An Introduction to Software Architecture and Design II

Dr. Mark C. Paulk

Mark.Paulk@utdallas.edu, Mark.Paulk@ieee.org

Some More Architectural Questions

How do we document architectures?

The agile methods have deprecated design… or at least design documentation.

- **If using agile, do we need to worry about architecture?**
- **What level of architectural documentation is needed / appropriate?**

If architecture should be viewed from the system's goals, how do we get those goals?

- **How can we select architecture tradeoffs in light of business goals?**

What is the future of architecture?

An Example Architecture

See wiki.sei.cmu.edu/sad/ for an example of a software architecture.

- **Adventure Builder – Software Architecture Document**

Includes

- **use cases (4)**
- **module views (5)**
- **C&C views (3)**
- **allocation views (2)**

Example software architecture done for Documenting Software Architectures, Views and Beyond, Second Edition (2010) by P. Clements, et al.

Problem Source

Adventure Builder Reference Application

- **Adventure Builder is a fictitious company that sells adventure packages for vacationers over the Internet.**

An adapted version of the Adventure Builder Reference application.

- **developed in the context of the Java BluePrints program at Sun Microsystems**
- **functionality is easy to understand**
- **source code, documentation, and other artifacts are publicly available for download.**
- **Singh book on Web services (2004) explains the design and implementation of the application**

Use Cases (UC1)

The user can visit the Adventure Builder Web site and browse the catalog of travel packages. Includes

- **flights to specific destinations**
- **lodging options**
- **activities that can be purchased in advance**

Activities include

- **mountain biking**
- **fishing**
- **surfing classes**
- **hot air balloon tours**
- **scuba diving**

The user can select transportation, accommodation, and various activities to build his/her own adventure trip.

Use Cases (UC2)

The user can place an order for a vacation package.

To process this order, the system has to interact with several external entities.

- **A bank will approve the customer payment.**
- **Airline companies will provide the flights.**
- **Lodging providers will book the hotel rooms.**
- **Businesses that provide vacation activities will schedule the activities selected by the customer.**

Use Cases (UC3)

After an order is placed, the user can return to check the status of his/her order.

- **This is necessary because some interactions with external entities are processed in the background and may take hours or days to complete.**

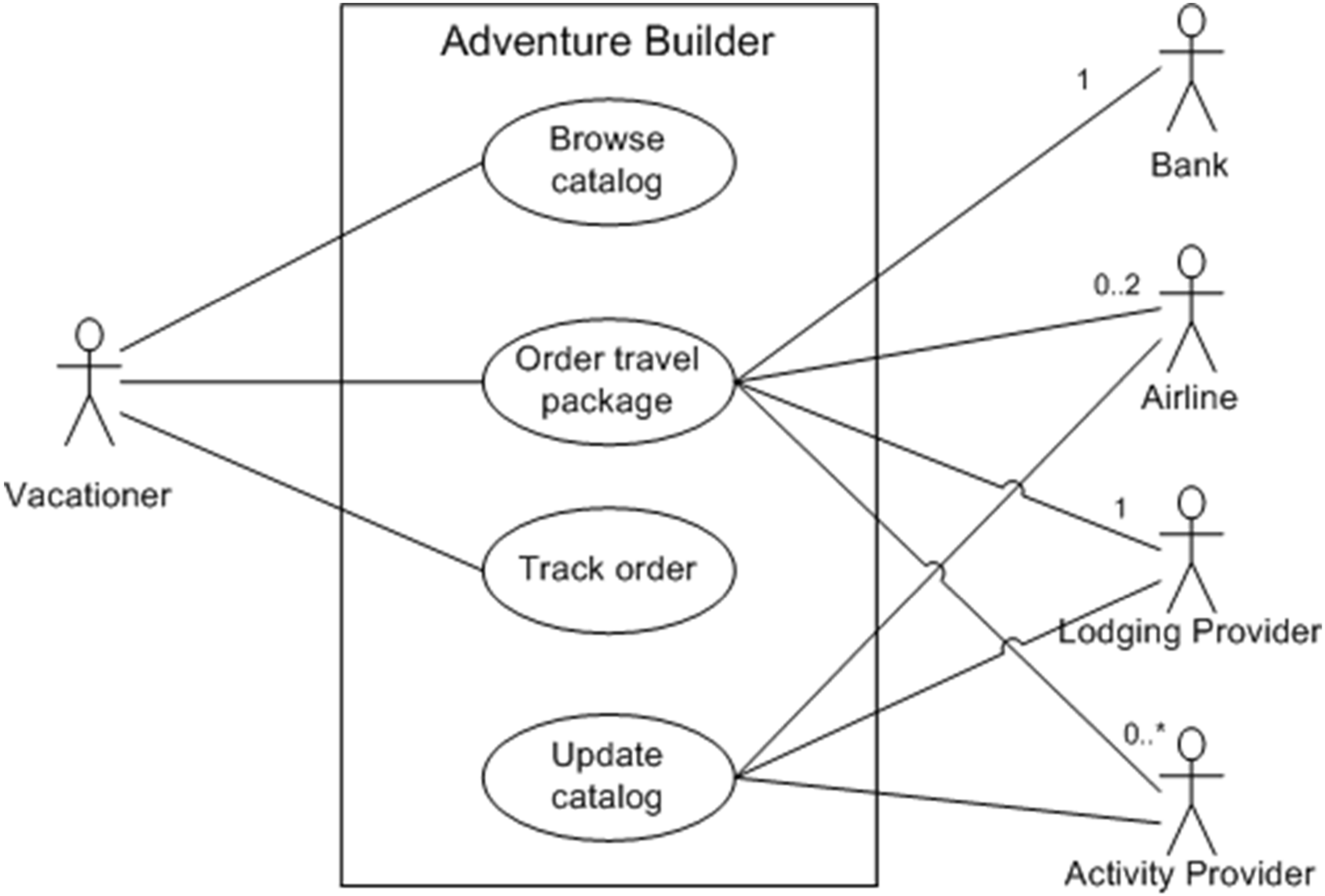
Use Cases (UC4)

The internal system periodically interacts with its business partners

- **transportation**
- **lodging**
- **activity providers**

to update the catalog with the most recent offerings.

Use Cases *(A Use Case Context Diagram)*



Quality Attribute Scenario Modifiability

A new business partner (airline, lodging, or activity 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.

The business goal is easy integration with new business partners.

Quality Attribute Scenario Performance

A user places an order for an adventure travel package to the Consumer Web site.

The user is notified on screen that the order has been successfully submitted and is being processed in less than five seconds.

Quality Attribute Scenario Performance

Up to 500 users click to see the catalog of adventure packages following a random distribution over 1 minute

- **the system is under normal operating conditions**
- **the maximal latency to serve the first page of content is under 5 seconds**
- **average latency for same is less than 2 seconds**

Quality Attribute Scenario Reliability

The Consumer Web site sent a purchase order request to the order processing center (OPC).

The OPC processed that request but didn’t reply to Consumer Web site within five seconds

- **the Consumer Web site resends the request to the OPC**

The OPC receives the duplicate request

- **the consumer is not double-charged**
- **data remains in a consistent state**
- **the Consumer Web site is notified that the original request was successful**

one hundred percent of the time

Quality Attribute Scenario Security

Credit approval and payment processing are requested for a new order.

In one hundred percent of the cases

- **the transaction is completed securely**
- **cannot be repudiated by either party**

The business goals are to provide customers and business partners confidence in security and to meet contractual, legal, and regulatory obligations for secure credit transactions.

Quality Attribute Scenario Security

The OPC experiences a flood of calls through the Web Service Broker endpoint that do not correspond to any current orders.

In one hundred percent of the times, the system

- **detects the abnormal level of activity**
- **notifies the system administrator**
- **continues to service requests in a degraded mode**

Quality Attribute Scenario Availability

The Consumer Web site is available to the user 24x7.

If an instance of OPC application fails, the fault is detected

- **the system administrator is notified in 30 seconds**
- **the system continues taking order requests**
- **another OPC instance is created**
- **data remains in consistent state**

Views Template

Primary presentation (graphic)

Element catalog

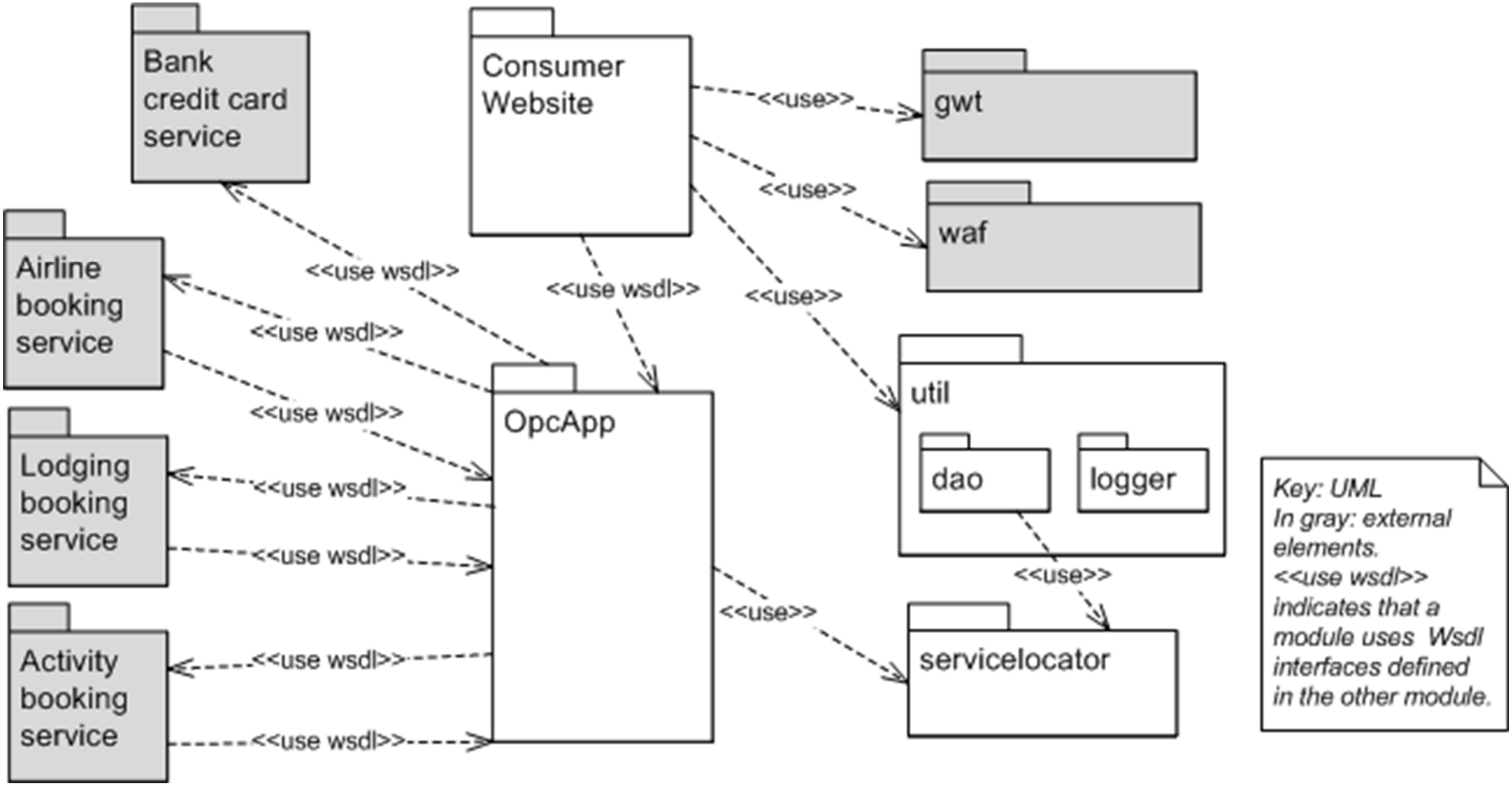
Context diagram

Variability guide

Rationale

Related views

Top Level Module Uses View (1)



Key: UML
In gray: external elements.
<<use wsdl>> indicates that a module uses Wsdl interfaces defined in the other module.

