Software Architecture DAT220/DIT544

Truong Ho-Quang

truongh@chalmers.se

Software Engineering Division

Chalmers | GU

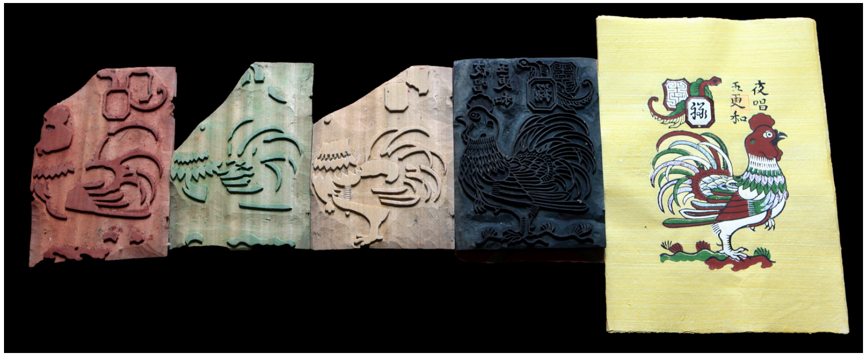
Decomposition and (Design) Patterns

- Đông Hồ painting
  (Đông Hồ folk woodcut painting)

- Materials: Điệp paper
- Colors: Nature-made, 3-4 colors
- Woodblocks: to apply colors

- How?
  - Decompose
  - Make patterns (woodblocks)
  - Apply colors

Decomposition strategy: using colors



Schedule

We are HERE!

Week		Date	Time	Lecture	
3	L1	Wed, 20 Jan	10:15 – 12:00	Introduction & Organization	Ho
3	L2	Thu, 21 Jan	13:15 – 15:00	Architecting Process & Views	Ho
4		Tue, 26 Jan	10:15 – 12:00	**Skip**	
4	S1	Wed, 27 Jan	10:15 – 12:00	<< Supervision: Launch Assignment 1>>	TAs
4	L3	Thu, 28 Jan	13:15 - 15:00	Roles/Responsibilities & Functional Decomposition	Truong Ho
5	L4	Mon, 1 Feb	10:15 – 12:00	Architectural Styles P1	Truong Ho
5	S2	Wed, 3 Jan	10:15 – 12:00	<< Supervision/Assignment>>	TAs
5	L5	Thu, 4 Jan	13:15 – 15:00	Architectural Styles P2	Truong Ho
6	L6	Mon, 8 Feb	10:15 – 12:00	Architectural Styles P3	Sam Jobara
6	S3	Wed, 10 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
6	L7	Thu, 11 Feb	13:15 – 15:00	Design Principles (Maintainability, Modifiability)	Truong Ho
7	L8	Mon, 15 Feb	10:15 – 12:00	Performance – Analysis & Tactics	Truong Ho
7	S4	Wed, 17 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
7	L9	Thu, 18 Feb	10:15 – 12:00	Tactics: Reliability, Availability, Fault Tolerance	TBD
8	L10	Mon, 22 Feb	13:15 – 15:00	Guest Lecture 1	TBD
8	S5	Wed, 24 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
8	L11	Thu, 25 Feb	10:15 – 12:00	Guest Lecture 2	TBD
9	L12	Mon, 1 Mar	13:15 – 15:00	Reverse Engineering & Correspondence	Truong Ho
9	S6	Wed, 3 Mar	10:15 – 12:00	<< Supervision/Assignment>>	TAs
9	L13	Thu, 4 Mar	13:15 – 15:00	To be determined (exam practice?)	Truong Ho
9		Fri, 5 Mar	Whole day	Group presentation of Assignment (TBD)	Teachers
11	Exam				

Voluntary student representatives

- 5 students (Chalmers and GU)
- Randomly generated
- Will be contacted shortly

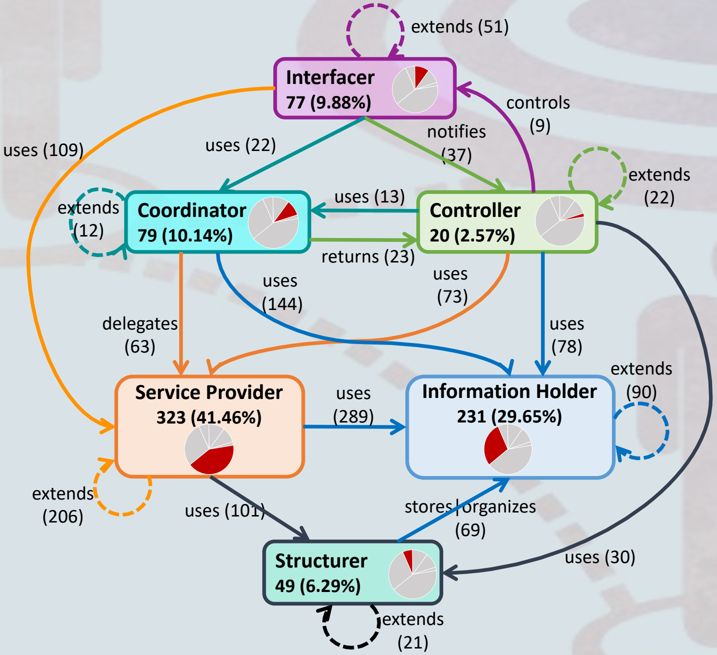
Recap of previous lectures

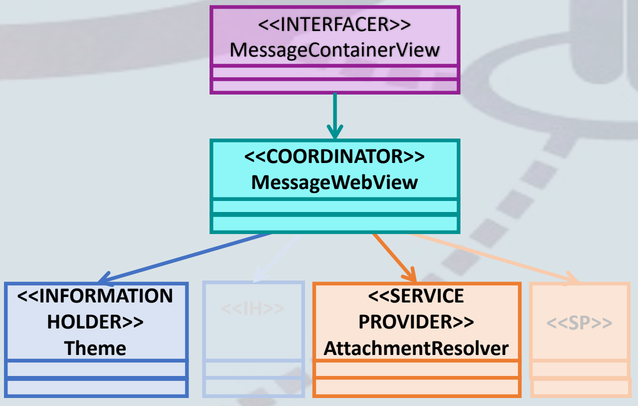
- L1: What, Why, How? SW Architecture
- L2: Architecting process, stakeholders, views

Goals of this lecture

- Understand notion of Role, Responsibilities in Software Architecture
- Understand functional decomposition and common way of doing that (via examples)
- Understand the difference and transition between analysis and design

Part I:
Roles & Responsibilities





Theme/Objective of this part

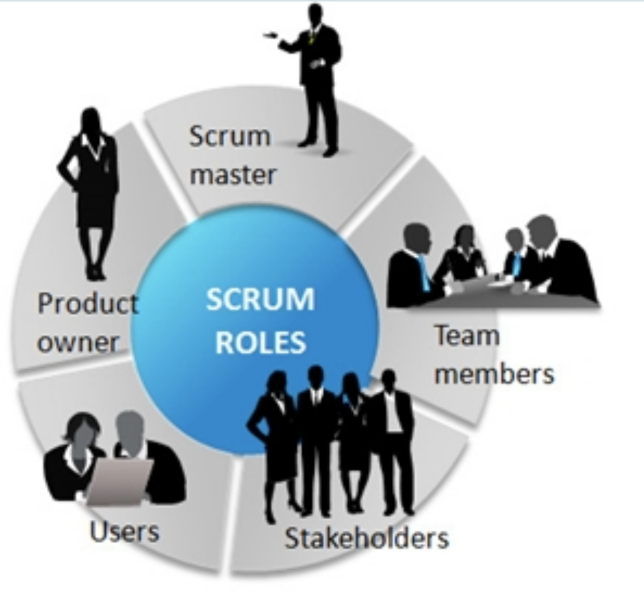
- Understand the importance of being aware of role when designing software.

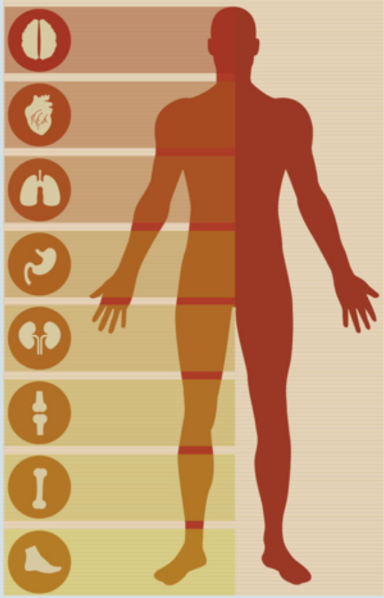
- Build vocabulary for characterizing role/responsibility
    - a set of six(6) common roles (role stereotypes)
    - collaborations between role stereotypes

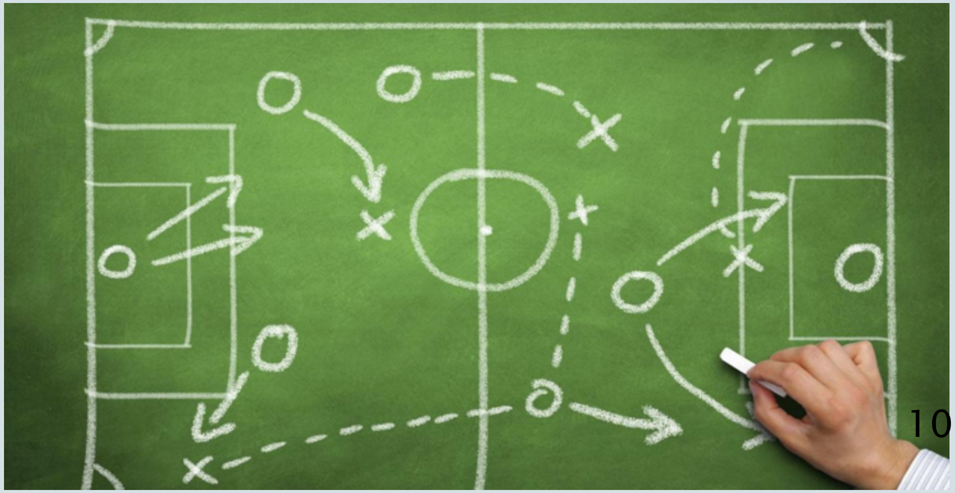
- Exploring impacts of role/stereotype in design quality metrics in two realistic cases

What is role & responsibility?

Where to find role/responsibility?







Why defining role is so important?

- To establish working scope
- To seek agreement
- To facilitate communication/collaboration when performing tasks
- Less waste

Role & Responsibility in Software Design

- Software is a set of components that
  - carry different roles
  - collaborate with different components
- Being aware of component’s role when designing would help to:
  - achieve better distribution of responsibility
  - manage complexity/communication
  - avoid redundancy
  - increase mainteability

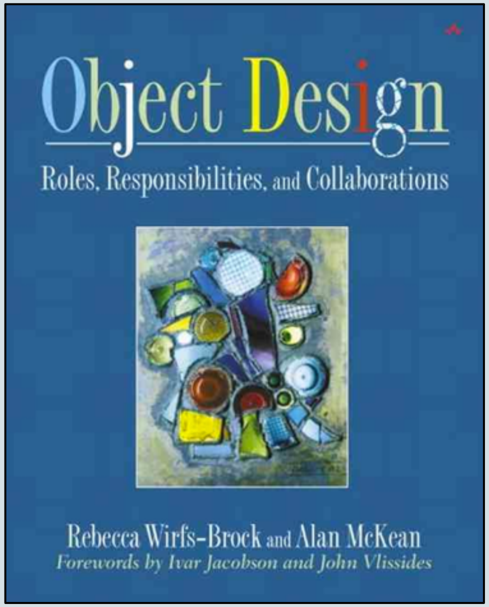
Role Stereotypes

- Definition
- Relationships between role stereotypes

Stereotype
A conventional, formulaic, and oversimplified conception, opinion, or image
(www.thefreedictionary.com)

Role Stereotypes

- The concept “role stereotype” was introduced by Rebecca Wirfs-Brock.
- The concept indicates generic roles that an software object plays in the design.
- It is recommended that each object carries a single role/responsibility.


Object Design
Roles, Responsibilities, and Collaborations
Rebecca Wirfs-Brock and Alan McKean
Forewords by Ivar Jacobson and John Vlissides

Object Design: Roles, Responsibilities and Collaborations, Rebecca Wirfs-Brock and Alan McKean, Addison-Wesley, 2003

- **Service providers** do things
- **Interfacers** translate requests and convert from one level of abstraction to another
- **Information holders** know things
- **Controllers** direct activities
- **Coordinators** delegate work
- **Structurers** manage object relations or organize large numbers of similar objects

Role stereotypes

Service Provider (SP)

- performs specific work
- offers services

’-er’, ’-or’; public static methods; might contains some logics to do specific tasks.

?

Information Holder (IH)

- knows/keeps information
- provides information

data encapsulation; get/set methods; private/internal methods

?

Interfacer (IF)

- transforms/converts information and requests btw SW layers

GUI-related; storefront; API; extension points

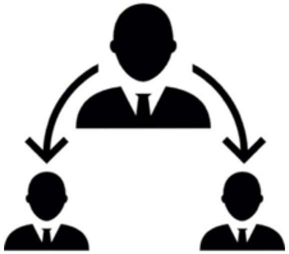
?

Controller (CT)

- makes decision
- control complex tasks

’controller’, ’manager’; logic statements; knows IH, SP, CO

?

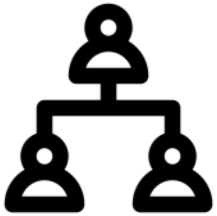


Coordinator (CO)

- delegates works
- forwards info/requests

no/simple logic; knows requester & requestee

?



Structurer (ST)

- keeps/maintains relationship
- pool, collects & arranges objs

Collection; sort(); compare(); validate(); add(); remove(); …

?

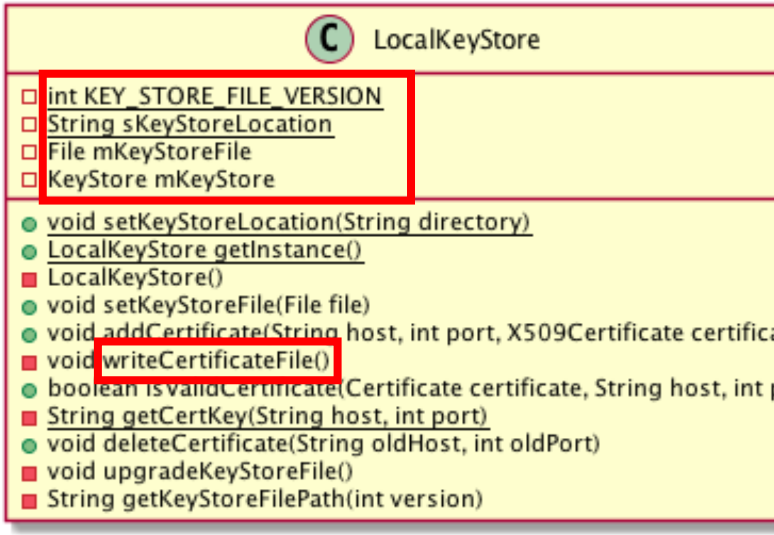
Information Holder (IH)

is a software element that

- keeps/knows information
- provides information to other elements

Example: An IH *class* might be characterized by:

- The class may just contains attributes
- Methods, if any, could be
  - Getters and setter
  - Persistence methods, eg. saving to database or implements Java's Serializable interface
  - Methods that are only used within the class



```
LocalKeyStore
□ int KEY_STORE_FILE_VERSION
□ String sKeyStoreLocation
□ File mKeyStoreFile
□ KeyStore mKeyStore
● void setKeyStoreLocation(String directory)
● LocalKeyStore getInstance()
■ LocalKeyStore()
● void setKeyStoreFile(File file)
● void addCertificate(String host, int port, X509Certificate certifica
■ void writeCertificateFile()
● boolean isValidCertificate(Certificate certificate, String host, int p
■ String getCertKey(String host, int port)
● void deleteCertificate(String oldHost, int oldPort)
■ void upgradeKeyStoreFile()
■ String getKeyStoreFilePath(int version)
```

Service Provider (SP)

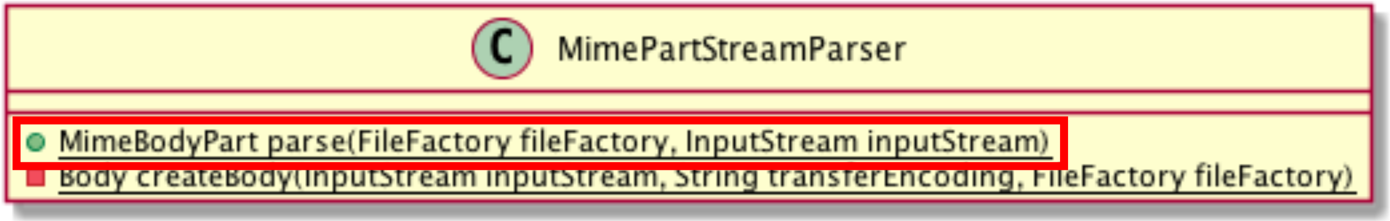
is a software element that

- performs specific works
- offers services to other elements on demand

A SP class can be characterized by:

