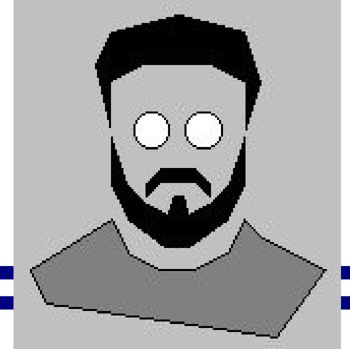
A Short Introduction

Alar Raabe

Alar Raabe



- Over 30 years in IT
  - held various roles from programmer to a software architect
- 15 years in insurance and last 4 years in banking domain
  - developed model-driven technology for insurance applications product-line (incl. models, method/process, platform/framework and tools)
  - developing/implmenting business architecture methods for a banking group
- Interests
  - software engineering (tools and technologies)
  - software architectures
  - model-driven software development
  - industry reference models (e.g. IBM IAA, IFW)
  - domain specific languages

Copyright © Alar Raabe 2011

Content

Software architecture is what software architects do

Kent Beck

- What is Software Architecture
  - Design vs. Architecture, Early Views and Software Architecture Discipline
  - Software Architecture related Concepts and Terminology
- Software Architectural Styles
  - Classification of Software Architectural Styles
  - Examples of Different Software Architectural Styles
- Software Quality Attributes
  - Categories of Software Quality Attributes
  - Quality Attribute Driven Design
- Value of Software Architecture
  - How to Evaluate Software Architectural Decisions
  - Value and Cost of Architecture
- Conclusions

Copyright © Alar Raabe 2011

Architecture

Architecture is about:

❖ Durability (*firmitas*)
❖ Utility (*utilitas*)
❖ Beauty (*venustas*)

Vitruvius
(Rome, 1 BC)

- Merriam-Webster :: Architecture (n)
  - art or science of building
  - unifying or coherent form or structure
  - manner in which the components of the system are organized and integrated

- Wikipedia :: Architecture
  (Greek: *αρχιτεκτονική* and Latin: *architectura*)
  - 2011
    - art and science of designing (*buildings and other physical*) structures
    - style and method of design and construction of (*buildings and other physical*) structures
    - …
  - *2009*
    - *…*
    - *as documentation, usually based on drawings, architecture defines the structure and/or behavior of a system that is to be or has been constructed*

Copyright © Alar Raabe 2011

Design vs. Architecture

All architecture is design but not all design is architecture

Grady Booch

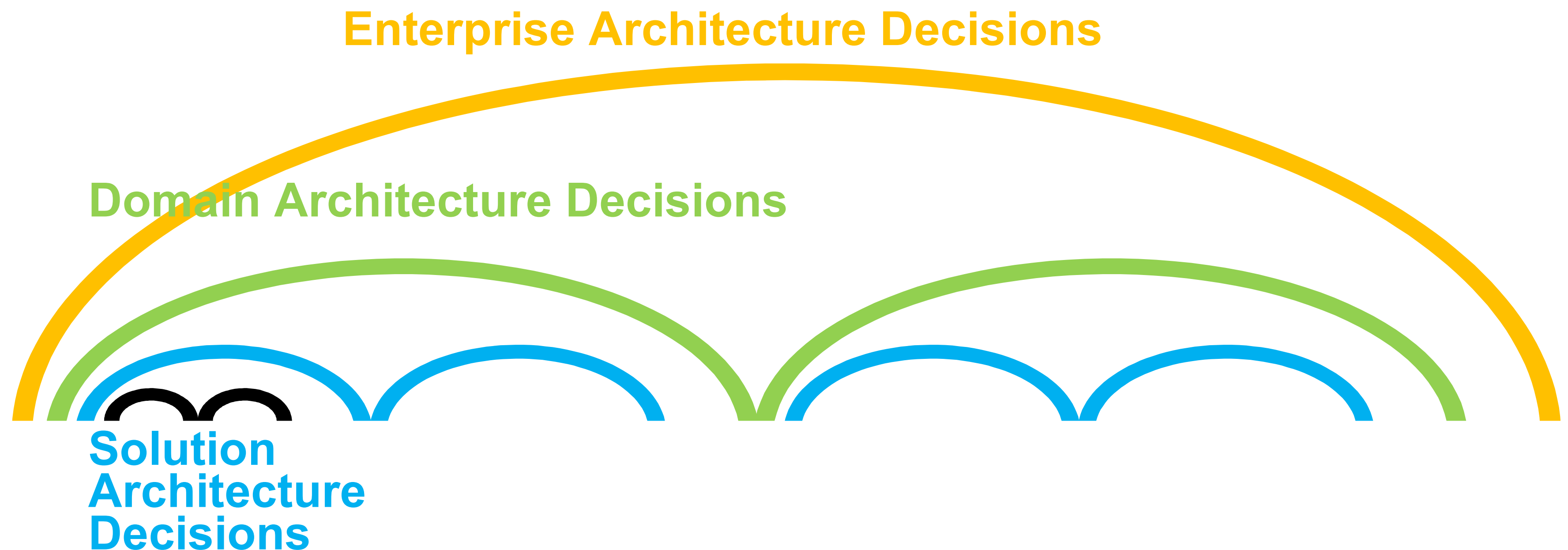
- Design = Plan
  - adaptation of means (*what we have*) to ends (*what we want*)
- Software Design can be viewed on many levels
  - design of higher levels is architecture for the lower levels
- Booch
  - architecture represents significant design decisions that shape a system, where *significant is measured by cost of change*
- Eden
  - Architectural decisions and specifications are
    - **intensional** (*generic – applicable to many implementations*), and
    - **non-local** (*applicable to entire system*)

The Intension/Locality Thesis		
Non-Local	**Intensional**	**Architecture**
Local	Intensional	Design
Local	Extensional	Implementation

Copyright © Alar Raabe 2011

Different Architecture Levels – Decision Scopes

- Enterprise Architecture – a **holistic view** on whole enterprise
  - a description of the enterprise that provides a common understanding and a **formal link between strategic objectives and tactical execution**



Copyright © Alar Raabe 2011

Early Views on Software Architecture

Structure matters !

- Turing & Wheeler (1946-50)
  - reuse of program code and modularization – (closed) subroutine
  - subroutine library (reusability, reliability, unit testing (testability), multiple versions with different non-functional qualities, …)
- Iverson & Brooks (1964-69)
  - architecture is a conceptual structure
  - architecture is the complete and detailed specification of the user interface (!)
- Dijkstra, Parnas & Jackson (1972-76)
  - separation of concerns – isolation, encapsulation, modularization
  - program families can be described by a decision trees
  - *structure influences non-functional ‘qualities’ of system*
  - *structure of program is defined by domain structures*

Copyright © Alar Raabe 2011

Software Architecture as Discipline

elements + form/structure + rationale/principles

- Perry & Wolf (1992)
  - Software Architecture = {Elements, Form, Rationale}
    - a set of architectural (or, if you will, design) ***elements*** that have a particular form (of three different classes: processing, data, and connecting elements),
    - architectural ***form***, consisting of weighted properties and relationships, and
    - ***rationale*** for various choices made in defining an architecture
- Garlan & Shaw (1994)
  - a collection of computational components – or simply ***components*** – together with a description of the interactions between these components – the ***connectors***
- Bass, Clements, Kazman (1997)
  - the structure or structures of the system, which comprise software ***components***, the externally visible properties of those components, and the ***relationships*** among them
- Eden, Kazman (2003)
  - strategic design decisions/statements (global design constraints like programming paradigms, architectural styles, component-based software engineering standards, design principles, and law-governed regularities)

Copyright © Alar Raabe 2011

Agile and Software Architecture

Architecture is the important stuff – whatever that is

Ralph Johnson

- Johnson (…)
  - a shared understanding of the system design of the expert developers working on the project (incl. how the system is divided into components and how the components interact through interfaces)
  - the decisions that you wish you could get right early in a project

- Beck (2000)
  - Expressed in XP through system metaphor, which “helps everyone to understand basic elements and their relationships”
  - Should be created by first iteration

- Fowler (2003)
  - a word we use when we want to talk about design but want to puff it up to make it sound important

Copyright © Alar Raabe 2011

Software Architecture Standards

architecture ≠ architecture description

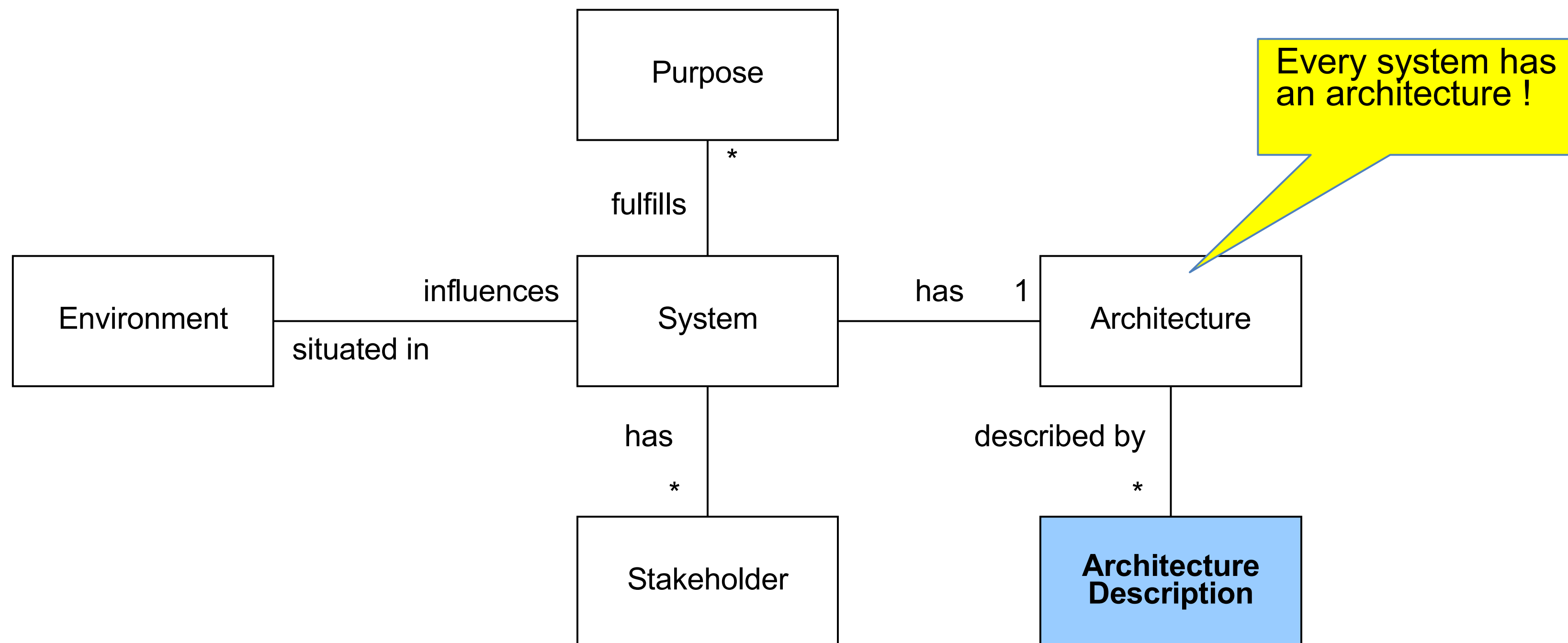
- Open Group TOGAF 9 Enterprise Architecture Framework
  - a formal description of a system, or a detailed plan of the system at component level to guide its implementation
  - the structure of ***components***, their ***interrelationships***, and the ***principles and guidelines*** governing their design and evolution over time

