2IW80 Software specification and architecture

Software architecture: Domain-Specific Software Architecture and Architectural Patterns

Alexander Serebrenik

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Before we start…

- A way of looking at a system from the position of a certain stakeholder with a particular concern is called

A. view B. viewpoint C. model D. architecture

Before we start…

- A way of looking at a system from the position of a certain stakeholder with a particular concern is called

A. view [✓ B. viewpoint] C. model D. architecture

- In Kruchten's 4+1 components, functions, subsystems, modules and packages are discussed in the

A. logical view B. development view
C. process view D. deployment view
E. scenarios

Before we start…

- A way of looking at a system from the position of a certain stakeholder with a particular concern is called

  A. view [✓ B. viewpoint] C. model D. architecture

- In Kruchten's 4+1 components, functions, subsystems, modules and packages are discussed in the

  A. logical view [✓ B. development view]
  C. process view D. deployment view
  E. scenarios

Before we start

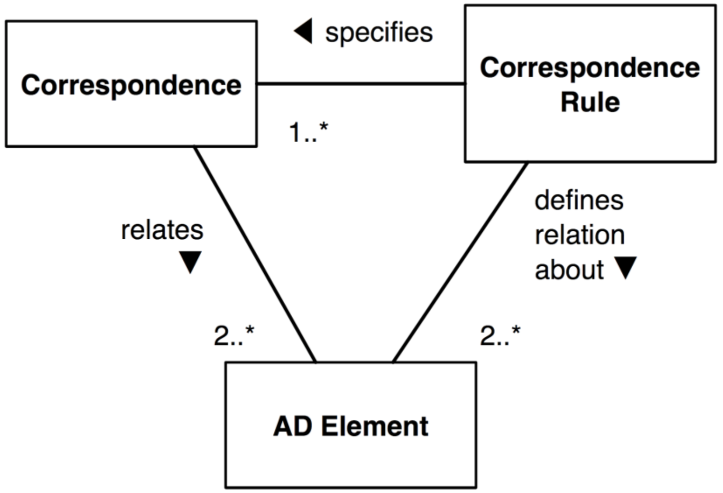
- **Correspondence** records relations between … architecture description elements

a) at least two
b) two
c) at most two
d) any number of
e) I have no clue

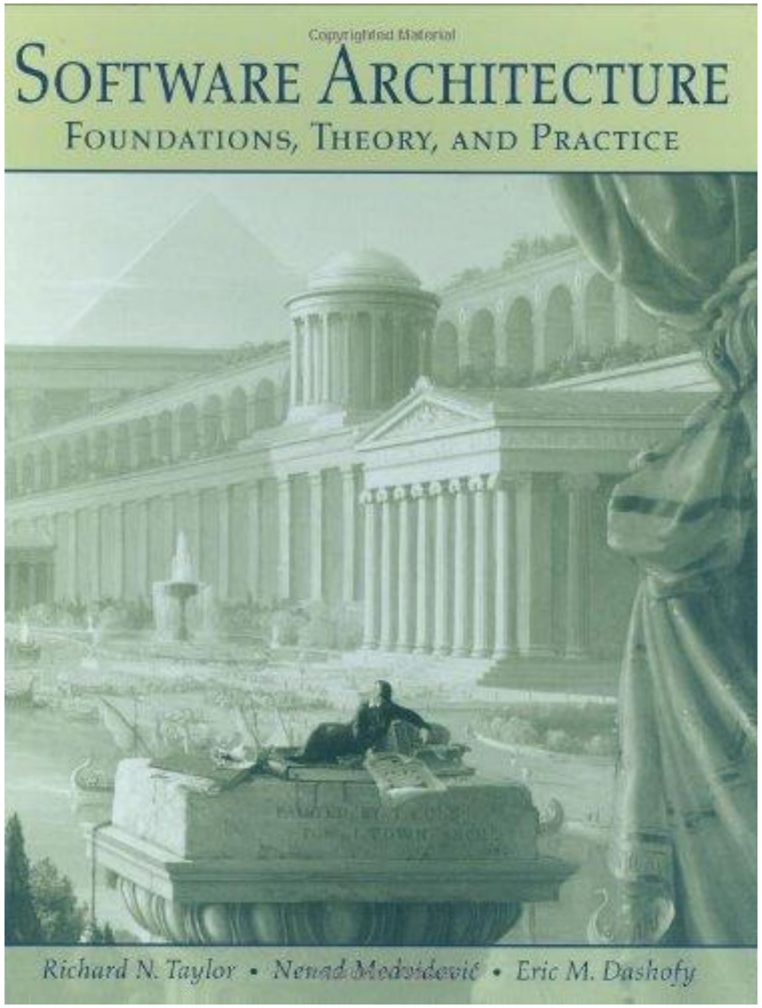
Before we start

- **Correspondence** records relations between … architecture description elements

a) at least two
b) two
c) at most two
d) any number of
e) I have no clue



This week sources


Copyrighted Material
SOFTWARE ARCHITECTURE
FOUNDATIONS, THEORY, AND PRACTICE
Richard N. Taylor • Nenad Medvidović • Eric M. Dashofy

Slides by

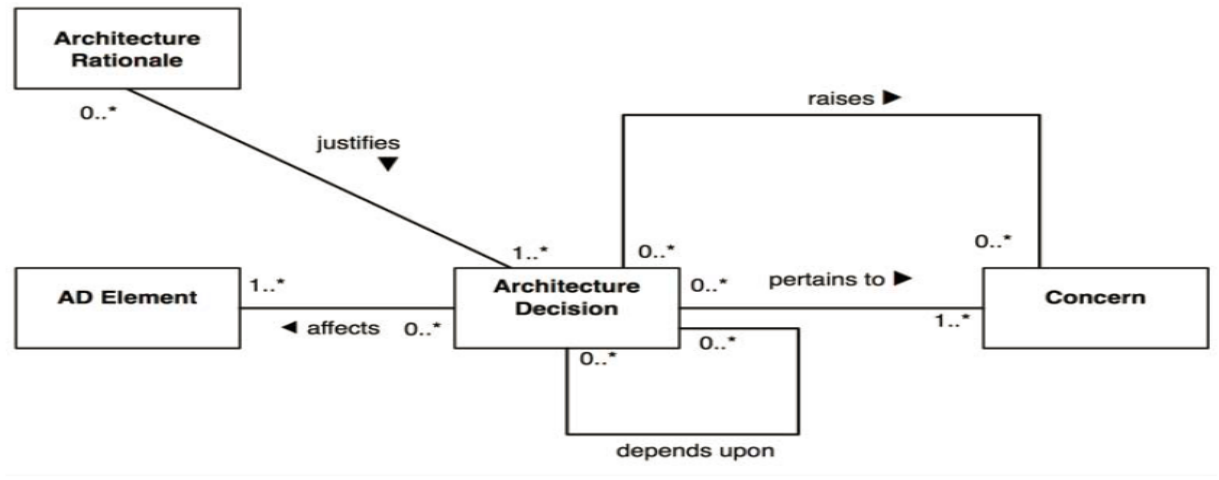
Dietmar Pfahl

Rudolf Mak

Johan Lukkien

TU/e Technische Universiteit Eindhoven University of Technology

Recall



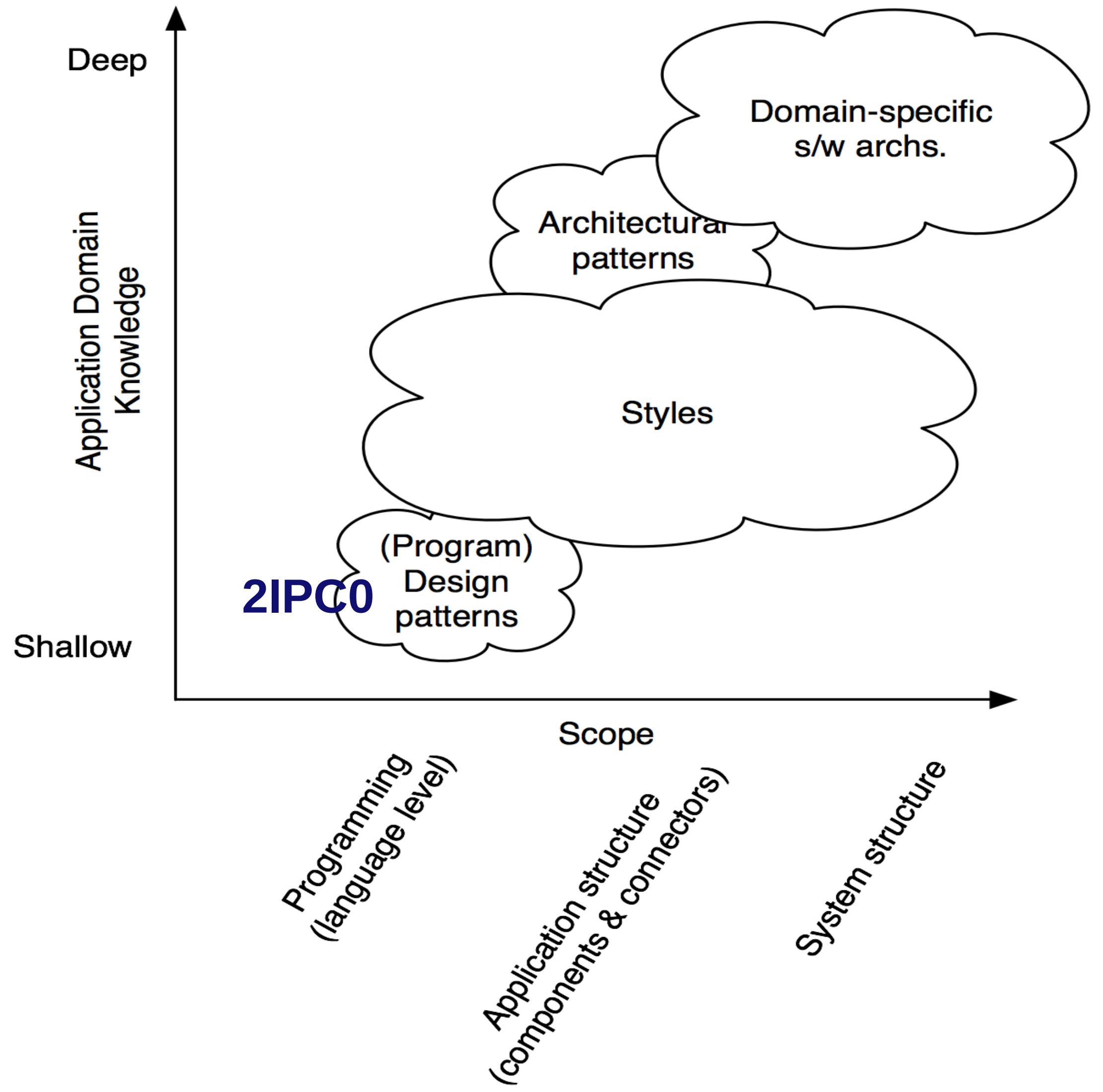
- **Architecture decisions** are important
  - Depend on the stakeholders’ concerns

- How to make right decisions?
  - Learn from successes/failure of other engineers

Learning from Others: Patterns, Styles, and DSSAs

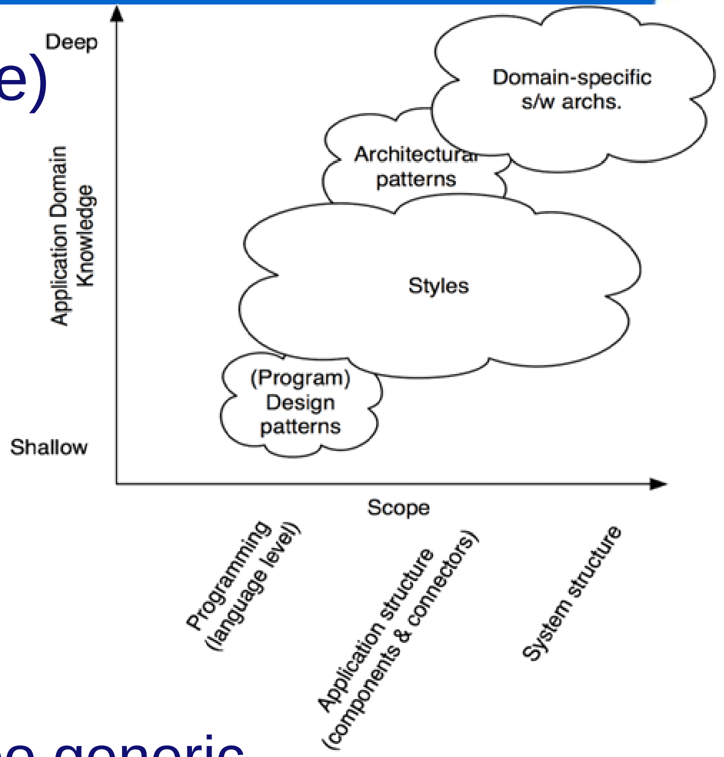
- **Experience** is crystallized as guidelines, **best practices**, do's and don'ts

- **Best practices** have different forms.