Consumer Website

The web-based user interface of the Adventure Builder is implemented in this module

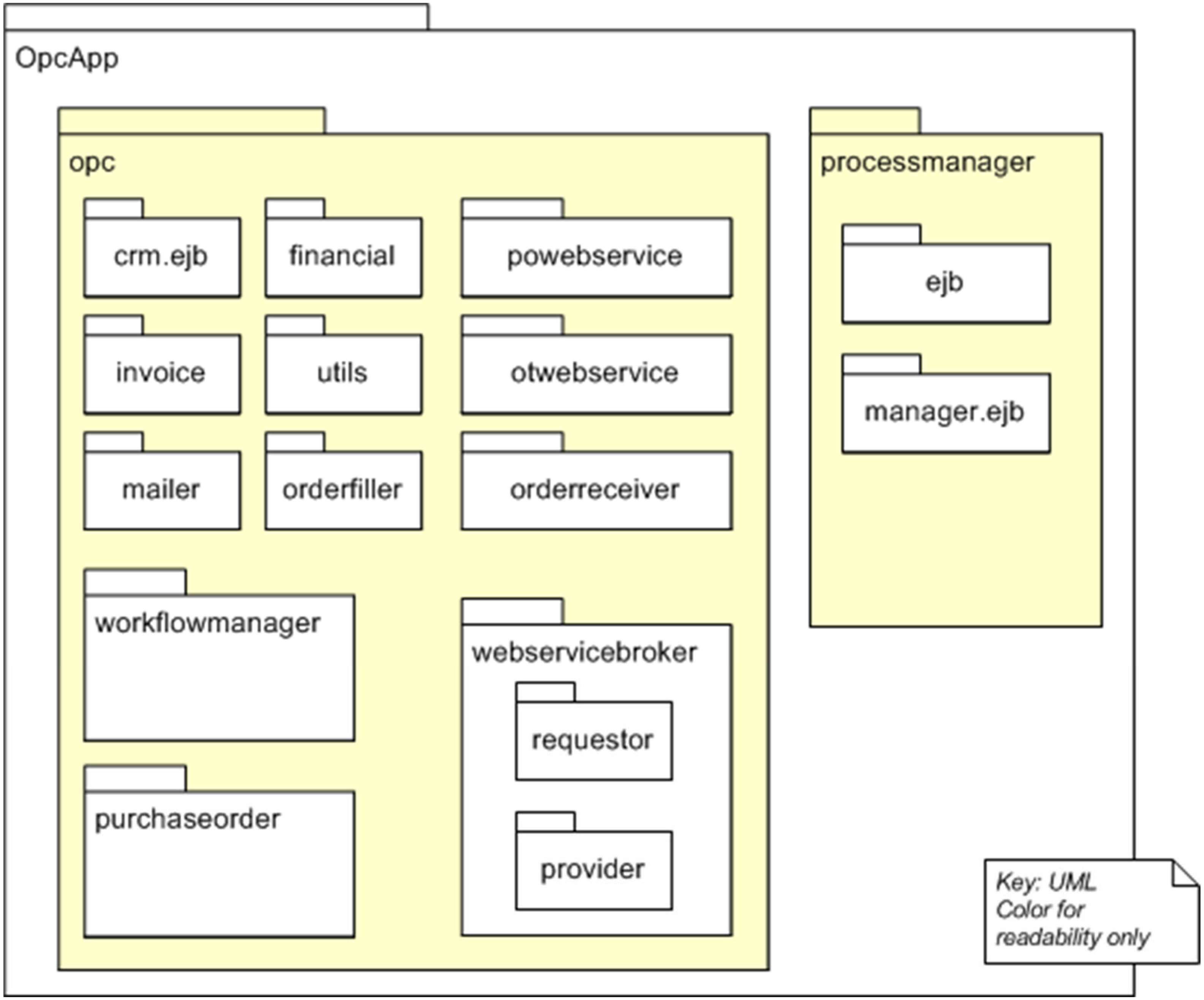
- **lets the user browse the catalog of travel packages**
- **place a new purchase order**
- **track the status of existing orders**
- **creates purchase orders based on user input and passes them to OpcApp for processing**
- **uses an implementation of the Model View Controller pattern called the Web Application Framework (waf)**
  - **model implemented using Entity beans**
  - **controller implemented using servlets**
  - **view is a collection of JSPs and static HTML pages**
- **part of the client-facing code is implemented using the GWT framework**

Order Processing Center Application OpcApp

The business logic of the Adventure Builder is implemented in this module.

- **Accepting purchase order requests from the ConsumerWebsite for processing by hosting the Purchase Order Web Service.**
- **Provide a mechanism for the Consumer Website to query the current status of a purchase order by hosting the Order Tracking Web Service.**
- **Communicate with external suppliers to process and maintain the status of a purchase order.**
- **Upon completion of processing a purchase order, send an email to the customer of its success or failure.**

OPC Module Decomposition View (2)

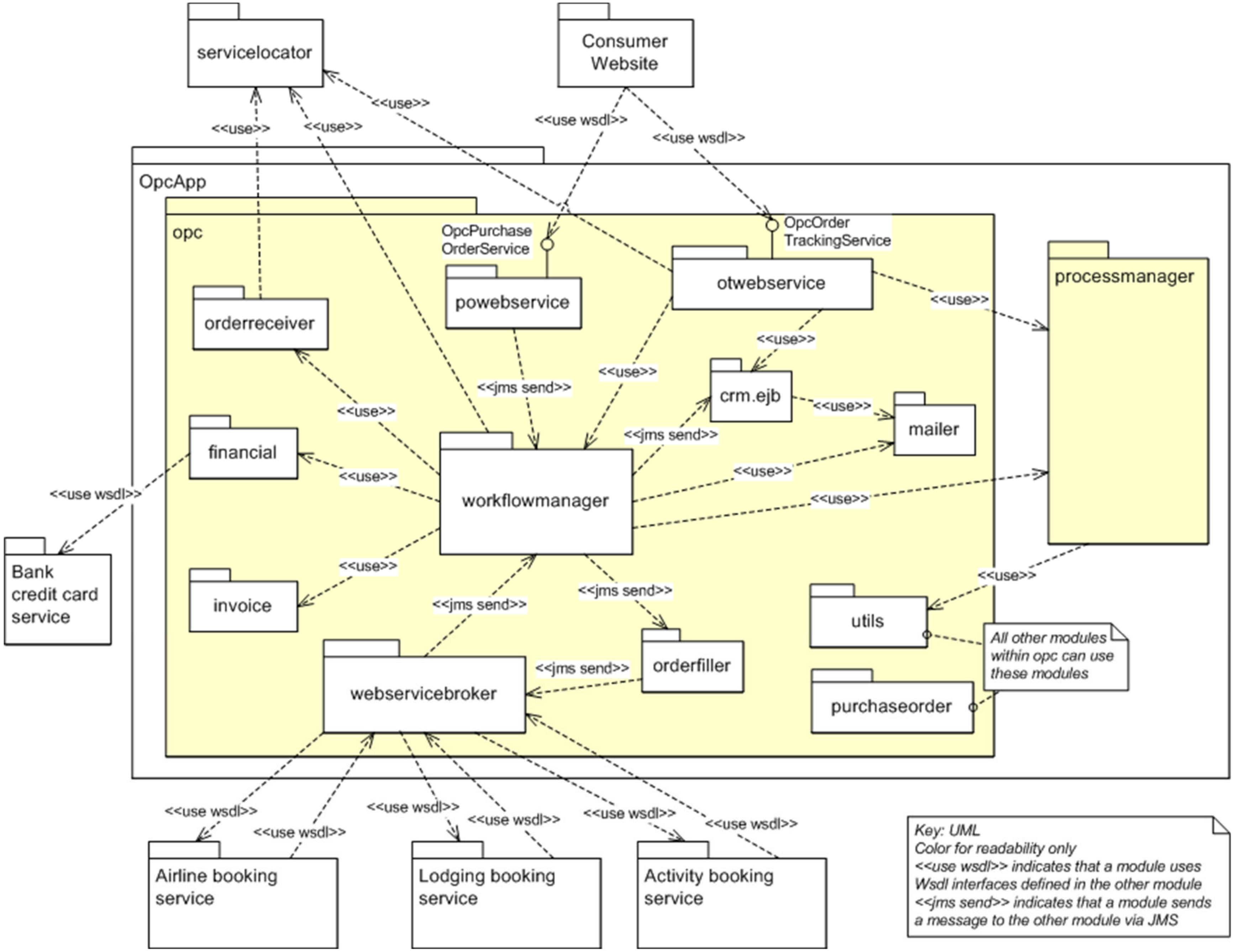


Rationale

The choice of EJBs in the implementation, including session beans, message-driven beans and entity beans is based on:

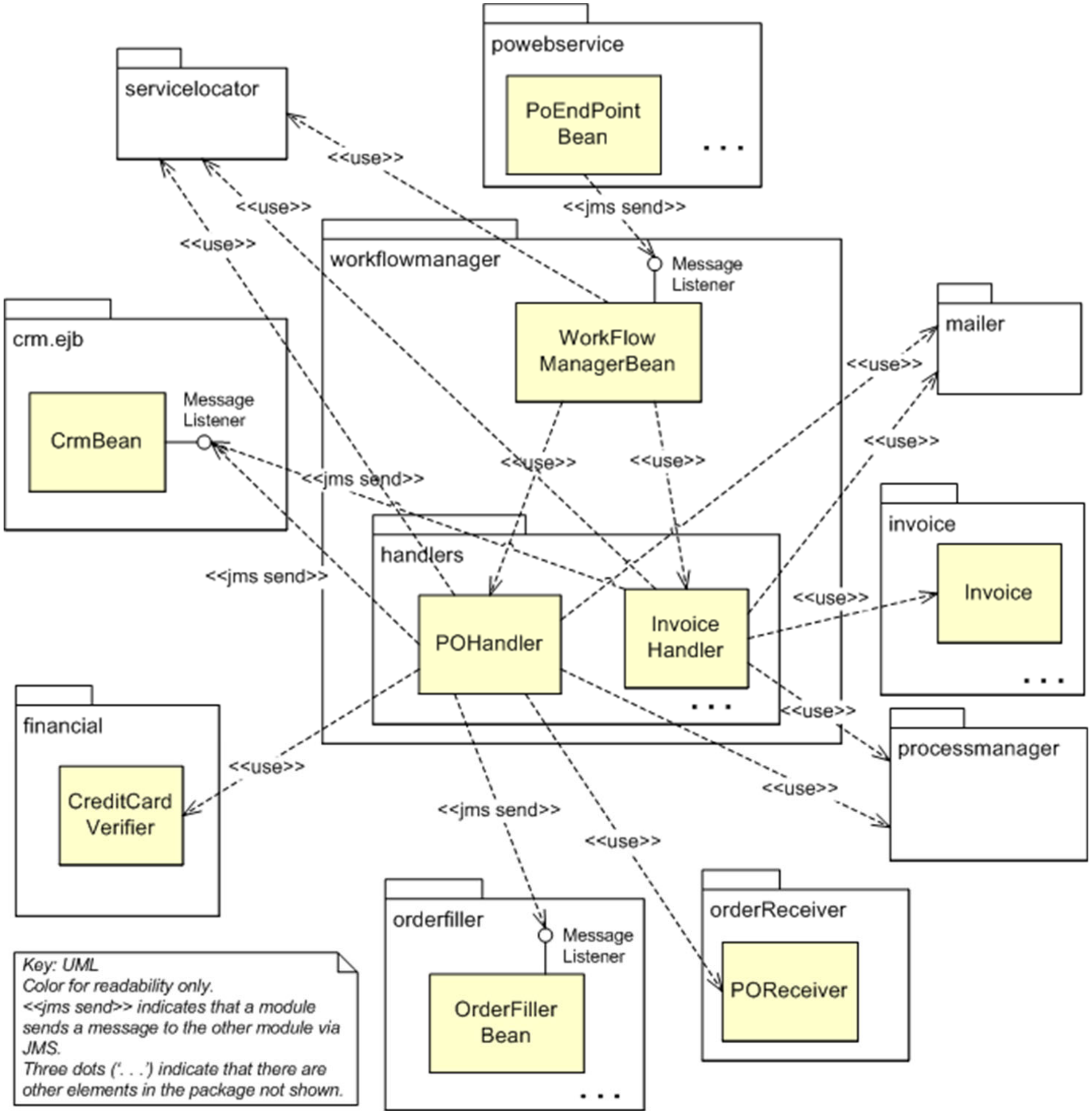
- **Developers are familiar with EJB development and component-based development.**
- **These highly modular EJB components promote reuse.**

OPC Module Uses View (3)



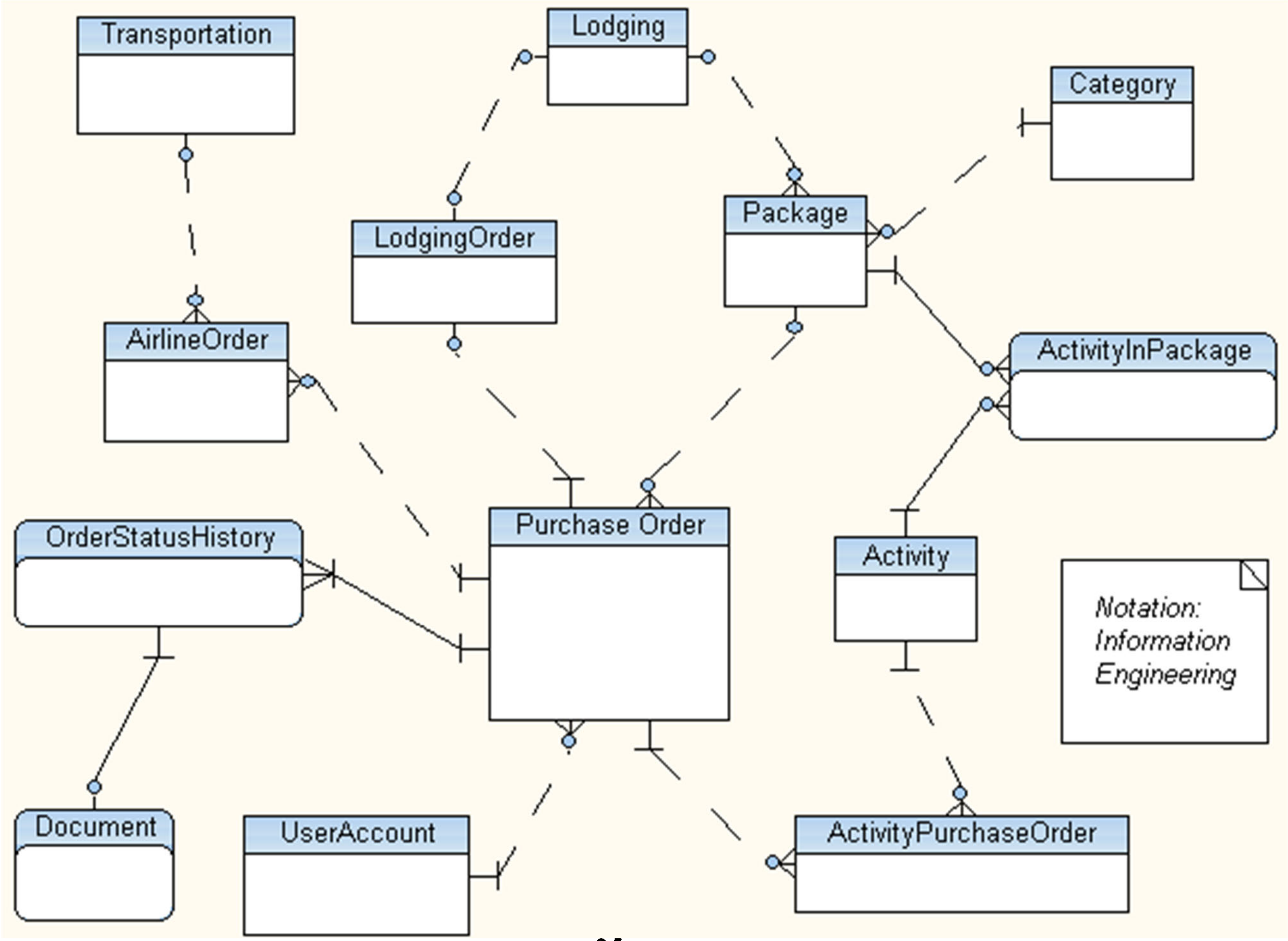
Key: UML
Color for readability only
<<use wsdl>> indicates that a module uses Wsdl interfaces defined in the other module
<<jms send>> indicates that a module sends a message to the other module via JMS

Workflowmanager Module Uses View (4)



Key: UML
Color for readability only.
<<jms send>> indicates that a module sends a message to the other module via JMS.
Three dots ('. . .') indicate that there are other elements in the package not shown.