- having name ended with “-er” (eg. *Provider*) or “-or” (eg. *Creator*, *Detector*)
- has methods and attributes are easily accessed by other classes (often static and public, or protected, not private)
- could be realization of a Interface
- decision making in methods should be at basic level, only to support specific work

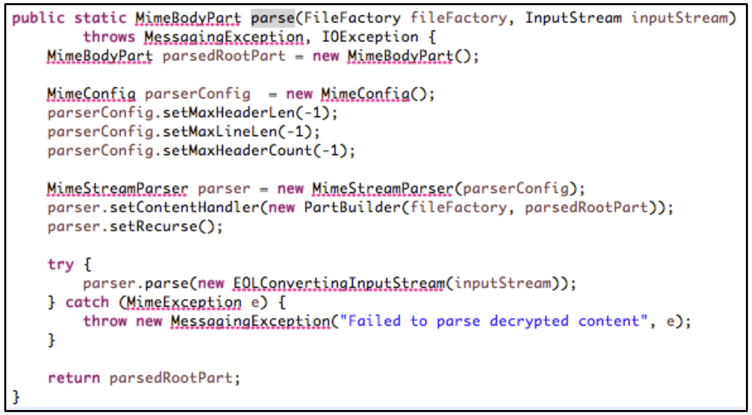
Service Provider Class - Example



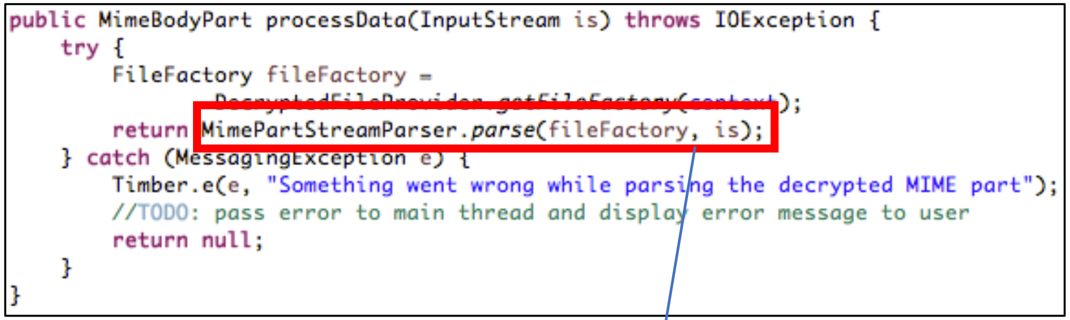
C MimePartStreamParser

● MimeBodyPart parse(FileFactory fileFactory, InputStream inputStream)

■ Body createBody(InputStream inputStream, String transferEncoding, FileFactory fileFactory)



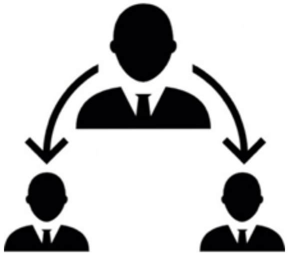
```
public static MimeBodyPart parse(FileFactory fileFactory, InputStream inputStream)
        throws MessagingException, IOException {
    MimeBodyPart parsedRootPart = new MimeBodyPart();

    MimeConfig parserConfig  = new MimeConfig();
    parserConfig.setMaxHeaderLen(-1);
    parserConfig.setMaxLineLen(-1);
    parserConfig.setMaxHeaderCount(-1);

    MimeStreamParser parser = new MimeStreamParser(parserConfig);
    parser.setContentHandler(new PartBuilder(fileFactory, parsedRootPart));
    parser.setRecurse();

    try {
        parser.parse(new EOLConvertingInputStream(inputStream));
    } catch (MimeException e) {
        throw new MessagingException("Failed to parse decrypted content", e);
    }

    return parsedRootPart;
}
```



```
public MimeBodyPart processData(InputStream is) throws IOException {
    try {
        FileFactory fileFactory =
                DecryptedFileProvider.getFileFactory(context));
        return MimePartStreamParser.parse(fileFactory, is);
    } catch (MessagingException e) {
        Timber.e(e, "Something went wrong while parsing the decrypted MIME part");
        //TODO: pass error to main thread and display error message to user
        return null;
    }
}
```

calls service

Coordinator (CO)



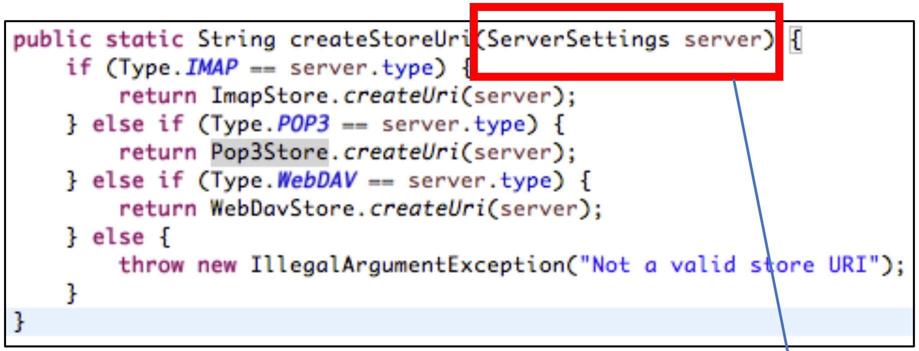
is a software element that

- does not make decisions
- delegates work to other objects
- forwards info/requests

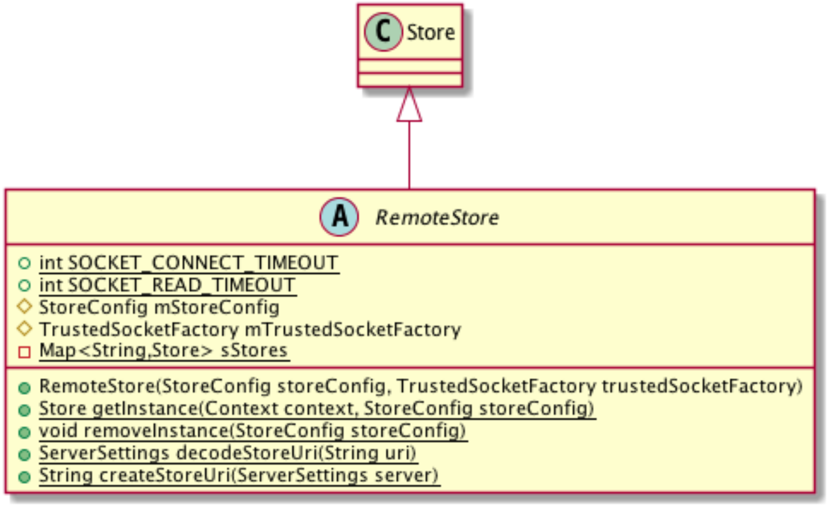
Signs of a CO class:

- Holding connection between working objects (SP, CT)
- Forwarding information and requests
  - it is important to define which classes are **requester** and **requestee**
  - information: method parameters; variables …
- When a Service Provider becomes too big, it evolves into Coordinator
  - Results of refactoring god classes

Coordinator Class - Example



```
public static String createStoreUri(ServerSettings server) {
    if (Type.IMAP == server.type) {
        return ImapStore.createUri(server);
    } else if (Type.POP3 == server.type) {
        return Pop3Store.createUri(server);
    } else if (Type.WebDAV == server.type) {
        return WebDavStore.createUri(server);
    } else {
        throw new IllegalArgumentException("Not a valid store URI");
    }
}
```



C Store

A RemoteStore

- ○ int SOCKET_CONNECT_TIMEOUT
- ○ int SOCKET_READ_TIMEOUT
- ◇ StoreConfig mStoreConfig
- ◇ TrustedSocketFactory mTrustedSocketFactory
- □ Map<String,Store> sStores

- ● RemoteStore(StoreConfig storeConfig, TrustedSocketFactory trustedSocketFactory)
- ● Store getInstance(Context context, StoreConfig storeConfig)
- ● void removeInstance(StoreConfig storeConfig)
- ● ServerSettings decodeStoreUri(String uri)
- ● String createStoreUri(ServerSettings server)

coordinates the works to ImapStore, Pop3Store, WebDavStore

Controller (CT)

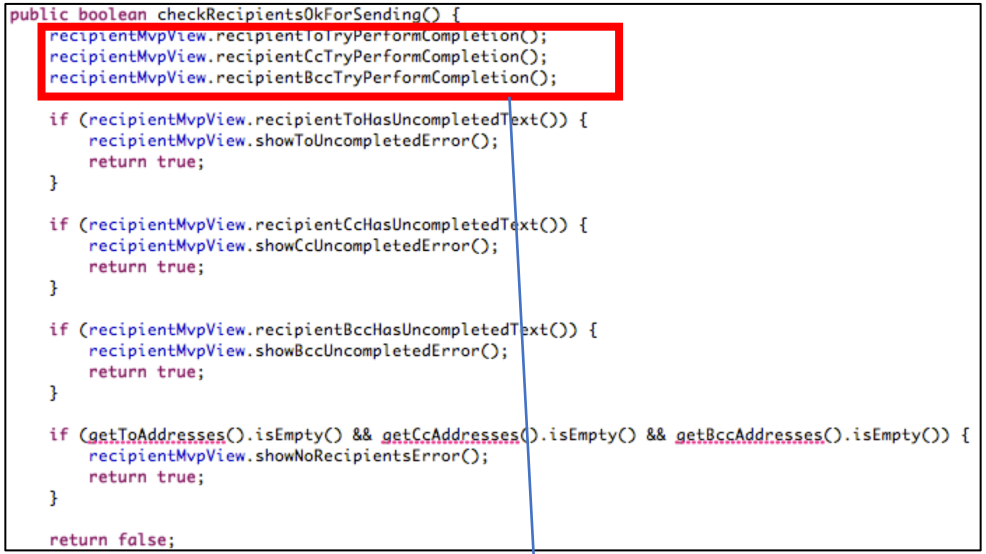
is a software element that

- make decisions
- control complex tasks

A CT class might be characterized by:

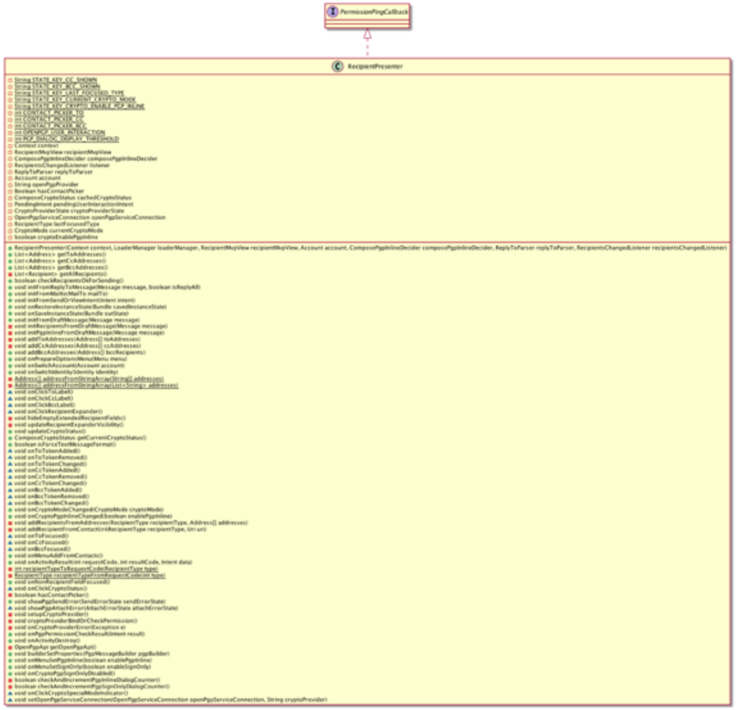
- having class name ended with “Controller”, “Manager”
- Should have access to information holders, coordinators, or service provider
- Its main responsibility is to make decision to control the flow of the application
  - Should contain condition statements (e.g. IF, IF ELSE, SWITCH CASE, x : ?)
  - The decision should be at the higher level than decision made at SP/CO.

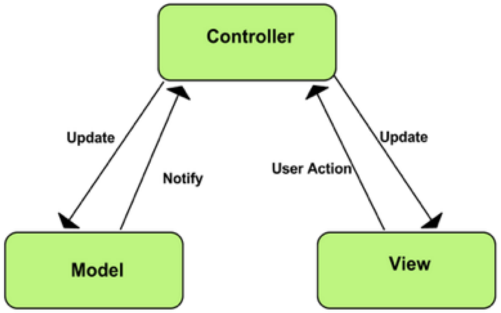
Controller Class - Example



```java
public boolean checkRecipientsOkForSending() {
    recipientMvpView.recipientToTryPerformCompletion();
    recipientMvpView.recipientCcTryPerformCompletion();
    recipientMvpView.recipientBccTryPerformCompletion();

    if (recipientMvpView.recipientToHasUncompletedText()) {
        recipientMvpView.showToUncompletedError();
        return true;
    }

    if (recipientMvpView.recipientCcHasUncompletedText()) {
        recipientMvpView.showCcUncompletedError();
        return true;
    }

    if (recipientMvpView.recipientBccHasUncompletedText()) {
        recipientMvpView.showBccUncompletedError();
        return true;
    }

    if (getToAddresses().isEmpty() && getCcAddresses().isEmpty() && getBccAddresses().isEmpty()) {
        recipientMvpView.showNoRecipientsError();
        return true;
    }

    return false;
```

Delegating the work


PermissionPingCallback
RecipientPresenter
String STATE_KEY_CC_SHOWN
String STATE_KEY_BCC_SHOWN
String STATE_KEY_LAST_FOCUSED_TYPE
String STATE_KEY_CURRENT_CRYPTO_MODE
String STATE_KEY_CRYPTO_ENABLE_PGP_INLINE
int CONTACT_PICKER_TO
int CONTACT_PICKER_CC
int CONTACT_PICKER_BCC
int OPENPGP_USER_INTERACTION
int PGP_DIALOG_DISPLAY_THRESHOLD
Context context
RecipientMvpView recipientMvpView
ComposePgpInlineDecider composePgpInlineDecider
RecipientsChangedListener listener
ReplyToParser replyToParser
Account account
String openPgpProvider
Boolean hasContactPicker
ComposeCryptoStatus cachedCryptoStatus
PendingIntent pendingUserInteractionIntent
CryptoProviderState cryptoProviderState
OpenPgpServiceConnection openPgpServiceConnection
RecipientType lastFocusedType
CryptoMode currentCryptoMode
boolean cryptoEnablePgpInline
RecipientPresenter(Context context, LoaderManager loaderManager, RecipientMvpView recipientMvpView, Account account, ComposePgpInlineDecider composePgpInlineDecider, ReplyToParser replyToParser, RecipientsChangedListener recipientsChangedListener)
List<Address> getToAddresses()
List<Address> getCcAddresses()
List<Address> getBccAddresses()
List<Recipient> getAllRecipients()
boolean checkRecipientsOkForSending()
void initFromReplyToMessage(Message message, boolean isReplyAll)
void initFromMailto(MailTo mailTo)
void initFromSendOrViewIntent(Intent intent)
void onRestoreInstanceState(Bundle savedInstanceState)
void onSaveInstanceState(Bundle outState)
void initFromDraftMessage(Message message)



Structurer (ST)

is a software element that

- maintains relationships between software components
- pools/collects/arranges a set of elements

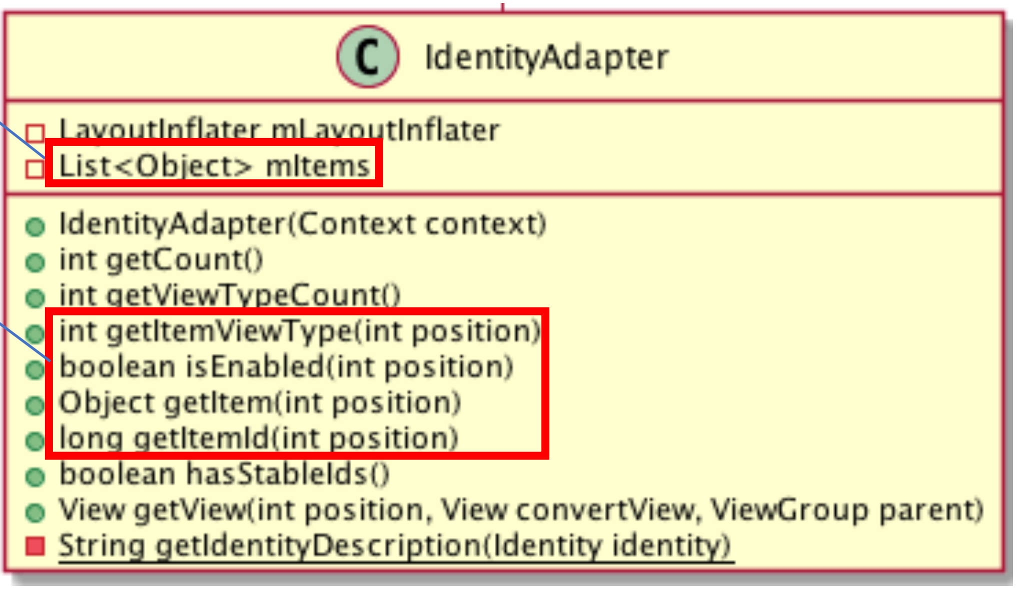
A ST class might be characterized by:

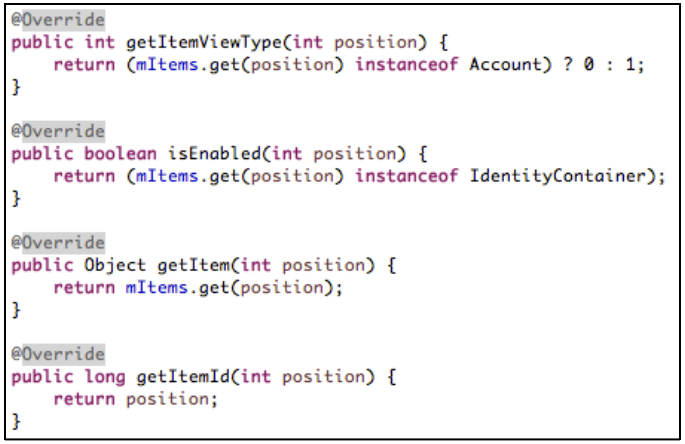
- extends Java’s Collections framework
- contains a collection of objects (of other classes)
- has methods that maintaining relationships between objects in the collection
    - methods that manipulate the collection such as sort(), compare(), validate(), remove(), updates(), add(), delete() …
    - methods that give access to the objects such as get(index), next(), hasNext() ...

