersome
  - Can impact network performance by encouraging more frequent client-server requests and responses
- Visibility
  - In case code-on-demand is used to extend the client
- Development lacks supporting software tools

REST Examples

- WWW (World Wide Web)
- CMIP/CMOT (Common Management Information Protocol)
- Amazon (?)
- IBM WebSphere Portal REST API (?)
- ...

Group Work: CDI-Hub

- relevant architectural styles
- properties of interest

Quality Attributes of Software

- (Software) Quality
  - the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [ISO/IEC 9126]
- (Software) Quality Attribute
  - characteristic of software that affects its quality
- Categorization of Software Quality Attributes
  - End User's View – Developer's View – Business's View
  - Runtime Qualities – Non-Runtime Qualities
  - Quality Attributes Specific to the Architecture

Quality Attributes of Software

CMU SEI

- End User's View
  - *Functionality*
  - *Interoperability*
  - *Security*
  - *Performance (Efficiency)*
  - *Resource Efficiency*
  - *Availability and Reliability*
  - *Recoverability*
  - *Usability*
- Developer's View
  - *Modifiability*
  - *Portability (Extensibility)*
  - Reusability
  - Integrability
  - Testability
- Business's View
  - Time To Market
  - Cost vs. Benefits
  - Projected Life-time
  - Targeted Market
  - Integration with Legacy
  - Roll-out (Roll-back) Schedule

Discussion

What is/are for us ...

- Concepts of
  - (Software) System
  - (Software) Architecture
  - (Software) Architecture Description
- Value of
  - (Software) Architecture
  - (Software) Architecture Description
- Most Relevant (Software) Architecture Styles
- Main Stakeholders
- Main System Concerns
- (Software) Architecture Framework
- (Software) Quality Attributes

Conclusion

- Value of (Software) Architecture
  - as fundamental conception of (software) system, architecture allows us to reason (answer questions) about the (software) system
  - as specific architectural styles address certain concerns (cause certain properties/qualities) of (software) systems, architecture allows us to address concerns (achieve required properties or qualities) of (sofware) systems
- Value of Architecture Description
  - as document, it provides guidance for constructing and evolving the (software) system, and allows us to record and communicate our knowledge and decisions about the (software) system architecture
  - as model, it allows us to reason (answer questions) about the (software) system architecture

Leftovers

- Conway's law (1968)
  - organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Thank You!

Terms (Glossary)

ISO/IEC 42010:2007

architecture	fundamental conception of a system in its environment embodied in elements, their relationships to each other and to the environment, and principles guiding system design and evolution
architecture decision	choice made from among possible options that addresses one or more architecture-related concerns
architecture description	collection of work products used to describe an architecture
architecture model	work product from which architecture views are composed
architecture rationale	explanation or justification for an architecture decision
architecture view	work product representing a system from the perspective of architecture-related concerns
architecture viewpoint	work product establishing the conventions for the construction, interpretation and use of architecture views
environment	context determining the setting and circumstances of developmental, technological, business, operational, organizational, political, regulatory, social and any other influences upon a system
model correspondence	relation on two or more architecture models
stakeholder	individual, team, organization, or class thereof, having concerns with respect to a system
purpose	*{one of system concerns}*
system	*{a conceptual entity defined by its boundaries}*
system concern	area of interest in a system pertaining to developmental, technological, business, operational, organizational, political, regulatory, social, or other influences important to one or moreof its stakeholders

Definitions 1

- System
  - a collection of interacting components organized to accomplish a specific function or set of functions within a specific environment
- Interface (Connection)
  - a shared boundary between two functional units, defined by various characteristics of the functions
  - component that connects two or more other components for the purpose of passing information from one to the other
- Module (Component)
  - a logically separable part of a system
- Encapsulation
  - isolating some parts of the system from the rest of the system
  - a module has an outside that is distinct from its inside (an external interface and an internal implementation)

Definitions 2

- Modularity
  - the degree to which a system is composed of discrete components such that a change to one component has minimal impact on other components
  - the extent to which a module is like a black box
- Coupling
  - the manner and degree of interdependence between modules
  - the strength of the relationships between modules
  - a measure of how closely connected two modules are
- Cohesion
  - the manner and degree to which the tasks performed by a single module are related to one another
  - a measure of the strength of association of the elements within a module

Definitions 3

- Model
  - an interpretation of a theory for which all the axioms of the theory are true
  - a semantically closed abstraction of a system or a complete description of a system from a particular perspective
  - anything that can be used to answer questions about system
    - to an observer B, an object M_A is a model of an object A to the extent that B can use M_A to answer questions that interest him about A

      Marvin Minsky
    - M is a model of A with respect to question set Q if and only if M may be used to answer questions about A in Q within tolerance T

      Doug Ross

Comparing CMU SEI and ISO/IEC Software Quality Attributes

• Functionality • Interoperability • Security	• Functionality – Suitability, Accuracy, Interoperability, Compliance, Security
• Availability and Reliability • Recoverability	• Reliability – Maturity, Recoverability, Fault Tolerance
• Usability – Learnability, Memorability, Error avoidance & handling, Satisfaction	• Usability – Learnability, Understandability, Operability
• Performance (Efficiency) • Resource Efficiency	• Efficiency – Time Behaviour, Resource Behaviour
• Modifiability	• Maintainability – Stability, Analyzability, Changeability, Testability
• Portability (Extensibility) – Installability, Replaceability, Adaptability, Conformance	• Portability – Installability, Replaceability, Adaptability, Conformance

Questions

- How & who is using design artifacts?
- How to measure the cost and value of design knowledge?
- Who wants to pay for documents?
- Who wants to pay for exploring of various design alternatives?
- How tests are debugged?
- How to select architectural style?
- How to recover concepts?
- How to measure cost of having (or not having) architecture?
- How to evaluate the goodness of a method?