Deep
Shallow
Application Domain Knowledge
Domain-specific s/w archs.
Architectural patterns
Styles
(Program) Design patterns
2IPC0
Scope
Programming (language level)
Application structure (components & connectors)
System structure

How to solve a problem

- Solve the problem (design architecture) **from scratch**
  - Unexpected solutions can be found
  - Labor-intensive and error-prone
- Apply a **generic solution/strategy (style/pattern)** and adapt it to the problem at hand
  - Reuse, less work and less errors
  - Generic solution might be ill-fitting or too generic, requiring rework
- Apply a solution **specific for your domain (DSSA)**
  - Highest amount of reuse
  - What if such solution does not exist?



Deep
Application Domain Knowledge
Shallow
Domain-specific s/w archs.
Architectural patterns
Styles
(Program) Design patterns
Scope
Programming (language level)
Application structure (components & connectors)
System structure

Technische Universiteit Eindhoven University of Technology

Domain-Specific Software Architecture

- Highest reuse: **Domain-Specific Software Architecture**
  - Naïve: architecture recommended for software in a certain **domain**

- **Examples** of domains
  - Compilers
  - Consumer electronics
  - Electronic commerce system/Web stores
  - Video game
  - Business applications
- **Subdivision** of a domain:
  - Avionics systems -> Boeing Jets -> Boeing 747-400

Technische Universiteit Eindhoven University of Technology

Domain-Specific Software Architectures

- Formally:

A **Domain-Specific Software Architecture** (DSSA) is an assemblage of software components

- specialized for a particular **domain**,
- generalized for **effective** use across that domain, and
- composed in a **standardized structure** (topology) effective for building successful applications.

- DSSAs are the pre-eminent means for **maximal reuse of knowledge and prior development**.

Domain-Specific Software Architecture

(Hayes-Roth)

- A **domain-specific software architecture** comprises:
  - **a reference architecture**, which describes a general computational framework for a significant domain of applications;
  - **a component library**, which contains reusable chunks of domain expertise; and
  - **an application configuration method** for selecting and configuring components within the architecture to meet particular application requirements.

- Examples:

  ADAGE for avionics, AIS for adaptive intelligent systems, and MetaH for missile guidance, navigation, and control systems

Reference architecture

Reference architectures is the set of **principal design decisions** that are simultaneously applicable to **multiple related systems**, typically within an application domain, with **explicitly defined points of variation**.

Reference architecture

Reference architectures is the set of **principal design decisions** that are simultaneously applicable to **multiple related systems**, typically within an application domain, with **explicitly defined points of variation**.

Architecture, hence can be described through multiple views.

Should all follow those principal decisions.

Cover all expected variation aspects.

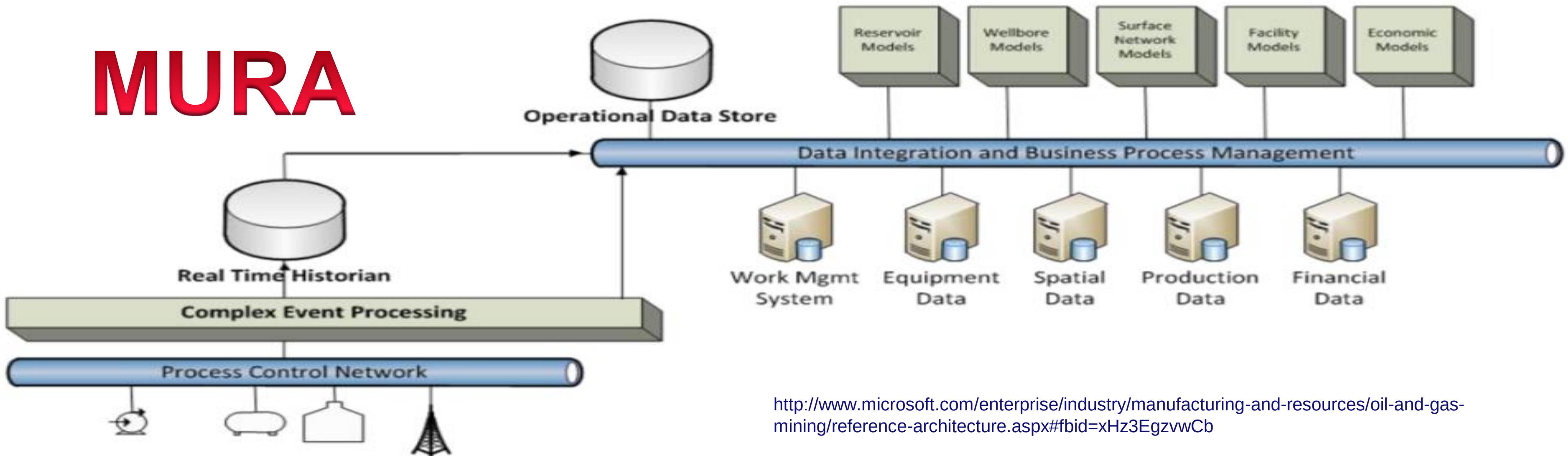
Reference architecture

Reference architectures is the set of **principal design decisions** that are simultaneously applicable to **multiple related systems**, typically within an application domain, with **explicitly defined points of variation**.

$\longrightarrow$ ***Data Integration and Business Process Management***

$\longrightarrow$ ***Which models exactly, what integration mechanisms…***

MURA



http://www.microsoft.com/enterprise/industry/manufacturing-and-resources/oil-and-gas-mining/reference-architecture.aspx#fbid=xHz3EgzvwCb

Domain-Specific Software Architecture also includes…

A component library contains reusable chunks of domain expertise.

REMINDER

Component: a modular unit with well-defined interfaces that is replaceable within its environment (UML Superstructure Specification, v.2.0, Chapter 8)

Domain-Specific Software Architecture also includes…

A component library contains reusable chunks of domain expertise.

REMINDER **Component**: a modular unit with well-defined interfaces that is replaceable within its environment (UML spec)

A **software component** is an architectural entity that

- encapsulates a subset of the system's functionality and/or data
- restricts access to that subset via an explicitly defined interface
- has explicitly defined dependencies on its required execution context (Taylor, Medvidovic, Dashofy)

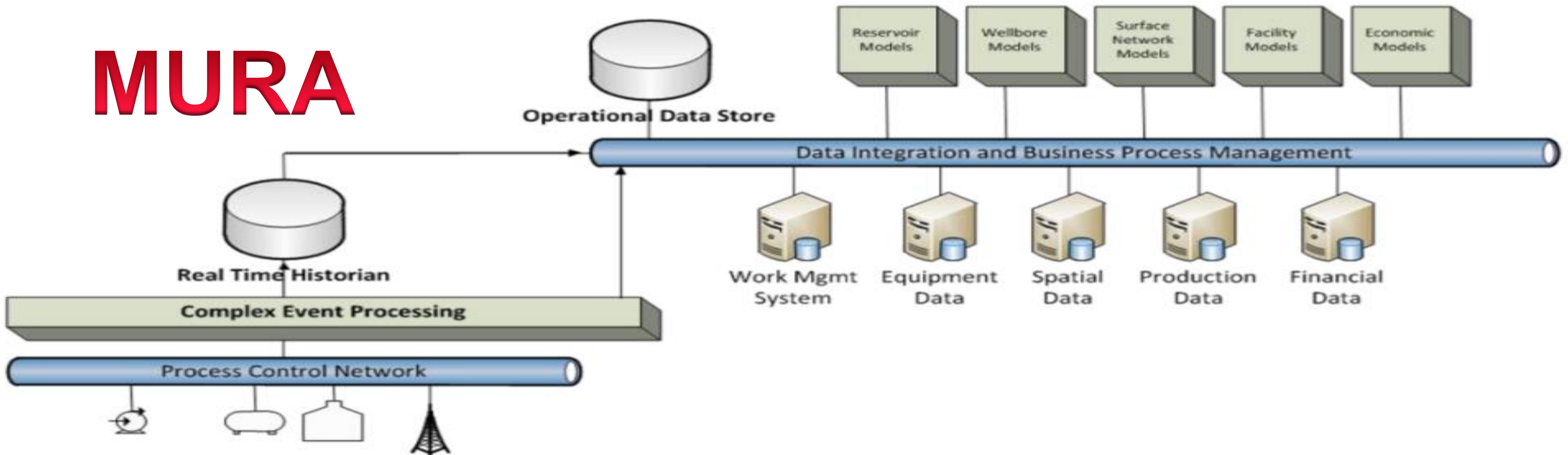
Domain-Specific Software Architecture also includes…

A component library contains reusable chunks of domain expertise.

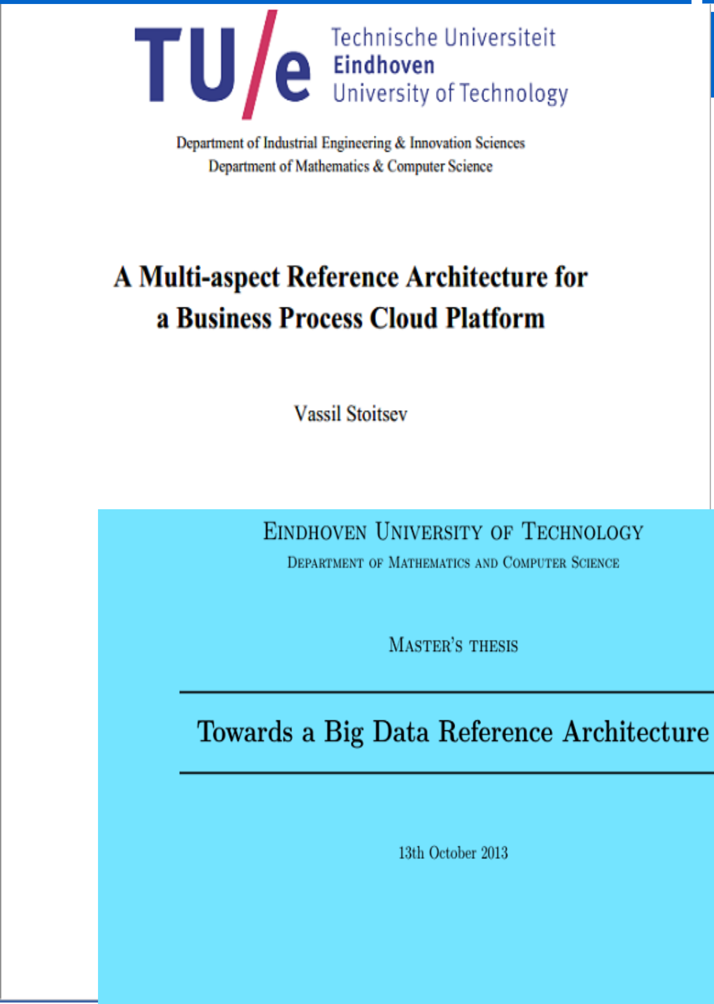
BizTalk (integration), SQL Server (data store), …

An **application configuration method** for selecting and configuring components within the architecture to meet particular application requirements.

Mapping MURA Guiding Principles to Microsoft Technology



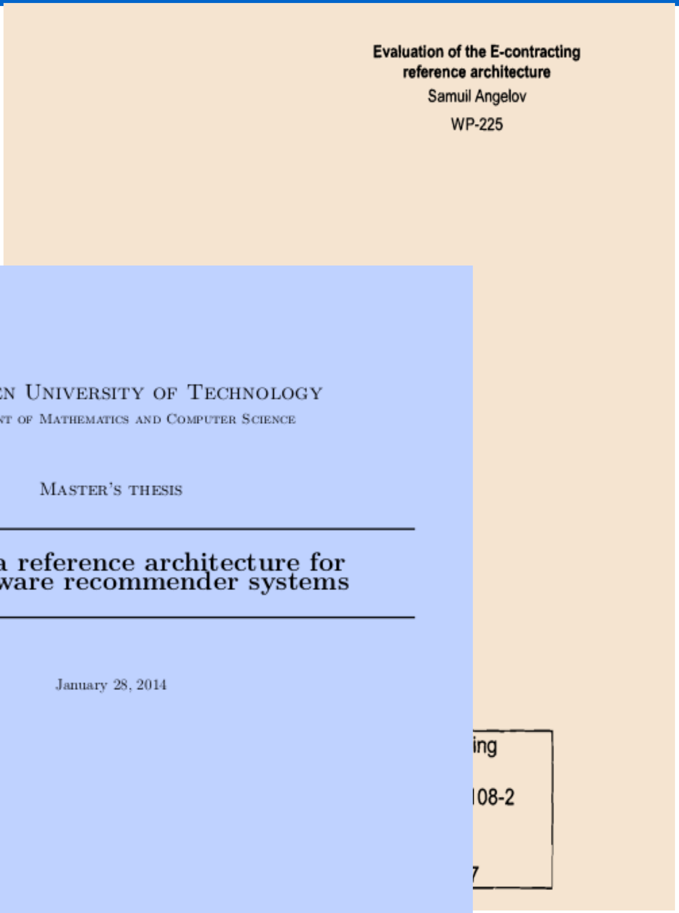
Reference architectures @TU/e


Technische Universiteit Eindhoven University of Technology

Department of Industrial Engineering & Innovation Sciences
Department of Mathematics & Computer Science