Structurer - Example

holds a collection

accessing to the collection


C IdentityAdapter
□ LayoutInflater mLayoutInflater
□ List<Object> mItems
● IdentityAdapter(Context context)
● int getCount()
● int getViewTypeCount()
● int getItemViewType(int position)
● boolean isEnabled(int position)
● Object getItem(int position)
● long getItemId(int position)
● boolean hasStableIds()
● View getView(int position, View convertView, ViewGroup parent)
■ String getIdentityDescription(Identity identity)



```
@Override
public int getItemViewType(int position) {
    return (mItems.get(position) instanceof Account) ? 0 : 1;
}

@Override
public boolean isEnabled(int position) {
    return (mItems.get(position) instanceof IdentityContainer);
}

@Override
public Object getItem(int position) {
    return mItems.get(position);
}

@Override
public long getItemId(int position) {
    return position;
}
```

Interfacer

is a software element that

- transforms information or requests between distinct parts of the system
  - User interfacer interacts with the users of the system, e.g. GUI components
  - Internal interfacer exists between sub parts of the system, e.g. Data Management Tier
  - External interfacer communicates with external systems, e.g. API, extension points of the system

A ST class can be characterized by:

- Contains Java Swing, AWT, and other UI components
- Manage user interface and handle user interaction
  - In Android apps, this extends Activity classes
- Encapsulates functions or objects in the system by providing an Interface or an abstract class that can be used outside of the system
- If an interface is created but never implemented: may be this serves as an extension point for the system

Role stereotypes

Service Provider (SP)

- performs specific work
- offers services

’-er’, ’-or’; public static methods; might contains some logics to do specific tasks.

Information Holder (IH)

- knows/keeps information
- provides information

data encapsulation; get/set methods; private/internal methods

Interfacer (IF)

- transforms/converts information and requests btw SW layers

GUI-related; storefront; API; extension points

Controller (CT)

- makes decision
- control complex tasks

’controller’, ’manager’; logic statements; knows IH, SP, CO

Coordinator (CO)

- delegates works
- forwards info/requests

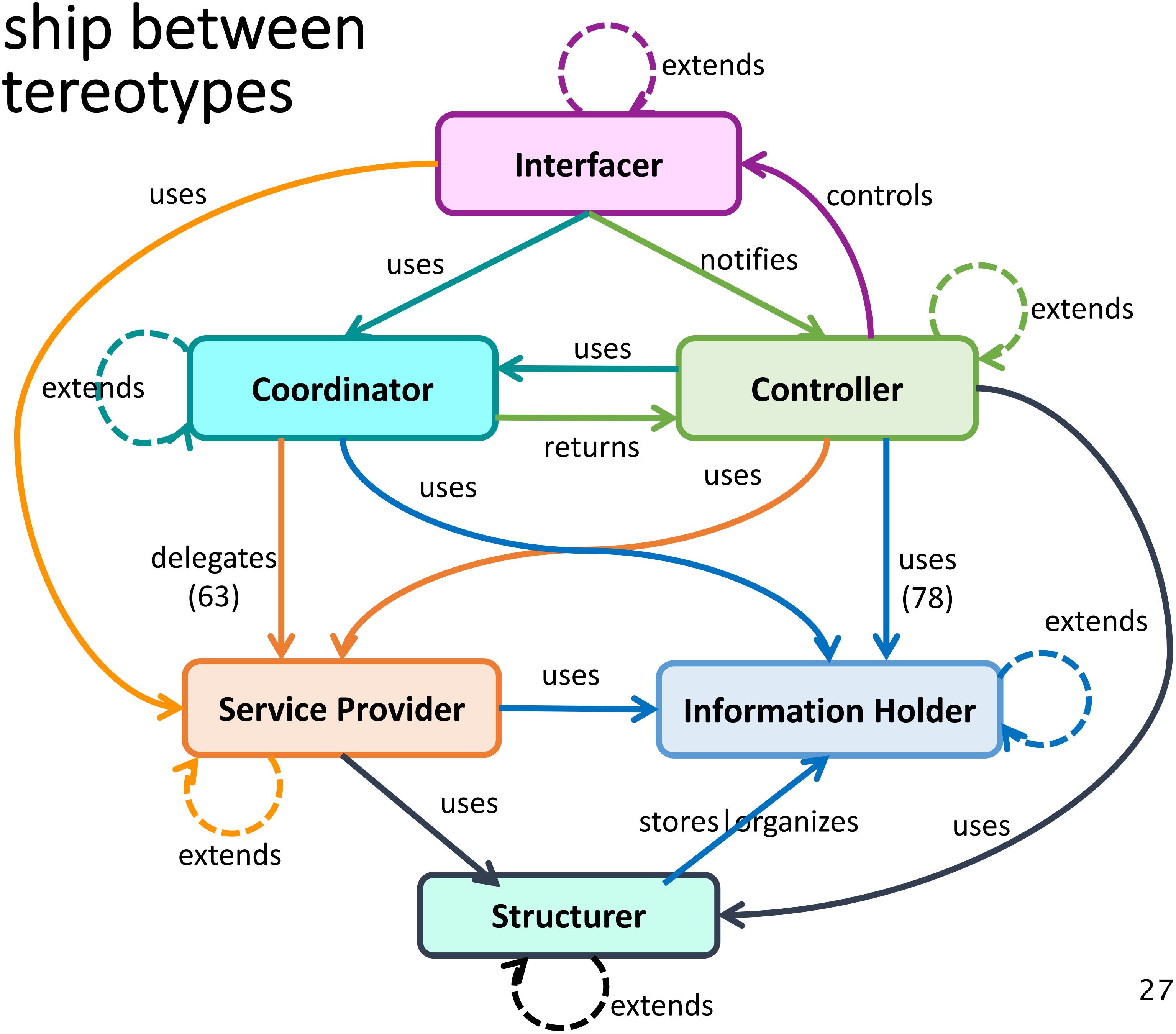
no/simple logic; knows requester & requestee

Structurer (ST)

- keeps/maintains relationship
- pool, collects, arranges objs

Collection; sort(); compare(); validate(); add(); remove(); …

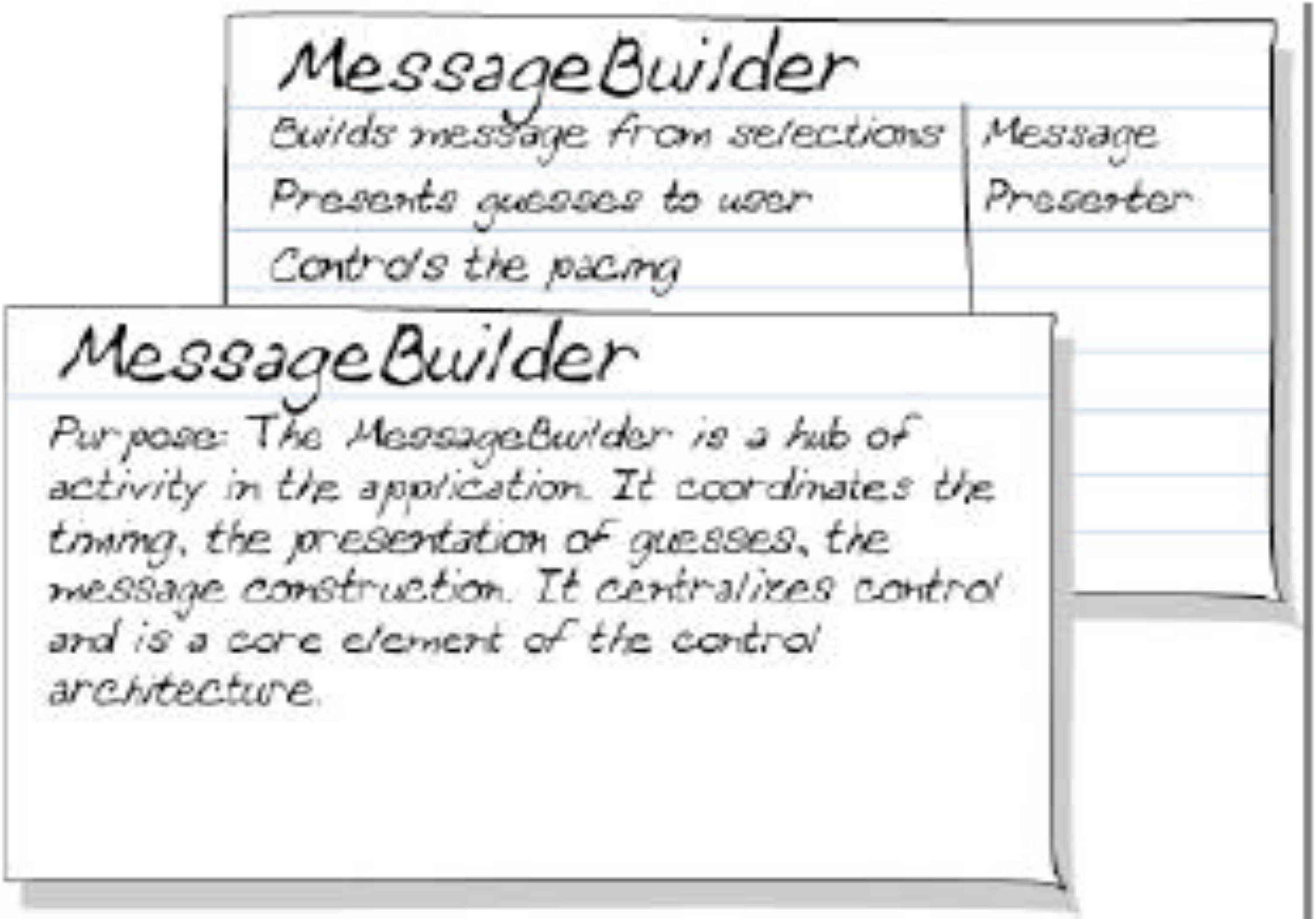
Relationship between role stereotypes



Analysing Responsibility and Collaborations of Objects using CRC Card

CRC Cards

Candidate, Responsibilities, Collaborators



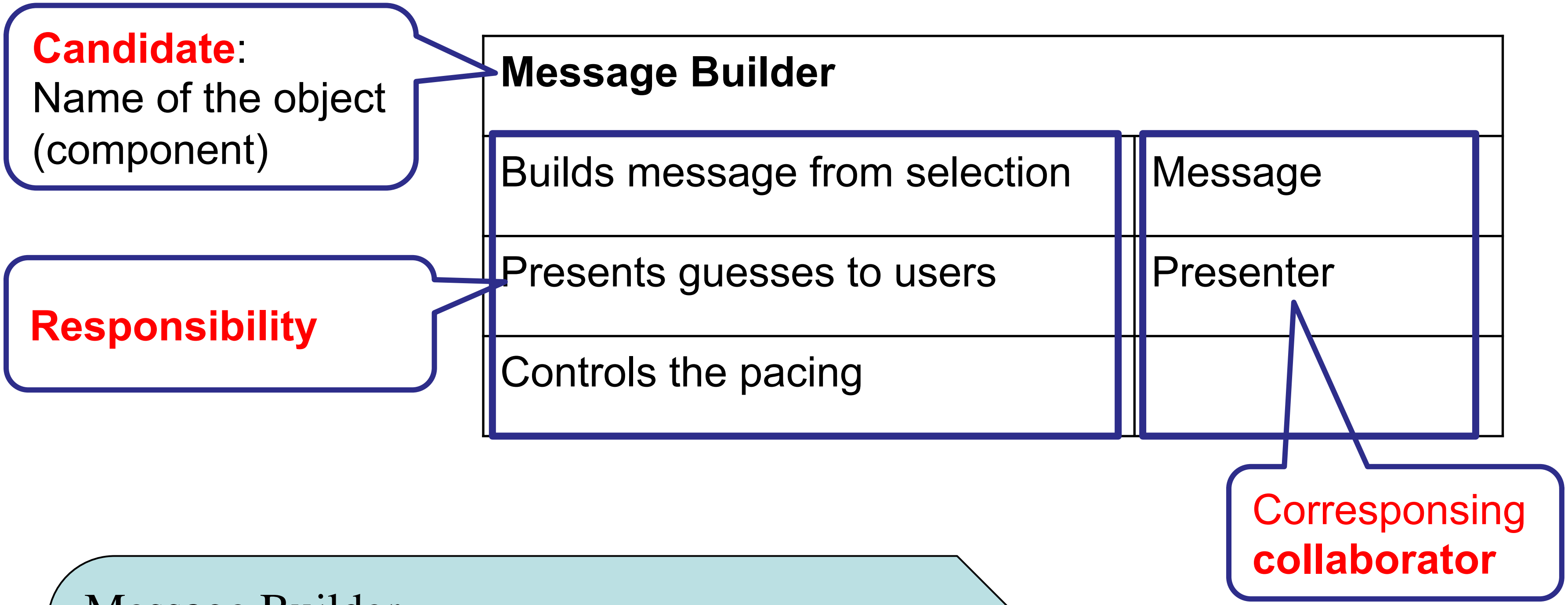
MessageBuilder

Builds message from selections	Message
Presents guesses to user	Presenter
Controls the pacing	

MessageBuilder

Purpose: The MessageBuilder is a hub of activity in the application. It coordinates the timing, the presentation of guesses, the message construction. It centralizes control and is a core element of the control architecture.

CRC Card



Candidate: Name of the object (component)

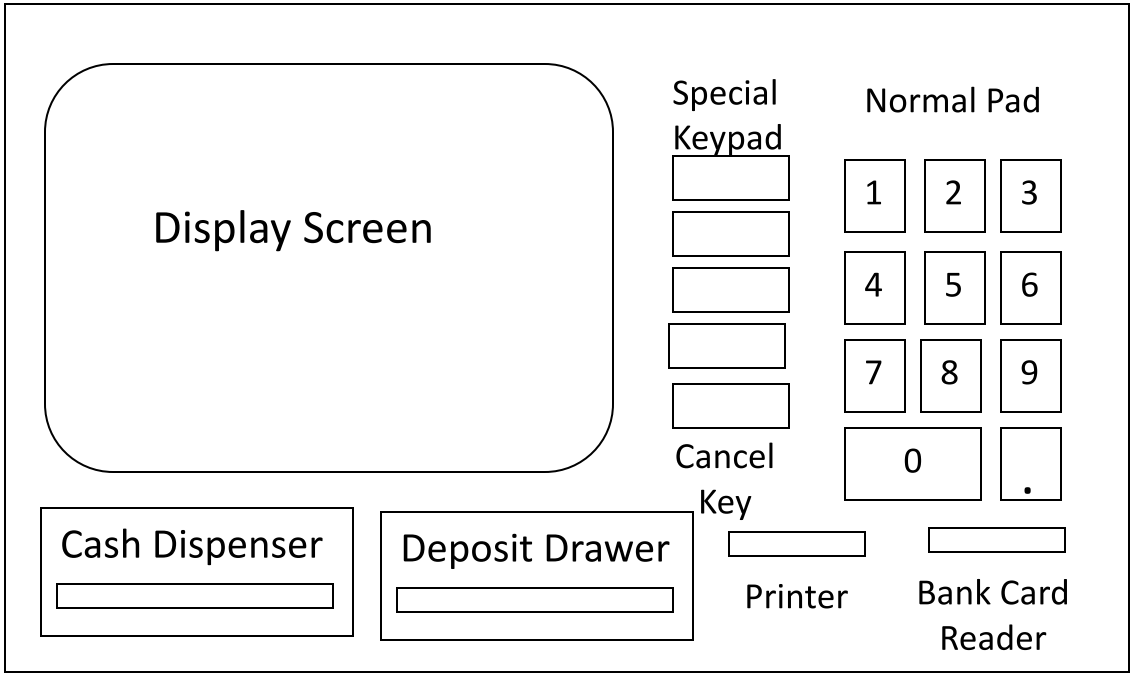
Responsibility

Corresponsing **collaborator**

Message Builder	
Builds message from selection	Message
Presents guesses to users	Presenter
Controls the pacing	

Message Builder
Purpose: The Message Builder is a hub of activity in the application. It coordinates the timing, the presentation of guesses, the message construction. It centralizes control and is a core element of the control architecture

Example: ATM system



Example: ATM system

An automated teller machine (ATM) is a machine through which bank customers can perform a number of financial transactions. The machine consists of a display screen, a bank card reader, input keys, a money dispenser slot, a deposit slot and a receipt printer. The main menu contains a list of the transactions that can be performed. These transactions include:

- deposit funds to an account
- withdraw funds from an account
- transfer funds from one account to the other
- query the balance of an account.