- IEEE 1741 | ISO/IEC 42010 Systems and Software Engineering – Architecture Description
  - the 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 descriptions are for …
    - *Communicating* among the system's stakeholders
    - *Planning and Managing* system development and operations
    - *Evaluating and Comparing* systems architectures, and verifying system's implementation for compliance with its intended architecture

Copyright © Alar Raabe 2011

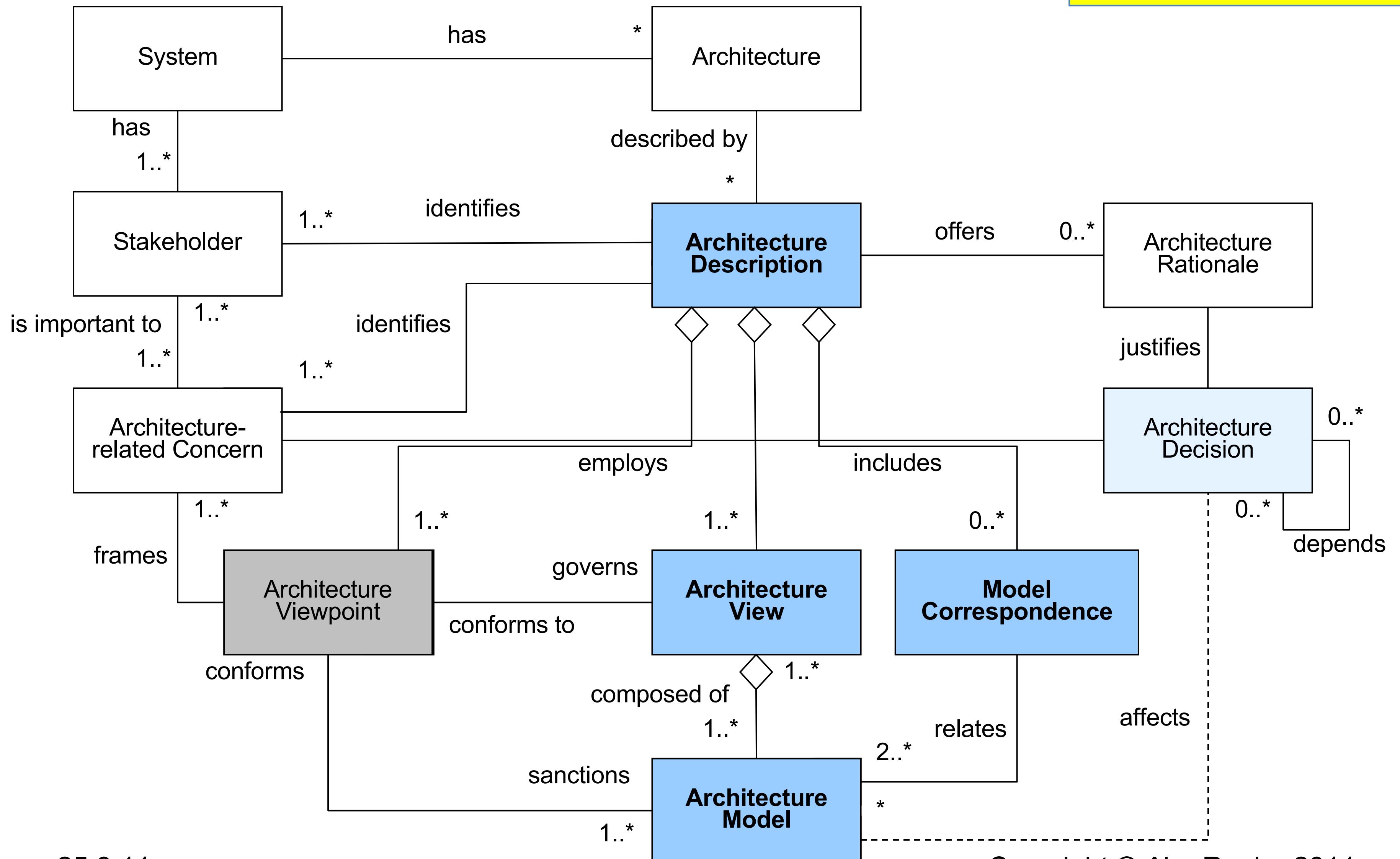
System, Architecture and Architecture Description

ISO/IEC 42010



Copyright © Alar Raabe 2011

Architecture Description – set of Views



Copyright © Alar Raabe 2011

Stakeholders & Concerns & Decisions

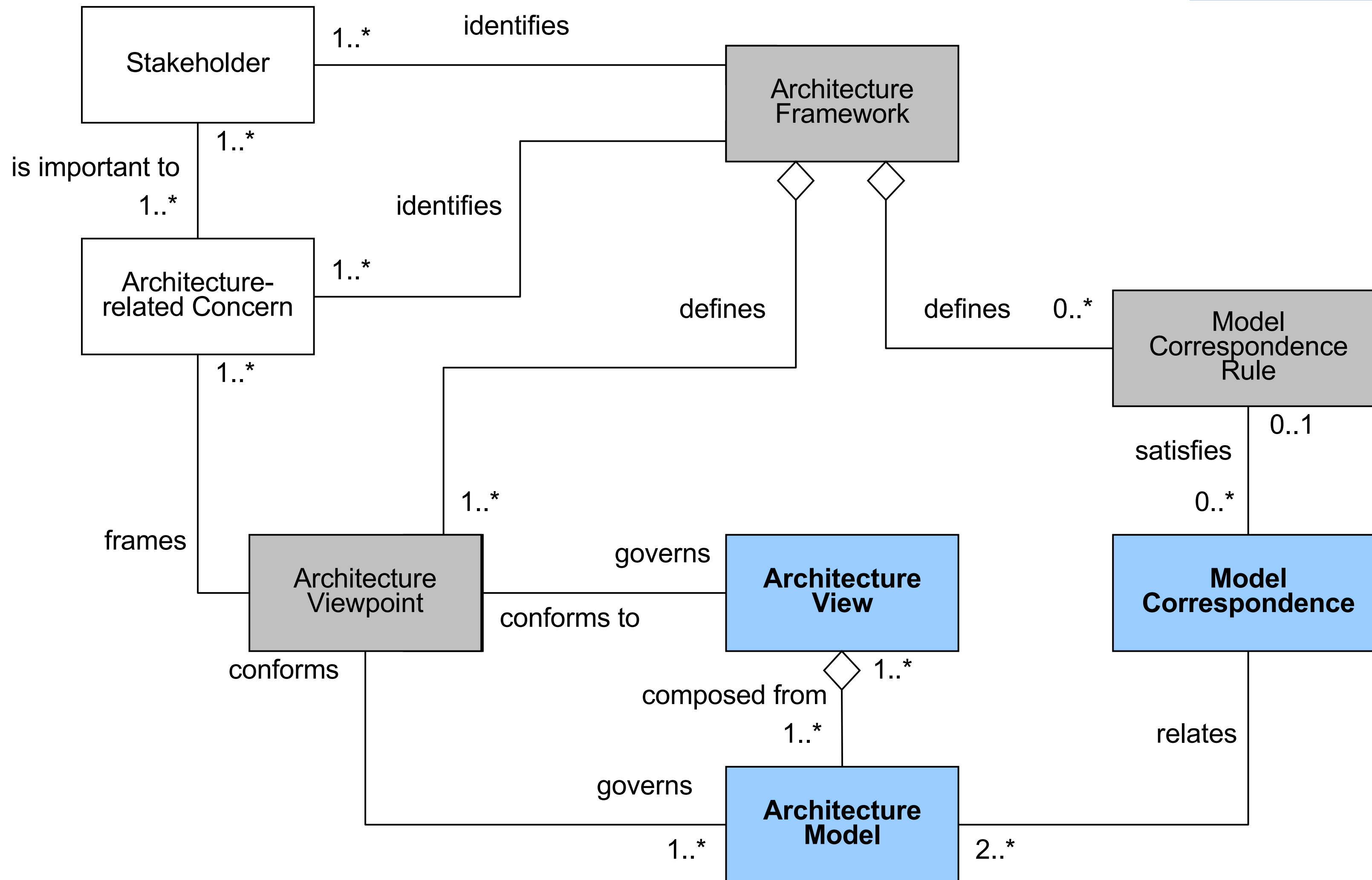
ISO/IEC 42010

- Stakeholders
  - Users and operators
  - Acquirers and owners
  - Suppliers, developers, builders and maintainers
- Architecture-related Concerns
  - The suitability of the architecture for achieving the system's purposes
  - The feasibility of constructing the system
  - The potential risks of the system to its stakeholders throughout its life cycle
  - Maintainability, deployability, and evolvability of the system
- Architecture Decisions are decisions
  - regarding architecturally significant requirements
  - needing a major investment of effort and time
  - affecting key stakeholders or a number of stakeholders
  - needing intricate or non-obvious reasoning
  - that are highly sensitive to changes
  - that could be costly to change

Copyright © Alar Raabe 2011

Architecture Framework – set of Viewpoints

ISO/IEC 42010



Copyright © Alar Raabe 2011

Example: Sensor Collection Service

ISO/IEC 42010

- Purpose (of the System)
  - Subscription-based service of providing access to a widely-distributed set of sensors
- Stakeholders
  - Users, developers, operators
- Architecture-related Concerns (by Stakeholders)
  - ROI (operators)
  - Timely delivery of sensor data (users)
  - Understanding of interactions between system elements (developers)
- Viewpoints (by Architecture-related Concerns)
  - Financial: cash-flow spreadsheet (ROI)
  - Operational: timeline diagram (timely delivery of sensor data)
  - System: system component diagram (understanding of interactions between system elements)
- View Consistency and Correspondence Rules
  - Each node in component diagram should appear at least once in timeline diagram
- Views (by Viewpoints)
  - Profit spreadsheet & profitability curve (cash-flow spreadsheet)
  - Timeline diagram (timeline diagram)
  - Dataflow diagram (system component diagram)

Copyright © Alar Raabe 2011

What is Software Architecture

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

- Software Architecture is a
  - *fundamental conception* of a (software) system in its
  - ***environment*** embodied in
  - ***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 ***related (corresponding) models***, organized into cohesive groups of
  - *synthetic* (constructed) or *projective* (derived) ***views***, defined by ***viewpoints*** according to the related set of ***concerns*** *(in architecture framework)*