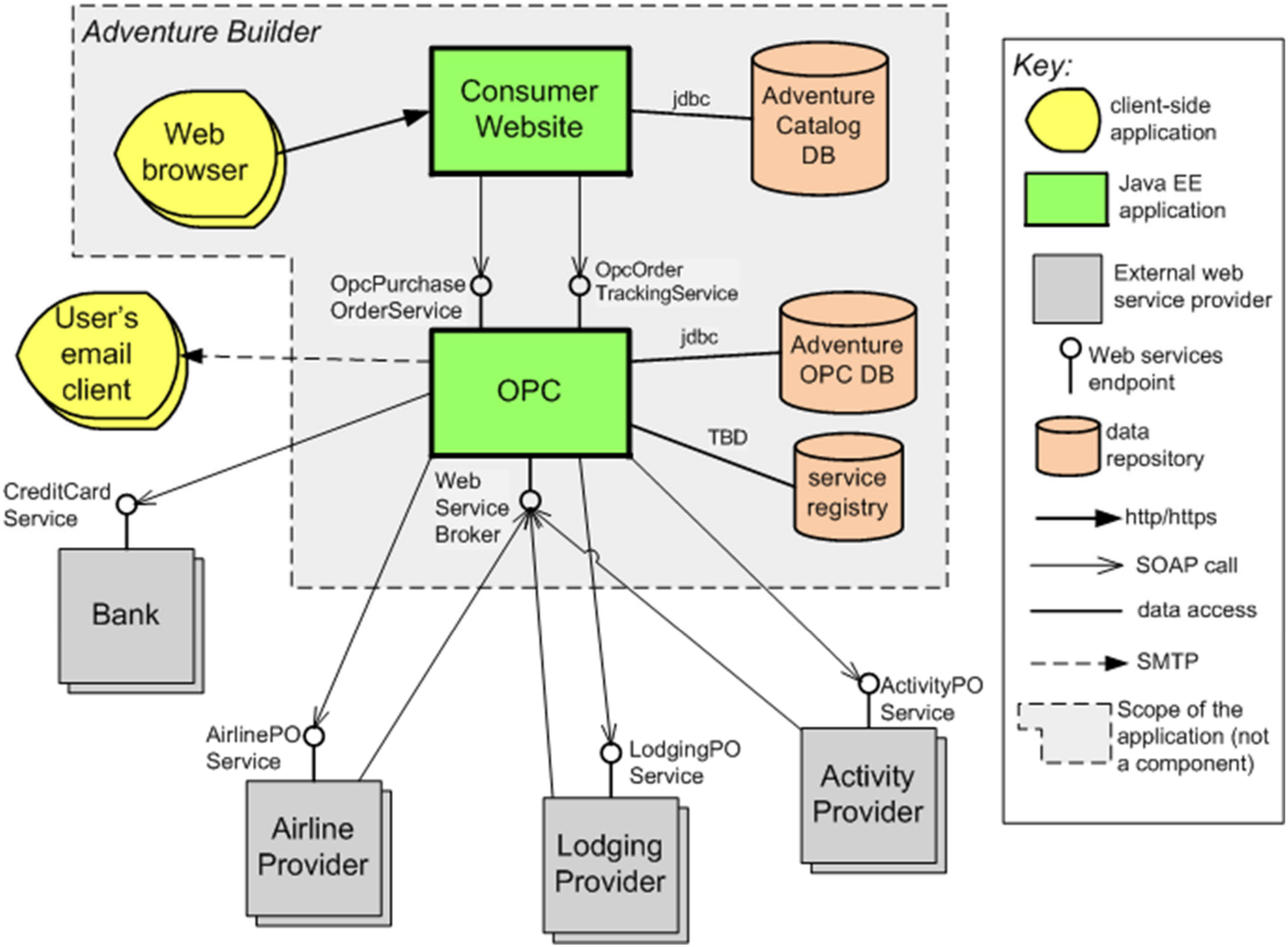
Data Model (5)



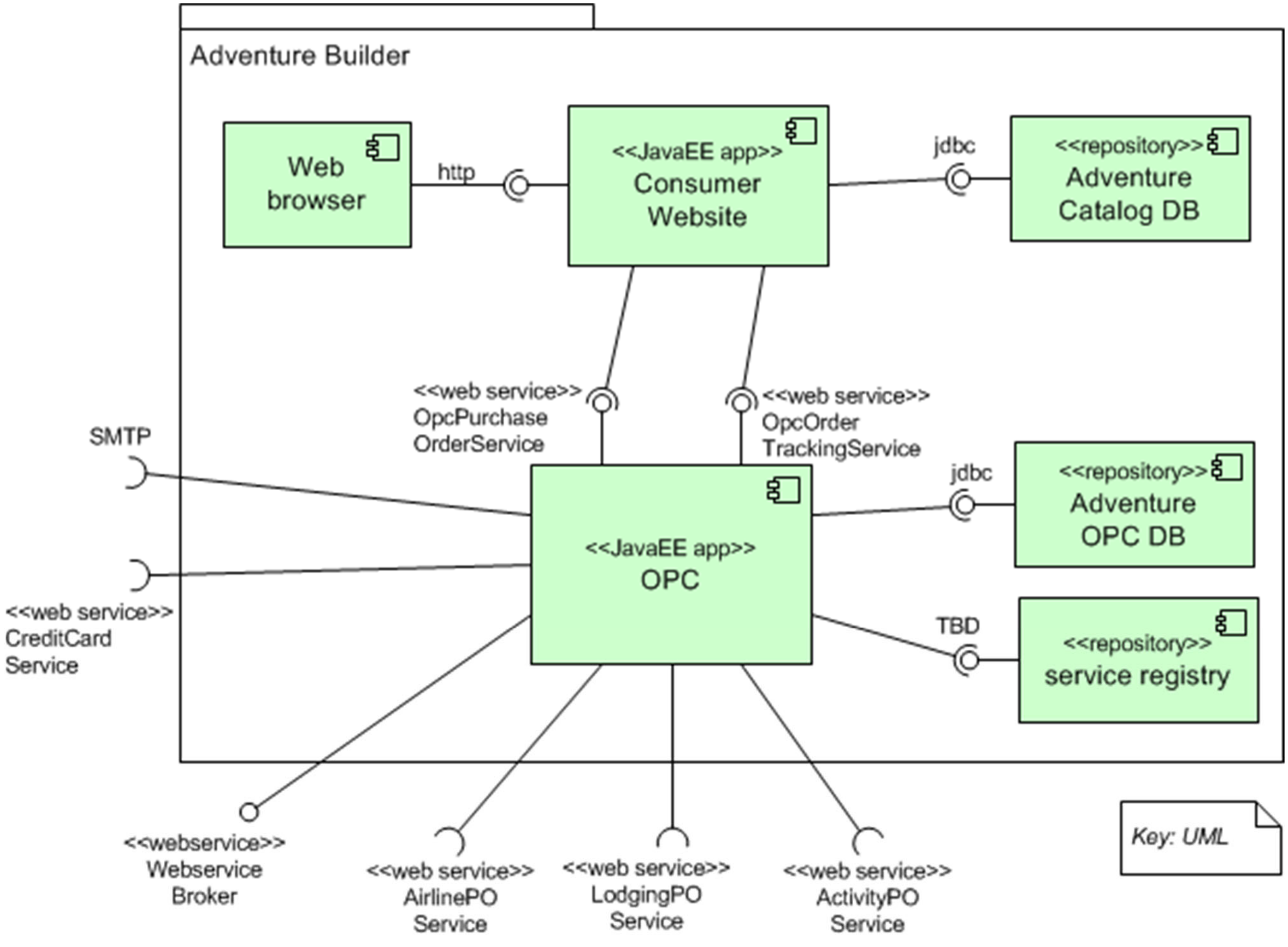
Notation: Information Engineering

Top Level SOA View (C&C 1)