ATM class

The ATM class represents the teller machine. Its main operations are to create and initiate transactions. This class acts the following roles:

- a **Controller** role to both the Financial Subsystem and the User Interface Subsystem.

ATM Class	
Initiate Transaction	User Interface
Execute Transaction	User Interface

Financial Subsystem

- The `Financial Subsystem` implements the financial aspects of a customer's interaction with the `ATM`. Its main operations are to execute the following financial transactions; `deposit(),withdraw(),transfer()`, and `balance()` on customer accounts. There is one `Financial Subsystem` contract that must execute all the transactions. This subsystem acts as a **Service Provider** which provides banking services for `ATM Class.`

Financial Subsystem	
Deposit	ATM Class
Withdraw	ATM Class
Transfer	ATM Class
Balance	ATM Class

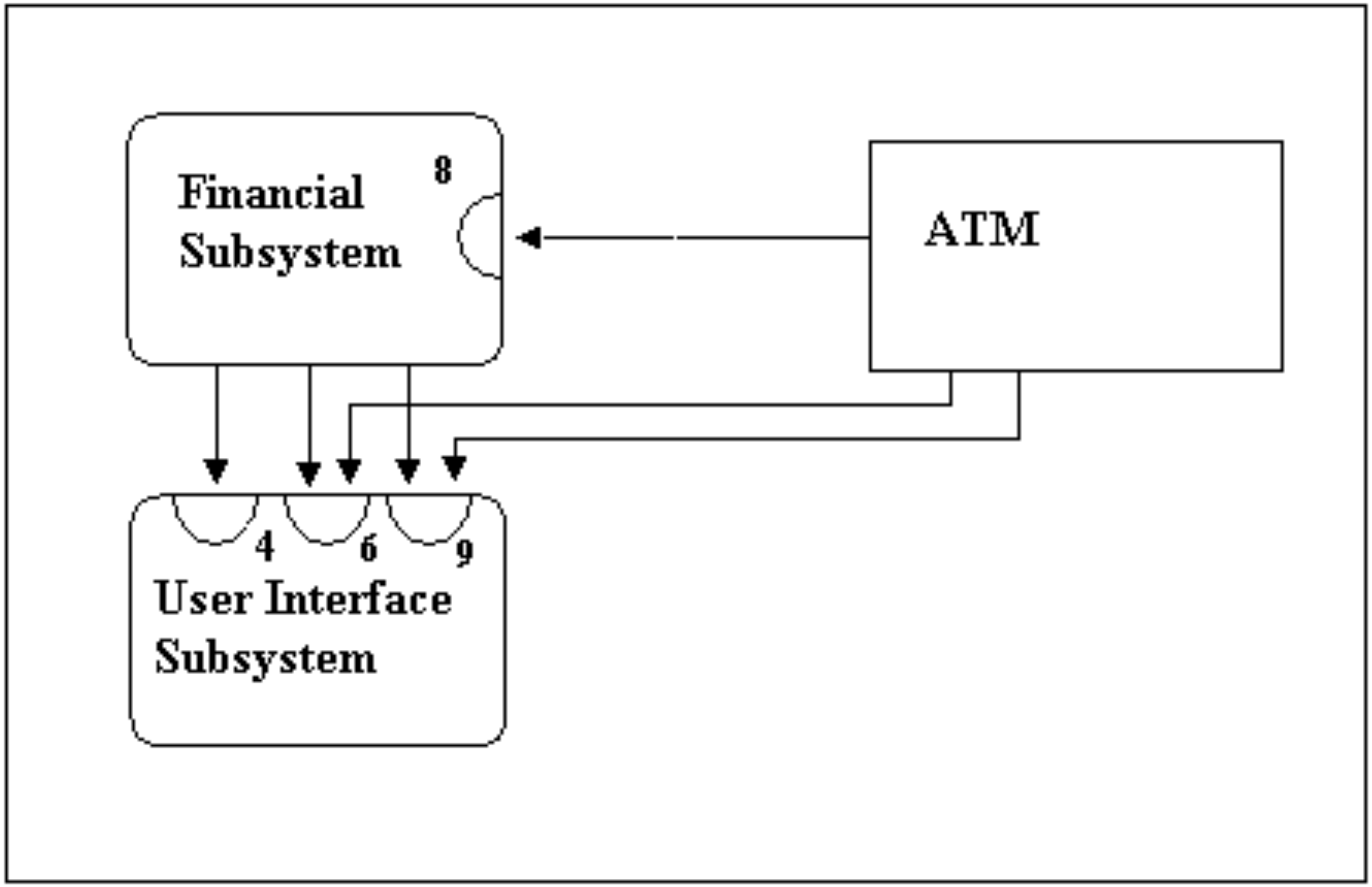
User Interface Subsystem

The `User Interface Subsystem` implements the interface between the ATM and the bank customer. The `User Interface Subsystem` has three responsibilities 1)To get numeric values from users. 2) Get users selection from menu. 3) To display messages and wait for events.

This subsystem acts as an **Interfacer** role to receive and transform requests from users to the system.

User Interface Subsystem	
Get numeric values	ATM Class, Financial Subsystem
Get users selection	ATM Class, Financial Subsystem
Display messages	ATM Class, Financial Subsystem

ATM collaboration graph



Does using role stereotype help in improving design quality?

Boeing Brewery Case (1)

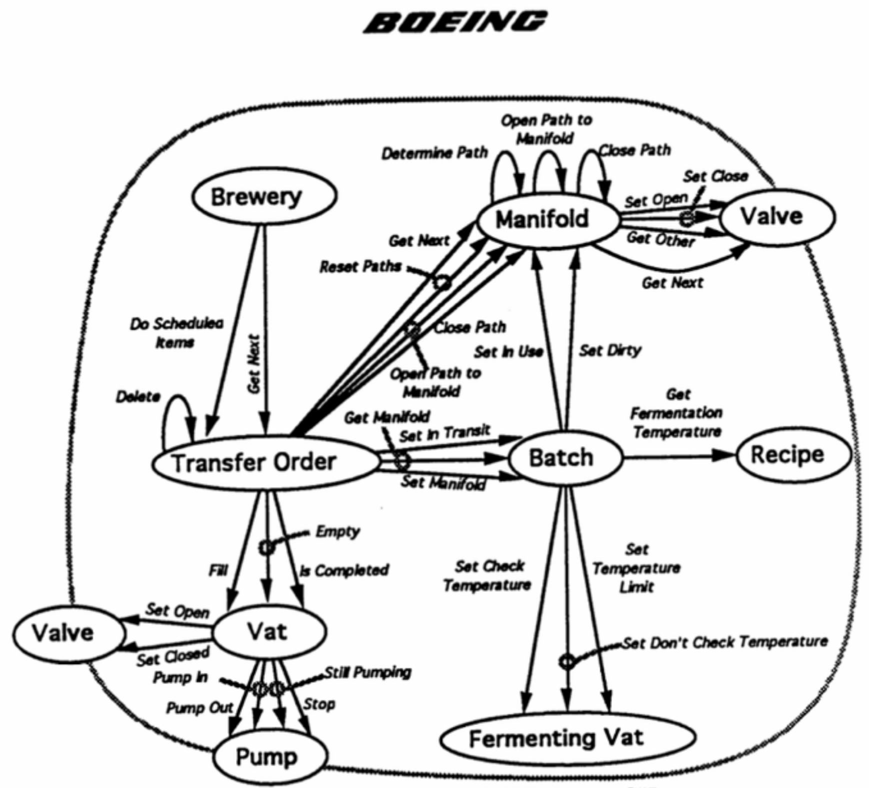


FIGURE 7.3-10 TRANSFER A BATCH

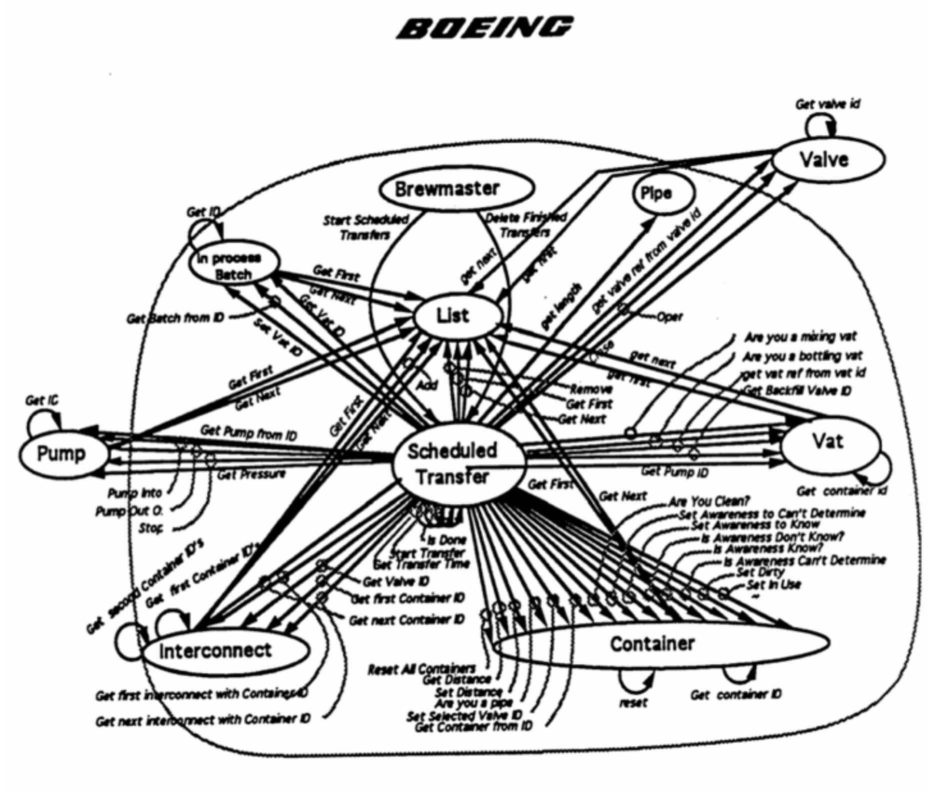


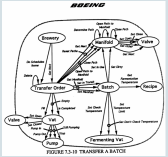
FIGURE 6.3-10 TRANSFER A BATCH

System 1: Responsibility-Focus

System 2: Data-Focus

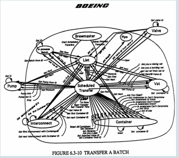
Case description: R. Sharble and S. Cohen “The Object-Oriented Brewery: A Comparison of Two ObjectOriented Development Methods” Boeing Technical Report no. BC2-G4059, October, 1992.

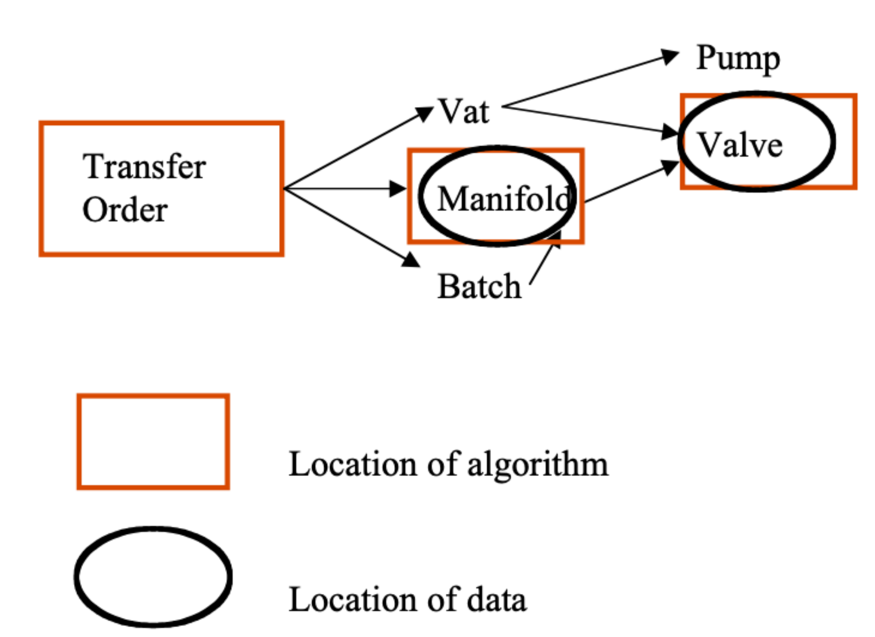
Boeing Brewery Case (2)

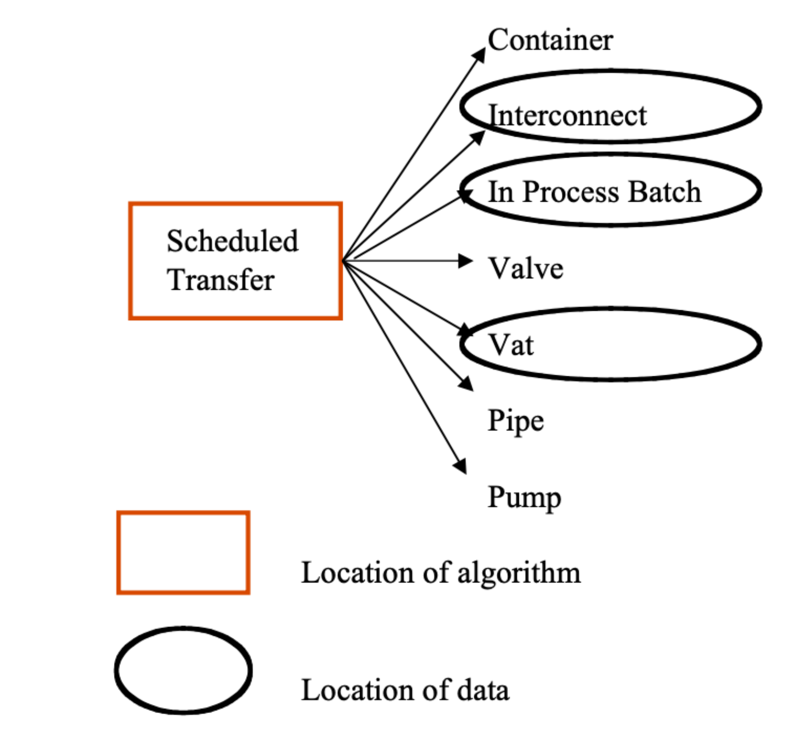


System 1: Responsibility-Focus

System 2: Data-Focus



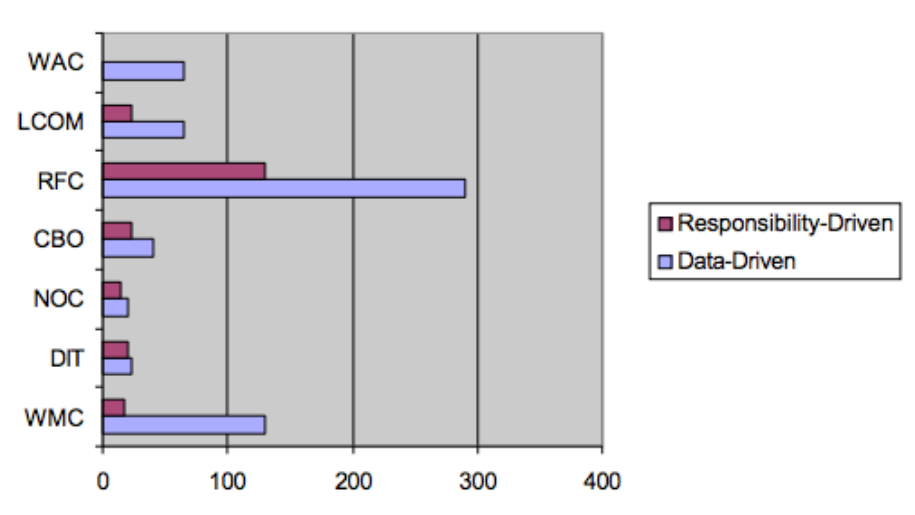




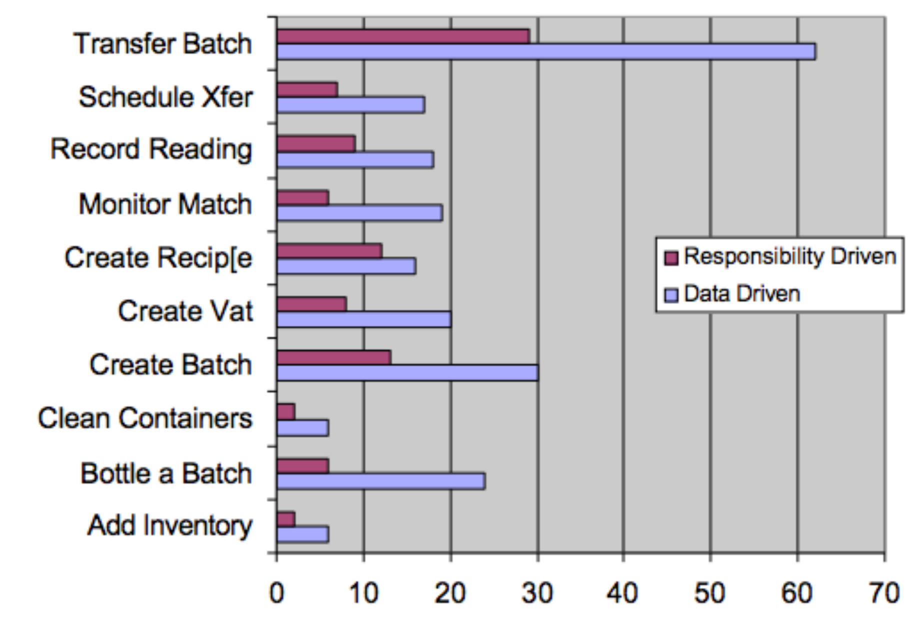
In the figure objects that were used by another have an arrow pointing at them. Objects enclosed in a rectangle performed work, objects whose state was set or queried are encircled.