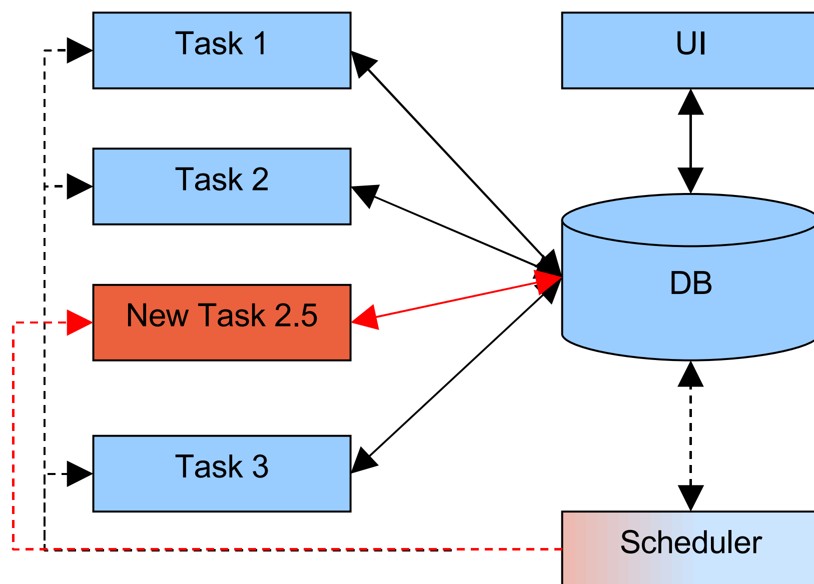
- Software Architecture Model is
  - ***work product*** that can be used to answer questions about the software system

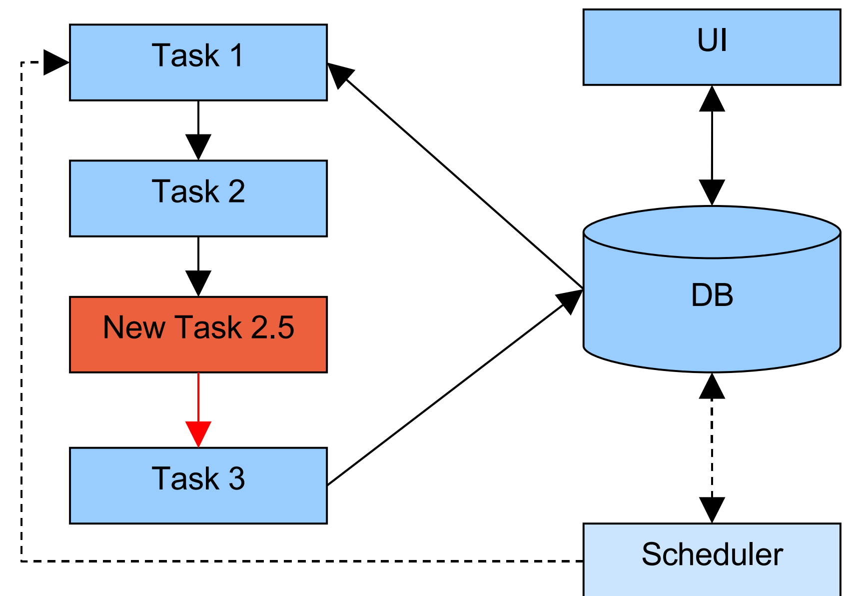
    - M. Minsky 1968: “to an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A”
    - IEEE SE VOCAB: an interpretation of a theory for which all the axioms of the theory are true, or a semantically closed abstraction of a system or a complete description of a system from a particular perspective

Copyright © Alar Raabe 2011

Different Architectures – Different Properties

adding new task

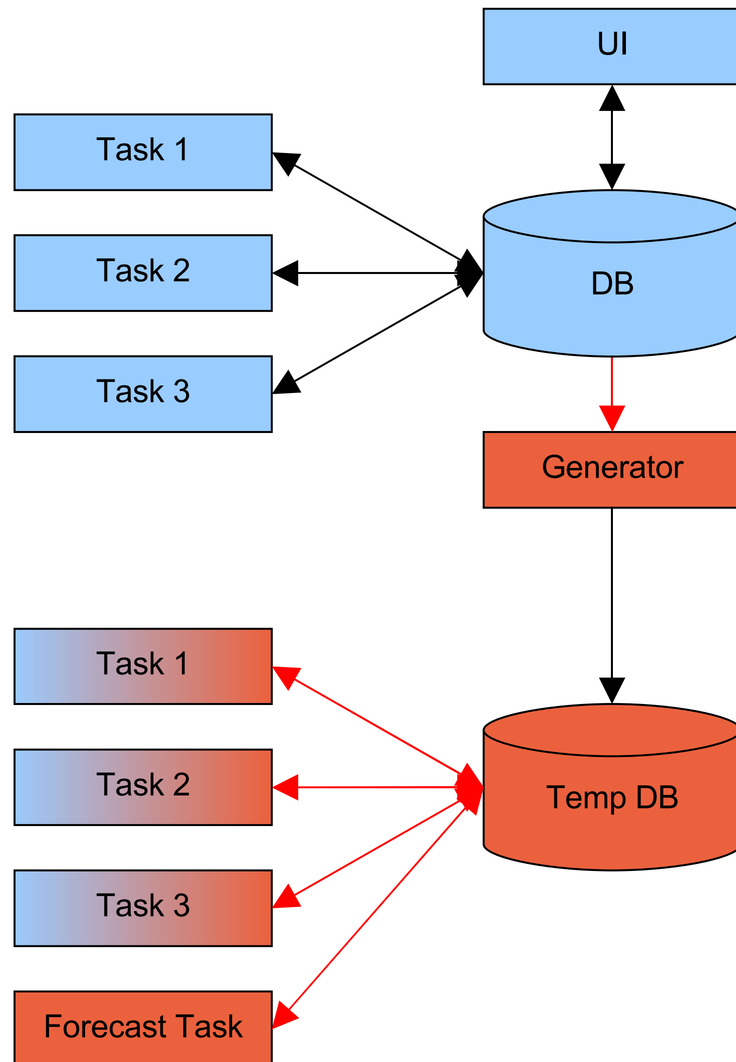


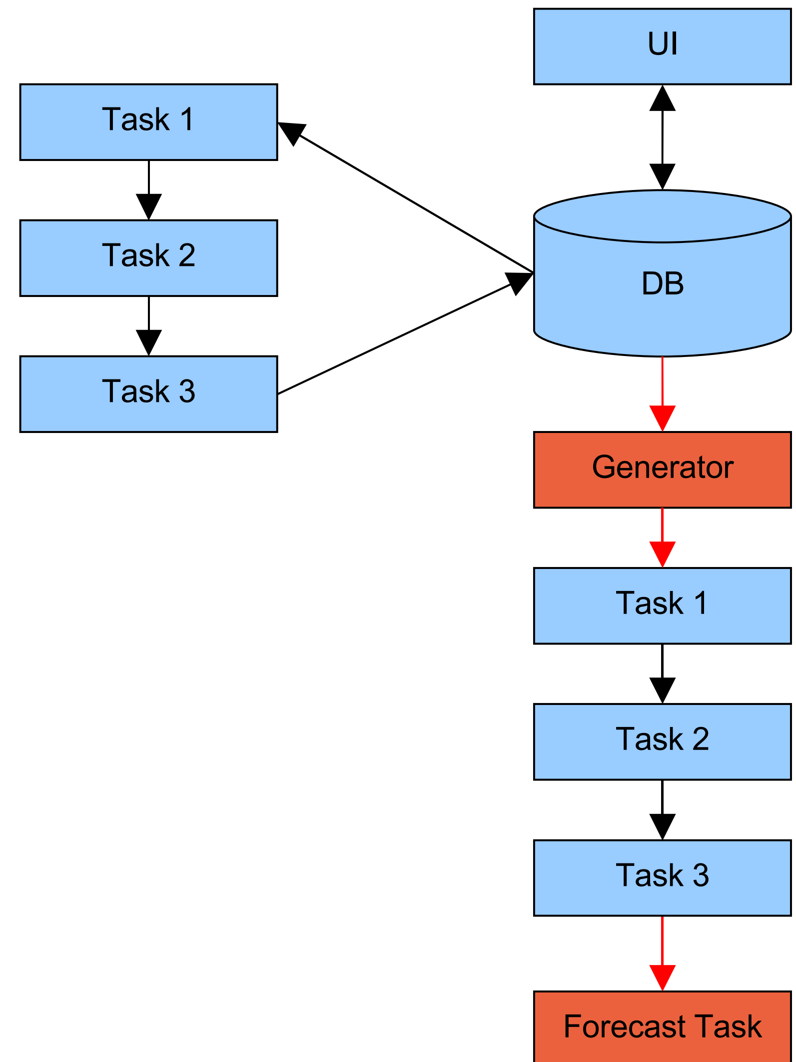


Copyright © Alar Raabe 2011

Different Architectures – Different Properties

adding forecasts of portfolio





Copyright © Alar Raabe 2011

Software Architectural Styles

families of system are related by shared structural and semantic properties

- Different Levels of Commonality: Idioms, Patterns, Styles
- Software Architectural Styles
  - What is a Software Architectural Style
  - Classification of Architectural Styles
- Examples of Different Software Architectural Styles
  - Dataflow Systems – Pipes and Filters
  - Data-Centric Systems (Repositories) – Blackboard
  - Independent Components – Service Oriented Architecture (SOA)
  - Complex (Compound) Styles – REST (Representational State Transfer)
- Building a Software Architectural Style
- Emerged Software Architecture

Copyright © Alar Raabe 2011

Different Levels: Idioms, Patterns, Styles

reuse of (design) knowledge

- Specific to a (programming) language
  - Software Idioms – coding/programming
    - describe usage of (programming) language for certain (simple) problems
  - Programming Style – programming
    - a consistent set of idioms (e.g. fluent style, functional style, …)
- (Programming) language independent
  - Design Patterns – design
    - describe standard solutions to certain common functional problems
  - Architecture Styles – architectural design
    - specific vocabulary and rules for architectural design
    - define a class of systems with specific properties
    - describe standard solution to a class of non-functional problems

Copyright © Alar Raabe 2011

What is a Software Architectural Style

a coherent package of pre-made 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
- Benefits
  - *Design Reuse* – well-understood solutions applied to new problems
  - *Code Reuse* – shared implementations of invariant aspects of a style
  - *Understandability of System Organization* – e.g. meaning of “client-server”
  - *Interoperability* – supported by style standardization
  - *Style-Specific Analysis* – enabled by the constrained design space
  - *Visualizations* – style-specific descriptions matching engineer’s mental models (e.g. stack diagrams for layers)

Copyright © Alar Raabe 2011

Classification of Architectural Styles

Boxology – Shaw & Clements

- Constituent 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
- Data Issues
  - Topology – geometric form of data flow
  - Continuity – new data generation (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