Informal Notation

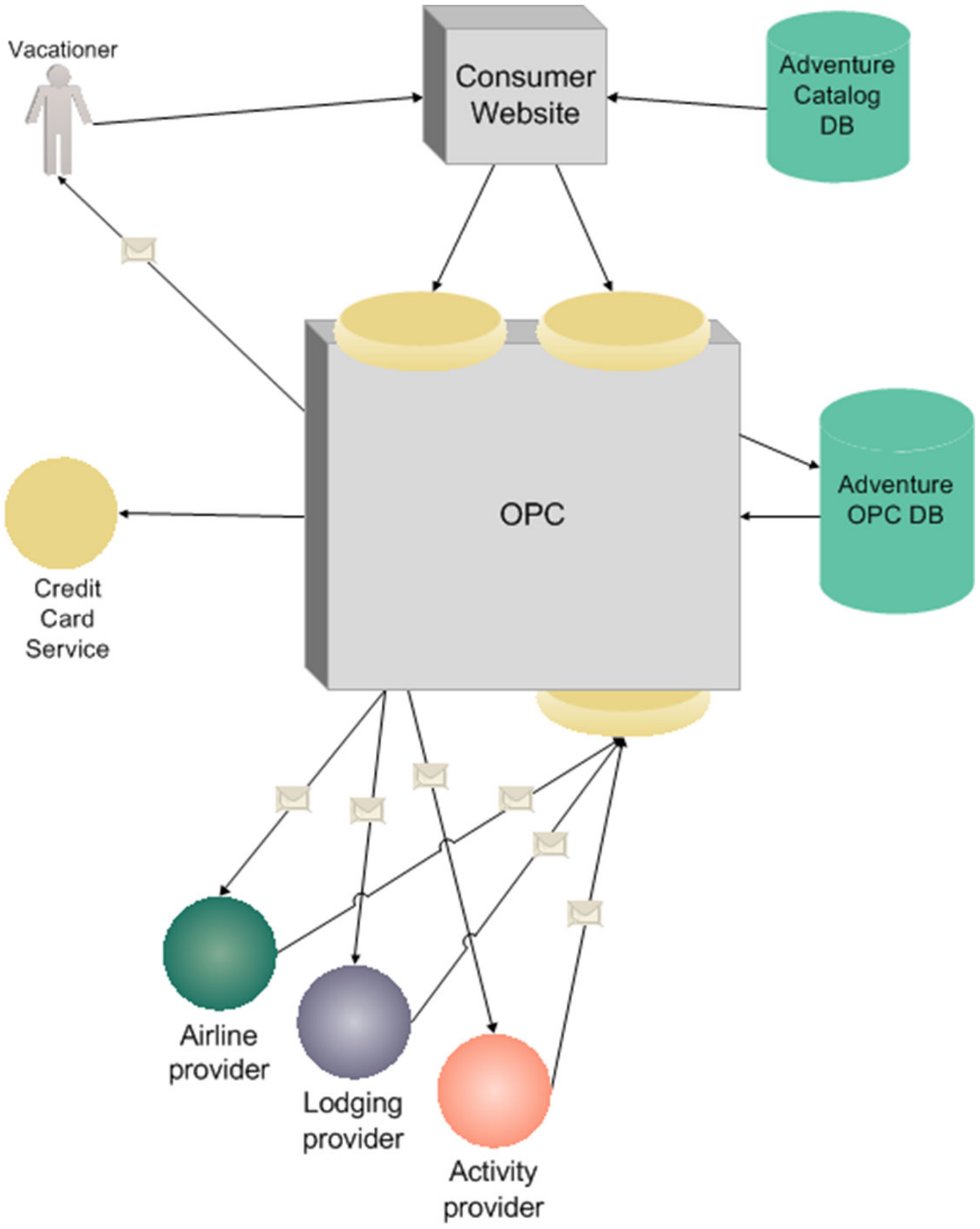


Top Level SOA View (C&C 1)
UML

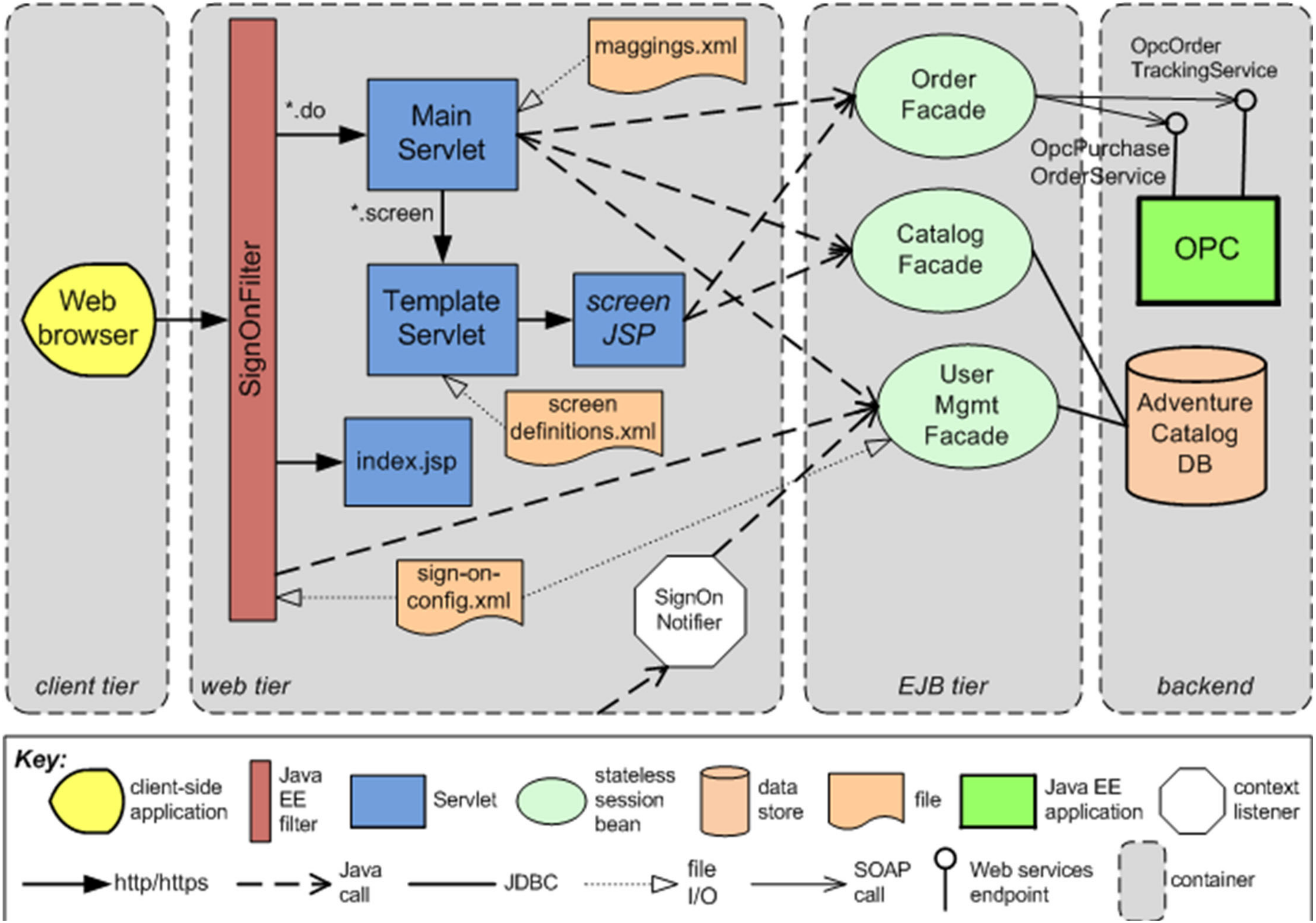


Top Level SOA View (C&C 1)

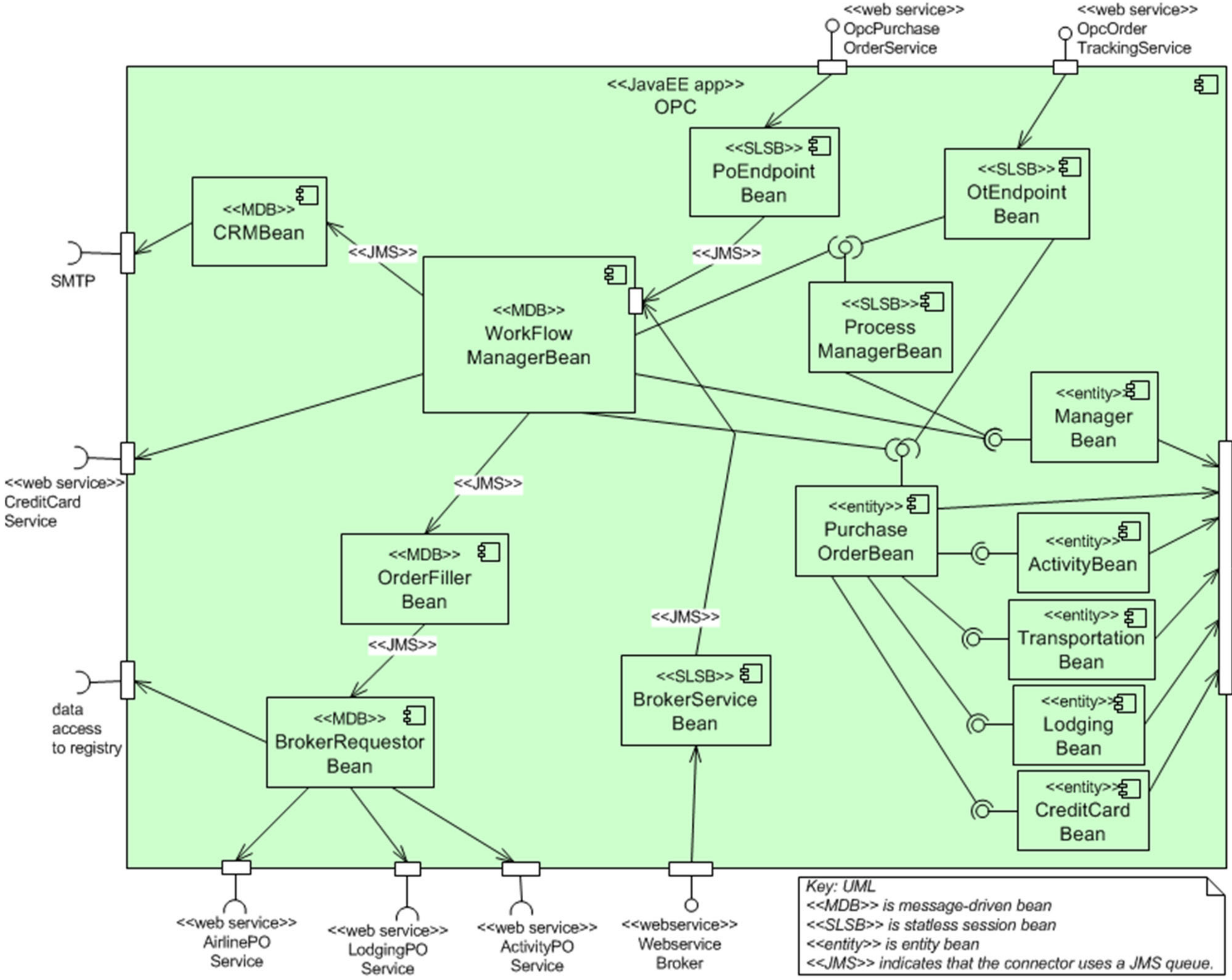
soapatterns.org Notation



Consumer Website Multi-Tier View (C&C 2)



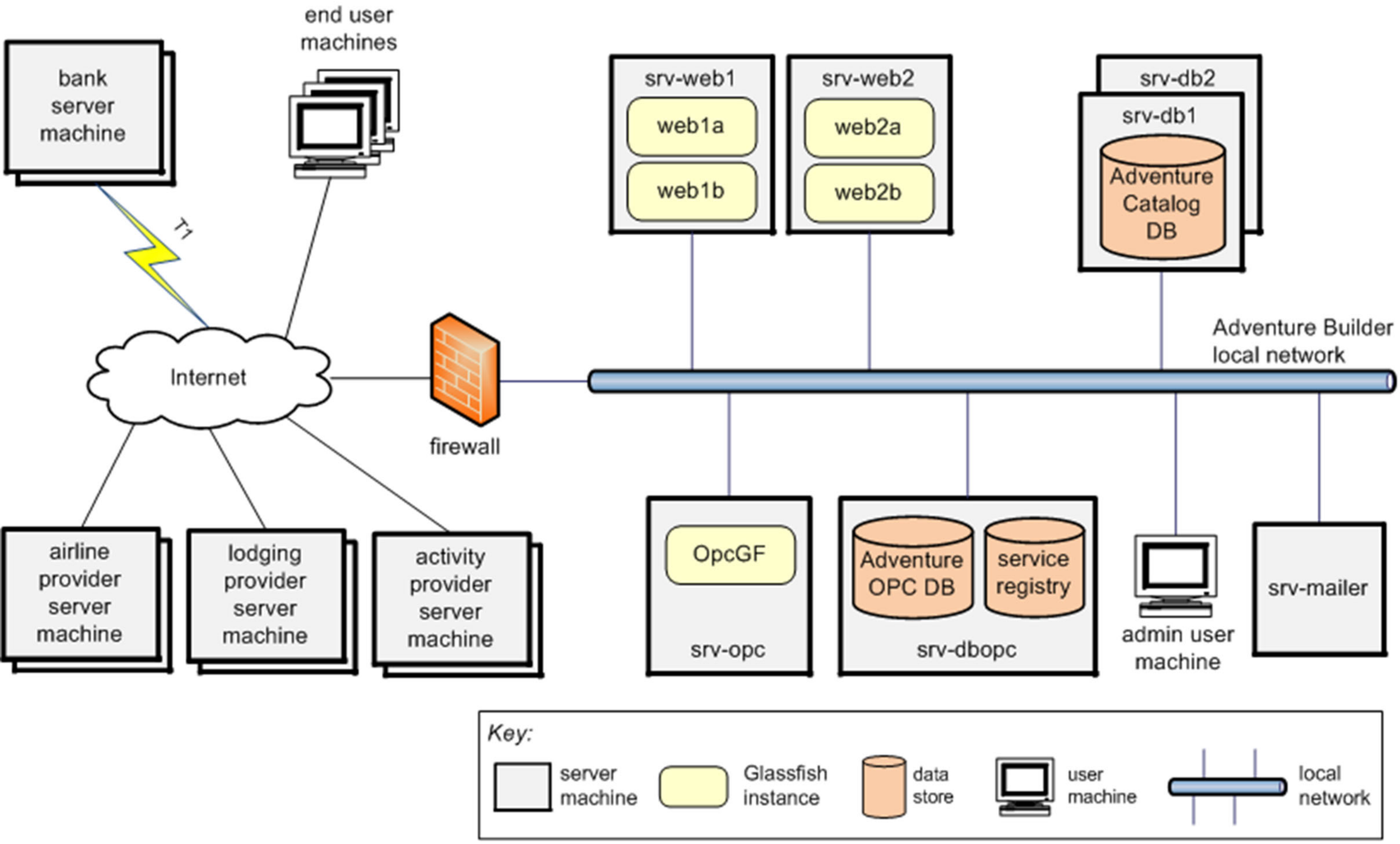
OPC View (C&C 3) UML



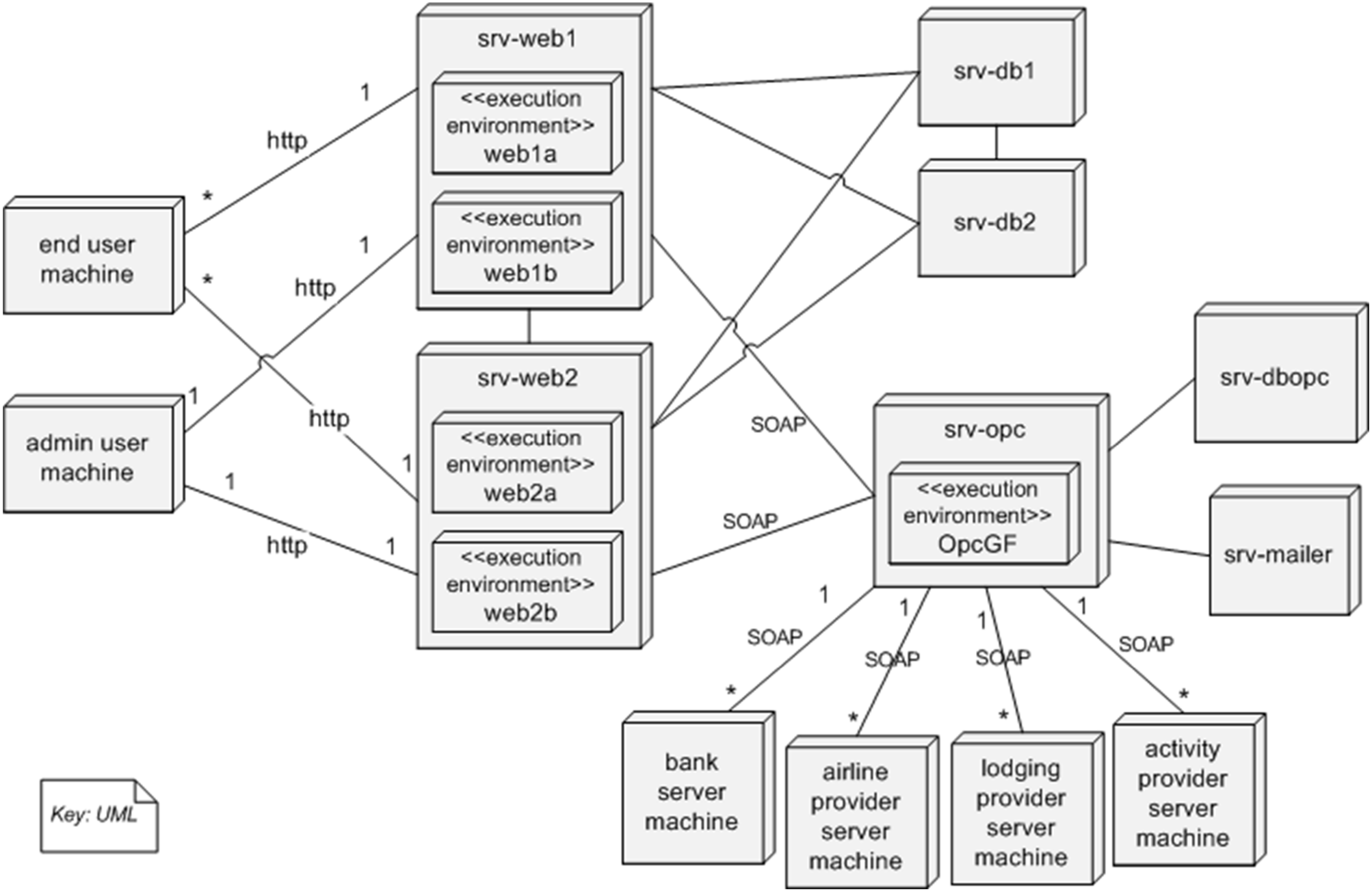
Key: UML
<<MDB>> is message-driven bean
<<SLSB>> is statless session bean
<<entity>> is entity bean
<<JMS>> indicates that the connector uses a JMS queue.