Comparison report: Report on Object Analysis and Design, Vol. 1, No. 4 by Rebecca Wirfs-Brock

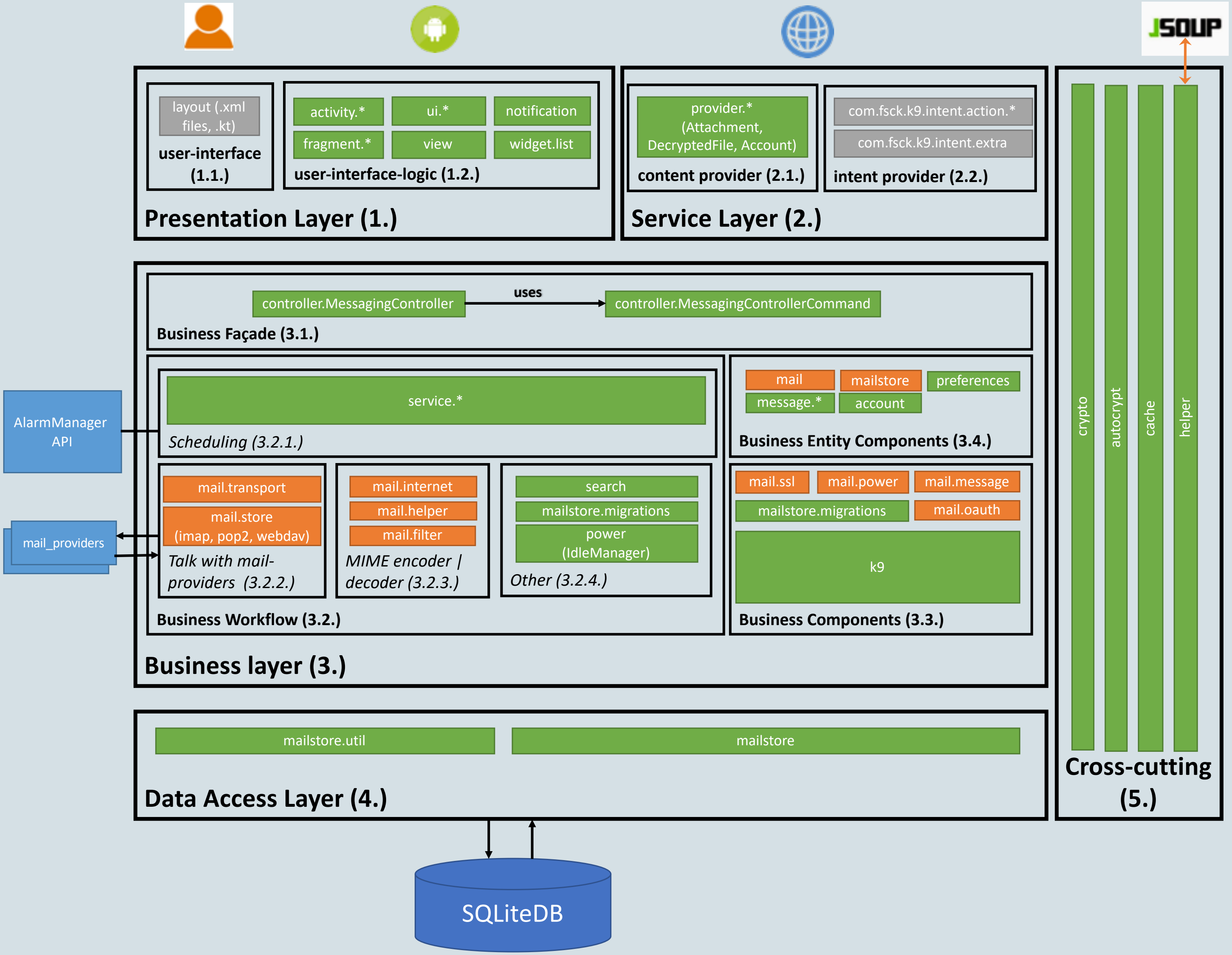
Boeing Brewery (3) – Design Quality Facts



C-K metrics
Weighted Methods per Class (**WMC**)
Depth of Inheritance (**DIT**)
Number of Children (**NOC**)
Coupling between Objects (**CBO**)
Response For a Class (**RFC**)
Lack of Cohesion in Methods (**LCOM**)



K9-Mail Case (1)



JSOUP

layout (.xml files, .kt)
user-interface (1.1.)

activity.*
ui.*
notification
fragment.*
view
widget.list
user-interface-logic (1.2.)

Presentation Layer (1.)

provider.* (Attachment, DecryptedFile, Account)
content provider (2.1.)

com.fsck.k9.intent.action.*
com.fsck.k9.intent.extra
intent provider (2.2.)

Service Layer (2.)

controller.MessagingController
uses
controller.MessagingControllerCommand
Business Façade (3.1.)

AlarmManager API

service.*
Scheduling (3.2.1.)

mail_providers

mail.transport
mail.store (imap, pop2, webdav)
Talk with mail-providers (3.2.2.)

mail.internet
mail.helper
mail.filter
MIME encoder | decoder (3.2.3.)

search
mailstore.migrations
power (IdleManager)
Other (3.2.4.)

Business Workflow (3.2.)

mail
mailstore
preferences
message.*
account
Business Entity Components (3.4.)

mail.ssl
mail.power
mail.message
mailstore.migrations
mail.oauth
k9
Business Components (3.3.)

Business layer (3.)

mailstore.util
mailstore

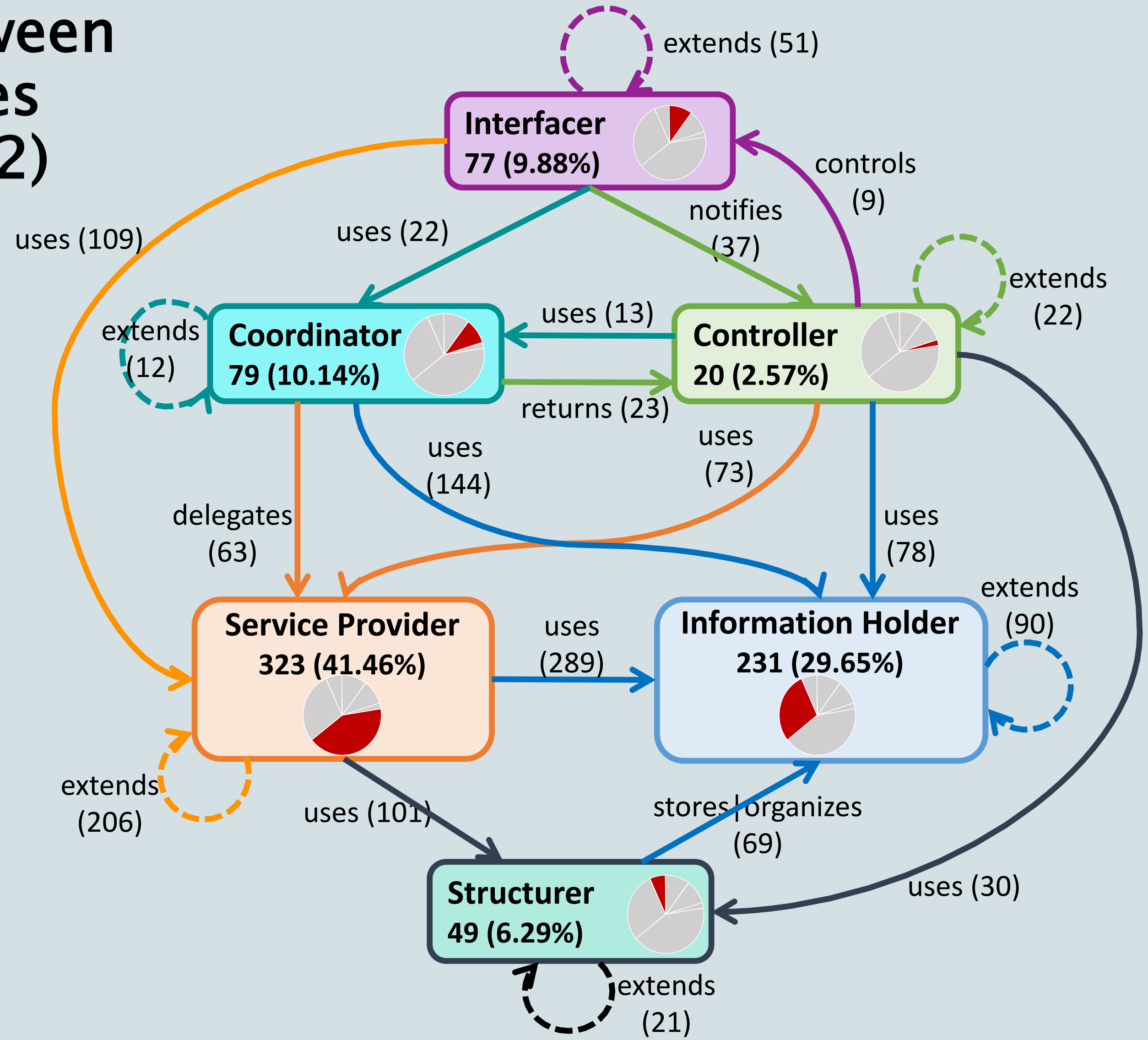
Data Access Layer (4.)

SQLiteDB

crypto
autocrypt
cache
helper

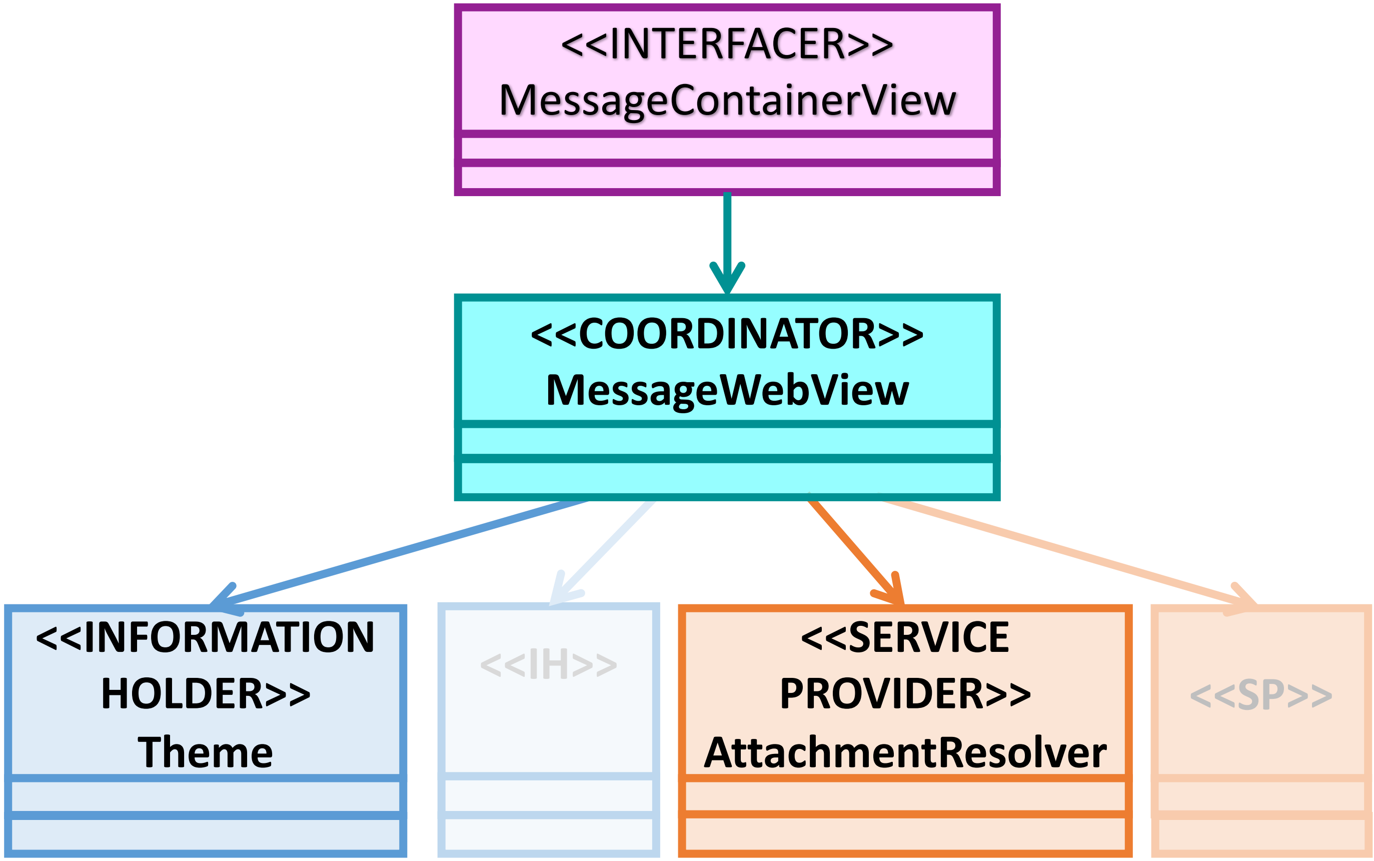
Cross-cutting (5.)