A Multi-aspect Reference Architecture for a Business Process Cloud Platform

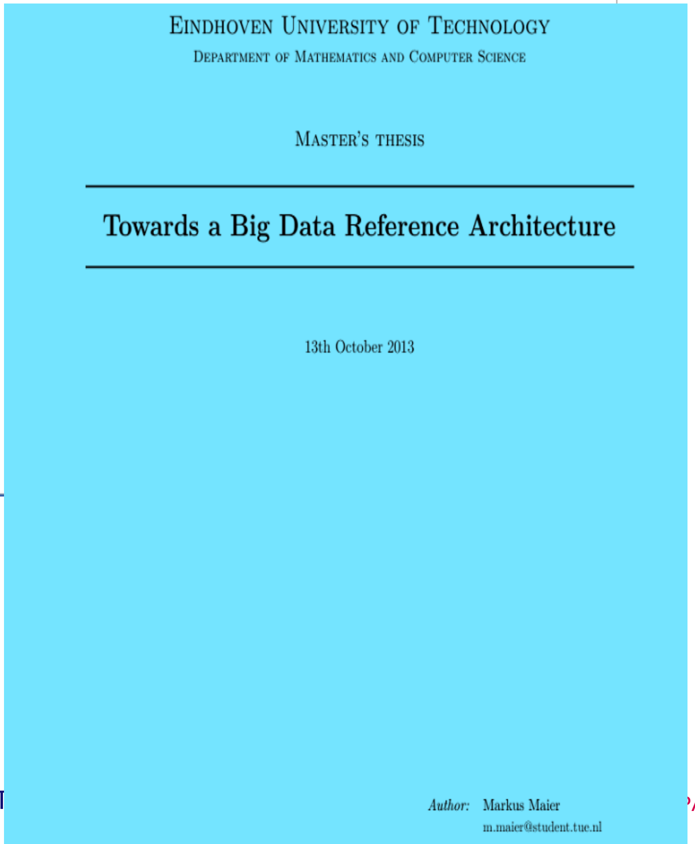
Vassil Stoitsev


Evaluation of the E-contracting reference architecture
Samuil Angelov
WP-225

ing

108-2

7


Eindhoven University of Technology
Department of Mathematics and Computer Science

Master's thesis

Towards a Big Data Reference Architecture

13th October 2013

Author: Markus Maier
m.maier@student.tue.nl

Eindhoven University of Technology
Department of Mathematics and Computer Science

Master's thesis

Towards a reference architecture for context-aware recommender systems

January 28, 2014

Author:	ing. B.M. Keijers b.m.keijers@student.tue.nl
Supervisor:	dr. M. Pechenizkiy m.pechenizkiy@tue.nl
Tutor:	Y. Kiseleva, MSc. j.kiseleva@tue.nl

Technische Universiteit Eindhoven University of Technology

Extreme case of Domain-Specific Software Architecture

- What happens when the domain becomes **narrower**?
  - Consumer Electronics ⇒ Sony WEGA TVs
  - Avionics ⇒ Boeing 747 Family
  - …

- **Engineering Product Line**: a set of products that have **substantial commonality** from a technical/engineering perspective

Technische Universiteit Eindhoven University of Technology

Engineering PL vs Business PL

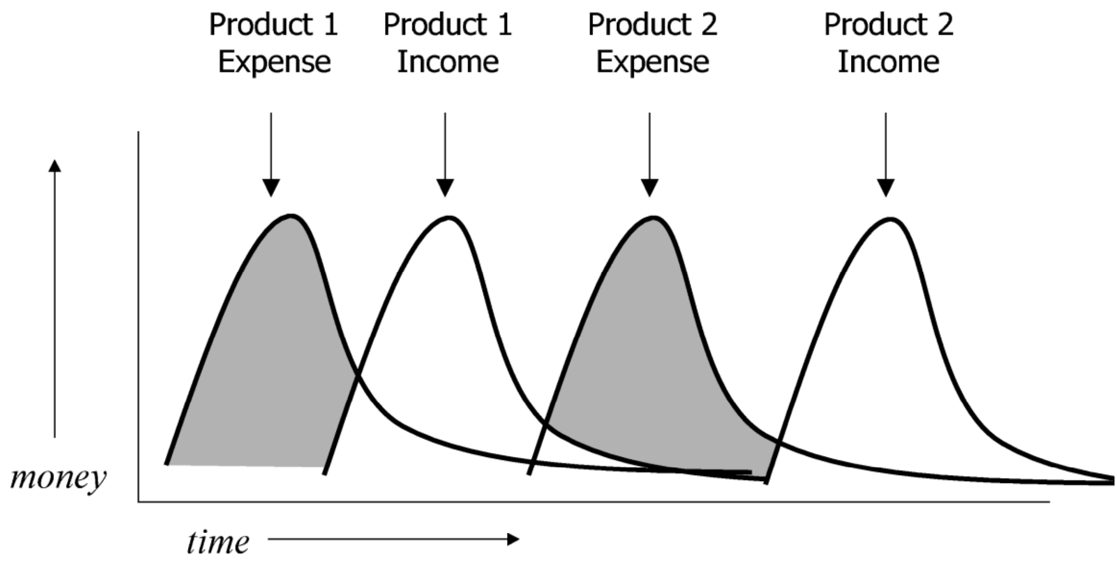
- **Engineering Product Line**: a set of products that have **substantial commonality** from a technical/engineering perspective

- **Business Product Line**: A set of products marketed under a **common banner** to increase sales and market penetration through bundling and integration

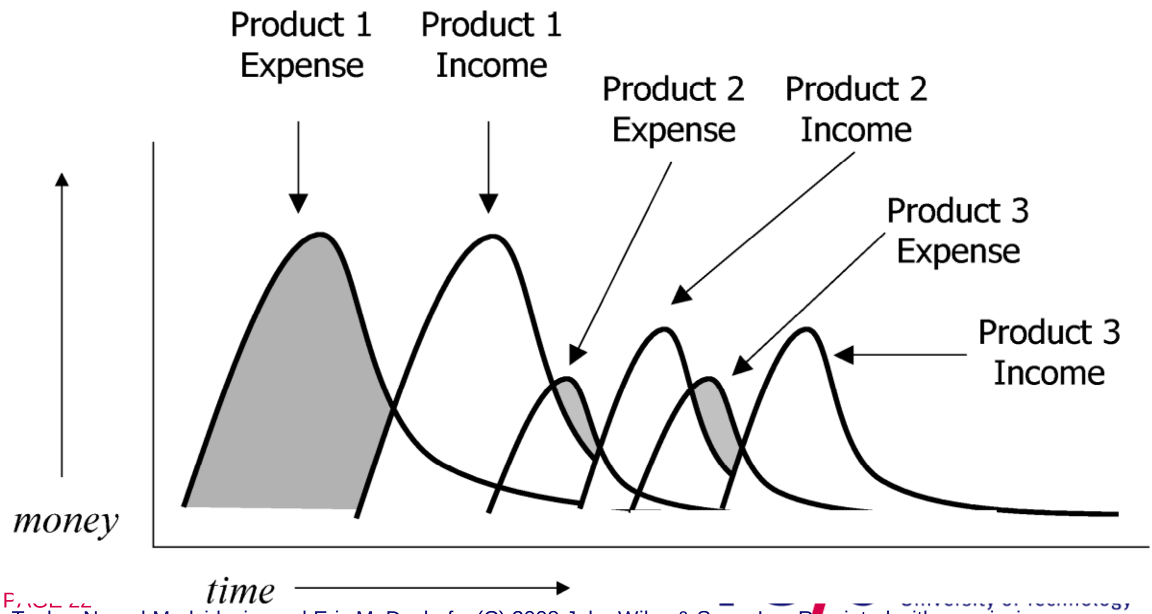
- Business product lines *usually are* engineering product lines and vice-versa, but not always
  - Applications bundled after a company acquisition
  - Chrysler Crossfire & Mercedes SLK V6

Product lines – why?

Traditional engineering



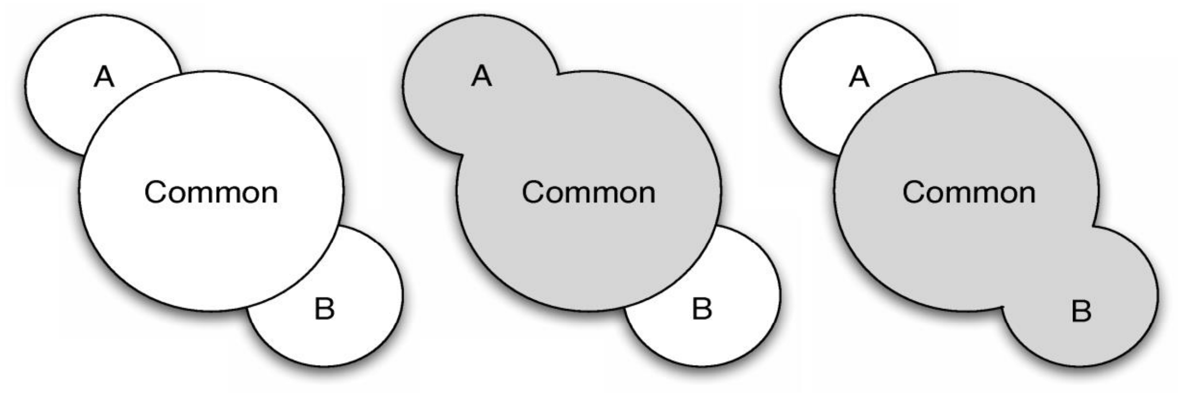
Product-line-based engineering



Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; (C) 2008 John Wiley & Sons, Inc. Reprinted with permission.

A Product-Line Architecture

- A **product-line architecture** captures the architectures of many related products simultaneously
  - Explicit **variation points**



- **Common**: features common to all products
- **A**: features specific to product A
  - Product A = Common + A
- **B**: features specific to product B
  - Product B = Common + B

How do product lines come to be?

- **Design**: expected variation points (now) / evolution scenarios (future)
  - List **current or envisioned features** of the product
    - If features are not explicit, list components and group them to (mostly) **orthogonal** features, or features that would be beneficial in **different products**/for different customers
  - Identify which **combinations of features** form feasible and marketable products
    - Only **some combinations** are meaningful!

How do product lines come to be?

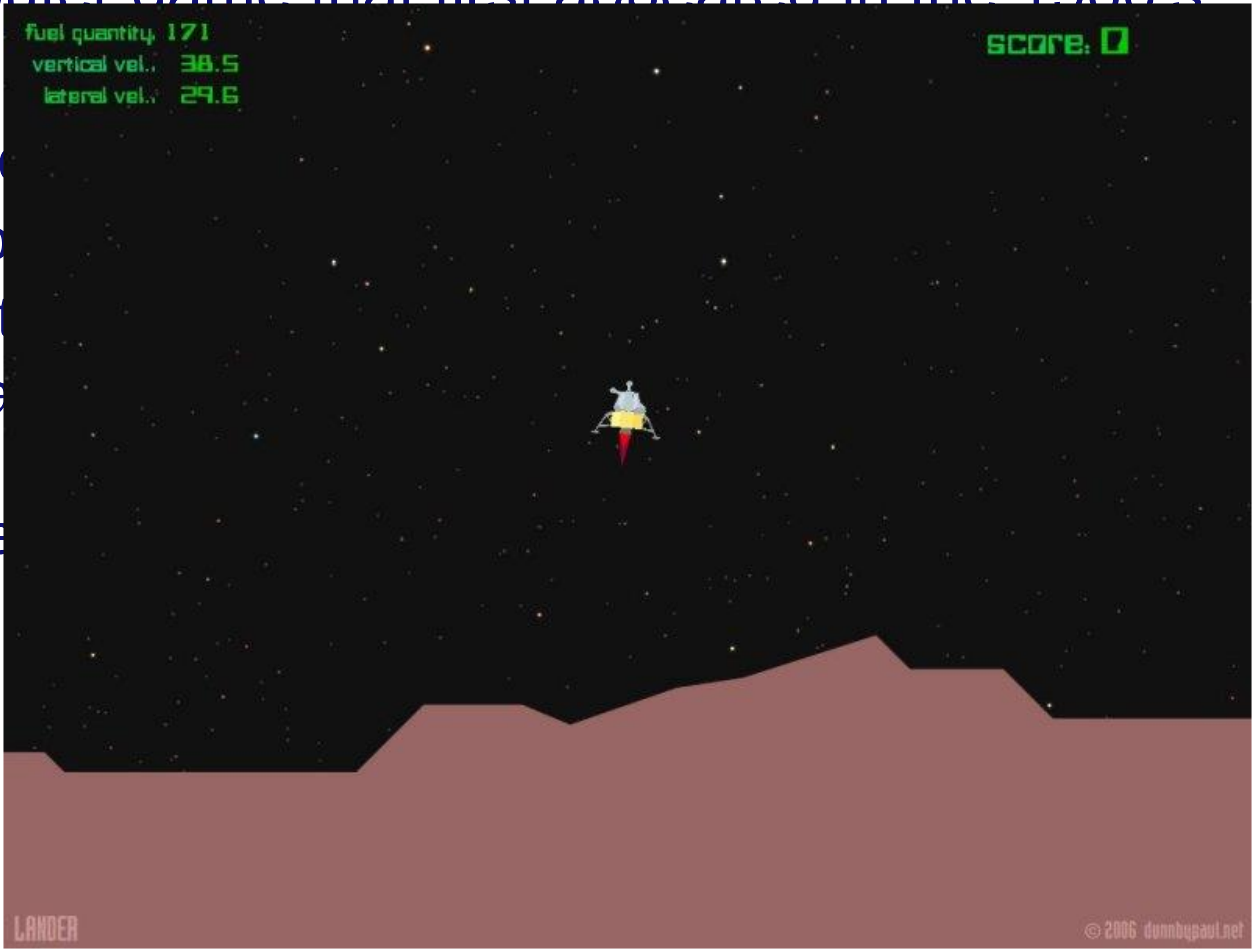
- **Unification**: *after* several products have been implemented and commonality is noticed
  - No product line
    - It may be more expensive to create a product line or there may not be enough commonality
  - One master product
    - One product architecture becomes the basis for the product line
  - Hybrid
    - A new product line architecture emerges out of many products
    - Seems ideal but can be hard in practice