Deployment View (Allocation 1)

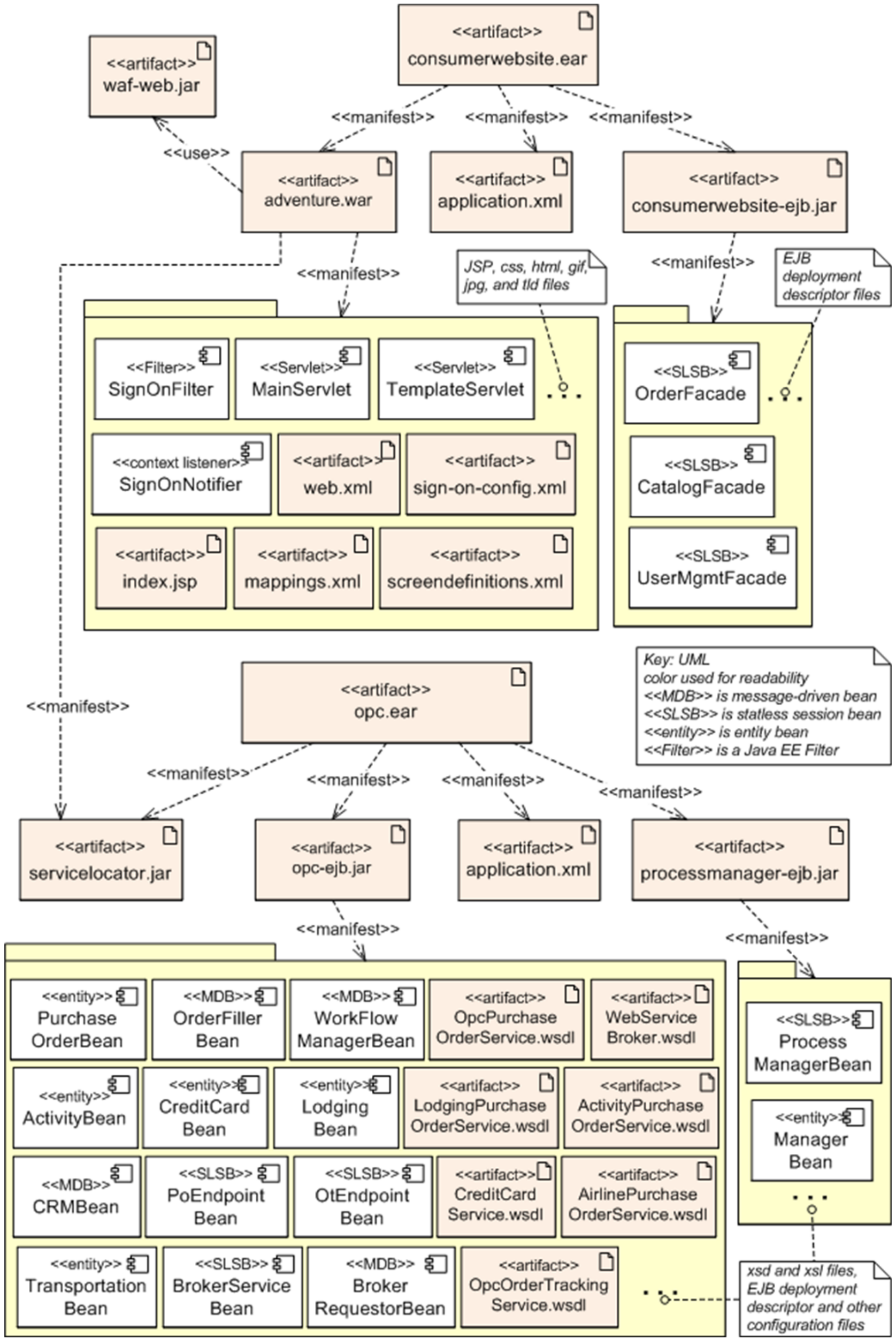
Informal Notation



Deployment View (Allocation 1) UML


srv-web1
<<execution environment>> web1a
<<execution environment>> web1b
srv-web2
<<execution environment>> web2a
<<execution environment>> web2b
srv-db1
srv-db2
srv-dbopc
srv-opc
<<execution environment>> OpcGF
srv-mailer
end user machine
admin user machine
bank server machine
airline provider server machine
lodging provider server machine
activity provider server machine
http
http
http
http
SOAP
SOAP
SOAP
SOAP
SOAP
SOAP
1
*
Key: UML

Install View (Allocation 2)



JSP, css, html, gif, jpg, and tld files

EJB deployment descriptor files

Key: UML
color used for readability
<<MDB>> is message-driven bean
<<SLSB>> is statless session bean
<<entity>> is entity bean
<<Filter>> is a Java EE Filter

xsd and xsl files, EJB deployment descriptor and other configuration files

What Is An "Agile Method"?

A software engineering "methodology" that follows the Agile Manifesto?

A method that supports responding rapidly to changing requirements?

- **Mark Paulk**

Does an agile method necessarily imply

- **Evolutionary / iterative / incremental development?**
- **Empowerment / participation of the development team?**
- **Active collaboration with the customer?**
- **…**

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

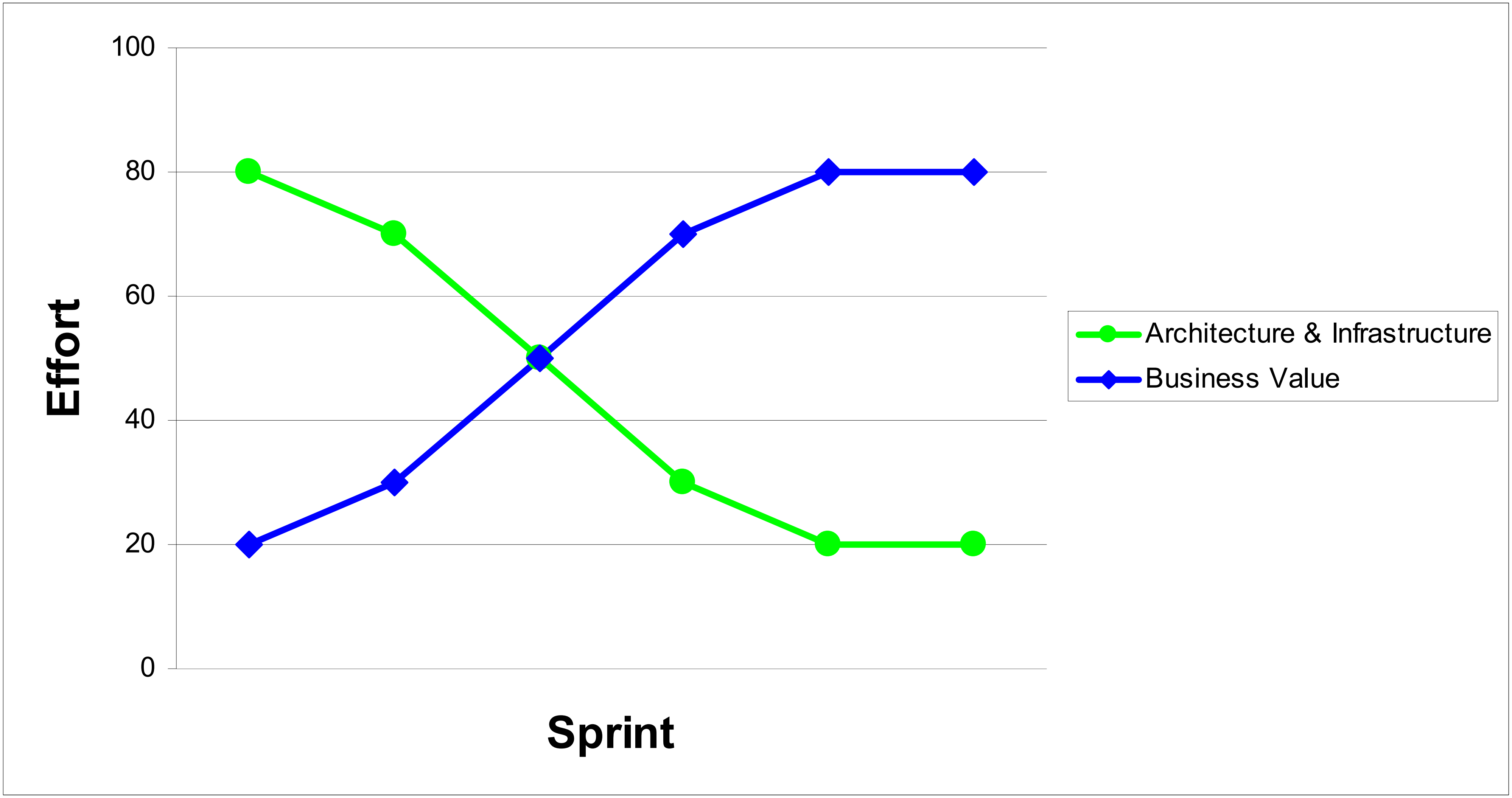
Architecture in an Agile Context

The best teams may be self-organizing, but the best architectures still require technical skill, deep experience, and deep knowledge.

A focus on early and continuous release of software, where "working" is measured in terms of customer-facing features, leaves little time for addressing the kinds of cross-cutting concerns and infrastructure critical to a high-quality large-scale system.

The issue is not agile vs architecture but how to best blend agile and architecture…

Building the Foundation



Documentation and YAGNI

Expect the greatest agile friction from evaluation and documentation.

Technical documentation principle: write for the reader.

- **No reader → no documentation**

The Views and Beyond approach (Clements 2002)

- **uses the architectural view as the “unit” of documentation**
- **prescribes producing a view if and only if it addresses substantial concerns of an important stakeholder community**
- **the view selection method prescribes producing the documentation in prioritized stages to satisfy the needs of the stakeholders who need it now**

Guidelines for Agile Architecture
(Booch)

All good software-intensive architectures are agile.

- **a successful architecture is resilient and loosely coupled**
- **composed of a core set of well-reasoned design decisions**
- **contains some "wiggle room" that allows modifications to be made and refactorings to be done**

An effective agile process will allow the architecture to grow incrementally as the system is developed and matures.

- **decomposability**
- **separation of concerns**
- **near-independence of the parts**

The architecture should be visible and self-evident in the code

- **make the design patterns, cross-cutting concerns, and other important decisions obvious, well communicated, and defended**
- **may, in turn, require documentation**
- **"socialize" the architecture**

Tradeoff Advice

Large and complex system with relatively stable and well-understood requirements

- **do a large amount of architecture work up front**

Big projects with vague or unstable requirements

- **quickly design a complete candidate architecture**
- **Cockburn's Crystal Clear "walking skeleton"**

Smaller projects with uncertain requirements,

- **try to get agreement on the central patterns**

Documenting Software Architectures

If it is not written down, it does not exist.

- **Philippe Kruchten**

If you don’t have it in writing, I didn’t make a commitment.

- **mcp**

(A lack of planning on your part does not constitute a crisis on my part.)

- **mcp**

Architecture has to be communicated in a way to let its stakeholders use it properly to do their jobs.

Uses of Architecture Documentation

As a means of education

- **introducing people to the system**

As a primary vehicle for communication among stakeholders

- **including the architect in the project’s future**

As the basis for system analysis and construction

Notations

Informal notations

- general-purpose diagramming and editing tools and visual conventions

Semiformal notations

- a standardized notation that prescribes graphical elements and rules of construction, e.g., UML

Formal notations

- has a precise (usually mathematically based) semantics
- formal analysis of both syntax and semantics is possible
- generally referred to as architecture description languages
- the use of such notations is rare

Module Views

A module is an implementation unit that provides a coherent set of responsibilities.

The relations that modules have to one another include *is part of*, *depends on*, and *is a*.

It is unlikely that the documentation of any software architecture can be complete without at least one module view.

Component-and-Connector Views

Show elements that have some runtime presence

- **processes, objects, clients, servers, and data stores**

Include as elements the pathways of interaction

- **communication links and protocols, information flows, and access to shared storage**

Components have interfaces called ports.

Connectors have roles, which are its interfaces, defining the ways in which the connector may be used by components to carry out interaction.

Allocation Views

Describe the mapping of software units to elements of an environment in which the software is developed or in which it executes.

The relation in an allocation view is <u>allocated to</u>.

The usual goal of an allocation view is to compare

- **the properties required by the software element with**
- **the properties provided by the environmental elements**

to determine whether the allocation will be successful or not.

Architectures Are Abstractions

Cannot be seen in the low-level implementation details

Tools aggregate abstractions

- **not a panacea**
- **no programming language construct for layer or connector or …**

Architecture reconstruction is an interpretive, interactive, iterative process

Workbench – open, integration framework

UML

The Unified Modeling Language (UML) is a visual language for specifying, constructing, and documenting the artifacts of systems.