Relationship between role stereotypes K9-Mail Case (2)



Nurwidyantoro, A., Ho-Quang, T. and Chaudron, M.R., 2019. Automated classification of class role-stereotypes via machine learning. In *Proceedings of the Evaluation and Assessment on Software Engineering* (pp. 79-88). https://dl.acm.org/doi/abs/10.1145/3319008.3319016

K9-Mail Case (3)
Collaboration Patterns between Role Stereotypes



<<INTERFACER>>
MessageContainerView

<<COORDINATOR>>
MessageWebView

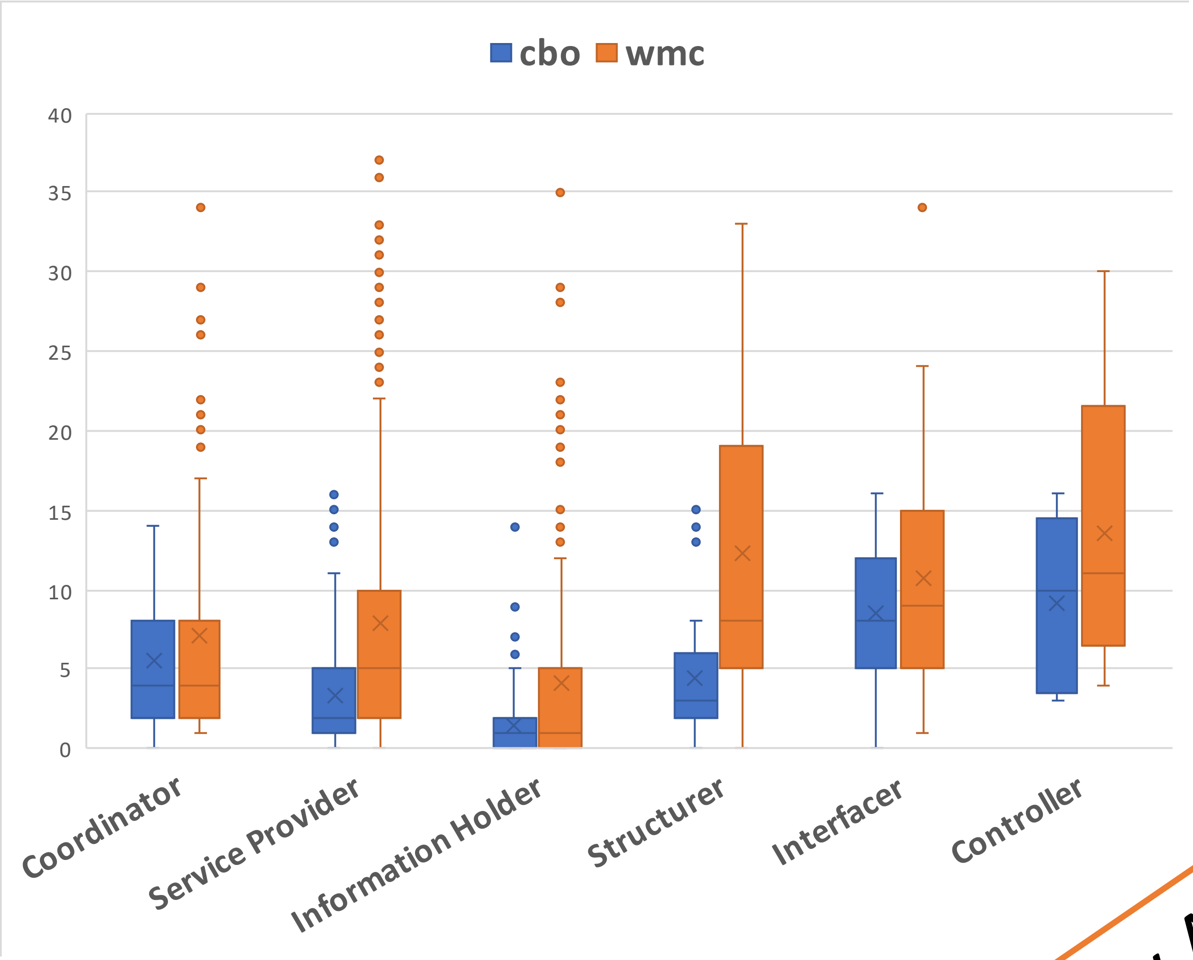
<<INFORMATION HOLDER>>
Theme

<<IH>>

<<SERVICE PROVIDER>>
AttachmentResolver

<<SP>>

K9-Mail Case (4)
Design Metrics of Role Stereotypes



Quality Assurance

Summary Part I

- Having a concrete view on role/responsibility is vital to software design/quality.
- Role stereotypes can be used as a tool for:
  - assigning roles to software elements (in design phase)
  - comprehending work breakdown and collaboration patterns in existing system
- Using CRC card when discussing/thinking of responsibilities and collaborations of an object (can be a component/subsystem/class)

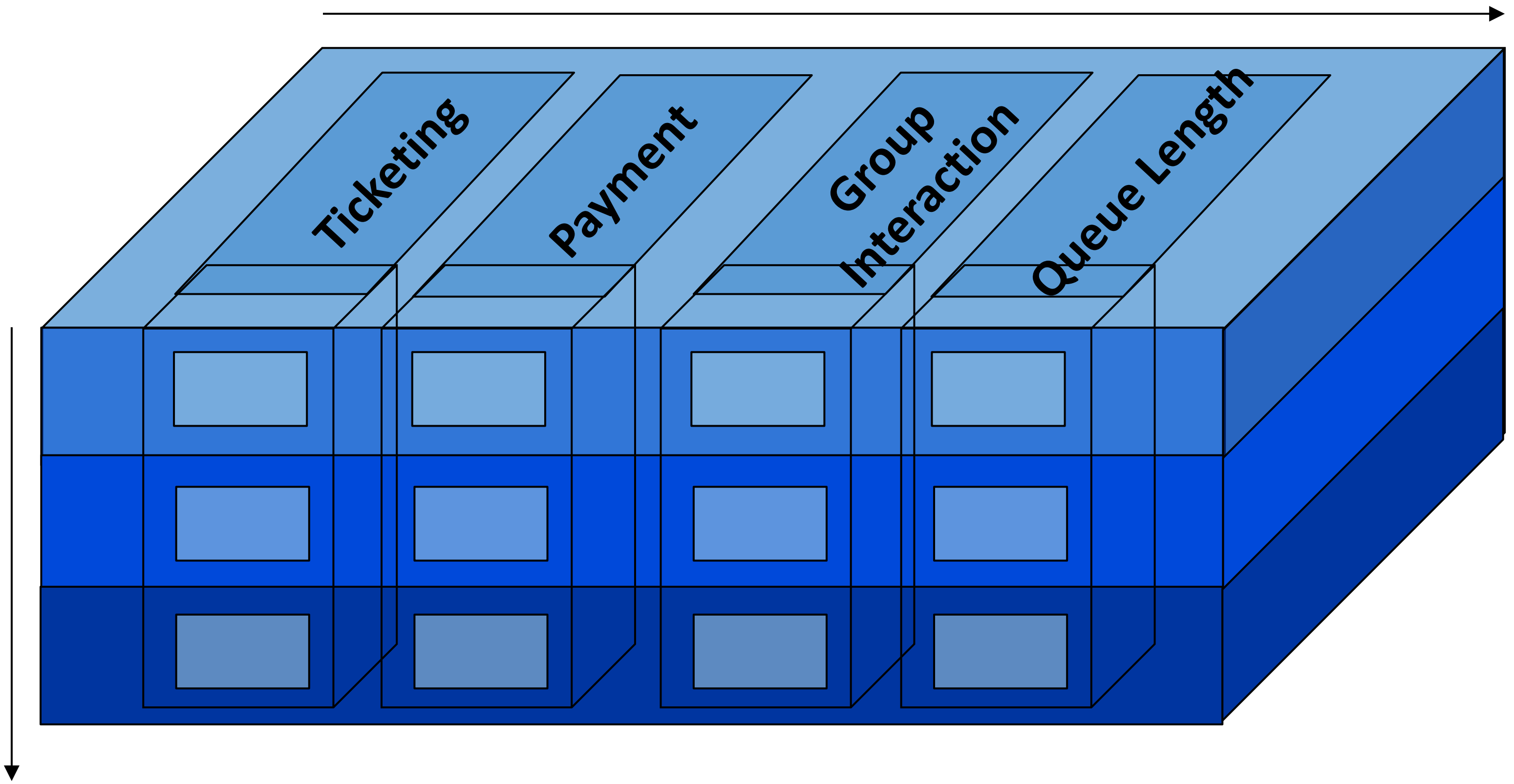
Decomposition of Functionality

- Functional decomposition answer the question: “What are the functions this software must provide?”
- Decomposing is needed to define fine-grain functions
- Functional requirements documents (FD) is a textual representation of functional decomposition. This can be used:
  - as the first step of development
  - as a base of contract with stakeholders

Subsystems vs Layering

Functional Dimension

(in the ‘problem domain’)



Ticketing

Payment

Group Interaction

Queue Length

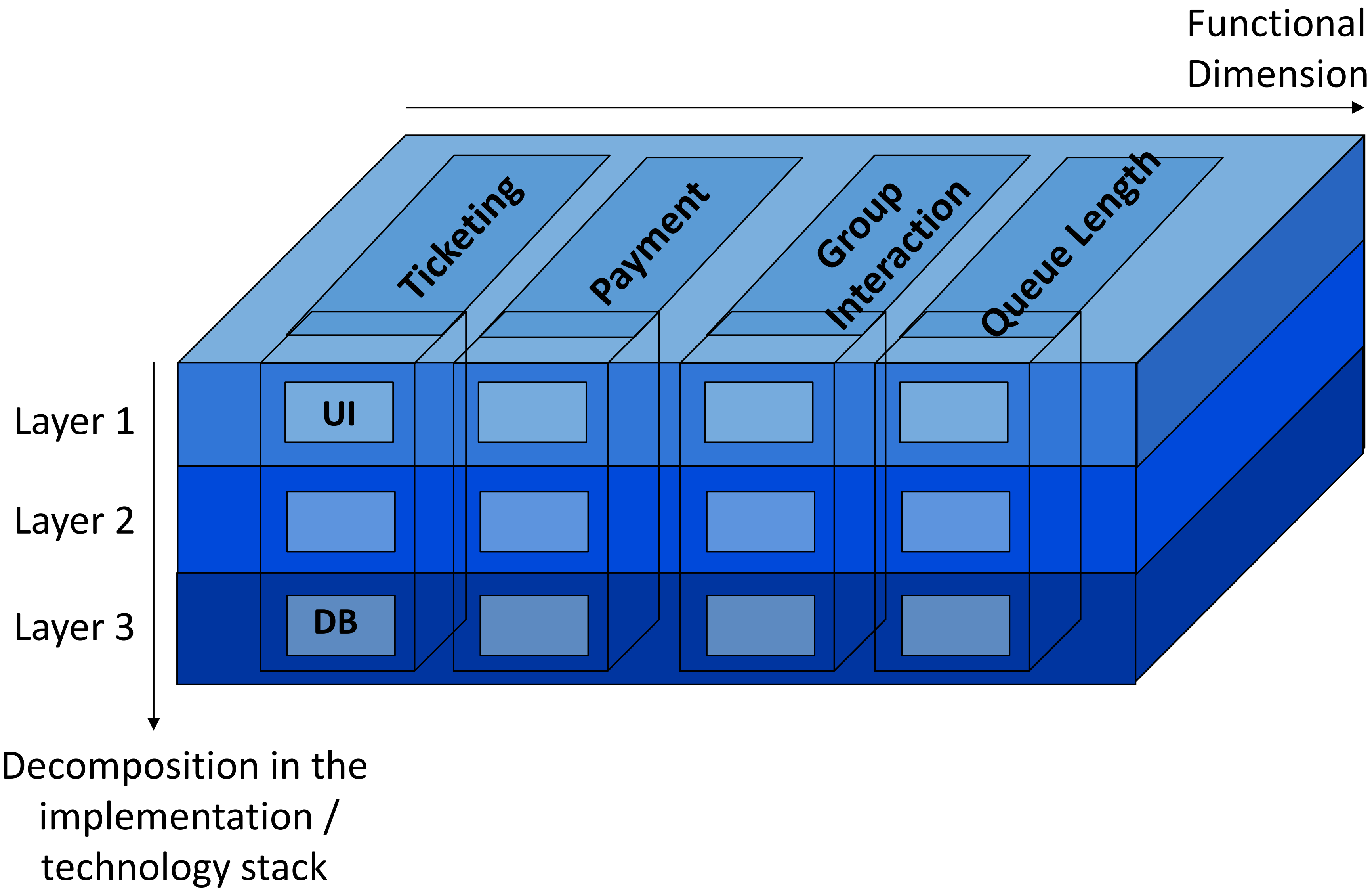
Layer 1

Layer 2

Layer 3

Decomposition in the implementation / technology stack

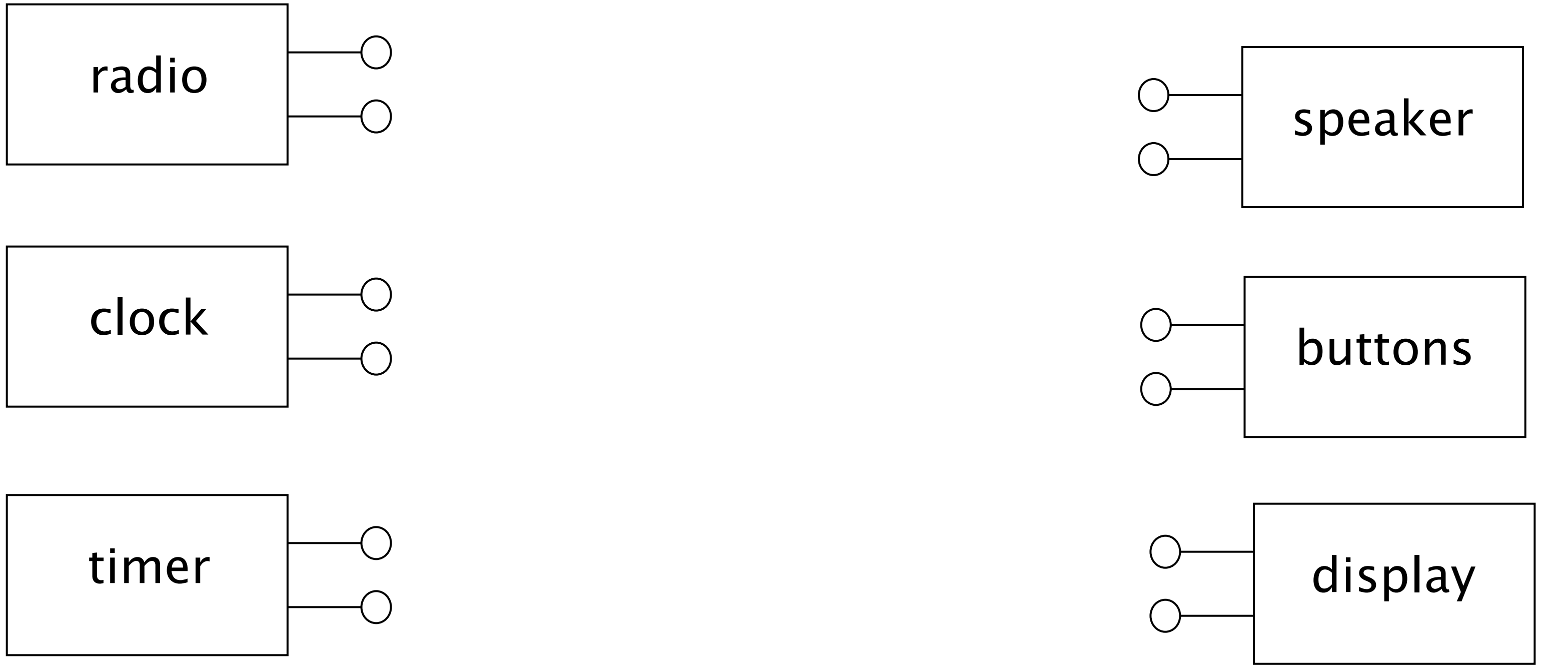
Subsystems vs Layering



Example 1: Radio-Alarm Clock

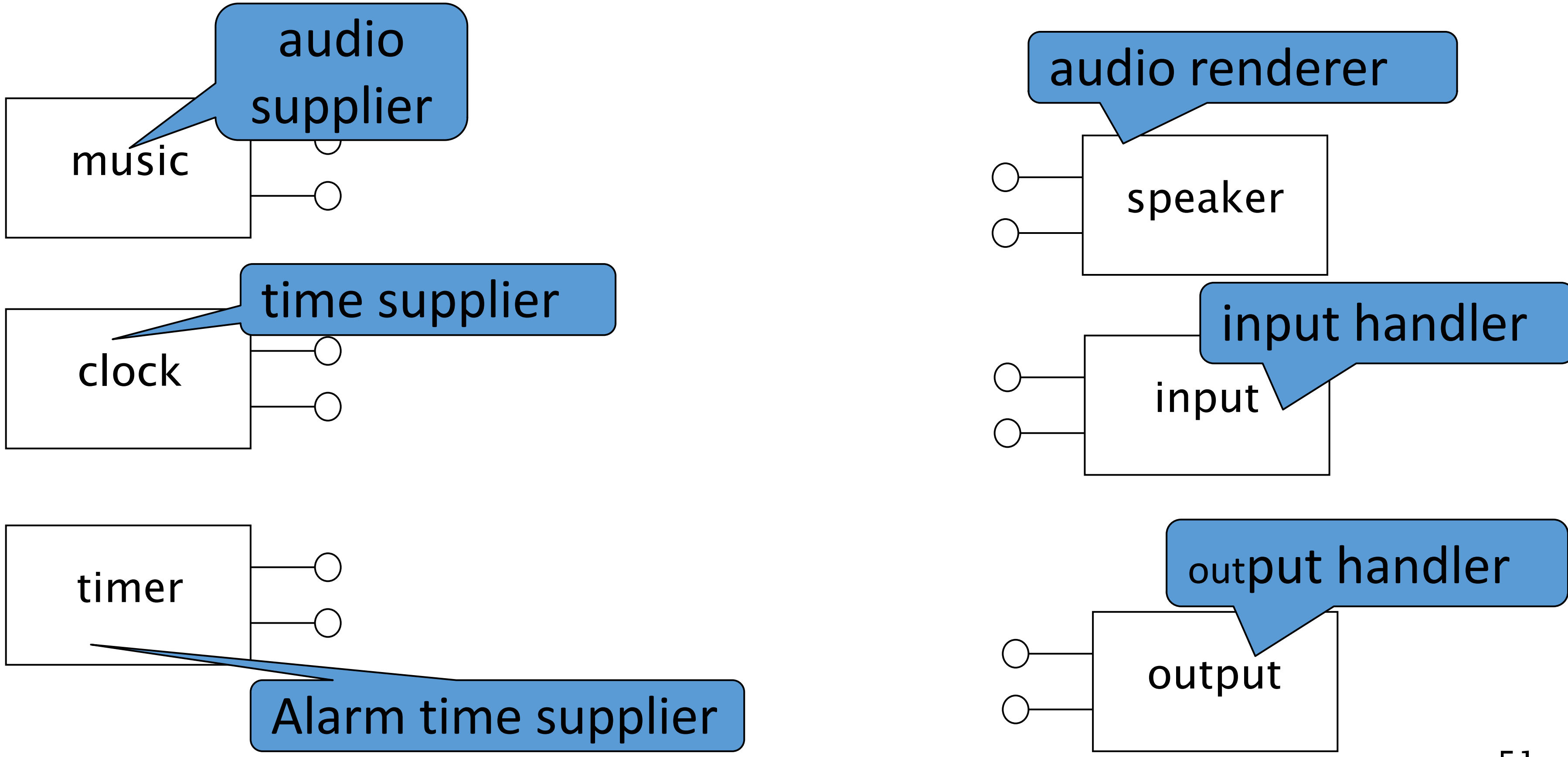
Radio Alarm Clock (initial)

Identify from subsystems the radio-alarm clock can be built?
What should be the responsibility of each component?