The Lunar Lander: A Running Example

- Computer game that first appeared in the 1960’s
- You control the descent rate of the Lunar Lander
  - Throttle setting controls descent engine
  - Limited fuel
  - Initial altitude and speed preset
  - If you land with a descent rate of < 5 fps: you win (whether there’s fuel left or not)
- “Advanced” version: joystick controls attitude & horizontal motion

The Lunar Lander: A Running Example

- Computer game that first appeared in the 1960's
- You c
  - Thro
  - Limi
  - Initia
  - If yo
    (whe



Product lines in the Lunar Lander

- We have a basic version
  - Components: data store, game logic, text-based UI

Product lines in the Lunar Lander

- We have a basic version
  - Components: data store, game logic, text-based UI

- We want to add a graphical UI and earn a lot of money
  - Free “Demo” with “Buy me” reminder when the game time expired
  - Components: data store, game logic, text-based UI, graphical UI, demo reminder, system clock

Product lines in the Lunar Lander

- We have a basic version
  - Components: data store, game logic, text-based UI

- We want to add a graphical UI and earn a lot of money
  - Free "Demo" with "Buy me" reminder when the game time expired
  - Components: data store, game logic, text-based UI, graphical UI, demo reminder, system clock

	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Basic	X	X	X			
Demo	X	X		X	X	X
Purchased	X	X		X		

Product lines: Components, Features, Products

1) List components

Components →	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Basic	X	X	X			
Demo	X	X		X	X	X
Purchased	X	X		X		

Product lines: Components, Features, Products

1) List components

Components →	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Basic	X	X	X			
Demo	X	X		X	X	X
Purchased	X	X		X		

2) Identify features

Features ↓	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Core	X	X				
Text UI			X			
Graphical UI				X		
Time-limited					X	X

Product lines: Components, Features, Products

1) List components

Components	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Basic	X	X	X			
Demo	X	X		X	X	X
Purchased	X	X		X		

2) Identify features

Features	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Core	X	X				
Text UI			X			
Graphical UI				X		
Time-limited					X	X

3) Construct intended products

Products	Core	Text UI	Graphical UI	Time-limited
Basic	X	X		
Demo	X		X	X
Purchased	X		X	

Technische Universiteit Eindhoven University of Technology

Product lines: Components, Features, Products

1) List components

Components →	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Basic	X	X	X			
Demo	X	X		X	X	X
Purchased	X	X		X		

2) Identify features

Features ↓	Data Store	Game Logic	Text-based UI	Graphical UI	Demo Reminder	System Clock
Core	X	X				
Text UI			X			
Graphical UI				X		
Time-limited					X	X

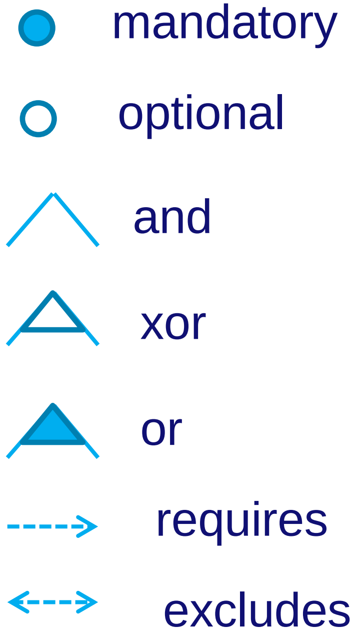
3) Construct intended products

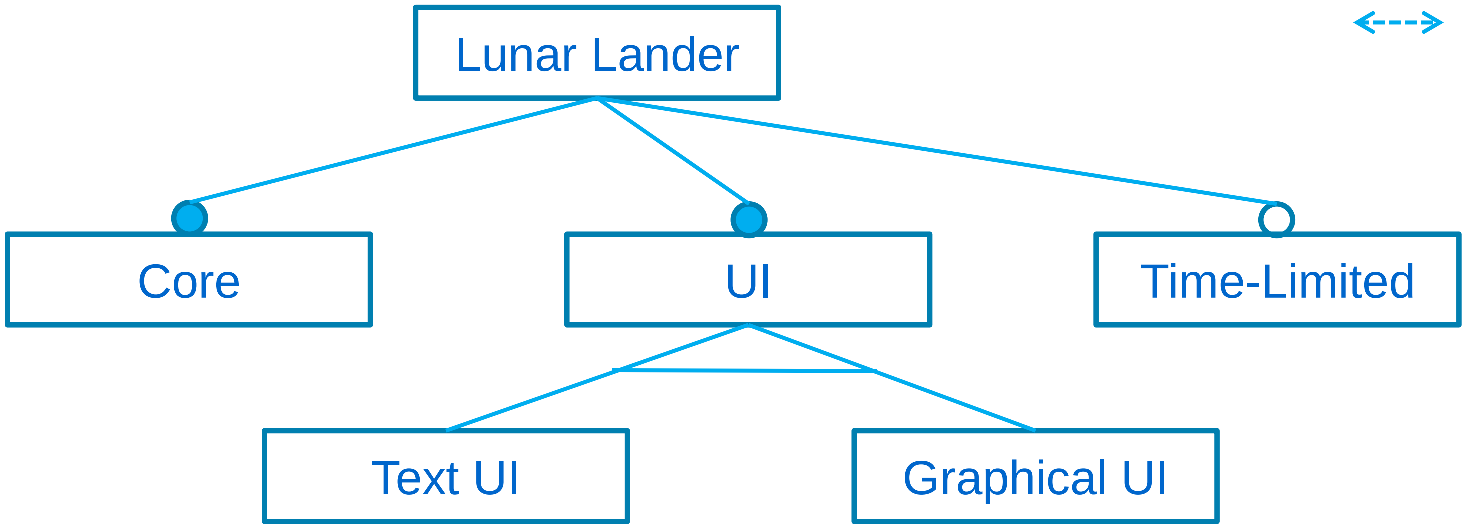
4) Identify new opportunities

Products ↓	Core	Text UI	Graphical UI	Time-limited
Basic	X	X		
Demo	X		X	X
Purchased	X		X	
Demo Text	X	X		X

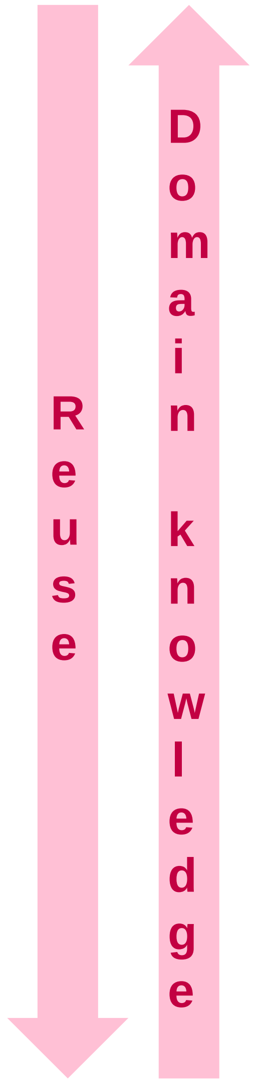
A better representation: variability model

Products	Core	Text UI	Graphical UI	Time-limited
Basic	X	X		
Demo	X		X	X
Purchased	X		X	
Demo Text	X	X		X





DSSA and Product Lines?



- **Product lines**
  - Explicit set of related products with common aspects
- **Domain-Specific Software Architectures**
  - Domain specific; includes elaborate domain model and specific reference architecture
- **Architectural Styles and Patterns**
- **Design Patterns (2IPC0)**

Architectural Patterns

- An **architectural pattern** is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears.

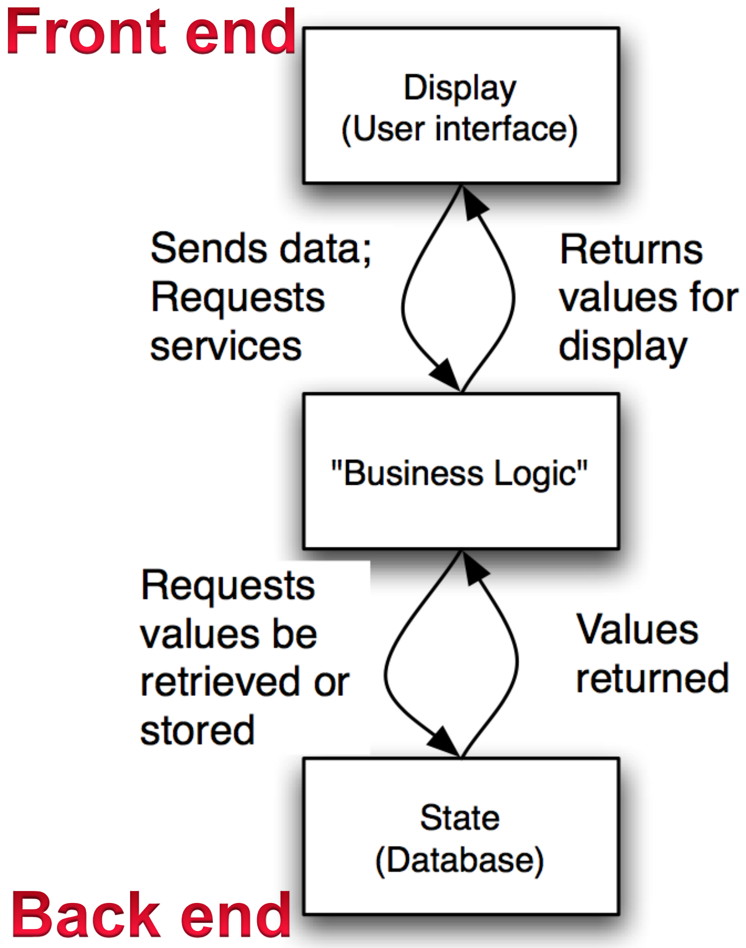
- Similar to DSSAs but applied “at a lower level” and within a much narrower scope.