- Object Management Group (OMG)
- UML 2.0 Infrastructure Specification

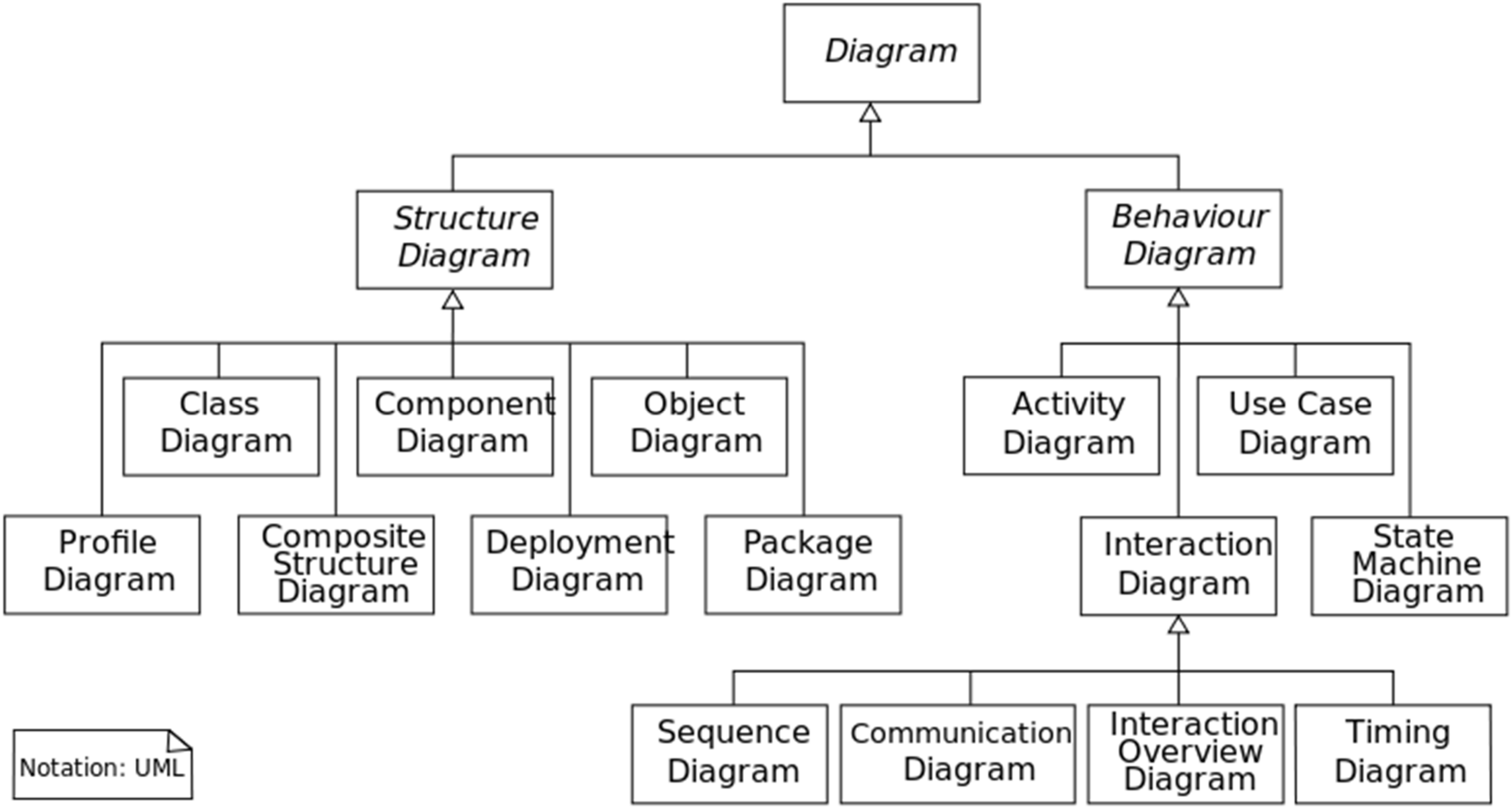
A model is a set of UML diagrams that represent various aspects of the software product.

- UML is the tool that we use to represent (model) the target software product

UML profiles

- specialized subsets of the notation for common subject areas
  - EJB profile for Enterprise JavaBeans

UML Diagrams



Applying UML

UML as <u>sketch</u>

- **informal and incomplete diagrams (often hand drawn on whiteboards) created to explore difficult parts of the problem or solution space**
- **emphasized in agile modeling**

UML as <u>blueprint</u>

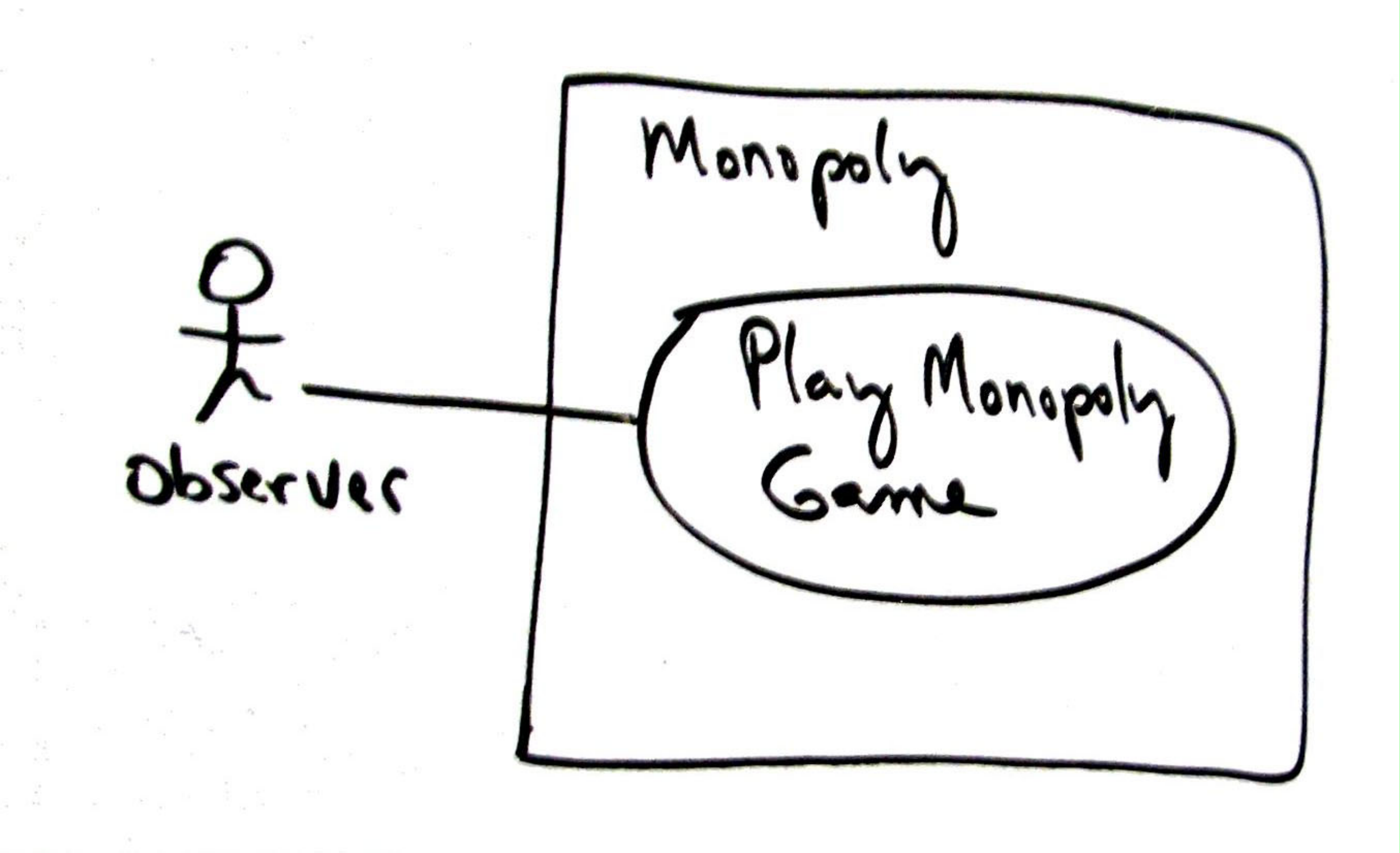
- **relatively detailed design diagrams used for reverse engineering or code generation**

UML as <u>programming language</u>

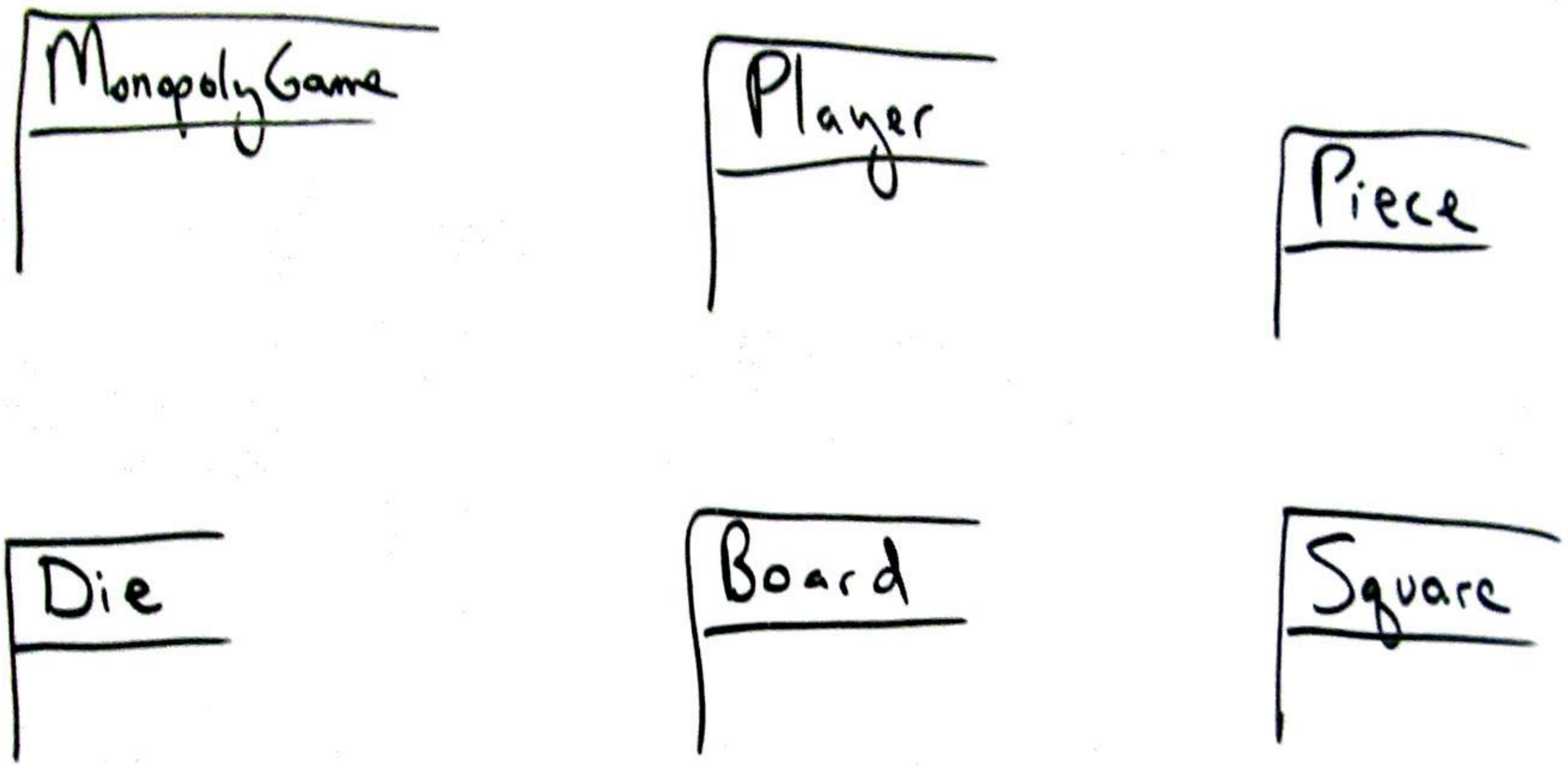
- **complete executable specification of a software system in UML**