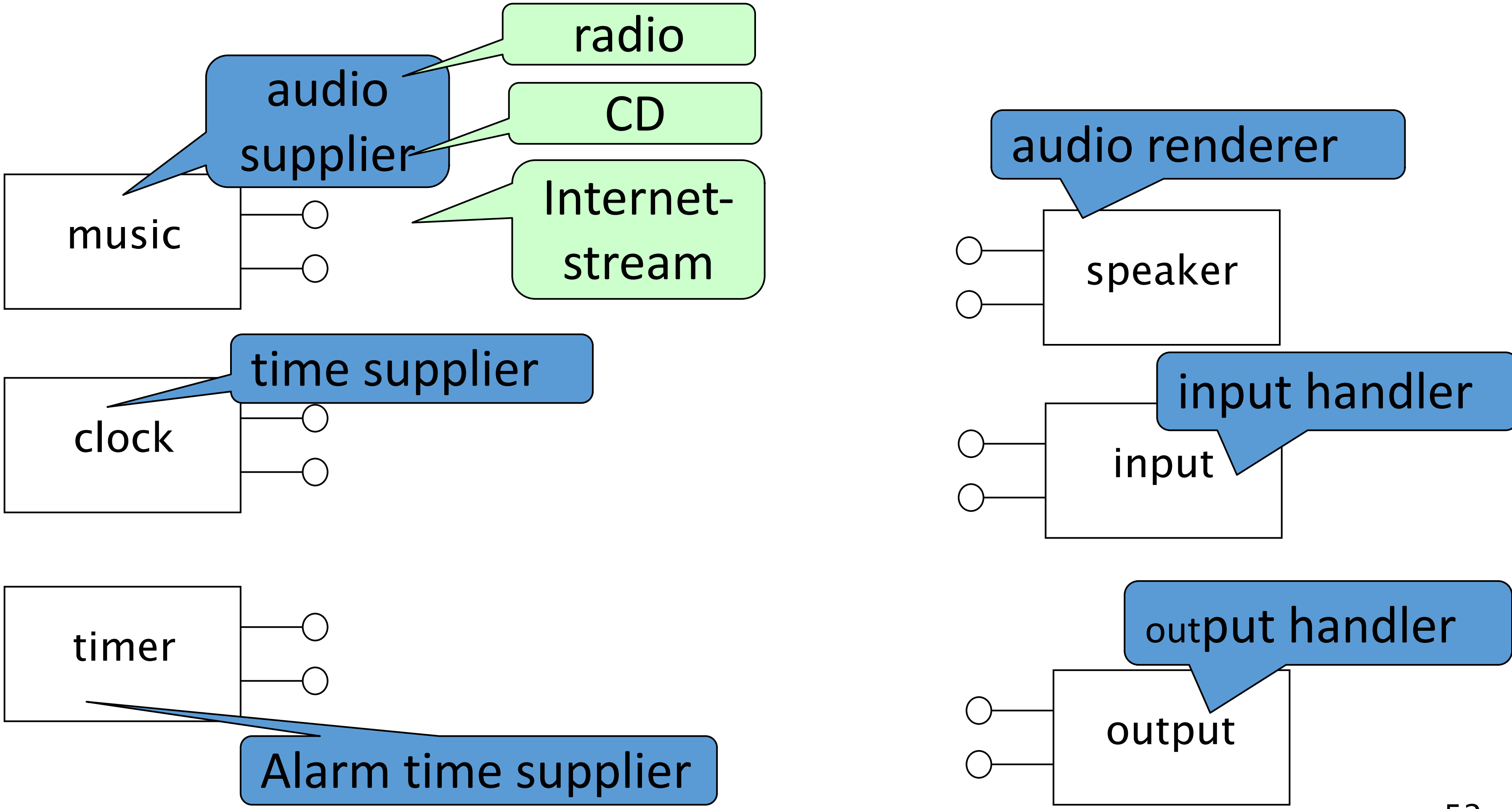


Radio Alarm Clock

Naming: aim for generality

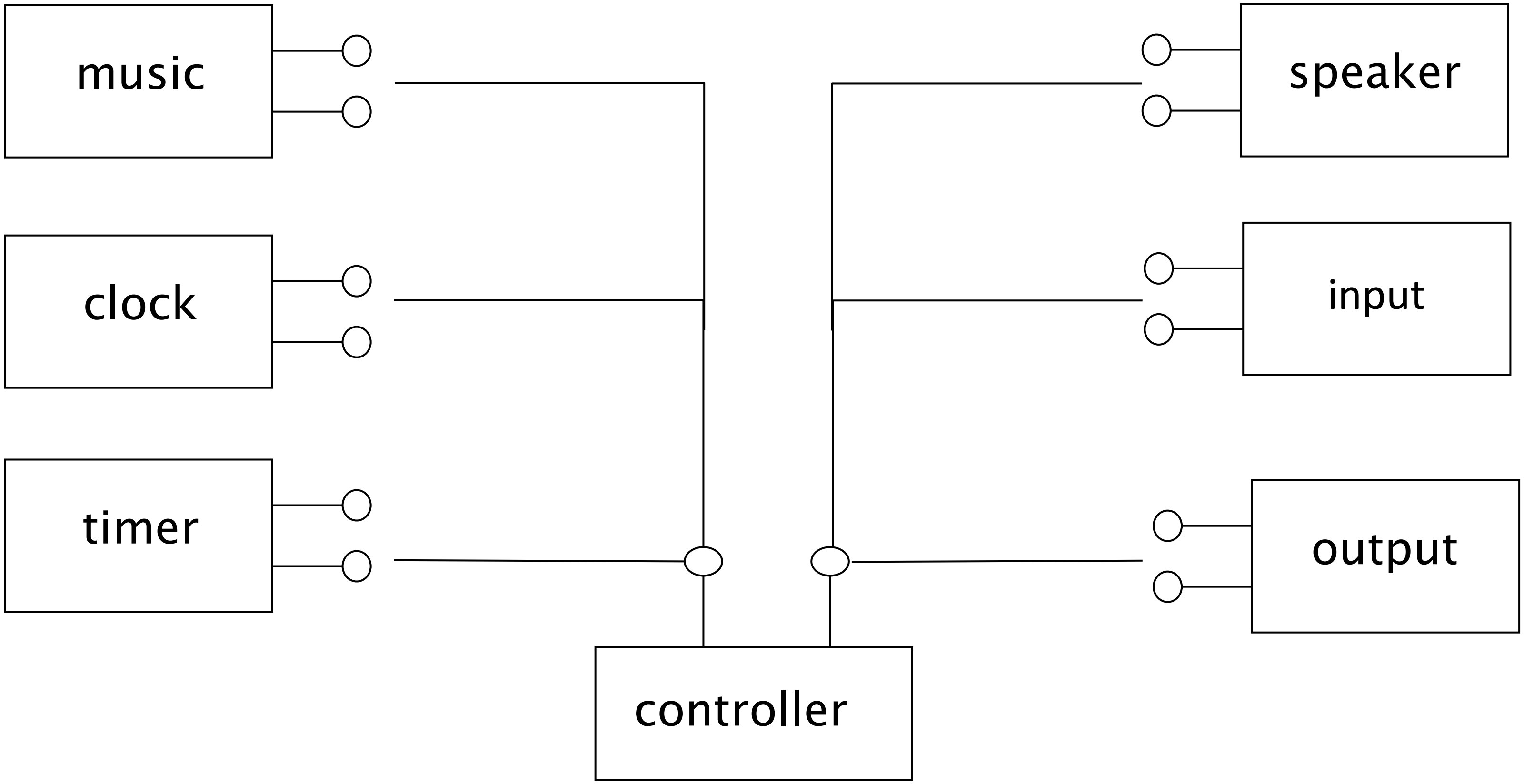


Radio Alarm Clock

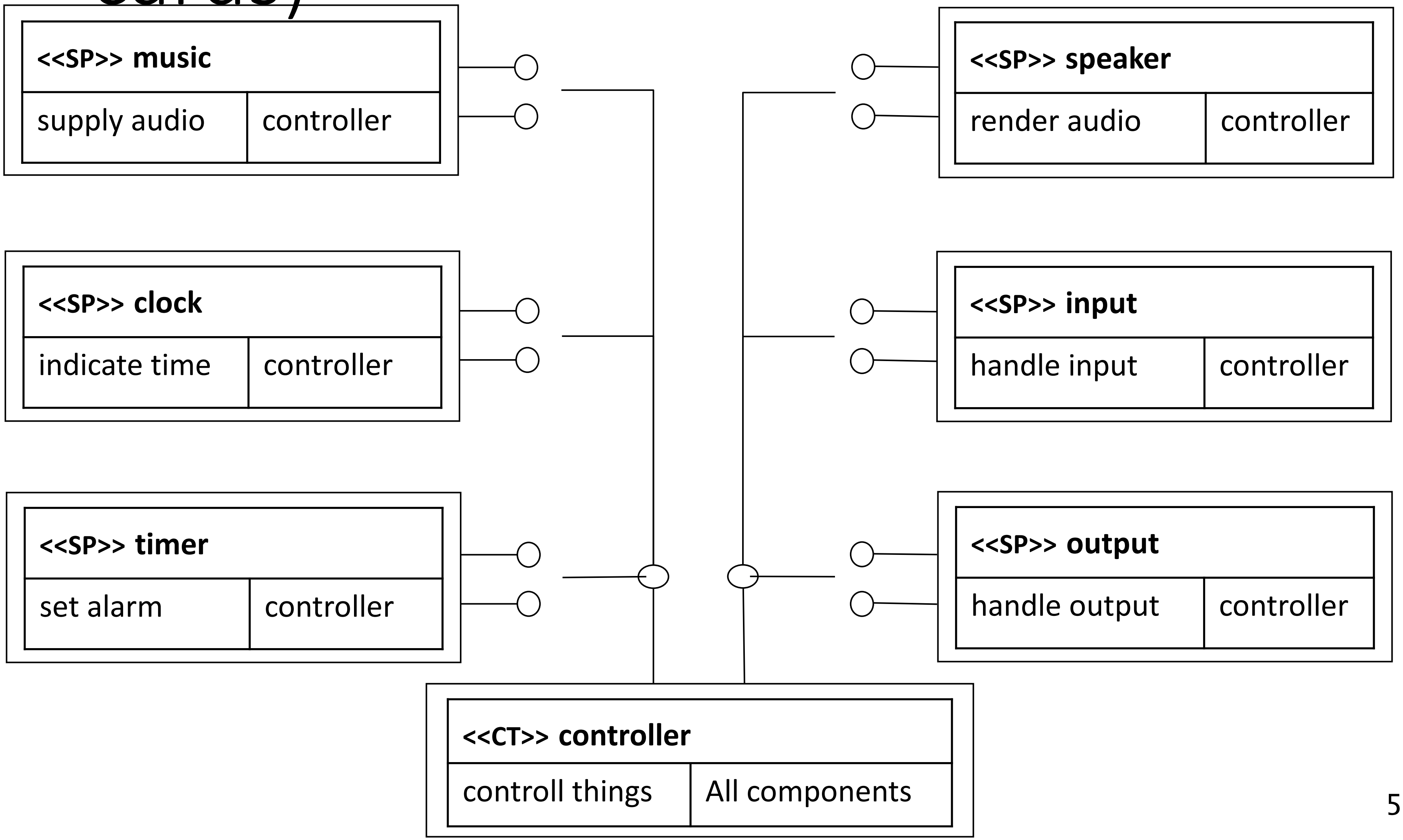


Radio Alarm Clock

A ‘controller’ is an ‘integrator’ of all functionalities



Radio Alarm Clock (with CRC cards)



<<SP>> music	
supply audio	controller

<<SP>> clock	
indicate time	controller

<<SP>> timer	
set alarm	controller

<<SP>> speaker	
render audio	controller

<<SP>> input	
handle input	controller

<<SP>> output	
handle output	controller

<<CT>> controller	
controll things	All components

Radio Alarm Clock

Can your design easily accommodate extensions?

lamp

temperature

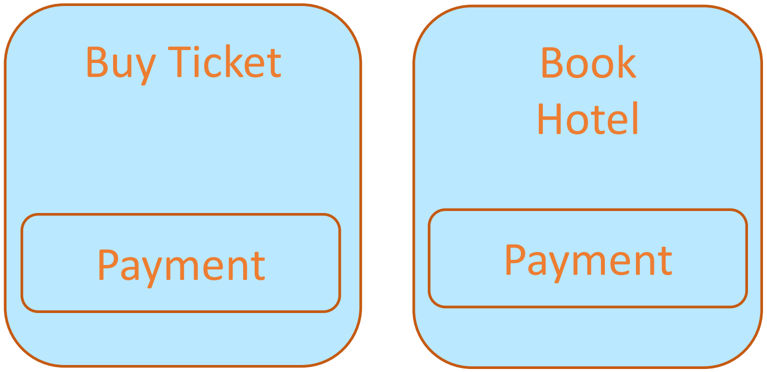
Alarm-time

Bat-alarm

(wireless) atomic clock

Train strike/traffic delays

Which Design and Why?



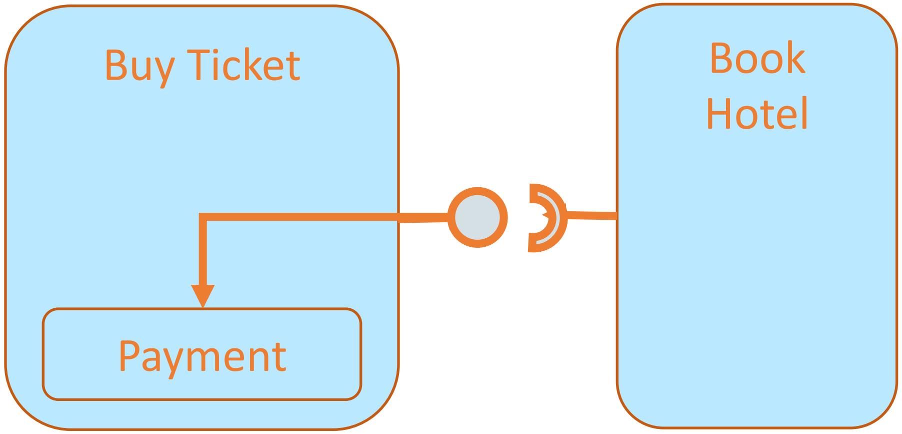
Buy Ticket

Payment

Book Hotel

Payment

A



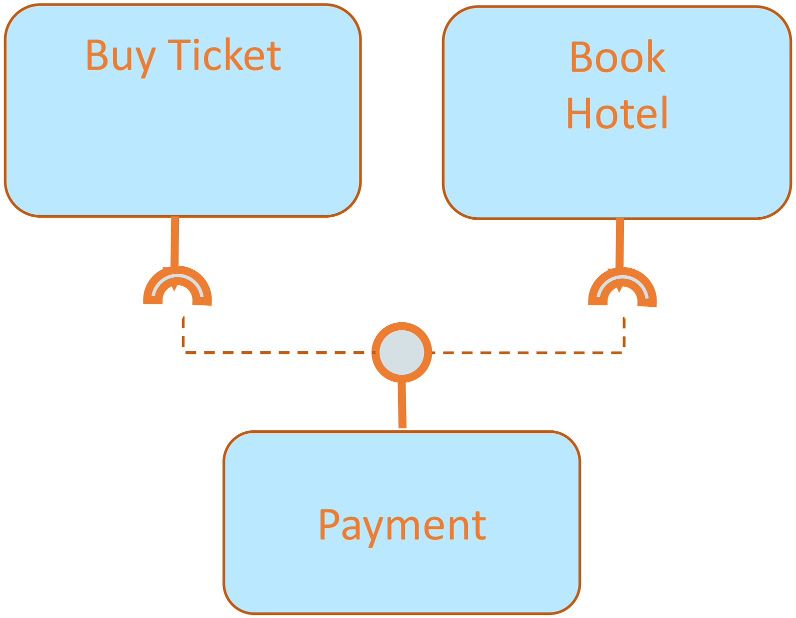
Buy Ticket

Payment

Book Hotel

B

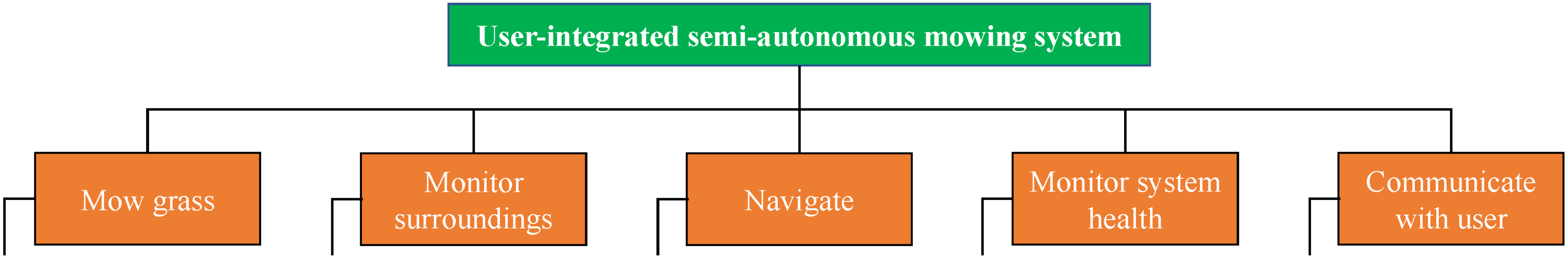
Factor out what is common



Payment-functionality is

1) a common, generic service
2) a clear cohesive responsibility
3) a unit of change