- Examples:
  - State-Logic-Display: Three-Tiered Pattern
  - Model-View-Controller
  - Sense-Compute-Control

Technische Universiteit **Eindhoven** University of Technology

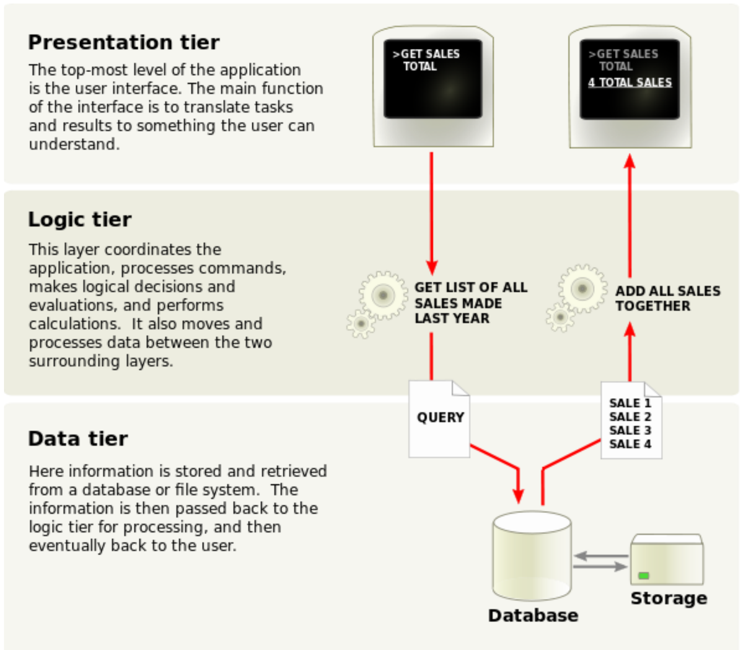
State-Logic-Display (a.k.a. Three-Tiered Pattern)

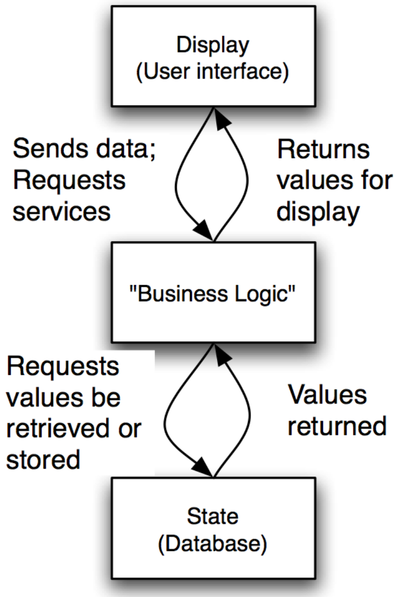
- **“Business logic”**
  - Tax calculation rules
  - Game rules
  - …
- **Application Examples**
  - Business applications
  - Multi-player games
  - Web-based applications



Front end
Display (User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State (Database)
Back end

Tiers and Layers


Presentation tier
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.
>GET SALES TOTAL
>GET SALES TOTAL
4 TOTAL SALES
Logic tier
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
GET LIST OF ALL SALES MADE LAST YEAR
ADD ALL SALES TOGETHER
Data tier
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.
QUERY
SALE 1
SALE 2
SALE 3
SALE 4
Database
Storage


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

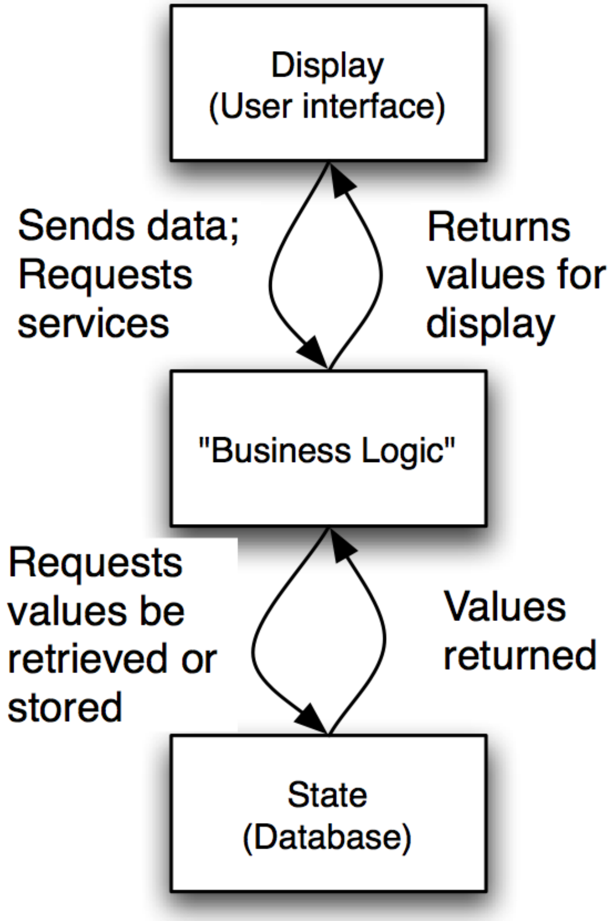
http://upload.wikimedia.org/wikipedia/commons/5/51/Overview_of_a_three-tier_application_vectorVersion.svg

- **Tiers**: physical distribution of components of a system on separate servers, computers, or networks (nodes)
- **Layers**: logical grouping of components
  - Components may or may not be located on the same node
- The middle tier may be multi-tiered itself (resulting in an "n-tier architecture")

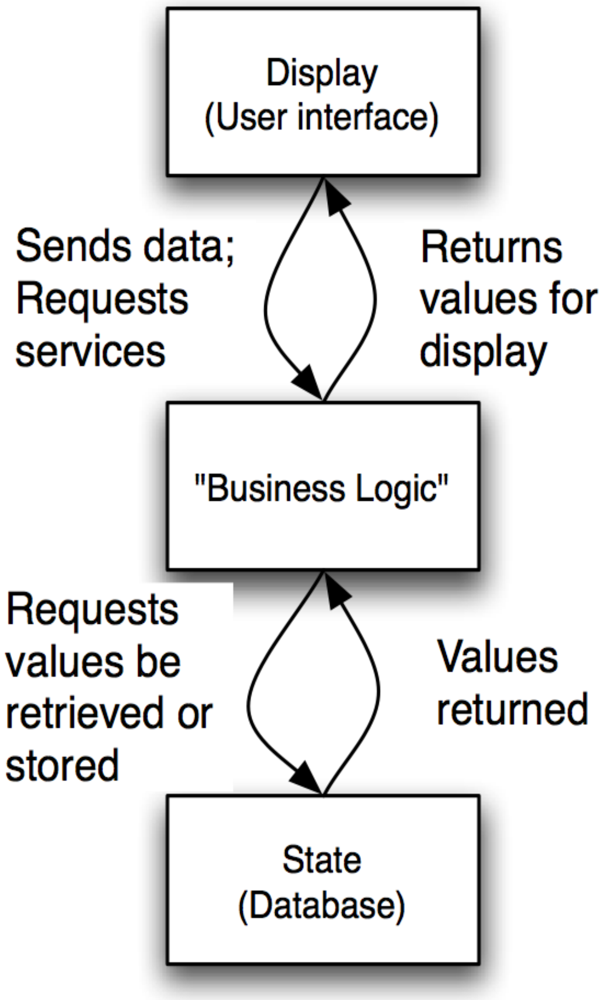
State-Logic-Display (a.k.a. Three-Tiered Pattern)

- **Fundamental rule:**
  - No direct communication between Display and State

- Display, Logic and State
  - are developed and maintained as independent modules,
  - most often on separate platforms
  - often using different technologies


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

State-Logic-Display in Web development


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

Static or cached dynamic content rendered by the browser.
JavaScript, Ajax, Flash, jQuery…

Dynamic content processing and generation level application server
Java, .NET, ColdFusion, PHP, Perl, Rails…

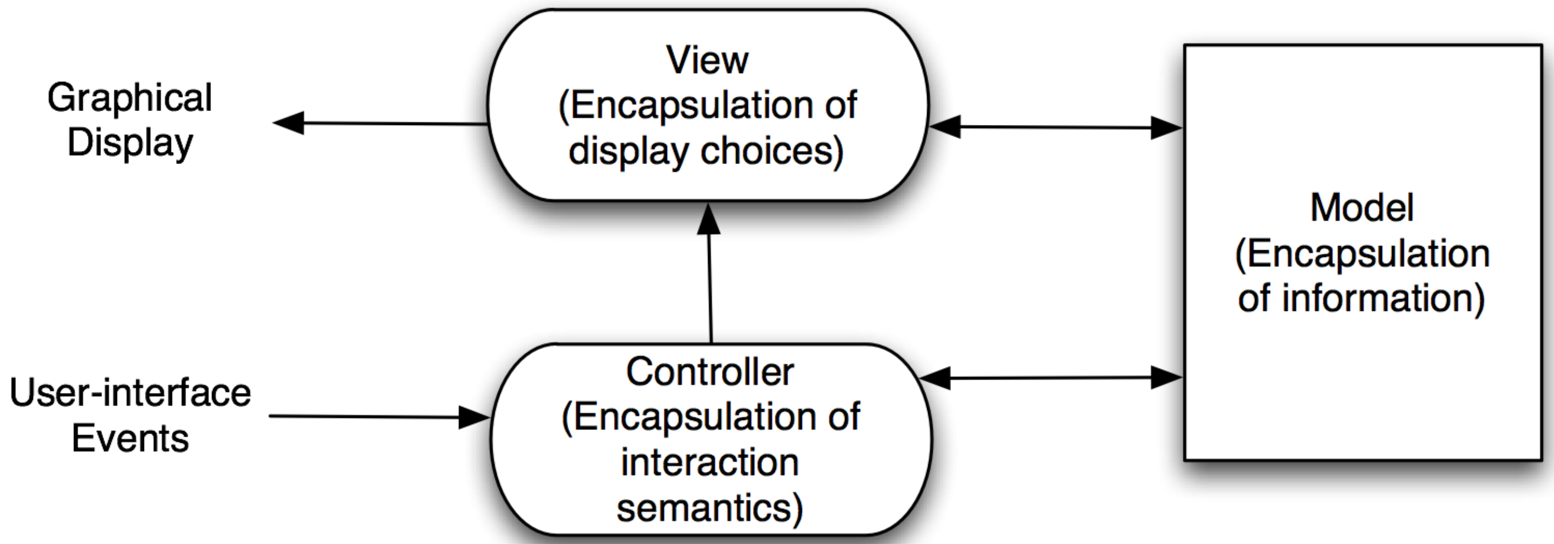
Database + connection (e.g., ORM like Hibernate, Java Persistence API, …)

Model-View-Controller (MVC)

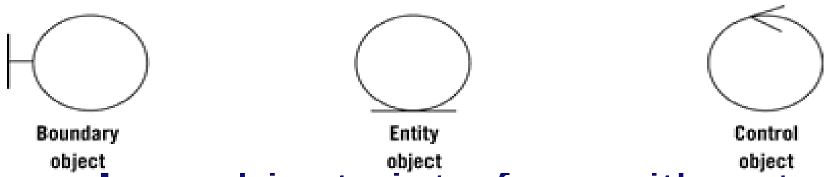
- **Objective:** Separation between information, presentation and user interaction.

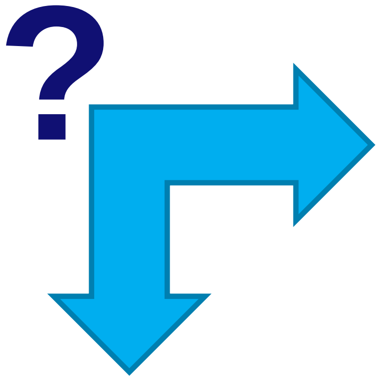
- When a **model** object value changes, a notification is sent to the **view** and to the **controller**.
  - view updates itself
  - controller modifies the view if its logic so requires.
- User input is sent to the controller
  - If a change is required, the controller updates the model.

Model-View-Controller

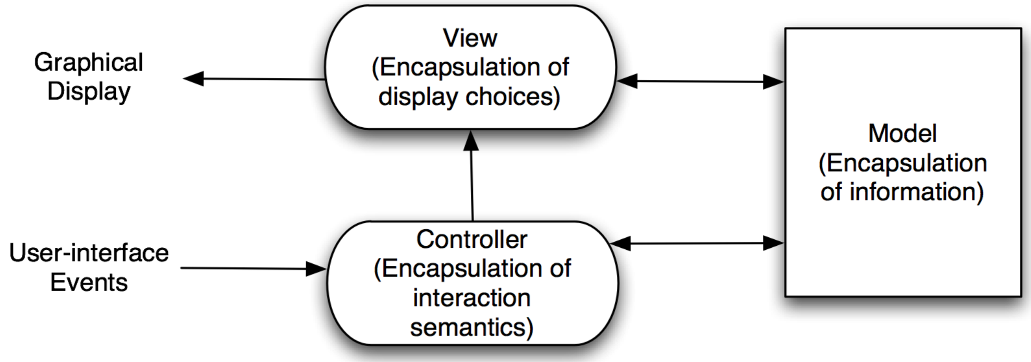

Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

Do you recall?



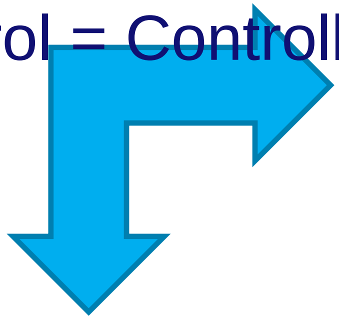


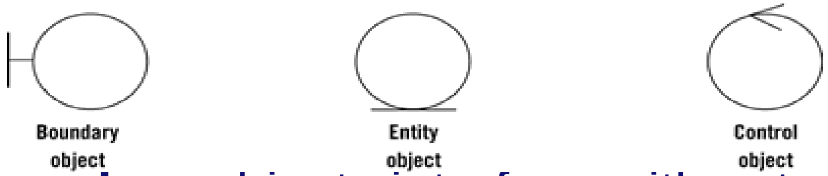
- **Boundary** objects interface with actors.
- **Entity** objects represent system data, often from the domain.
- **Control** objects glue boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

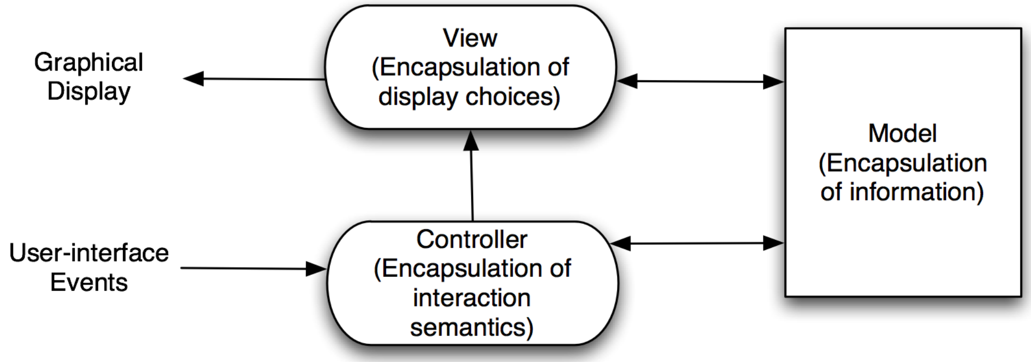
Do you recall?