Copyright © Alar Raabe 2011

Classes of Architectural Styles

Boxology – Shaw & Clements

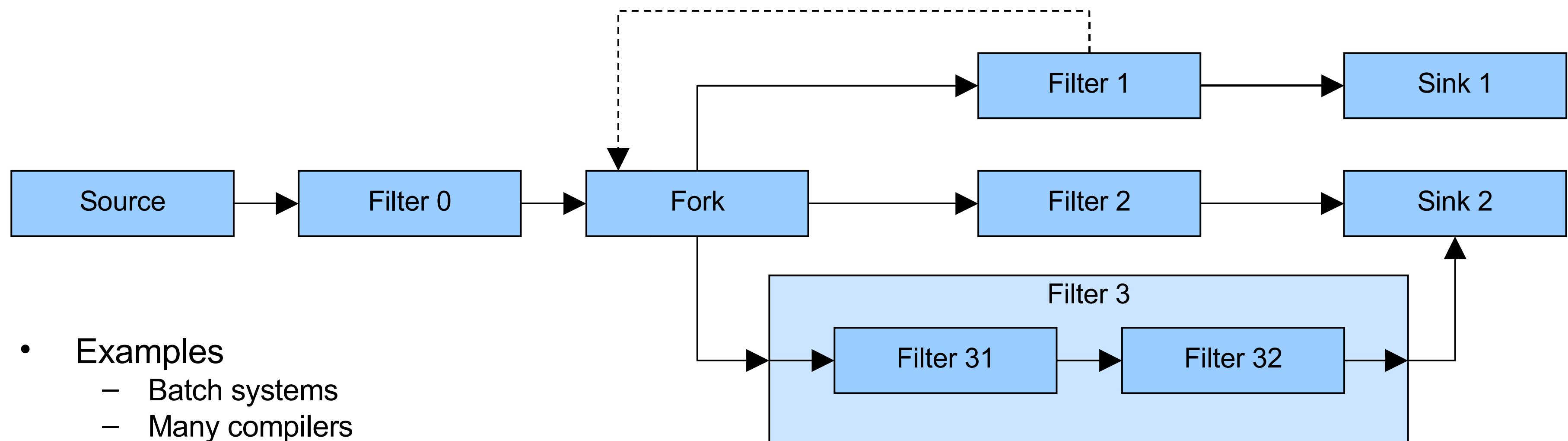
- Dataflow Systems
  - Batch sequential, Data Flow Network, Pipes and Filters
- Call-and-return Systems (*explicit calls*)
  - Main programs and subroutines, Abstract Data Types, OO systems, Client-Server, Layered (hierarchical layers), Three-tier
- Independent Components (*implicit calls*)
  - Communicating processes, Event-driven systems, SOA
- Virtual Machines
  - Interpreters, Rule-based systems
- Data-Centered Systems (Repositories)
  - Database-centric, Hypertext systems, Blackboards
- Complex (Compound) Styles
  - REST, C2 (Chiron-2), …

Copyright © Alar Raabe 2011

Dataflow System

shared nothing !

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



- Examples
  - Batch systems
  - Many compilers
  - Unix pipelines
  - Spreadsheets
  - JDPF (Java Data Processing Framework)
  - Signal and Graphic processors

Copyright © Alar Raabe 2011

Dataflow System – Evaluation

shared nothing !

- Advantages
  - Modifiability & Reuse (low coupling, encapsulation)
    - Filters can be treated as black boxes
  - Ease of construction
    - System can be hierarchically composed (new higher order filters can be created by combining lower order pipes and filters, etc.)
  - Flexibility
    - Construction (system configuration) can often be delayed until runtime (late binding)
  - Run-time scalability
    - It is easy to run a pipe-and-filter system on parallel processors
  - Understandability/Analyzability
    - Supports well certain analyses (throughput, latency, deadlock)

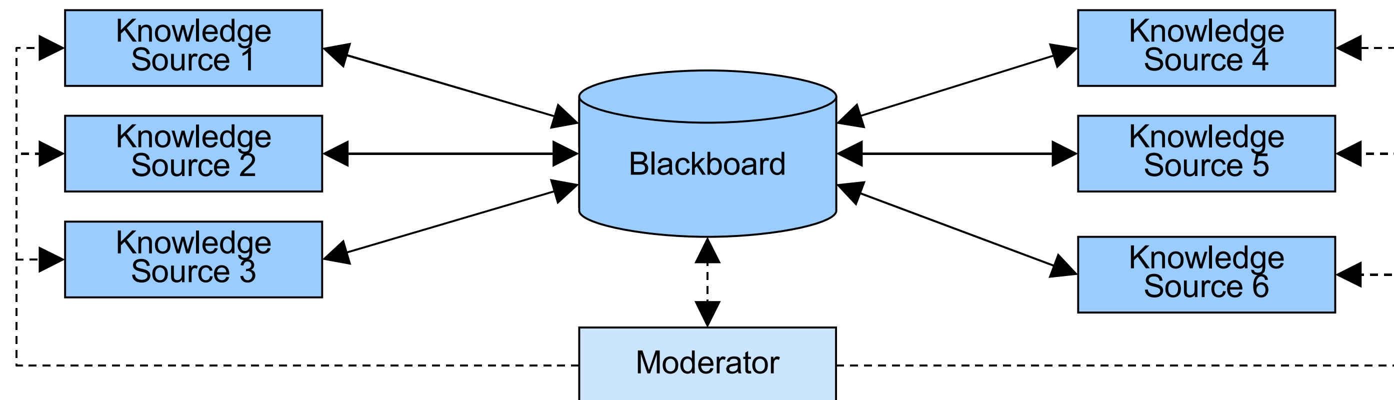
- Disadvantages
  - Difficult to create interactive applications
  - Common data representation
    - The lowest common denominator (typically byte or character streams)
  - Parsing overhead
    - Every filter may introduce parsing and un-parsing of the data stream
  - Unknown memory requirements and deadlock possibility (e.g. sort filter has this problem)

Copyright © Alar Raabe 2011

Data-Centered Systems (Repositories)

shared everything !

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



- Examples
  - Many expert systems (e.g. Hearsay II)
  - Many language compilers and IDEs
  - Systems with global database
  - GBBopen (based on Common Lisp)
  - Java Spaces
  - Blackboard Event Processor (JVM-based, JavaScript, Jruby)

Copyright © Alar Raabe 2011

Data-Centered Systems (Repositories) – Evaluation

shared everything !

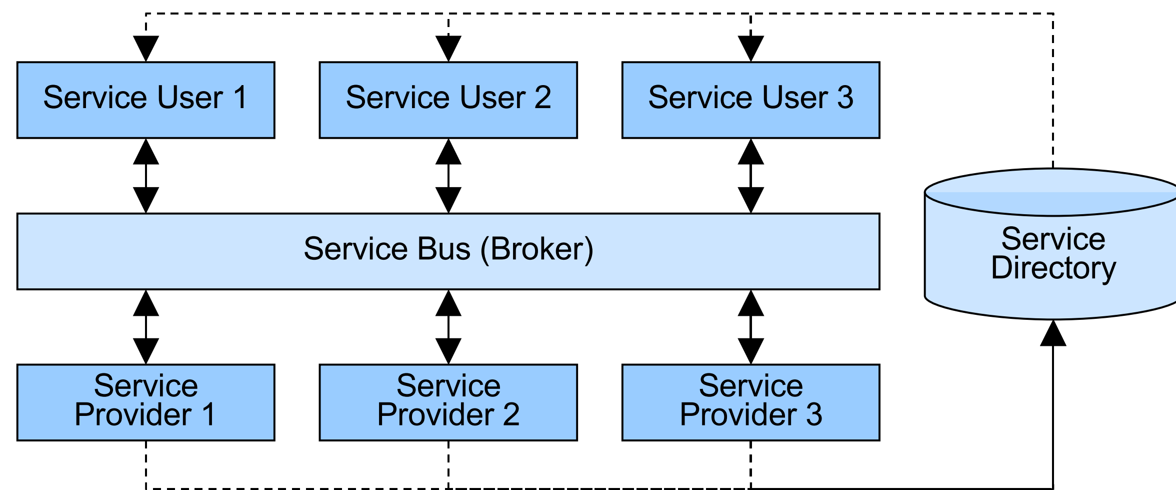
- Advantages
  - Scalability
    - Easy to add more knowledge sources
    - Knowledge sources can run in parallel and are synchronized through the central repository
  - Separation of concerns (problem partitioning)
    - Each knowledge source performs separate function
    - Each knowledge source solves part of the problem
  - Coupling
    - Loose coupling between knowledge sources
  - Modifiability
    - Knowledge sources can be modified independently
- Disadvantages
  - Scalability
    - Blackboard becomes bottleneck with too many knowledge sources
  - Coupling
    - Tight coupling between knowledge sources and blackboard
  - Understandability/Analyzability
    - Difficult to analyze – non-deterministic behavior
    - System behavior emerges from the behaviors of knowledge sources

Copyright © Alar Raabe 2011

Independent Components (SOA)

bus and directory are optional !

- Independent Components – Service-Oriented Architecture (SOA)
  - Components (providers, users/consumers; opt. *bus, directory*)
  - Connectors (synchronous and asynchronous calls, messages)
  - Constraints (call style, …)
  - Theory (CSP (C.A.R. Hoare), π-calculus (Millner, Parrow), …)



- Examples
  - CORBA (IIOP)
    - ORB is bus
    - Naming Service is directory
  - DCOM (RPC)
  - Jini (RMI) – LUS as directory
  - Web Services (HTTP) – UDDI as directory
  - ESB (Mule, Apache ServiceMix)

Copyright © Alar Raabe 2011

Independent Components (SOA) – Evaluation

bus and directory are optional !

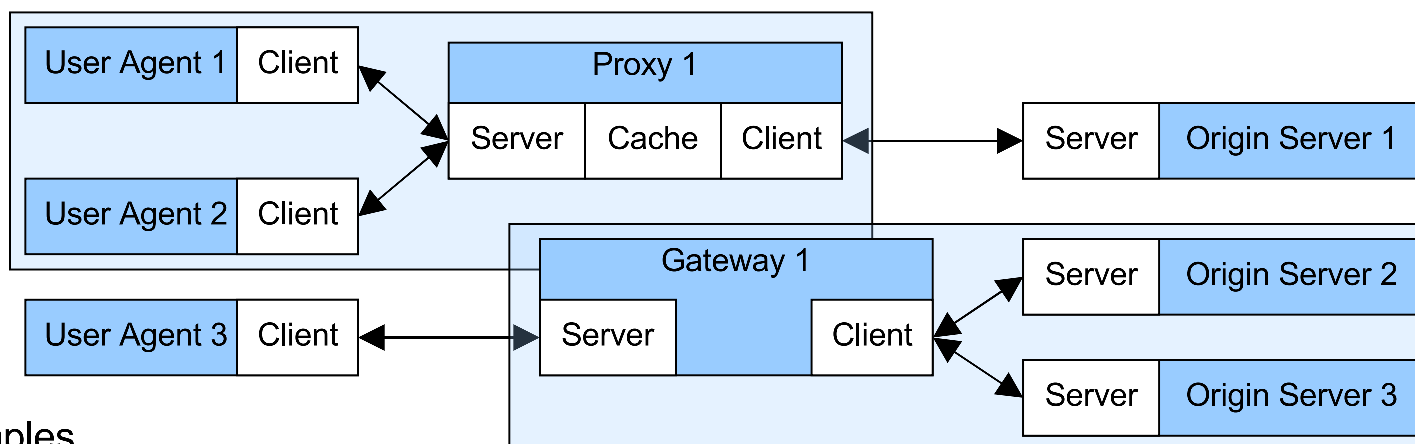
- Advantages
  - Coupling
    - Loose coupling – specially, if asynchronous calls are used
  - Interoperability
    - Service users can transparently call services implemented in disparate platforms using different languages
  - Modifiability
    - Loose coupling between service users and service providers
    - Services are self-contained and modular
  - Extensibility (adding new services is easy if bus is used)
  - Reliability (good fault tolerance, if asynchronous calls are used)
- Disadvantages
  - Performance
    - Network overhead
    - Overhead of intermediaries (like bus and 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 needed)

Copyright © Alar Raabe 2011

Compound Style (REST)

architecture of web !