Monopoly Case Study *(Larman)*

Use cases aren’t always best for behavior requirements…

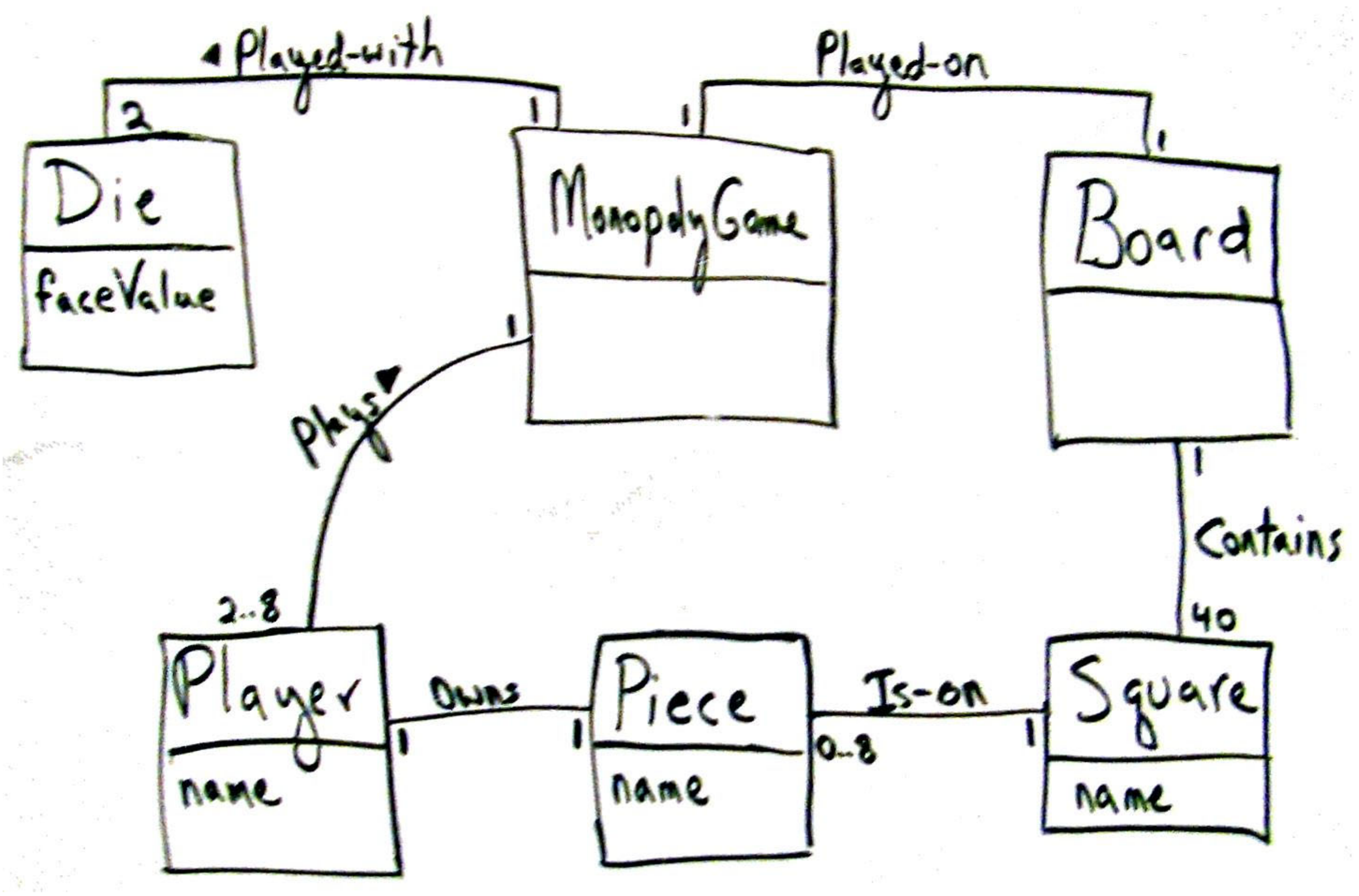


Initial Monopoly Domain Model

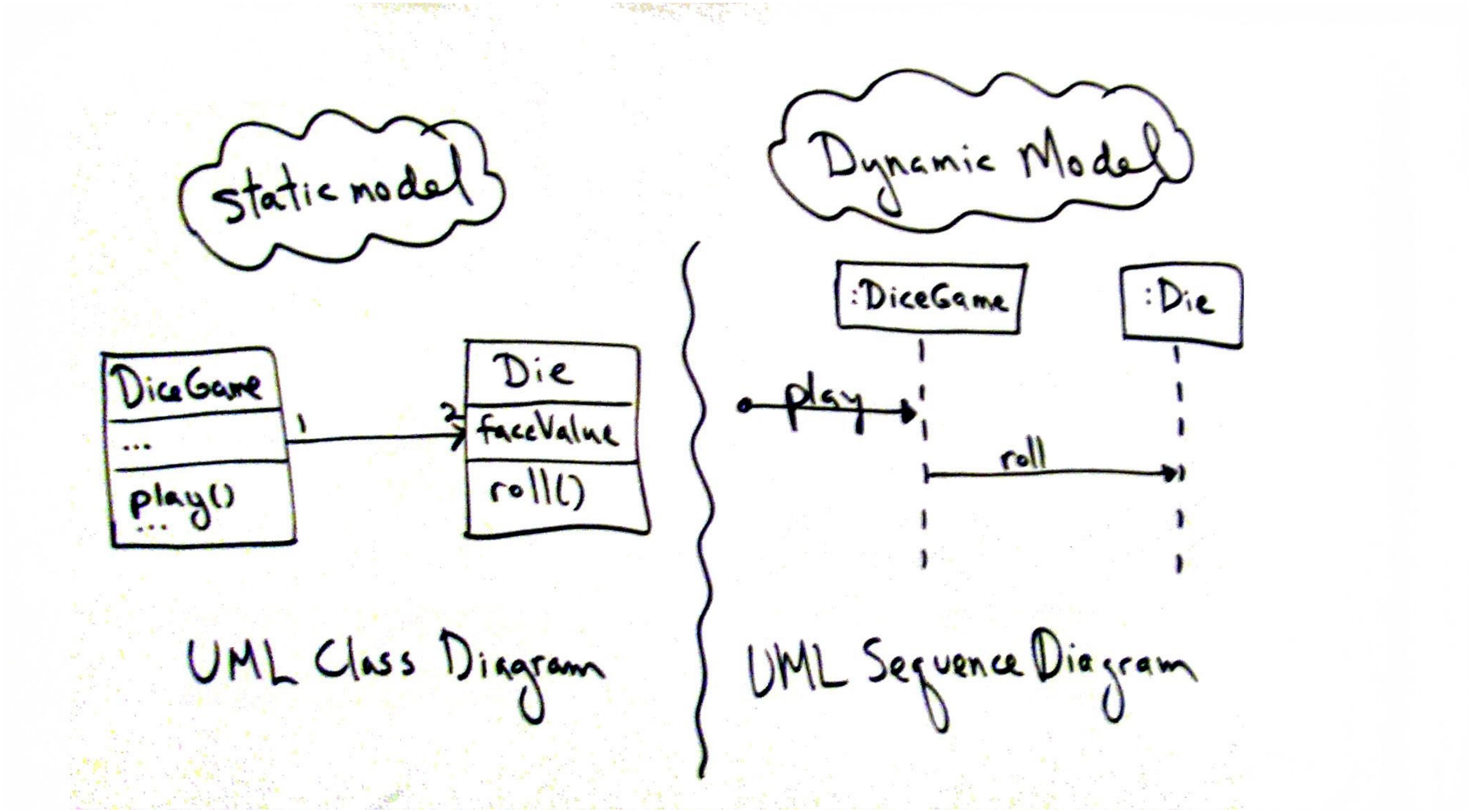


If someone wants the model maintained… redraw using a CASE tool. Who is going to use the updated model and why?

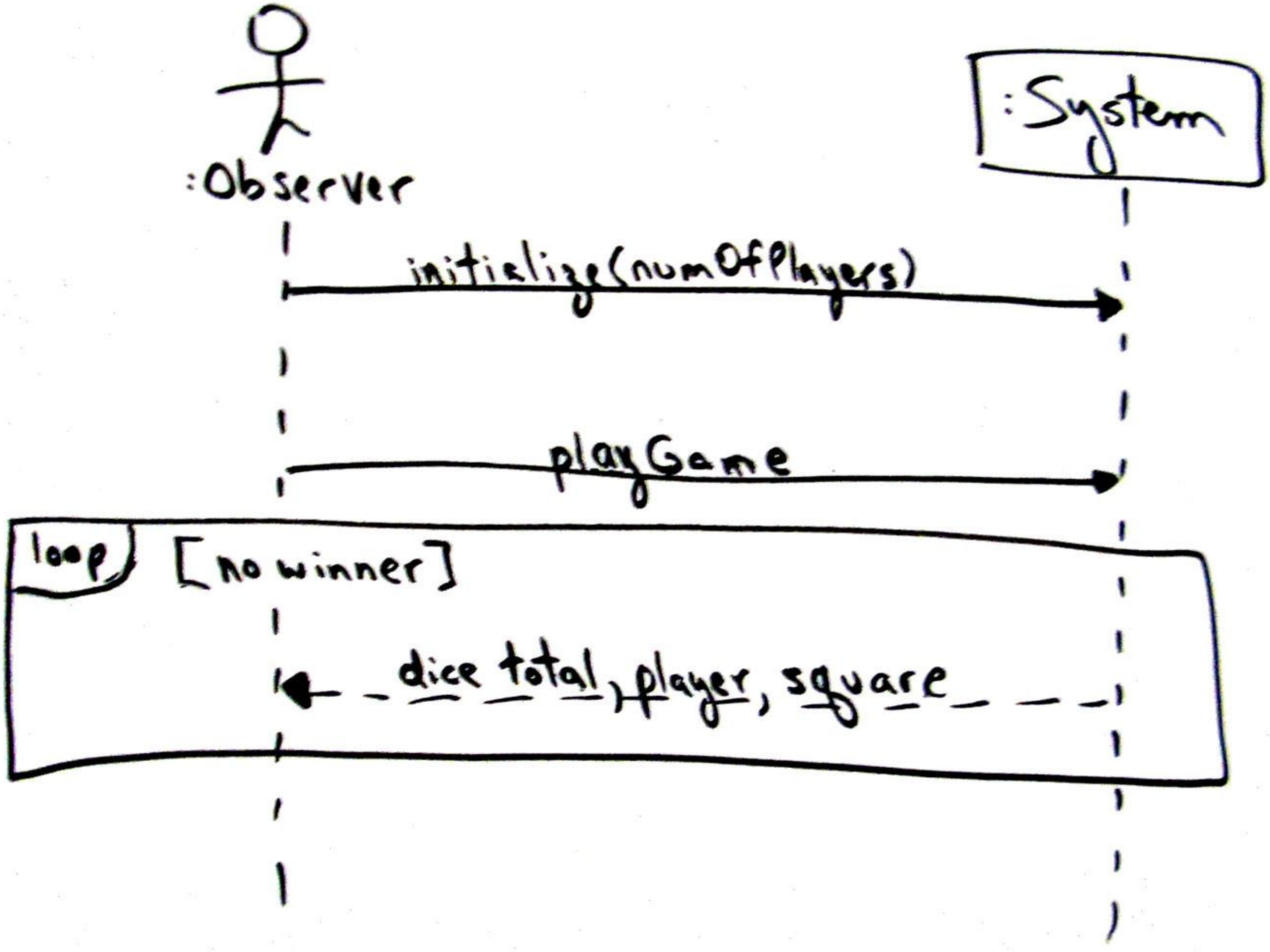
Monopoly Partial Domain Model



Static and Dynamic UML Diagrams



SSD for a PlayMonopolyGame Scenario



Documenting an Architecture

Case study of ~200KSLOC open source product

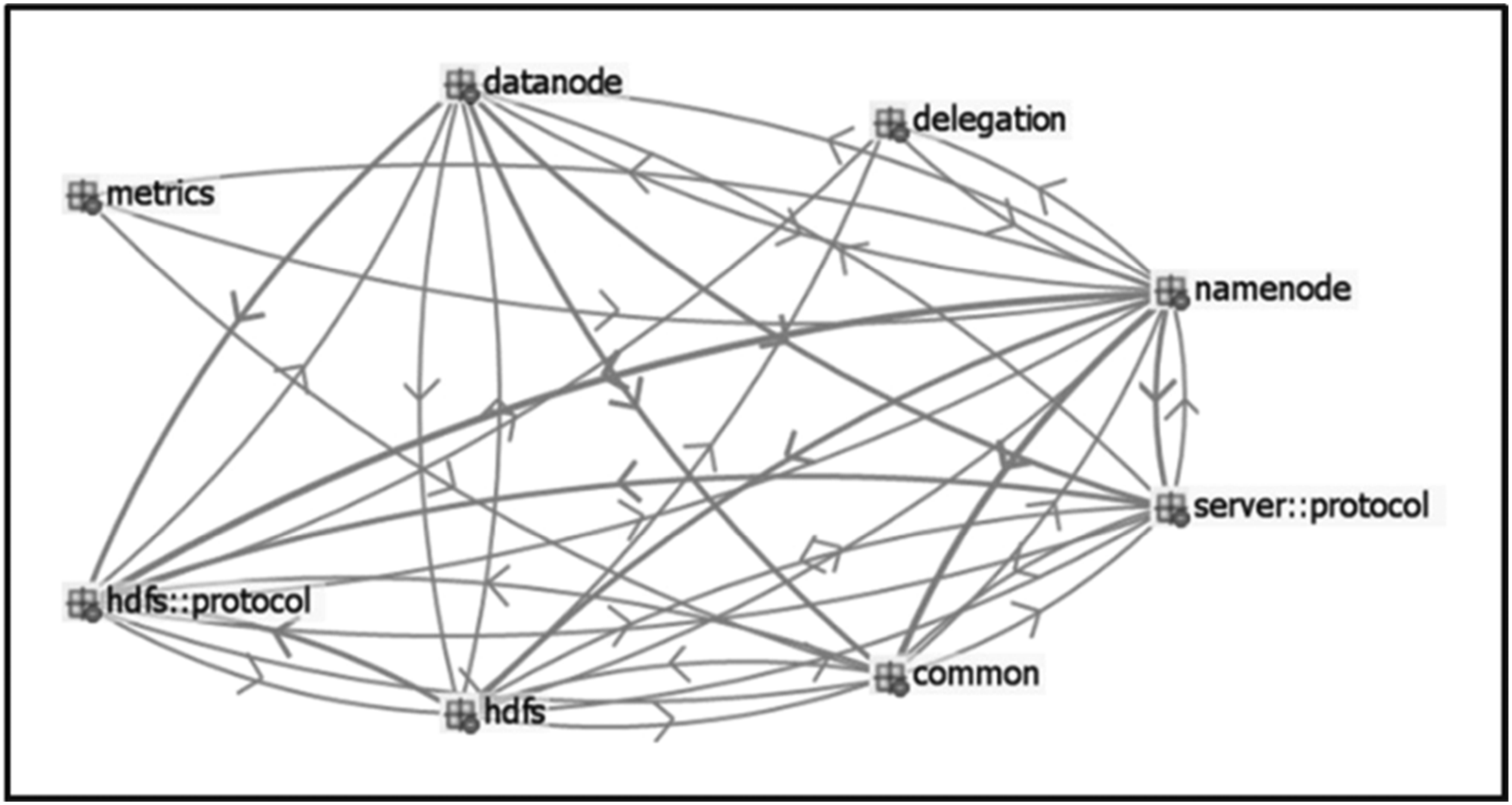
Very little architectural documentation

Team reverse-engineered the architecture (2-3 person weeks of effort) and provided the architecture to the developers