Boundary = View
Entity = Model
Control = Controller



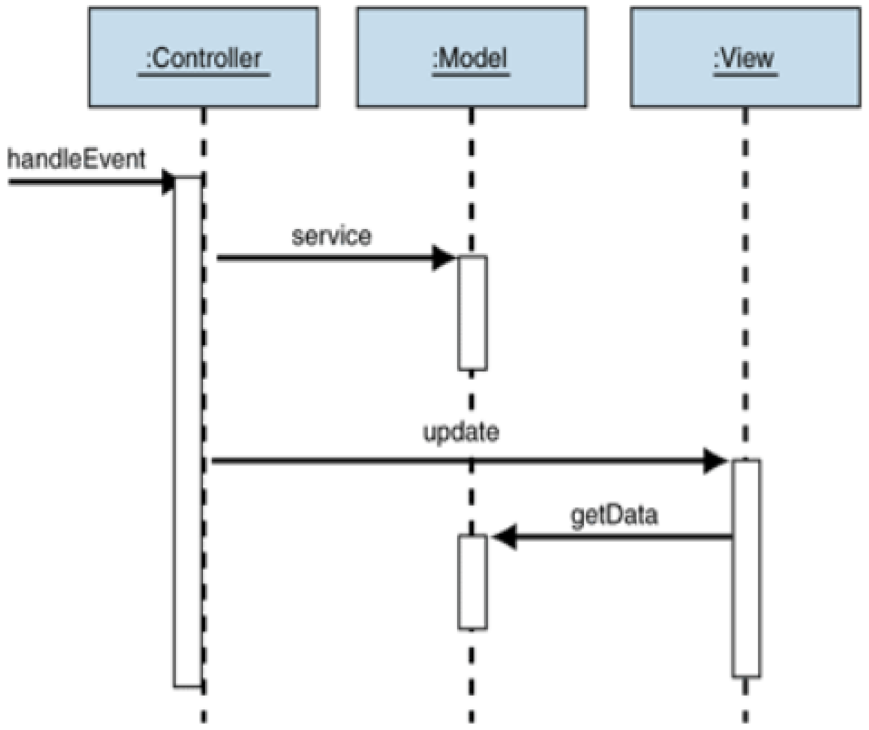


- **Boundary** objects interface with actors.
- **Entity** objects represent system data, often from the domain.
- **Control** objects glue boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

Two flavors of MVC: Passive model

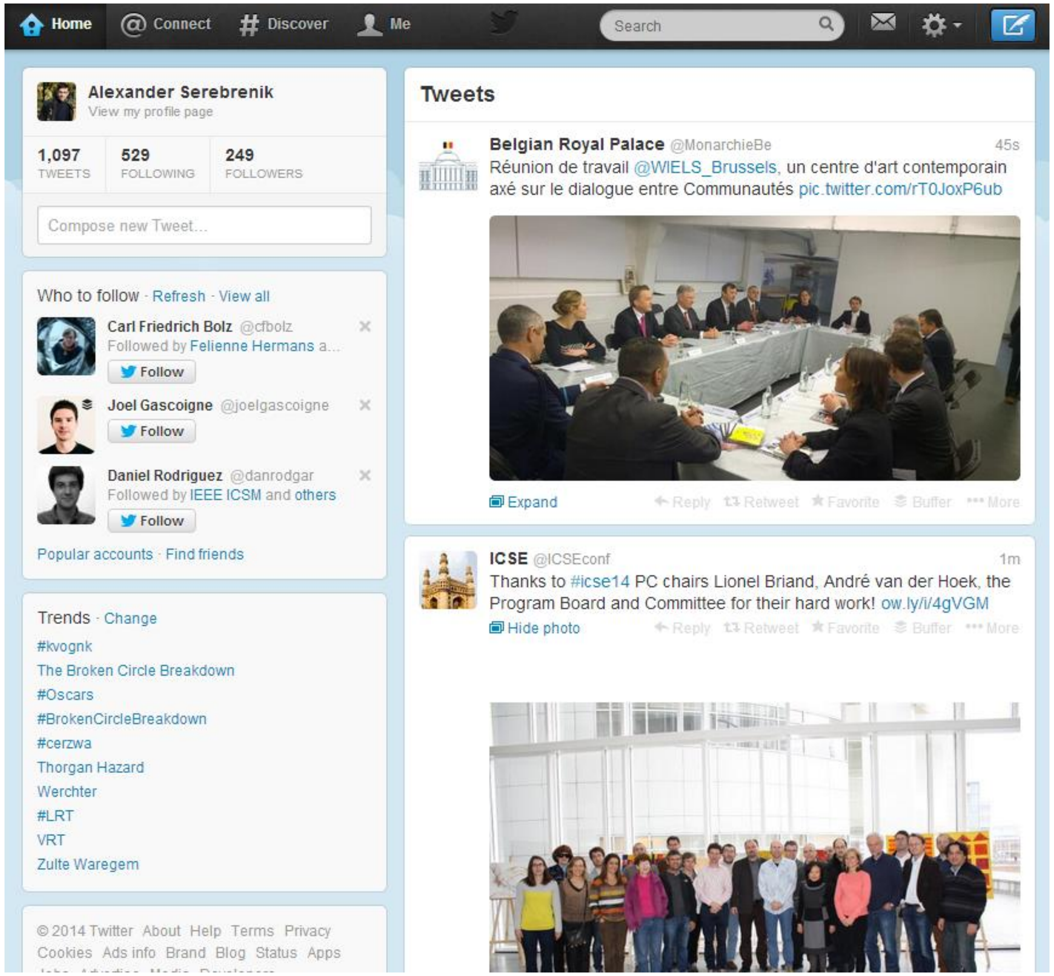
- **Passive model**
  - Model is completely controlled by the Controller and cannot change independently
  - Model change is *always* a reaction to user's actions.



- Example: HTTP
  - The browser displays the view and responds to user input, but it does not detect changes in the data on the server.

Two flavors of MVC: Active model

- **Active model**
  - Model can change without involving Controller
    - e.g., other sources are changing the data and the changes must be reflected in the views.



Alexander Serebrenik
View my profile page

1,097 TWEETS | 529 FOLLOWING | 249 FOLLOWERS

Who to follow · Refresh · View all

Carl Friedrich Bolz @cfbolz
Followed by Felienne Hermans a...

Joel Gascoigne @joelgascoigne

Daniel Rodriguez @danrodgar
Followed by IEEE ICSM and others

Popular accounts · Find friends

Trends · Change
#kvognk
The Broken Circle Breakdown
#Oscars
#BrokenCircleBreakdown
#cerzwa
Thorgan Hazard
Werchter
#LRT
VRT
Zulte Waregem

© 2014 Twitter About Help Terms Privacy
Cookies Ads info Brand Blog Status Apps

Tweets

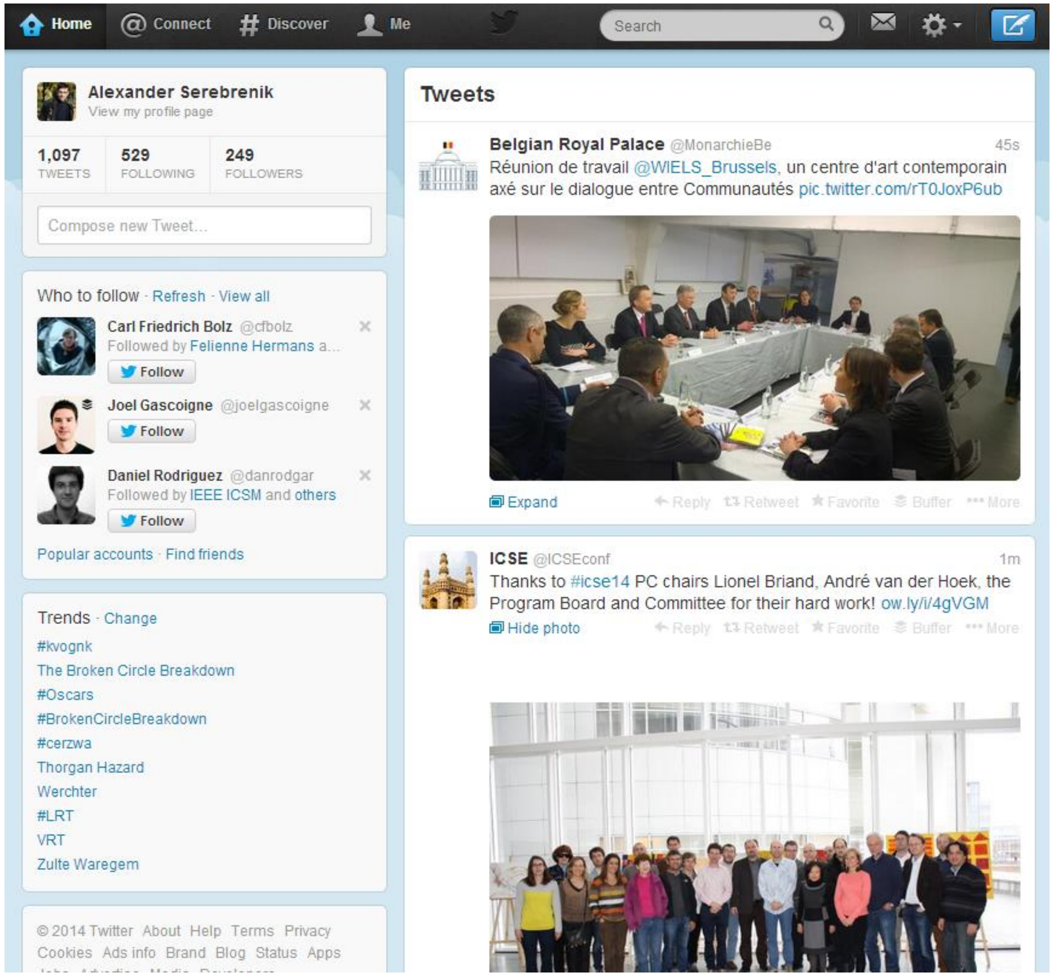
Belgian Royal Palace @MonarchieBe 45s
Réunion de travail @WIELS_Brussels, un centre d'art contemporain axé sur le dialogue entre Communautés pic.twitter.com/rT0JoxP6ub

ICSE @ICSEconf 1m
Thanks to #icse14 PC chairs Lionel Briand, André van der Hoek, the Program Board and Committee for their hard work! ow.ly/i/4gVGM

Two flavors of MVC: Active model

- **Active model**
  - Model can change without involving Controller
    - e.g., other sources are changing the data and the changes must be reflected in the views.
- However, Model should not be aware of its Views!

- **Software Science students**: which design pattern can solve this problem?



Alexander Serebrenik
View my profile page

1,097 TWEETS | 529 FOLLOWING | 249 FOLLOWERS

Compose new Tweet...