- Compound Style – REpresentational State Transfer (REST)
  - 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)



- Examples
  - WWW (World Wide Web)
  - Twitter, Yahoo, Amazon S3 API
  - CMIP/CMOT (Common Management Information Protocol)
  - IBM WebSphere Portal REST API

Copyright © Alar Raabe 2011

Compound Style (REST) – Evaluation

architecture of web !

- Advantages
  - Simplicity
    - No need for explicit resource discovery mechanism due to hyper-linking
  - 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
- Disadvantages
  - Limited functionality
    - Selected uniform interface (HTTP) is difficult for handling real time asynchronous events
  - 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)

Copyright © Alar Raabe 2011

Building a Software Architectural Style – REST

Fielding

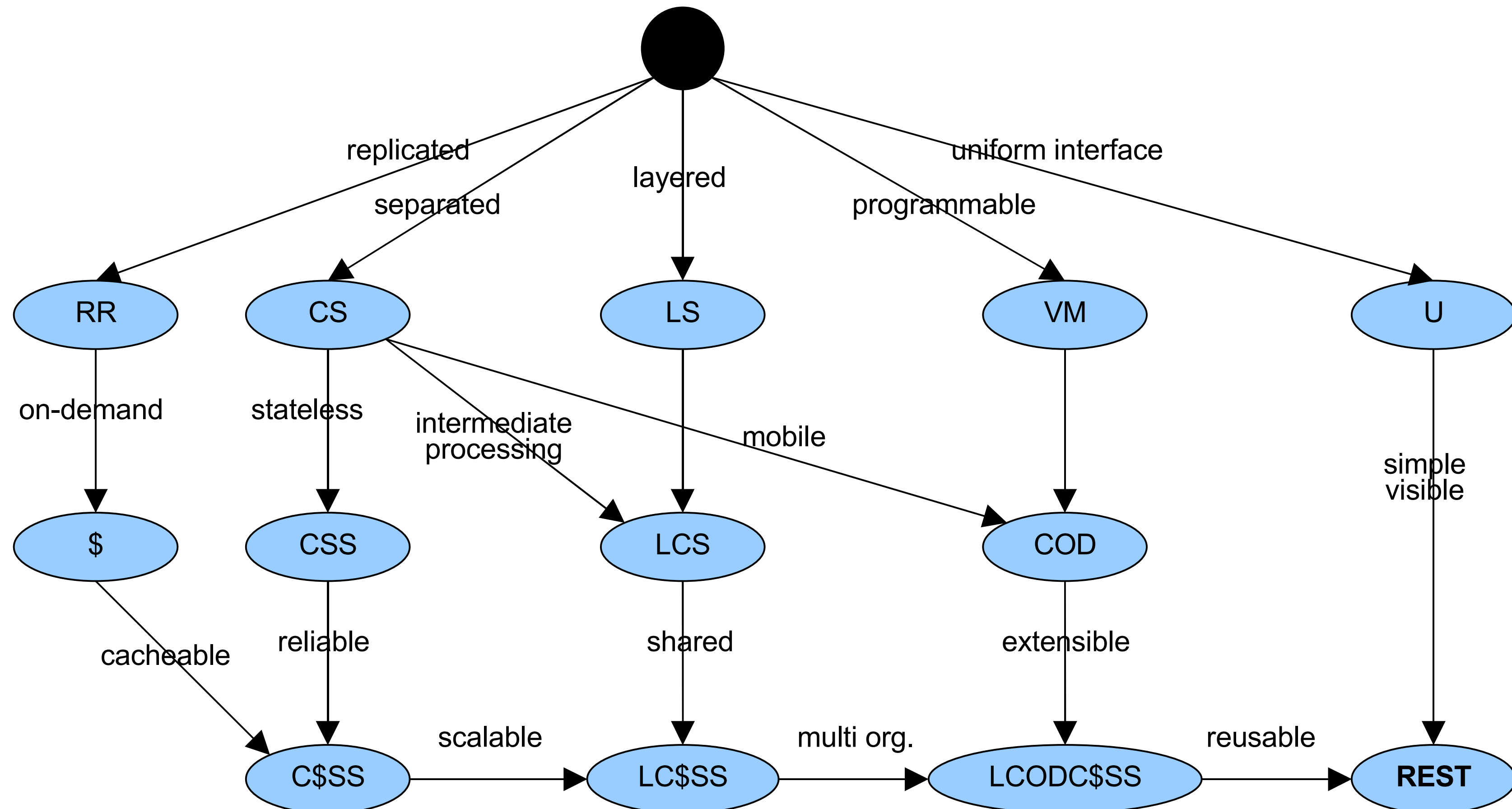
- Desired Properties
  - Performance (network and user-perceived performance, and network efficiency)
  - Scalability
  - Simplicity
  - Modifiability (evolvability and extensibility, customizability and configurability, reusability)
  - Visibility
  - Portability
  - Reliability

- Constituent Architecture Styles
  - Null Style – an empty set of constraints
  - Client-Server Style (CS) – separation of concerns → modifiability, independent evolution
  - Stateless Communication (S) – session state in client → visibility, reliability, scalability
  - Cache ($) – a variant of Replicated Repository (RR) → network efficiency
  - Uniform Interface (U) – a constrained set of well defined operations and content types → simplicity, portability
  - Layered System Style (LS) – hierarchical decomposition, managing complexity → simplicity, scalability
  - {*optional*} Code-on-Demand (COD) – simplified clients, but lower visibility → modifiability (extensibility), simplicity

Copyright © Alar Raabe 2011

Deriving REST from Constituents

Fielding



 Copyright © Alar Raabe 2011

Emerged Architecture – Big Ball of Mud !

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

- 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-time 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 – ***mostly business concerns !***
  - Quick to make → Time-to-Market
  - Cheap to make → Cost vs. Benefit
  - Does not need governance – just emerges
  - Does not need skills
- Disadvantages – ***mostly IT concerns !***
  - Maintainability – difficult and costly to maintain
  - Modifiability – hard and dangerous to change
  - Testability – difficult to test

Copyright © Alar Raabe 2011

Software Quality Attributes

architecture is the primary carrier of quality attributes

- Quality
  - fitness for use (J. M. Juran)
- (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

- Software Quality Attributes
  - Categorizations/Taxonomies of Software Quality Attributes
  - Tradeoffs between Quality Attributes
- Building for Software Quality Attributes
  - Architecture-Centric Methods – ATAM

Copyright © Alar Raabe 2011

Comparing CMU SEI and ISO/IEC 9126

- 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) Schedul

- ISO/IEC 9126
  - End User's View
    - *Functionality*
      - Suitability, Accuracy, Interoperability, *Security*
    - *Reliability*
      - Maturity, Fault Tolerance, *Recoverability*
    - *Usability*
      - Understandability, Learnability, Operability, Attractiveness
    - *Efficiency*
      - *Time Behavior, Resource Utilization*
  - Developer's View
    - Maintainability
      - Analyzability, Changeability, Stability, *Testability*
    - *Portability*
      - Adaptability, Installability, Co-Existence, Replaceability
  - Business's View
    - ***??? MISSING ???***

Copyright © Alar Raabe 2011

Quality Attribute Tradeoff Points

you can’t eat your cake and have it too !

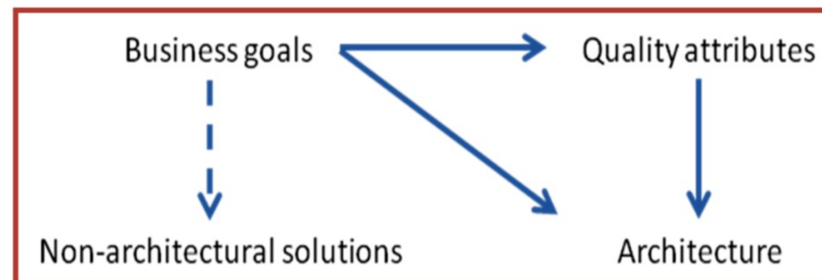
	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Copyright © Alar Raabe 2011

Architecture-Centric Methods

CMU SEI

- A Family of Scenario-Driven and Quality Attribute Driven Development Methods
  - Software Architecture Analysis Method (SAAM)
  - Architecture Tradeoff Analysis Method (ATAM)
    - To assess the consequences of architectural decision alternatives in light of quality attribute requirements – uses scenarios
  - Quality Attribute Workshop (QAW)
  - Cost-benefit Analysis Method (CBAM)
  - Active Reviews for Intermediate Design (ARID)
  - Attribute-Driven Design (ADD)
  - Pedigree Attribute eLicitation Method (PALM)
    - To elicit and capture business goals that lie behind the development of software-intensive system

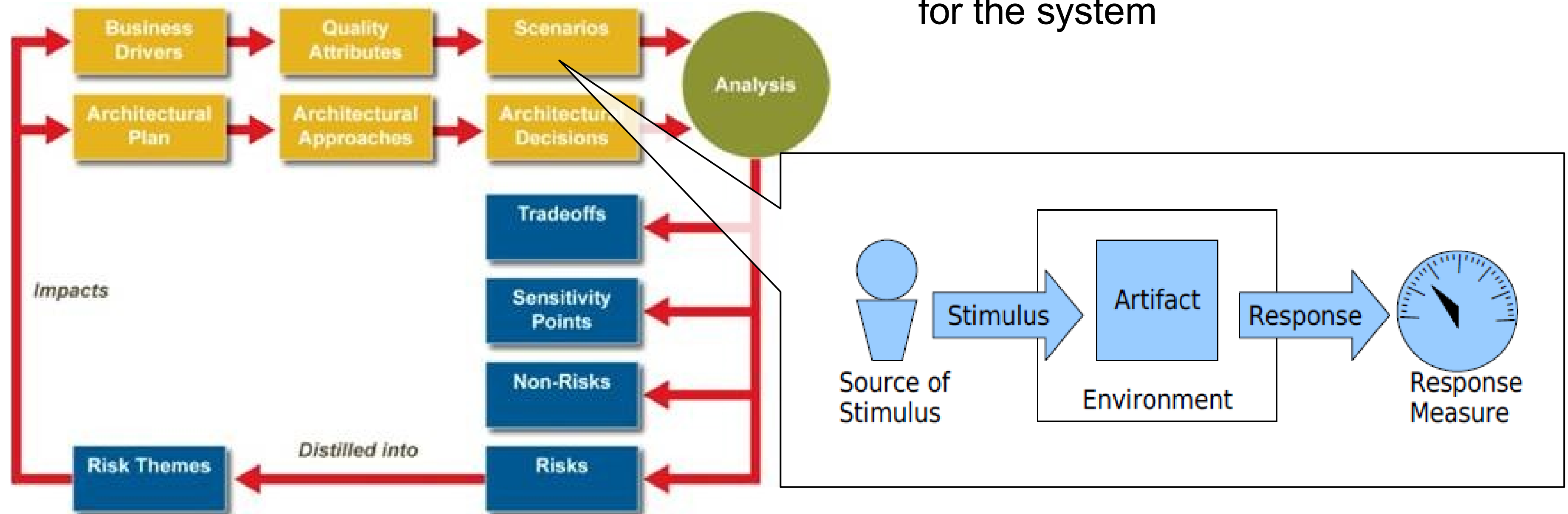


Copyright © Alar Raabe 2011

Architecture Tradeoff Analysis Method (ATAM)