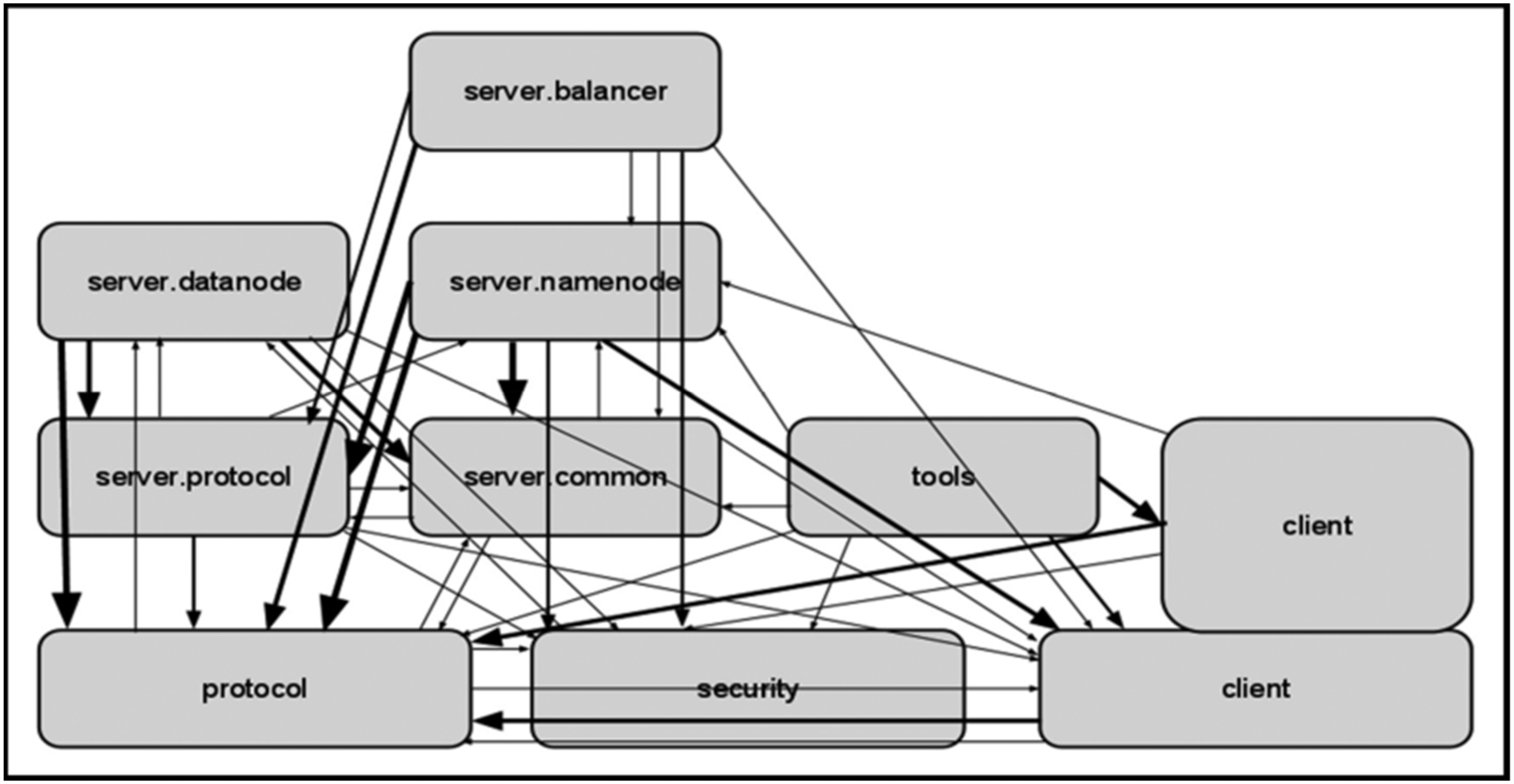
- **system could be characterized as poor quality architectural design (my opinion)**

R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto, "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project," IEEE Transactions on Software Engineering, March 2016.

Reverse-Engineered Module Relationships in HDFS *(Kazman 2016)*



Documented Module Relationships in HDFS *(Kazman 2016)*



Value of Architecture Documentation

“Committers” did not need or value the architecture documentation.

- **system was small enough to keep architectural details in their heads**

“Outsiders” were promoted to “committers” more quickly using the architecture documentation.

- **decentralization occurred**
- **developers looked at the documentation rather than asking one of the committers about the architecture**

Committers were unwilling to maintain the architecture documentation.

- **need to use tools to automatically extract and maintain architectural information**

Architecturally Significant Requirements (ASRs)

Requirements documents

- **most of what is in a requirements specification does not affect the architecture**
- **much of what is useful to an architect is not in even the best requirements document**
- **ASRs often derive from business goals in the development organization**
- **excavation and archaeology is required to dig ASRs from requirements documents**

Design Decision Category	Look for Requirements Addressing . . .
Allocation of Responsibilities	Planned evolution of responsibilities, user roles, system modes, major processing steps, commercial packages
Coordination Model	Properties of the coordination (timeliness, currency, completeness, correctness, and consistency) Names of external elements, protocols, sensors or actuators (devices), middleware, network configurations (including their security properties) Evolution requirements on the list above
Data Model	Processing steps, information flows, major domain entities, access rights, persistence, evolution requirements
Management of Resources	Time, concurrency, memory footprint, scheduling, multiple users, multiple activities, devices, energy usage, soft resources (buffers, queues, etc.) Scalability requirements on the list above
Mapping among Architectural Elements	Plans for teaming, processors, families of processors, evolution of processors, network configurations
Binding Time Decisions	Extension of or flexibility of functionality, regional distinctions, language distinctions, portability, calibrations, configurations
Choice of Technology	Named technologies, changes to technologies (planned and unplanned)

Interviewing Stakeholders

Architects often have good ideas what quality attributes are exhibited by similar systems and are reasonable.

Stakeholders often have no idea what quality attributes they want in a system.

Results of stakeholder interviews

- **a list of architectural drivers**
- **a set of quality attribute scenarios that the stakeholders (as a group) prioritized**

Quality Attribute Workshop

1) **QAW Presentation and Introductions**
2) **Business/Mission Presentation**
3) **Architectural Plan Presentation**
4) **Identification of Architectural Drivers**
5) **Scenario Brainstorming**
6) **Scenario Consolidation**
7) **Scenario Prioritization**
8) **Scenario Refinement**

Gathering ASRs by Understanding the Business Goals

Business goals are the reason for building a system.

- **often the precursor of requirements that may or may not be captured in a requirements specification**

Business goals often lead to quality attribute requirements.

- **every quality attribute requirement should originate from some higher purpose that can be described in terms of added value**

Business goals may directly affect the architecture without precipitating a quality attribute requirement at all.

Pedigreed Attribute eLicitation Method (PALM)

Day and a half workshop attended by architects and stakeholders who can speak to the business goals of the organizations involved

1) **PALM overview presentation**
2) **Business drivers presentation**
3) **Architecture drivers presentation**
4) **Business goals elicitation**
5) **Identification of potential quality attributes from business goals**
6) **Assignment of pedigree to existing quality attribute drivers**
7) **Exercise conclusion**

Utility Tree

Begins with the word "utility" as the root node.

List the major quality attributes that the system is required to exhibit.

- **under each quality attribute, record a specific refinement of that QA**
- **under each refinement, record the appropriate ASRs (usually expressed as QA scenarios)**

Evaluate against two criteria

- **the business value of the candidate ASR**
- **the architectural impact of including it**
  - **must-have, important, nice-to-have**

Tying the Methods Together

If you have a requirements process that gathers, identifies, and prioritizes ASRs, consider yourself lucky…

If nobody has captured the business goals behind the system you're building, then a PALM exercise.

If you feel that important stakeholders have been overlooked, capture their concerns through interviews.

- **Quality Attribute Workshop**

Building a utility tree is a good way to capture ASRs along with their prioritization.

Designing an Architecture

The building blocks for designing a software architecture:

- **locating architecturally significant requirements**
- **capturing quality attribute requirements**
- **choosing, generating, tailoring, and analyzing design decisions for achieving those requirements**

Now to pull the pieces together…

Attribute-Driven Design (ADD) Method

Produce a workable architecture quickly

Before beginning a design process, the requirements should (ideally) be known…

Requirements (changes) are continually arriving…

ADD can begin when a set of architecturally significant requirements is known.

Breadth vs Depth First

Personnel availability may dictate a refinement strategy.

Risk mitigation may dictate a refinement strategy.

Deferral of some functionality or quality attribute concerns may dictate a mixed approach.

All else being equal, a breadth-first refinement strategy is preferred because

- **it allows you to apportion the most work to the most teams soonest**
- **allows for consideration of the interaction among the elements at the same level**

Generate a Design Solution

Sources of design candidates— patterns, tactics, and checklists

- **initial candidate design will likely be inspired by a pattern**
- **possibly augmented by one or more tactics**
- **consider the design checklists for the quality attributes**

To the extent that the system you're building is similar to others, it is likely that the solutions you choose will solve a collection of ASRs simultaneously…

Verify and Refine Requirements

Your design solution may not satisfy all the ASRs.

Backtrack – reconsider the design.

Unsatisfied ASRs may relate to

- **A quality attribute requirement allocated to the parent element**
- **A functional responsibility of the parent element**
- **One or more constraints on the parent element**

What Requirements Are Left?

Requirements assigned to element are satisfied…

Delegate to one of the children

Distribute among the children

Cannot be satisfied with the current design

- **backtrack**
- **push back on the requirement**

Done?

Terminate with a sketch of the architecture…

- **flesh out the architecture consistent with the overall design approaches laid out**

Satisfy (contractual) specifications…

Exhaust design budget…

Terminating ADD and releasing the architecture are different decisions.

- **early architectural views can be usable**

Architecture and Business

Perhaps the most important job of an architect is to be a fulcrum where business and technical decisions meet and interact…

What are the economic implications of an architectural decision?

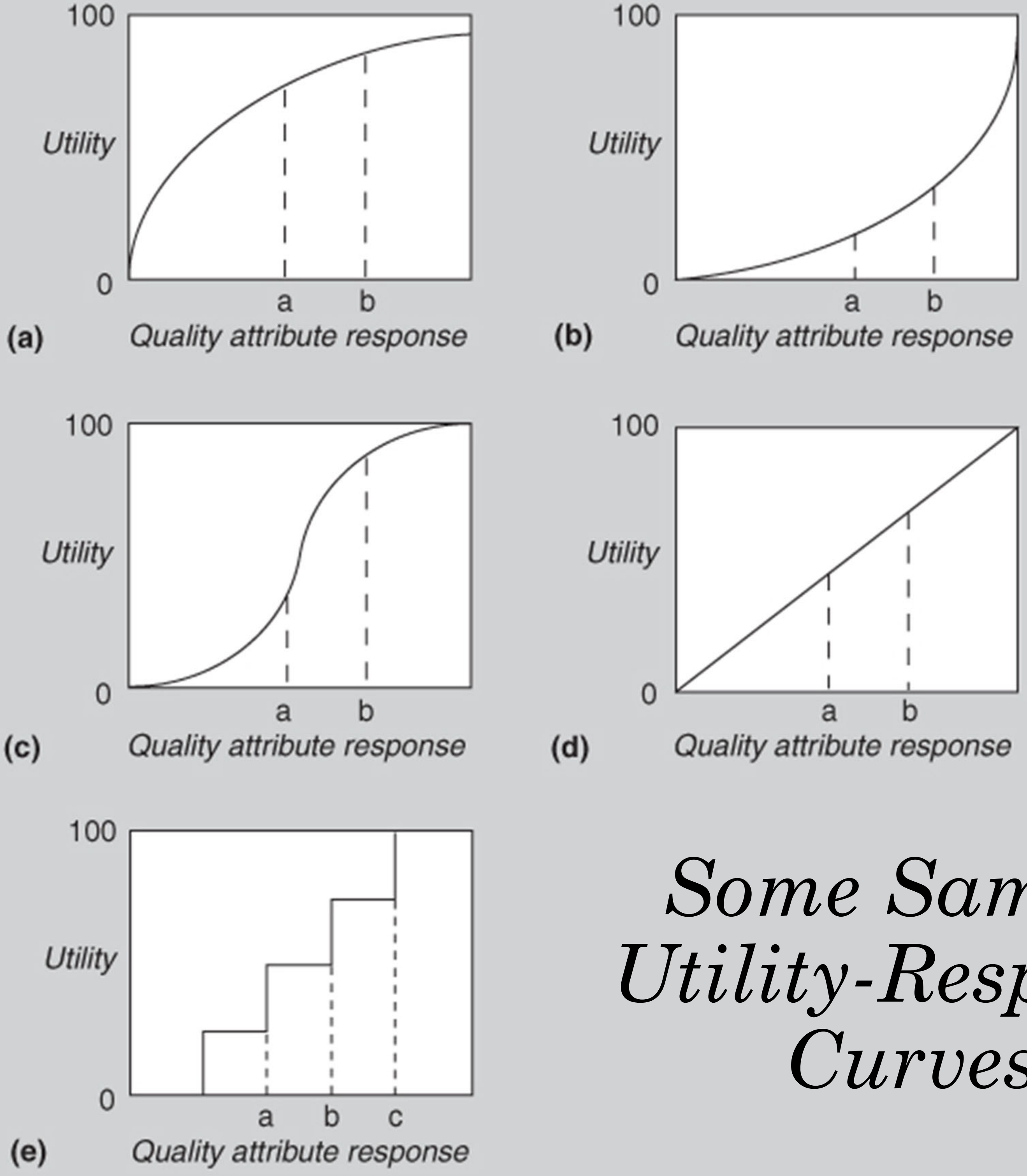
Utility Response Curves

Each scenario’s stimulus-response pair provides some utility (value) to stakeholders

The utility of different possible values for the response can be compared

Absolute numbers are not necessary to compare alternatives…

- **human beings are better at comparative estimation**



Some Sample Utility-Response Curves

Best and Worst Cases

Best-case quality attribute level – that above which the stakeholders foresee no further utility

Worst-case quality attribute level – the minimum threshold above which a system must perform, otherwise it is of no value to the stakeholders

Current quality attribute level

Desired quality attribute level

Anchor the utility levels on a scale of 0-100 with the worst and best cases

Questions and Answers