Who to follow · Refresh · View all

Carl Friedrich Bolz @cfbolz
Followed by Felienne Hermans a...
Follow

Joel Gascoigne @joelgascoigne
Follow

Daniel Rodriguez @danrodgar
Followed by IEEE ICSM and others
Follow

Popular accounts · Find friends

Trends · Change
#kvognk
The Broken Circle Breakdown
#Oscars
#BrokenCircleBreakdown
#cerzwa
Thorgan Hazard
Werchter
#LRT
VRT
Zulte Waregem

© 2014 Twitter About Help Terms Privacy
Cookies Ads info Brand Blog Status Apps

Tweets

Belgian Royal Palace @MonarchieBe 45s
Réunion de travail @WIELS_Brussels, un centre d'art contemporain axé sur le dialogue entre Communautés pic.twitter.com/rT0JoxP6ub
Expand · Reply · Retweet · Favorite · Buffer · More

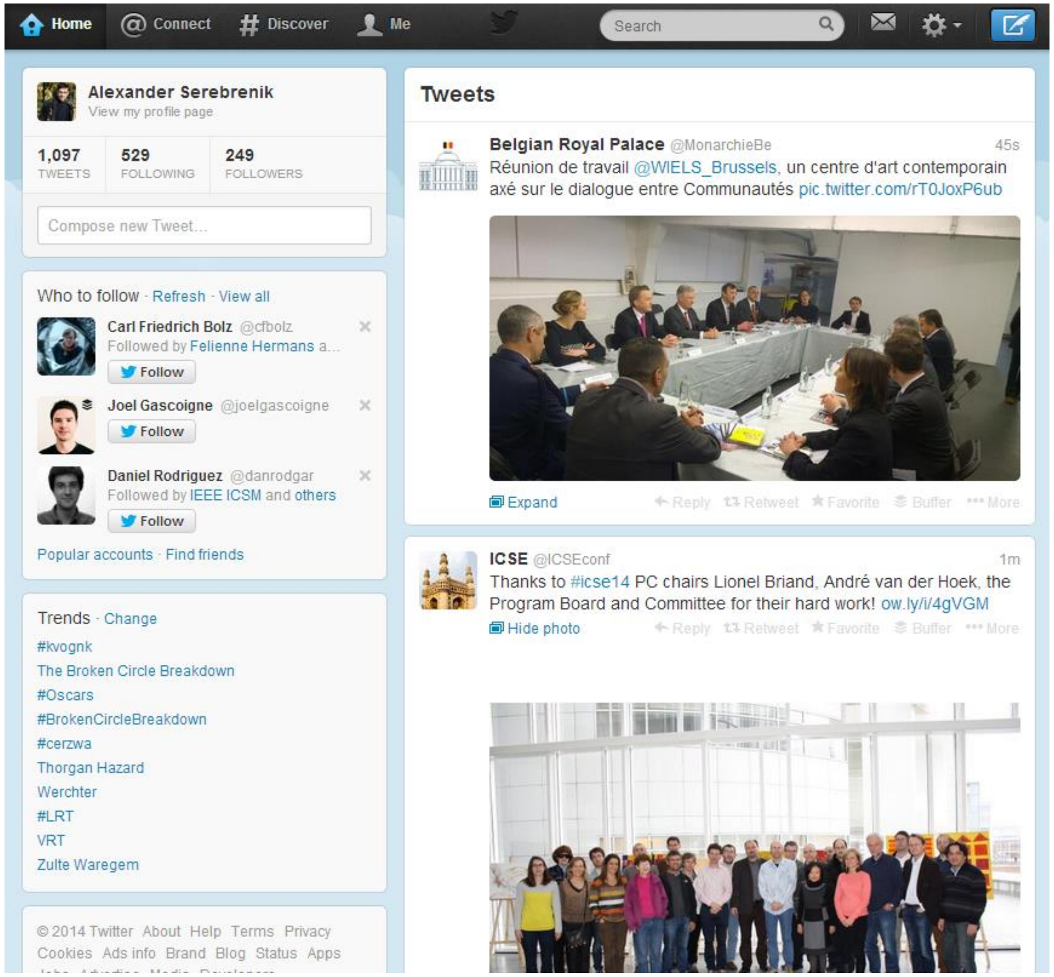
ICSE @ICSEconf 1m
Thanks to #icse14 PC chairs Lionel Briand, André van der Hoek, the Program Board and Committee for their hard work! ow.ly/i/4gVGM
Hide photo · Reply · Retweet · Favorite · Buffer · More

Two flavors of MVC: Active model

- **Active model**
  - Model can change without involving Controller
    - e.g., other sources are changing the data and the changes must be reflected in the views.
- However, Model should not be aware of its Views!

- **Software Science students**: which design pattern can solve this problem?

Observer



Alexander Serebrenik
View my profile page

1,097 TWEETS | 529 FOLLOWING | 249 FOLLOWERS

Compose new Tweet...

Who to follow · Refresh · View all

Carl Friedrich Bolz @cfbolz
Followed by Felienne Hermans a...
Follow

Joel Gascoigne @joelgascoigne
Follow

Daniel Rodriguez @danrodgar
Followed by IEEE ICSM and others
Follow

Popular accounts · Find friends

Trends · Change
#kvognk
The Broken Circle Breakdown
#Oscars
#BrokenCircleBreakdown
#cerzwa
Thorgan Hazard
Werchter
#LRT
VRT
Zulte Waregem

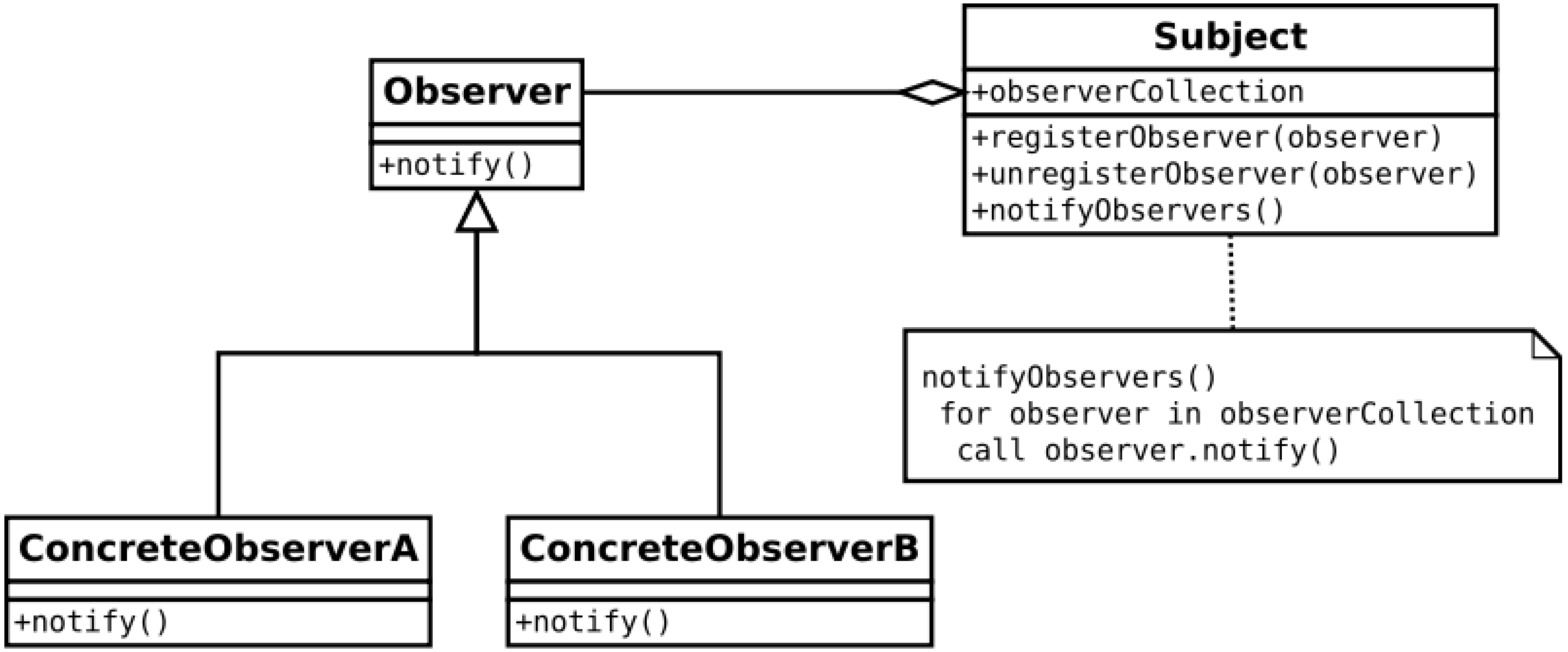
© 2014 Twitter About Help Terms Privacy
Cookies Ads info Brand Blog Status Apps

Tweets

Belgian Royal Palace @MonarchieBe 45s
Réunion de travail @WIELS_Brussels, un centre d'art contemporain axé sur le dialogue entre Communautés pic.twitter.com/rT0JoxP6ub
Expand Reply Retweet Favorite Buffer More

ICSE @ICSEconf 1m
Thanks to #icse14 PC chairs Lionel Briand, André van der Hoek, the Program Board and Committee for their hard work! ow.ly/i/4gVGM
Hide photo Reply Retweet Favorite Buffer More

Observer pattern



http://upload.wikimedia.org/wikipedia/commons/8/8d/Observer.svg

- Java: **Observer** as an interface, **Observable** as a class.
  - Model inherits from Observable, View/Controller implement Observer.

Benefits of MVC

- Supports **multiple** views
  - Users can individually change the appearance of the web-pages based on the same model

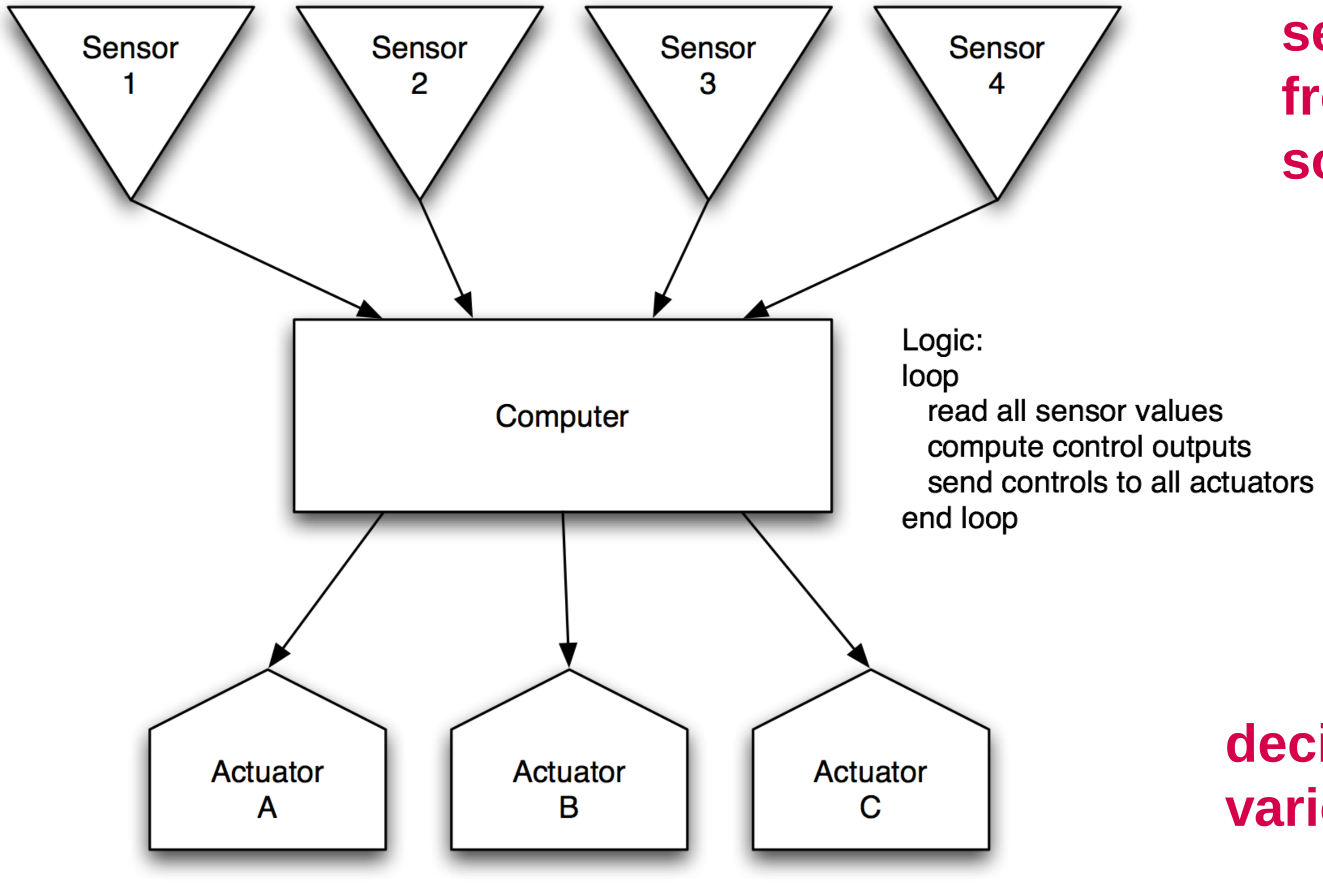
- Well-suited for **evolution**
  - User interface requirements change faster than the models
  - Changes are limited to the views only

Liabilities of MVC

- **Complexity**
  - new levels of indirection
  - behavior becomes more event-driven complicating debugging
- **Communication**
  - If model is frequently updated, it could flood the views with update requests.