CMU SEI

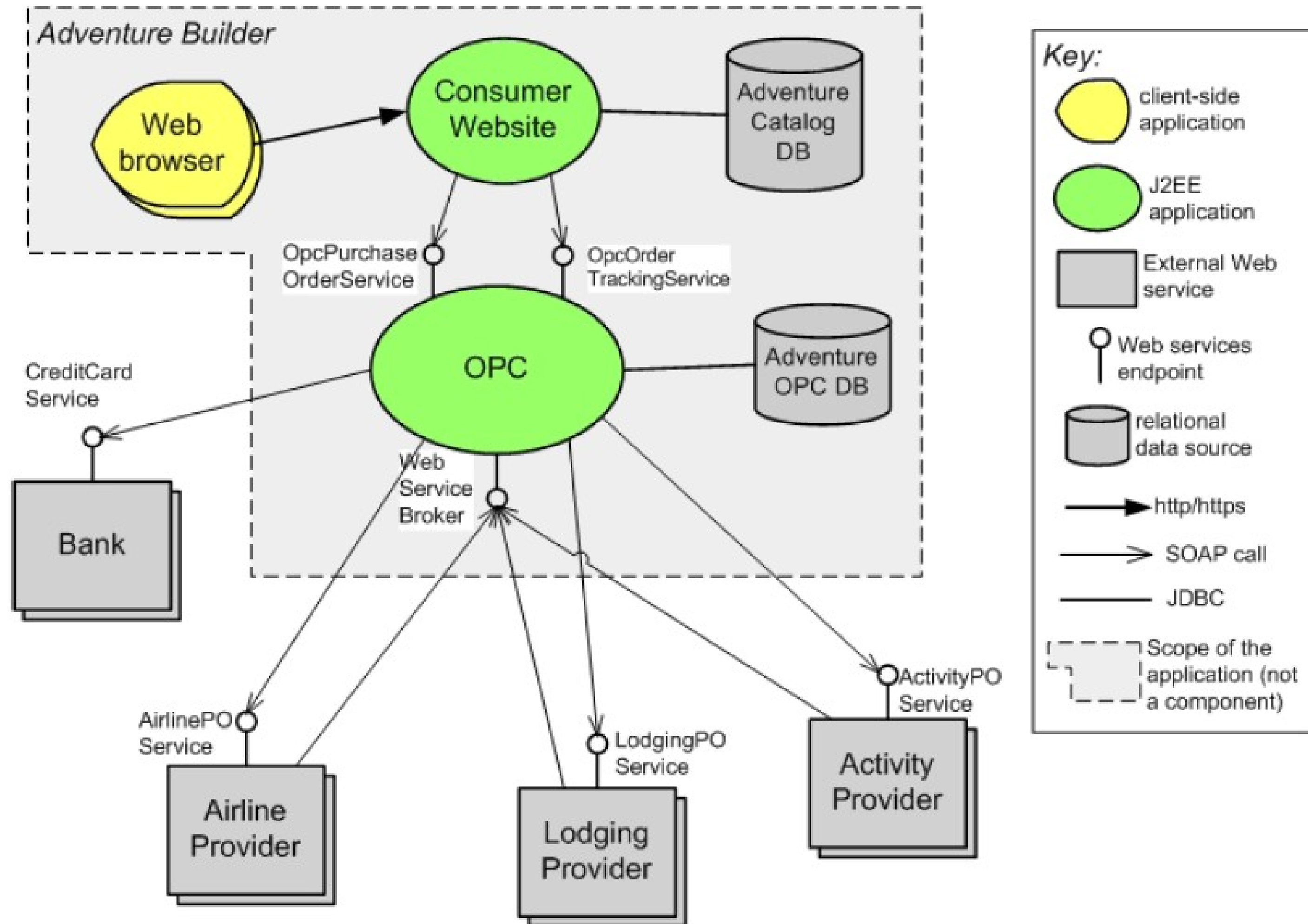
- Output
  - A set of identified architectural approaches
  - “utility tree” – driving architectural requirements
  - The set of scenarios mapped onto architecture
- …
  - A set of quality-attribute specific questions and responses
  - A set of identified risks
  - A set of identified non-risks
  - A set of risk themes that threaten to undermine the business goals for the system



Copyright © Alar Raabe 2011

Example: SOA Quality Attribute Scenario

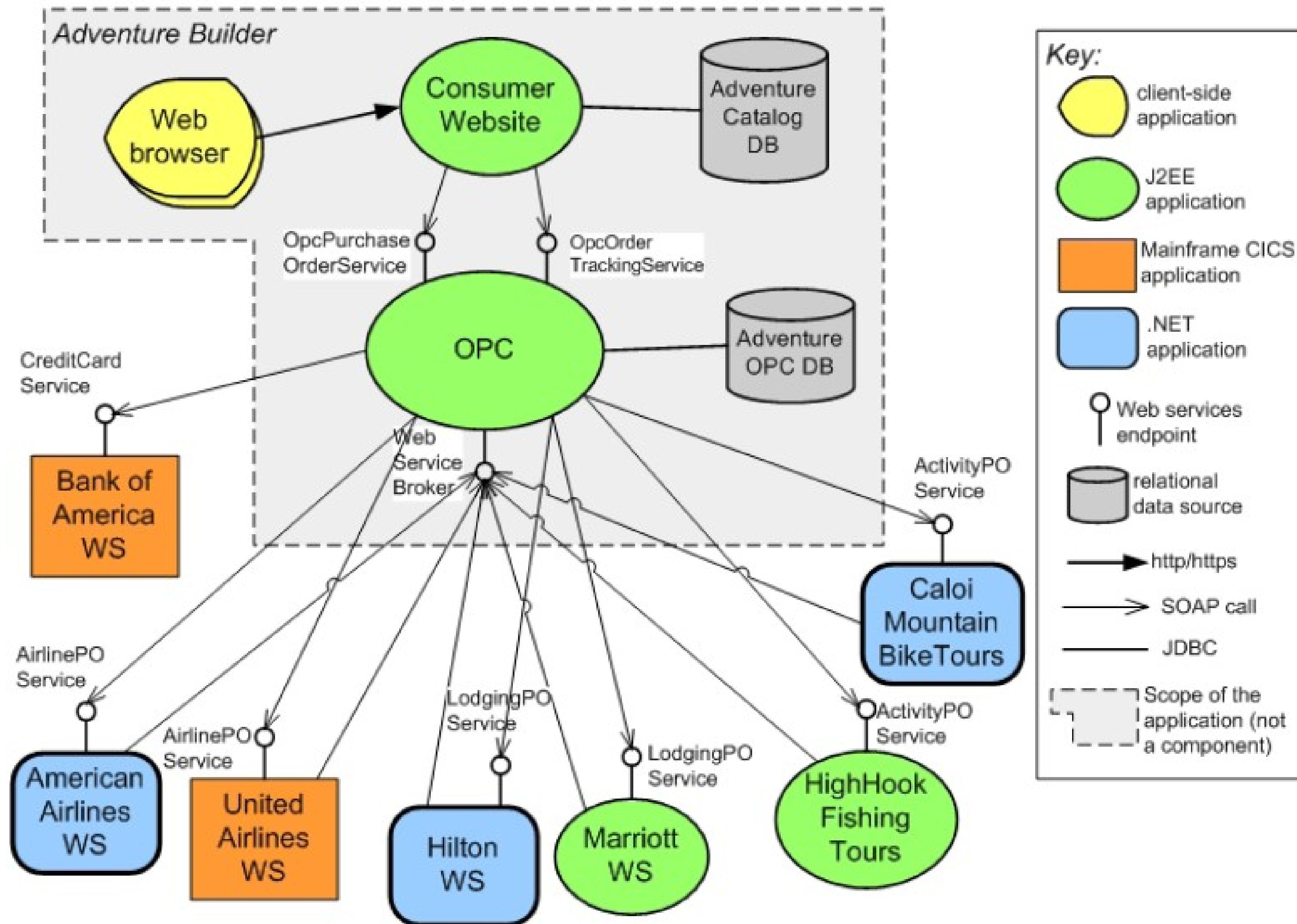
CMU SEI



 Copyright © Alar Raabe 2011

Example: SOA Quality Attribute Scenario

CMU SEI



Copyright © Alar Raabe 2011

Example: SOA Quality Attribute Scenario

CMU SEI

Quality Attribute	
…	…
Scenario 2 Modifiability	• (**Source**) Business Analyst/Customer • (**Stimulus**) Add a new airline provider that uses its own Web services interface. • (**Artifact**) OPC • (**Environment**) Developers have already studied the airline provider interface definition. • (**Response**) New airline provider is added that uses its own Web services. • (**Response Measure**) No more than 10 person-days of effort are required for the implementation (legal and financial agreements are not included).
…	…

Copyright © Alar Raabe 2011

Example: SOA Quality Attribute Scenario

CMU SEI

Analysis for Scenario 2	
Scenario Summary	A new airline provider that uses its own Web services interface is added to the system in no more than 10 person-days of effort for the implementation.
Business Goal(s)	Permit easy integration with new business partners.
Quality Attribute	Modifiability, interoperability
Architectural Approaches and Reasoning	• Asynchronous SOAP-based Web services • Interoperability is improved by the use of document-literal SOAP messages for the communication between OPC and external services. • Adventure Builder runs on Sun Java System Application Server Platform • Edition V8.1. This platform implements the WS-I Basic Profile V1.1, so interoperability issues across platforms are less likely to happen.
Risks	The design does not meet the requirement in this scenario, because it assumes that all external transportation providers implement the same Web services interface called 'AirlinePOService' (as shown in Figure 10 and Figure 11). The design does not support transportation providers that offer their own service interface.
Tradeoffs	The homogenous treatment of all transportation providers in OPC increases modifiability. However, intermediaries are needed to interact with external providers that offer heterogeneous service interfaces, as in this scenario. These intermediaries represent a performance overhead, because they may require routing messages and extensive XML processing.

Copyright © Alar Raabe 2011

Value of Software Architecture

80% of time during maintenance is spent in design-rediscovery

Davidson, 2002

Value of Architecture (Description)

- Users and operators of the system
  - Understand the external system behavior
  - Understand how to operate system
- Acquirers and owners of the system
  - Understand economical issues connected to the system
- Suppliers and developers of the system
  - Plan development and construction
  - Estimate system properties
- Builders and maintainers of the system
  - Understand the system internals

Good Architecture provides

- Users and operators of the system
  - High availability and performance
  - Survival from partial failure
- Acquirers and owners of the system
  - Easy integration into environment
- Suppliers and developers of the system
  - Speed and freedom
  - Guidance
  - Reuse of effort, skills and know-how
  - Ease of integration
- Builders and maintainers of the system
  - Survival of extension, adaptation, requirements changes, platform changes, etc.

Copyright © Alar Raabe 2011

Measuring Value of Software Architecture

> Focus on quality and cost will decrease
> Focus on costs and quality will decrease
>
> W. E. Deming

- Value of Software Architecture
  - Cost of realization of risks compared to cost of architecture

$$value_{arch} = \sum_{i=1}^{n} \left(cost_{risk} \left(concern_i \right) \right) - cost_{arch}$$

- Value of Software Architecture Description
  - Cost of performing activities without architecture description compared to cost of documenting architecture

$$value_{arch.desc} = \sum_{i=1}^{n} \left(cost_{performing} \left(activity_i \right) \right) - cost_{arch.desc}$$

Copyright © Alar Raabe 2011

Real Options for Valuation of Software Architecture

- Option is
  - A right, but not obligation to make a decision in the future
  - Difference from financial option (American call) – might be exercised multiple times
- Applicable when there is
  - ***Uncertainty***
  - ***Business goal***
    - Uncertainty is important for managing or achieving a business goal
  - ***New information***
    - New information should be exploited when it comes available
  - ***Action*** today should create
    - Possibility of ***future design choices***
    - Possibility of ***future value***
- Strategic Value with Real Options

$$NPV_{strategic} = NPV_{traditional} + Value_{real.options}$$

- Valuation of real options
  - Decision trees with probabilities (Markov processes)
  - Dynamic programming algorithms
  - Monte Carlo simulations

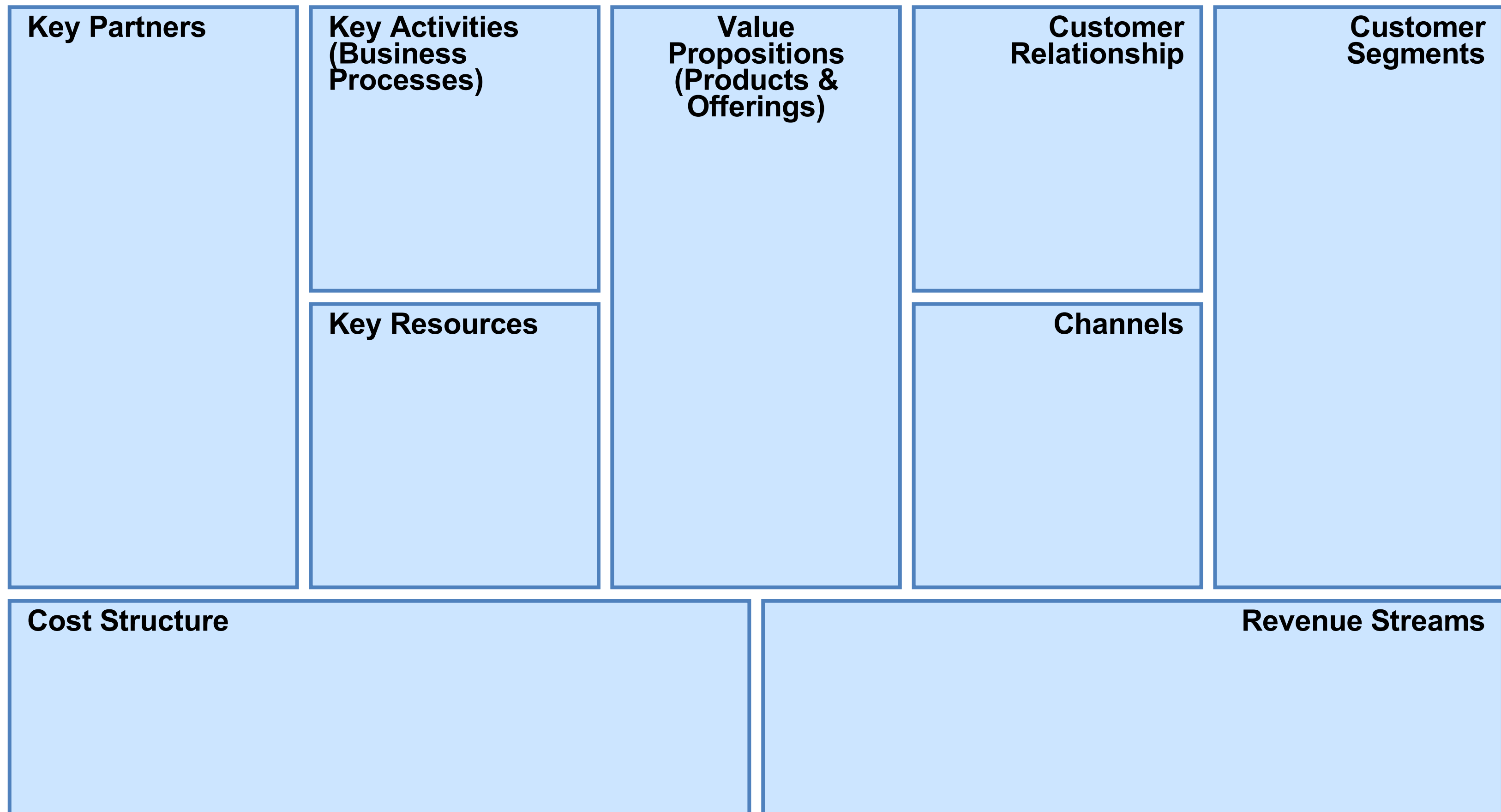
Copyright © Alar Raabe 2011

Economic Value of Architecture Decisions

- Real Option Theory (Sullivan)
  - Qualitative Design Principles
    - If at any time, the NPV of future profits is at least by value of investment more than the direct costs, then commit to the design decision, otherwise not
    - If the expected PV of the future profits that would flow from choice exceeds the direct cost of implementing it, then implement the choice, otherwise implement some other choice
    - If the expected PV of future profits that would flow from restructuring exceeds the direct cost of restructuring, then restructure, otherwise do not
    - If the cost to effect a software decision is sufficiently low, then the benefit of investing to effect it immediately outweighs the benefit of waiting, so the decision should be made immediately
    - With other factors, including the static NPV, remaining the same, the incentive to wait for better information before effecting a design decision increases with the risk (i.e. with the spread in possible benefits)
    - The incentive to wait before investing increases with the likelihood of unfavorable future events occurring
    - All else being equal, the value of the option to delay increases with variance in future costs

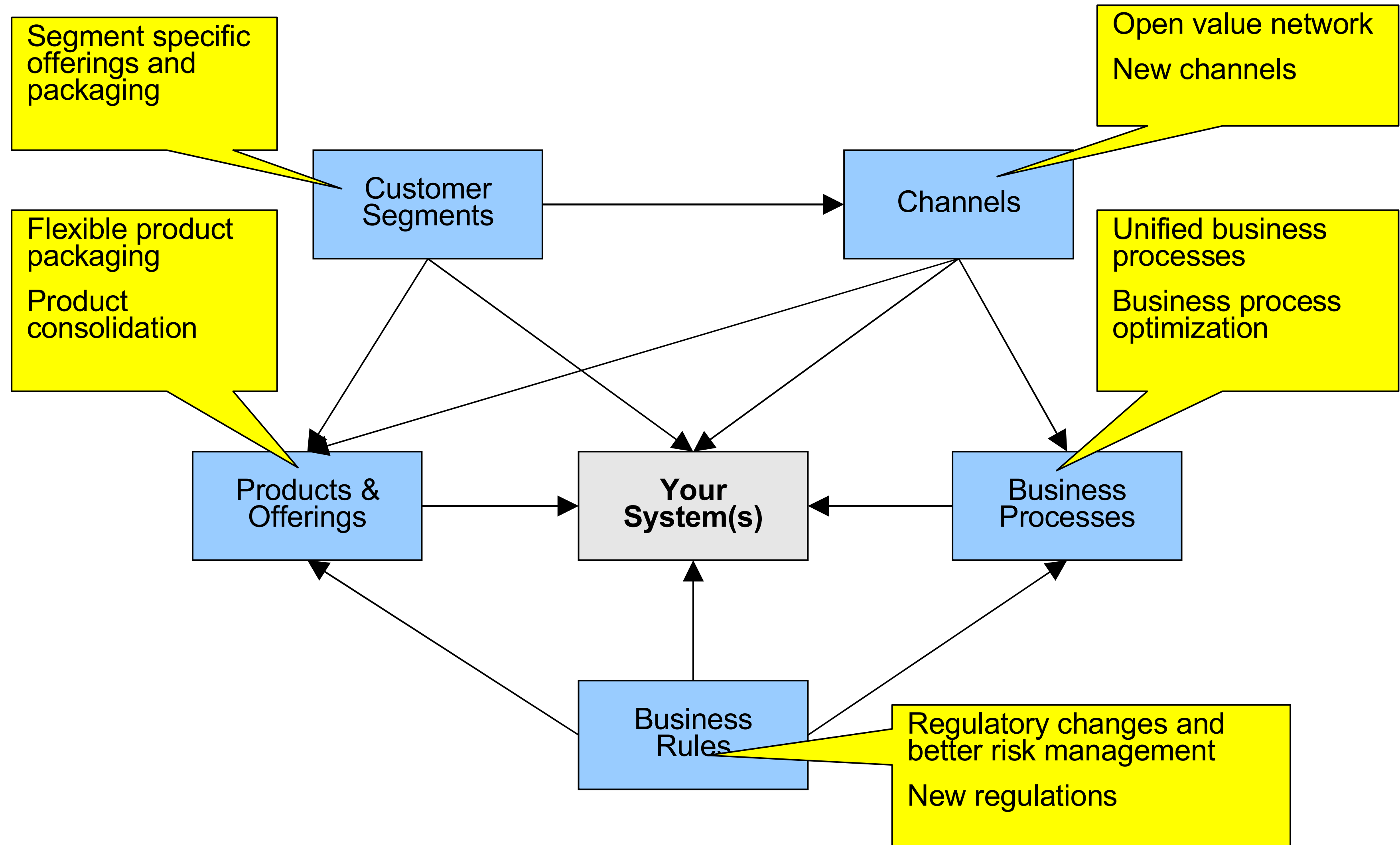
Copyright © Alar Raabe 2011

Business Model – How Business Works (Osterwalder)