Sense-Compute-Control

Objective: Structuring embedded control applications

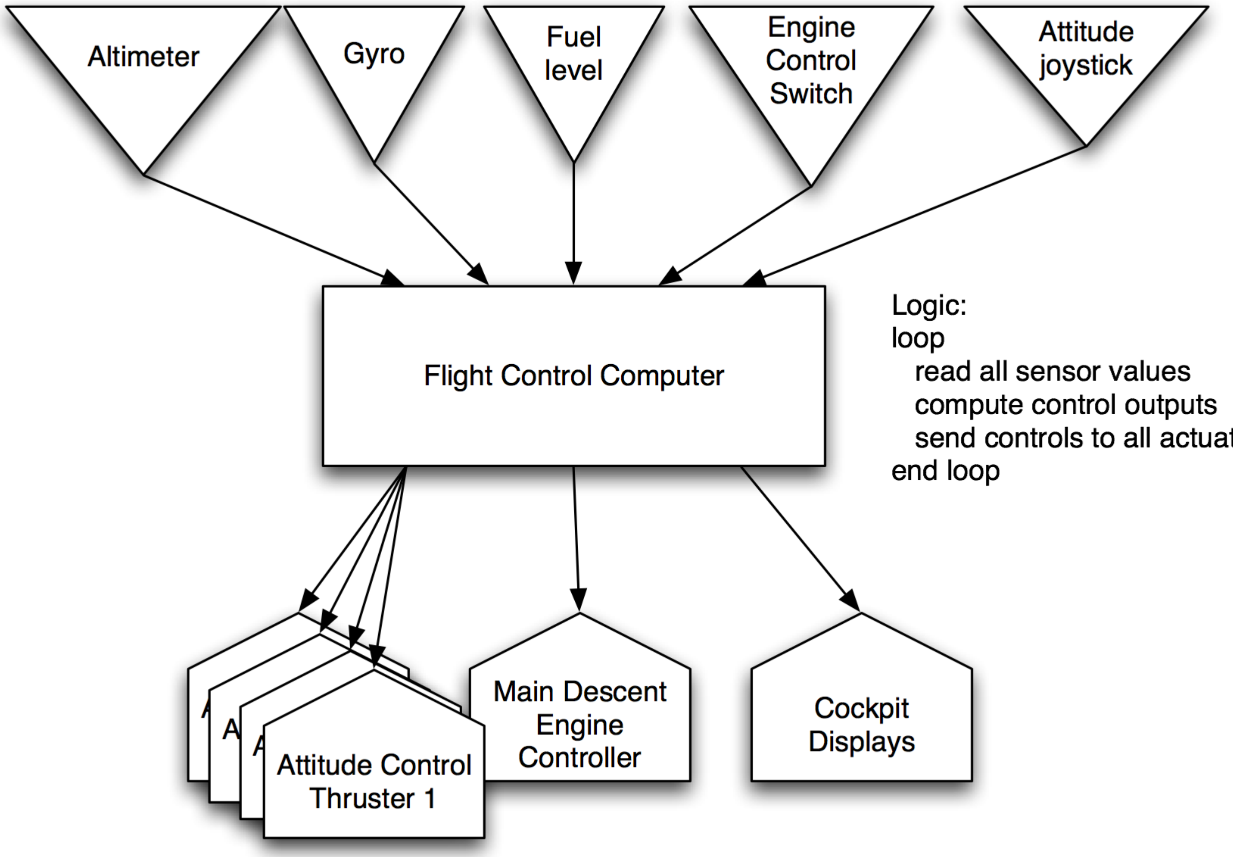


```
Logic:
loop
  read all sensor values
  compute control outputs
  send controls to all actuators
end loop
```

send information from various sources

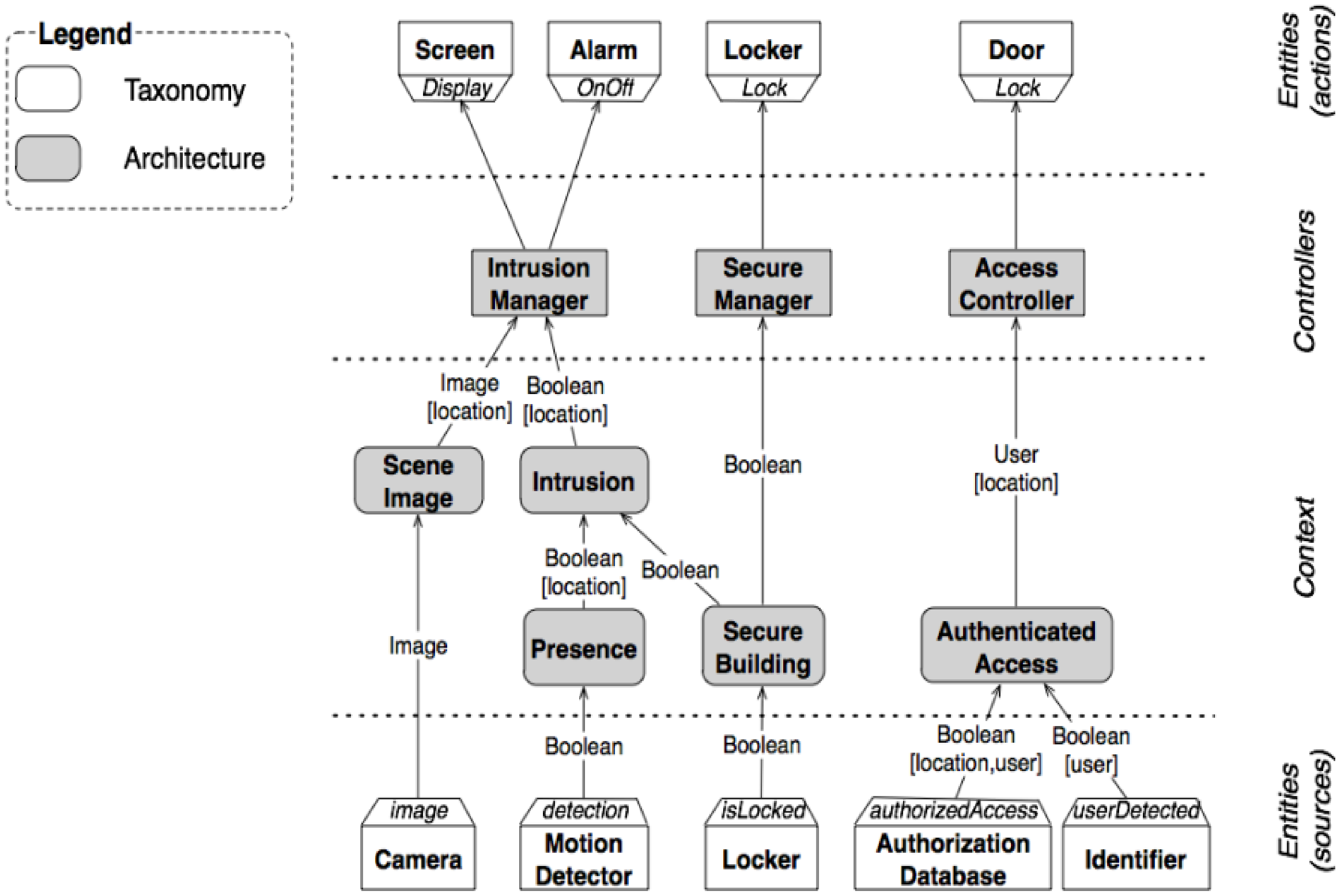
decide how to control various devices.

Sense-Compute-Control Lunar Lander



```
Logic:
loop
   read all sensor values
   compute control outputs
   send controls to all actuators
end loop
```

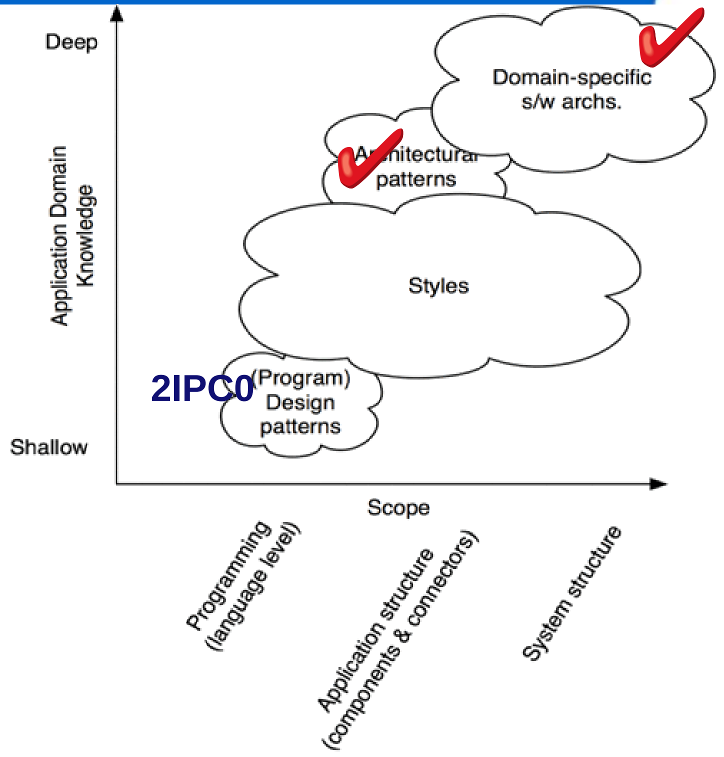
Example: Intrusion/Access Management



Architectural patterns vs. Architectural styles vs. Design patterns

Next time:

- **Architectural styles** define the components and connectors ('what?')
  - Less domain specific
- **Architectural patterns** define the implementation strategies of those components and connectors ('how?')
  - More domain specific
  - Difference pattern/style is not too sharp


Deep
Shallow
Application Domain Knowledge
Domain-specific s/w archs.
Architectural patterns
Styles
2IPC0
(Program) Design patterns
Scope
Programming (language level)
Application structure (components & connectors)
System structure

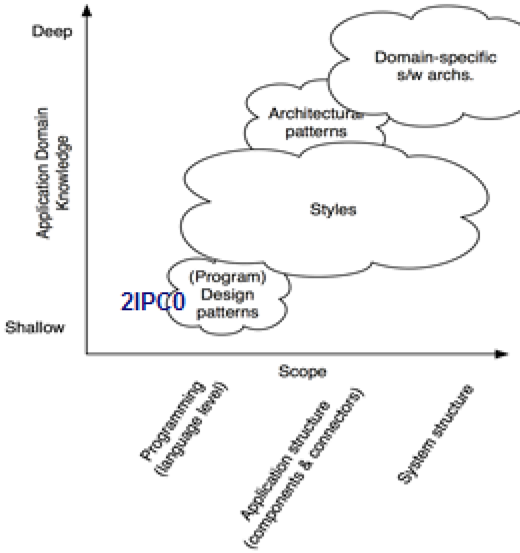
STOP!

- “**Architectural styles** define the components and connectors”

- A **software connector** is an architectural building block tasked with effecting and regulating interactions among components (Taylor, Medvidovic, Dashofy)
  - Procedure call connectors
  - Shared memory connectors
  - Message passing connectors
  - Streaming connectors
  - Distribution connectors
  - Wrapper/adaptor connectors
  - …

Learning from Others: Patterns, Styles, and DSSAs

- **Experience** is crystallized as guidelines, **best practices**, do's and don'ts
- **Best practices** have different forms.


Deep
Shallow
Application Domain Knowledge
Domain-specific s/w archs.
Architecture patterns
Styles
(Program) Design patterns
2IPC0
Scope
Programming (language level)
Application structure (components & connectors)
System structure

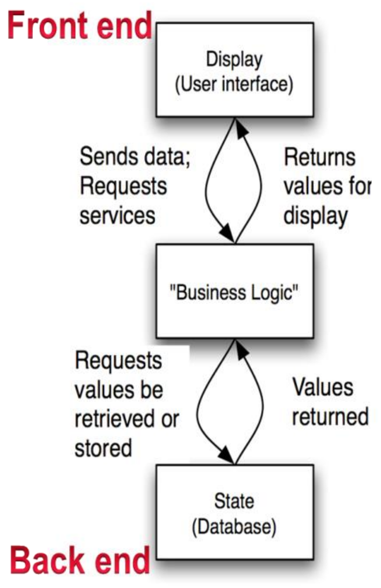
Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc. Reprinted with permission.

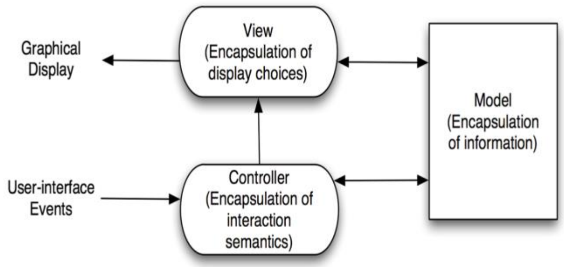
Domain-Specific Software Architecture

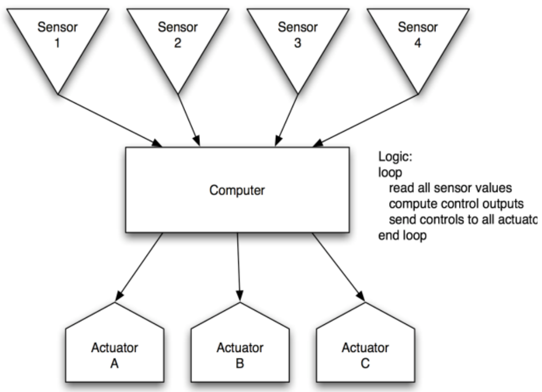
(Hayes-Roth)

- A **domain-specific software architecture** comprises:
  - **a reference architecture**, which describes a general computational framework for a significant domain of applications;
  - **a component library**, which contains reusable chunks of domain expertise; and
  - **an application configuration method** for selecting and configuring components within the architecture to meet particular application requirements.
- Examples:

  ADAGE for avionics, AIS for adaptive intelligent systems, and MetaH for missile guidance, navigation, and control systems


Front end
Display (User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State (Database)
Back end


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)



Components and connectors

- A **software component** is an architectural entity that
  - encapsulates a subset of the system's functionality and/or data
  - restricts access to that subset via an explicitly defined interface
  - has explicitly defined dependencies on its required execution context
- A **software connector** is an architectural building block tasked with effecting and regulating interactions among components