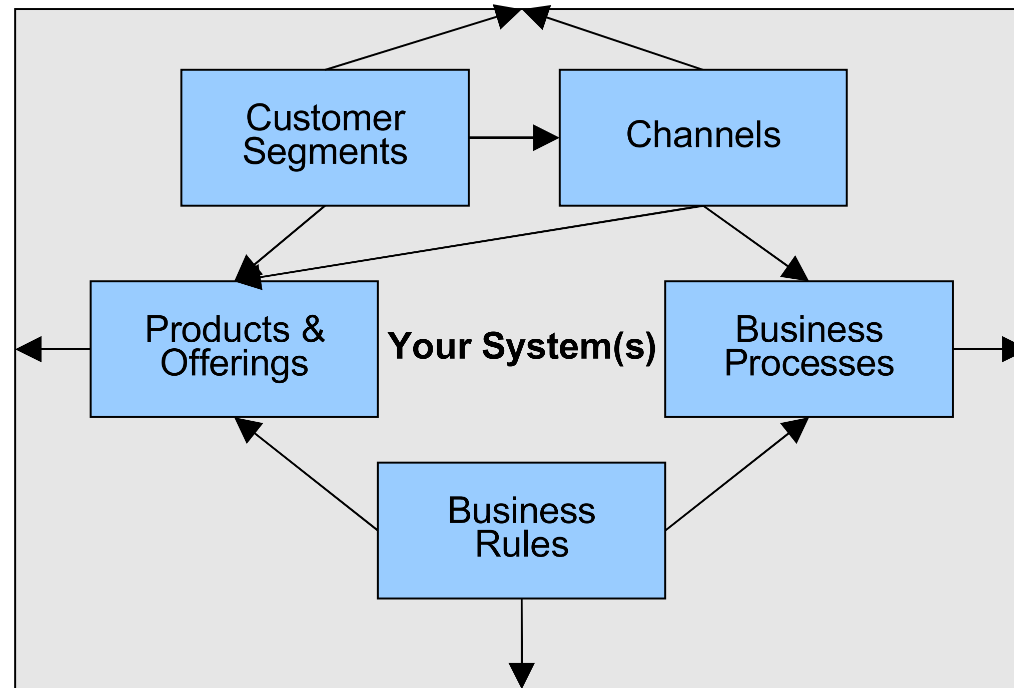
Copyright © Alar Raabe 2011

What Changes in Business and How Often ?



Copyright © Alar Raabe 2011

Make it Easy to Change – Make it Explicit !



Copyright © Alar Raabe 2011

Conclusions

not documenting, but understanding !

- 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 (software) 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

- Economic Value of Architecture
  - Architecture creates choices/options, which have value – designing and building an architecture is an investment activity

Copyright © Alar Raabe 2011

Conclusions

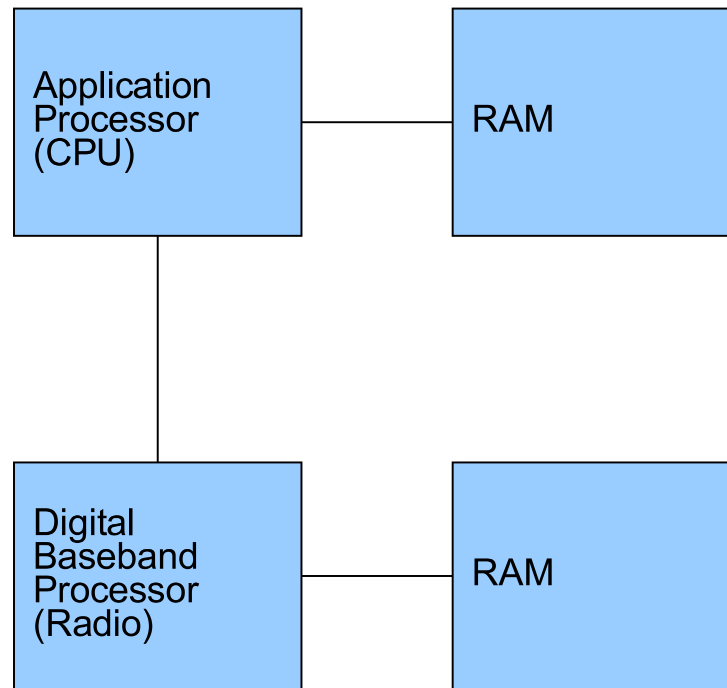
find out who are your business counterparties and communicate !

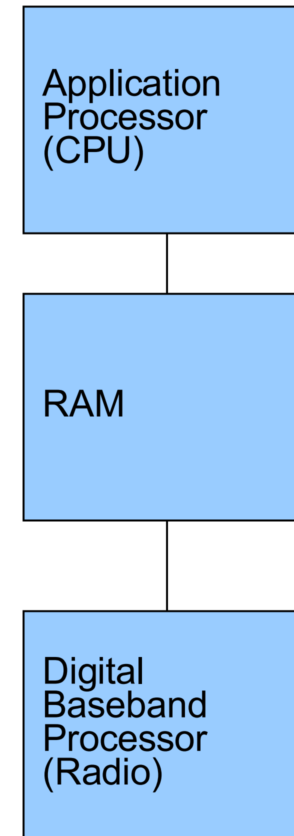
- Have rationale for your architecture
  - Connect your architecture descriptions (especially decisions) to the business goals and requirements
- Speak business language
  - Show the value (payback) of investments into architecture
- Gather data from development and operation
  - You will need this data to build a business case for architecture (to show the value of architecture)
- Find out from business
  - What will (or could) change
  - How probable and how frequent is the change
- Separate what will change from what will not, and group together things that change with same rate

Copyright © Alar Raabe 2011

Extra: Smartphone Architectures

do you know which one ?
who cares ?





Copyright © Alar Raabe 2011

Copyright © Alar Raabe 2011

Leftovers

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

Copyright © Alar Raabe 2011

Terms (Glossary)

ISO/IEC 42010

Term	Definition
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
architecture-related concern	area of interest in a system pertaining to developmental, technological, business, operational, organizational, political, regulatory, social, or other influences important to one or more of its stakeholders
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}*

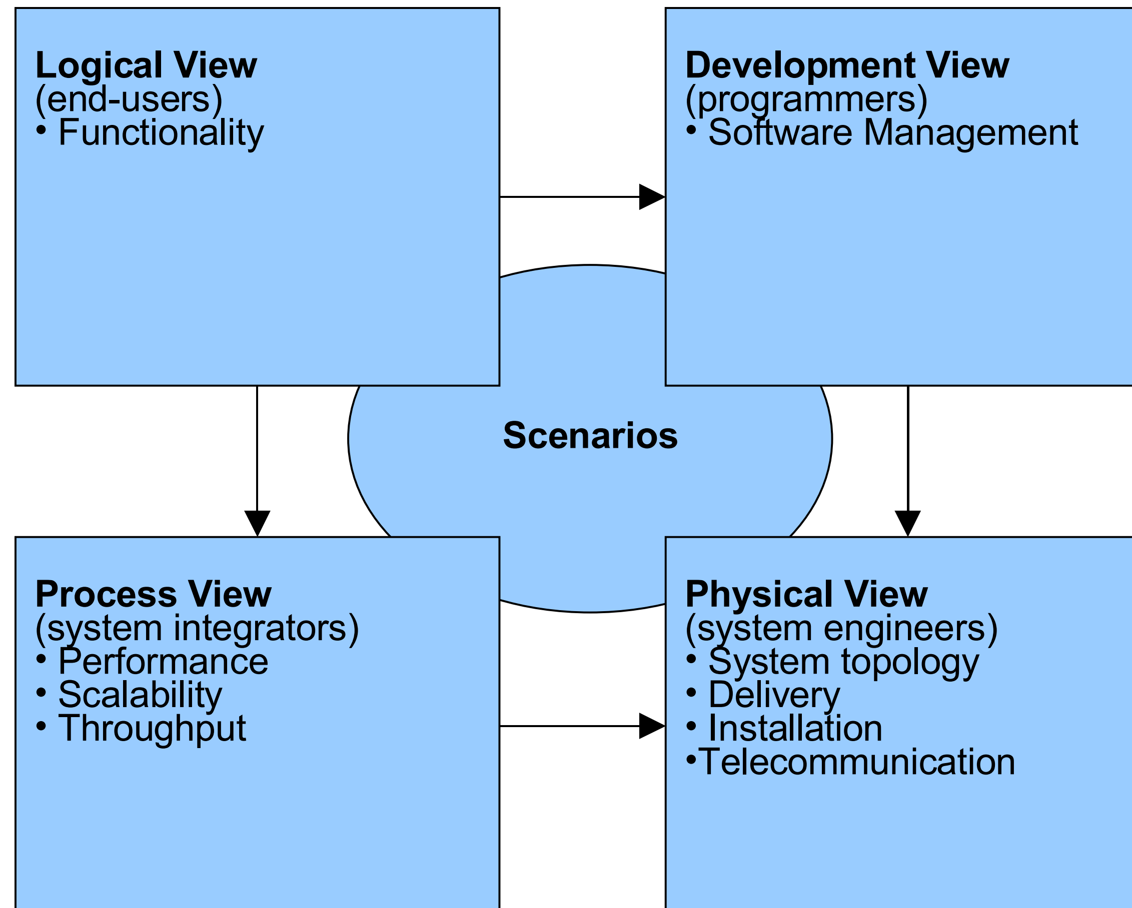
Copyright © Alar Raabe 2011

Viewtypes and Styles in Architecture Description

- CMU SEI
  - The Module
    - Decomposition
    - Uses
    - Generalization
    - Layered
  - Component-and-Connector
    - Datastream
    - Call-Return
    - Shared-Data
    - Publish-Subscribe
    - Communicating Processes
  - Allocation
    - Deployment
    - Implementation
    - Work Assignment
- Others
  - RUP / Kruchten 4+1
    - Logical view
    - Process view
    - Deployment view
    - Implementation view
    - Use-Case view
  - Siemens Four Views
    - Conceptual
    - Module
    - Code
    - Execution
  - C4ISR Framework
    - Operational architecture
    - System architecture
    - Technical Architecture

Copyright © Alar Raabe 2011

4+1 Views (Kruchten)



Copyright © Alar Raabe 2011

Architectural Design Decisions

- Kinds of Architectural Design Decisions
  - Existence Decisions (*ontocrises*)
    - Structural decisions
    - Behavioral decisions
    - Ban or non-existence decisions (*anticrises*)
  - Property Decisions (*diacrises*)
    - Constraints
    - Design rules
    - Guidelines
  - Executive Decisions (*pericrises*)
    - Organizational decisions
    - Process decisions
    - Technology decisions
    - Tool decisions

- Attributes of Architectural Design Decisions
  - Epitome (the Decision itself)
  - Rationale (“why”)
  - Scope
  - State (idea, rejected, tentative/challenged, decided, approved)
  - Author, Time-Stamp, History
  - Categories (usability, security, …)
  - Cost
  - Risk

- Relationships between Architectural Design Decisions
  - Constraints
  - Forbids (Excludes)
  - Enables
  - Subsumes
  - Conflicts with (mutually excluding)
  - Overrides
  - Comprises (is made of, decomposes into)
  - Is bound to (strong)
  - Is an alternative to
  - Is related to (weak)
  - Dependencies

- Relationship with External Artifacts
  - Traces to
  - Does not comply with

Copyright © Alar Raabe 2011