Example 2: User-integrated semi-autonomous mowing system



Shows:

- Decomposition into main functions of the system

Alt: responsibilities / tasks

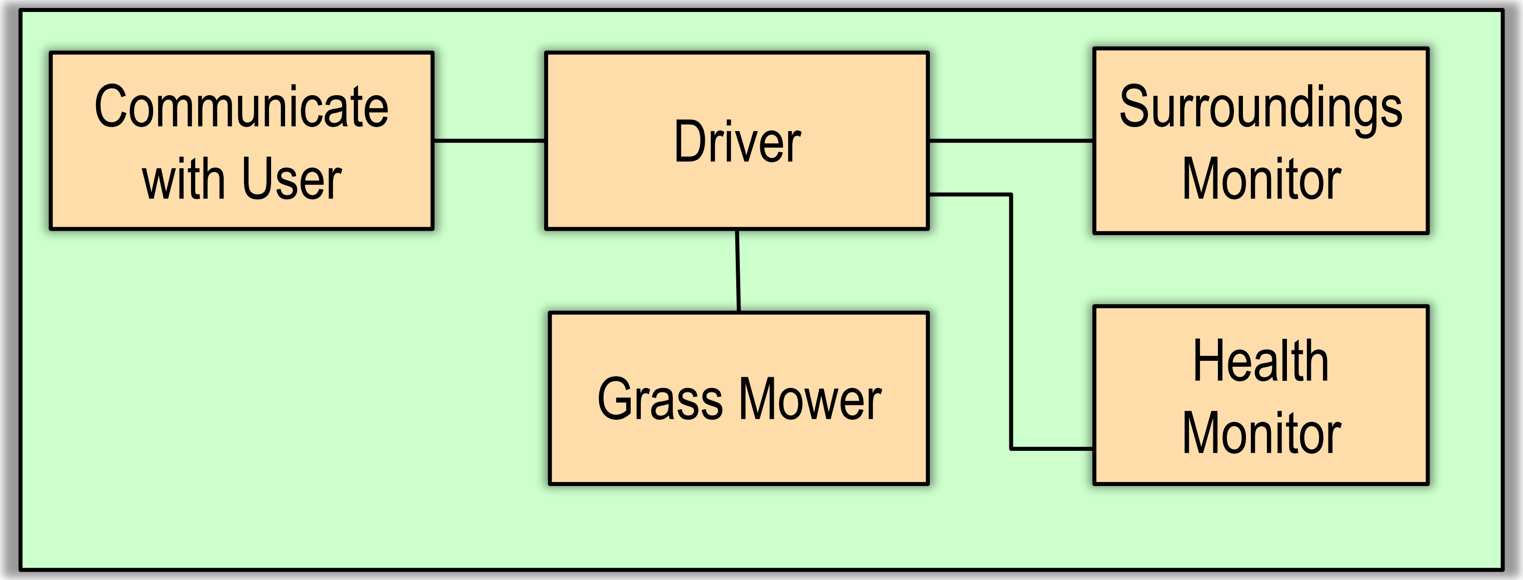
Does not show:

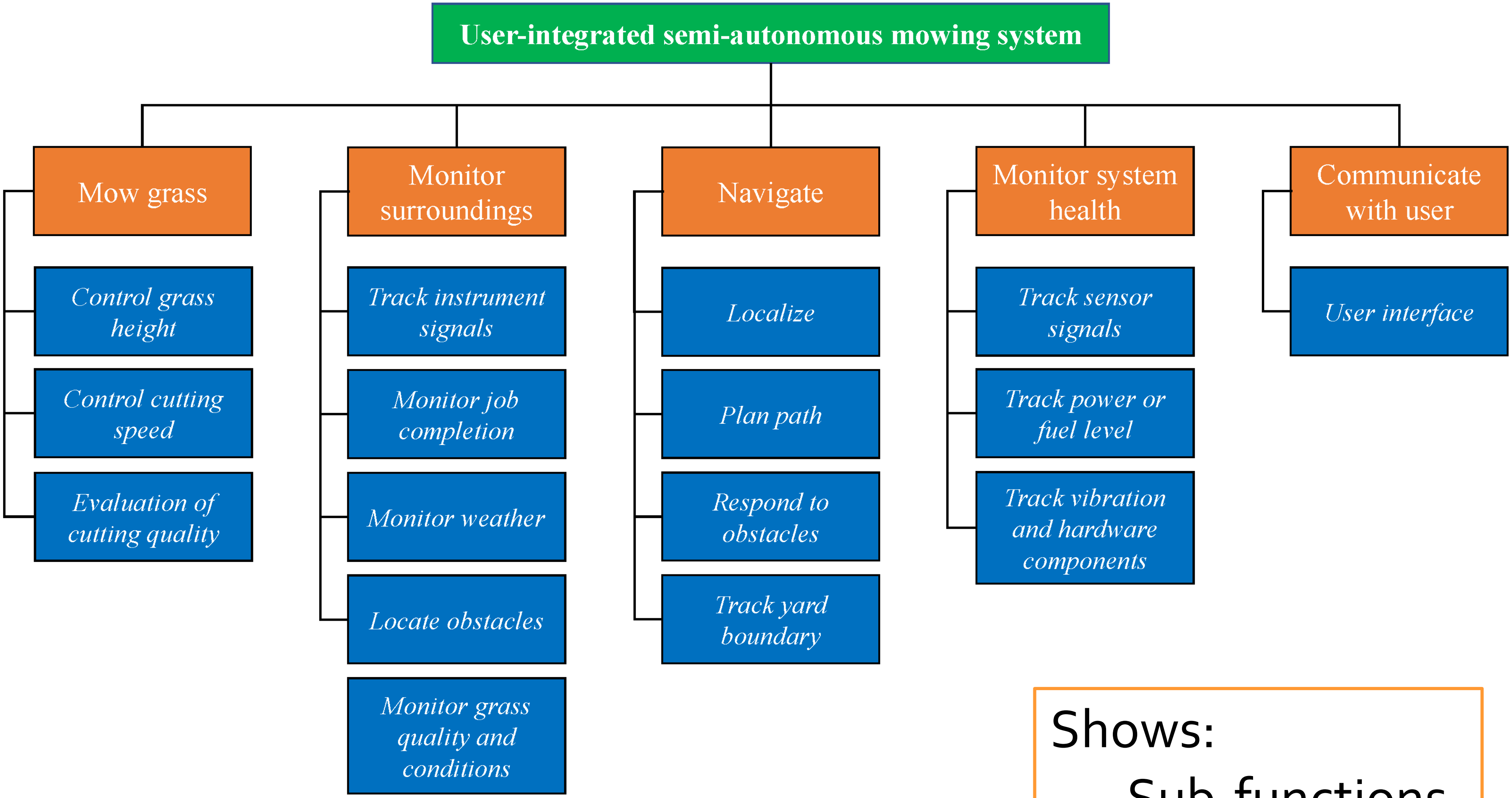
- How components are implemented
- IT does not show 'power' or 'memory'!

Not at this 'perspective'/abstraction

AgriEngineering 2019, 1(3), 453-474; https://doi.org/10.3390/agriengineering1030033
Development of User-Integrated Semi-Autonomous Lawn Mowing Systems: A Systems Engineering Perspective and Proposed Architecture by Albert E. Patterson,Yang Yuan and William R. Norris

Autonomous Grass Mower Subsystem decomposition





User-integrated semi-autonomous mowing system

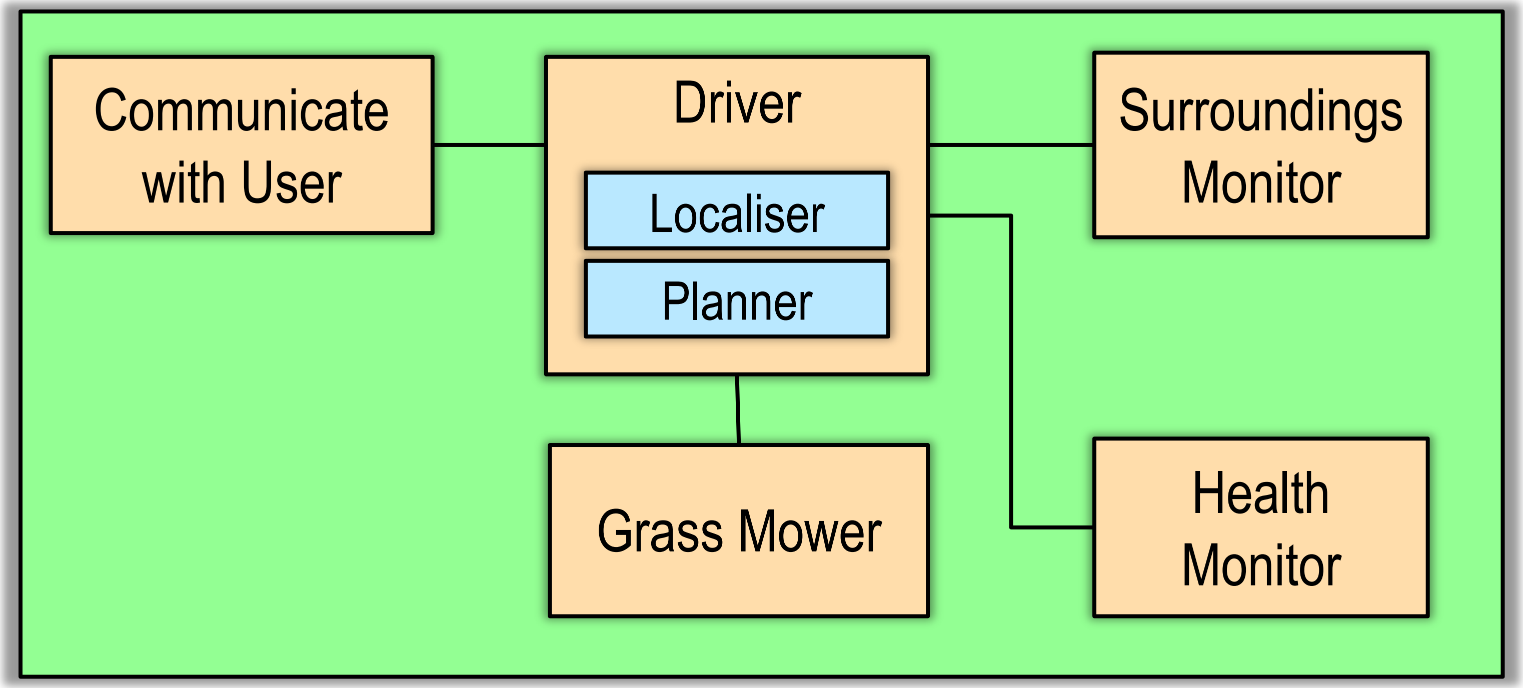
- Mow grass
  - *Control grass height*
  - *Control cutting speed*
  - *Evaluation of cutting quality*
- Monitor surroundings
  - *Track instrument signals*
  - *Monitor job completion*
  - *Monitor weather*
  - *Locate obstacles*
  - *Monitor grass quality and conditions*
- Navigate
  - *Localize*
  - *Plan path*
  - *Respond to obstacles*
  - *Track yard boundary*
- Monitor system health
  - *Track sensor signals*
  - *Track power or fuel level*
  - *Track vibration and hardware components*
- Communicate with user
  - *User interface*

Shows:

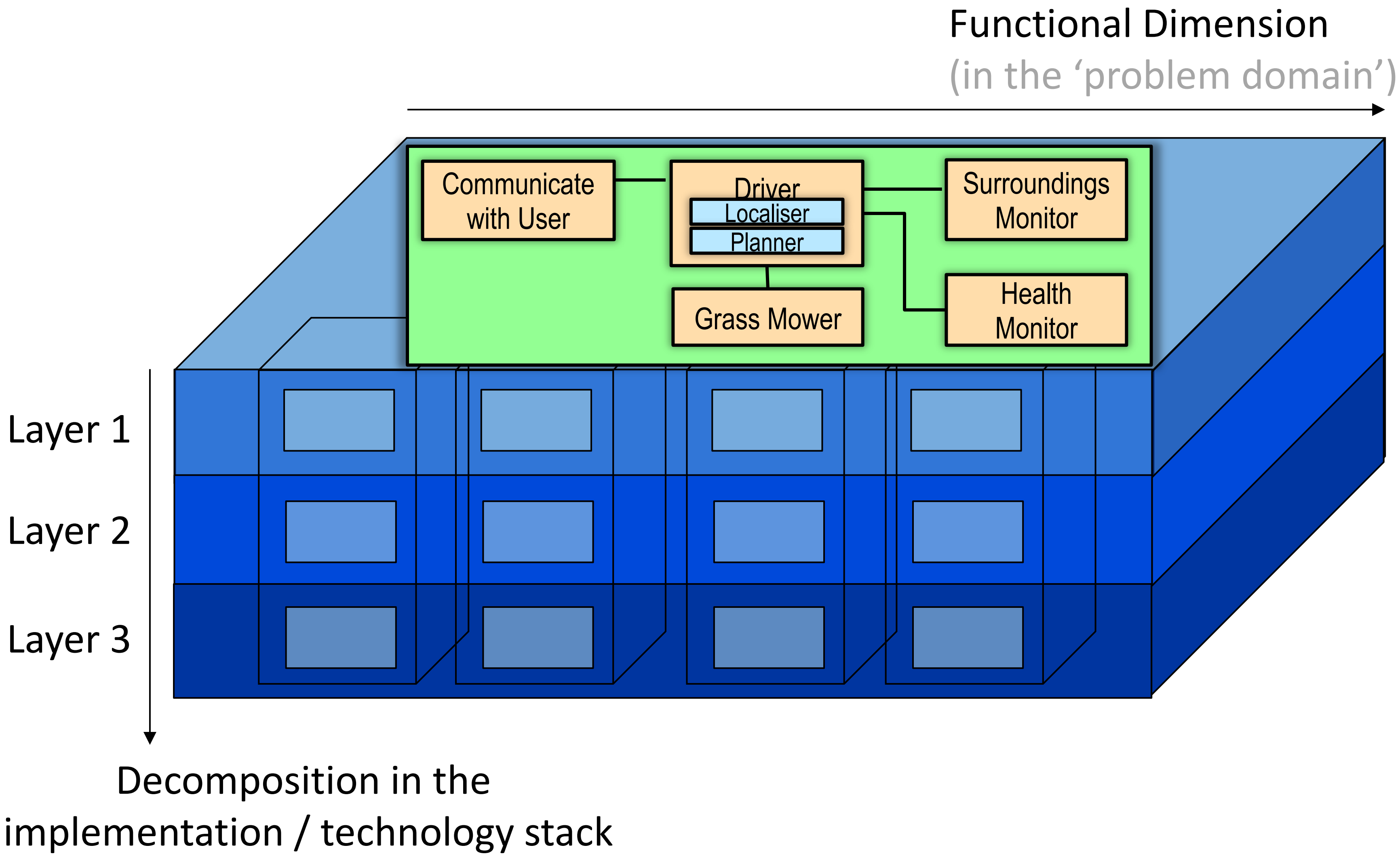
- Sub-functions

AgriEngineering 2019, 1(3), 453-474; https://doi.org/10.3390/agriengineering1030033
Development of User-Integrated Semi-Autonomous Lawn Mowing Systems: A Systems Engineering Perspective and Proposed Architecture by Albert E. Patterson,Yang Yuan and William R. Norris

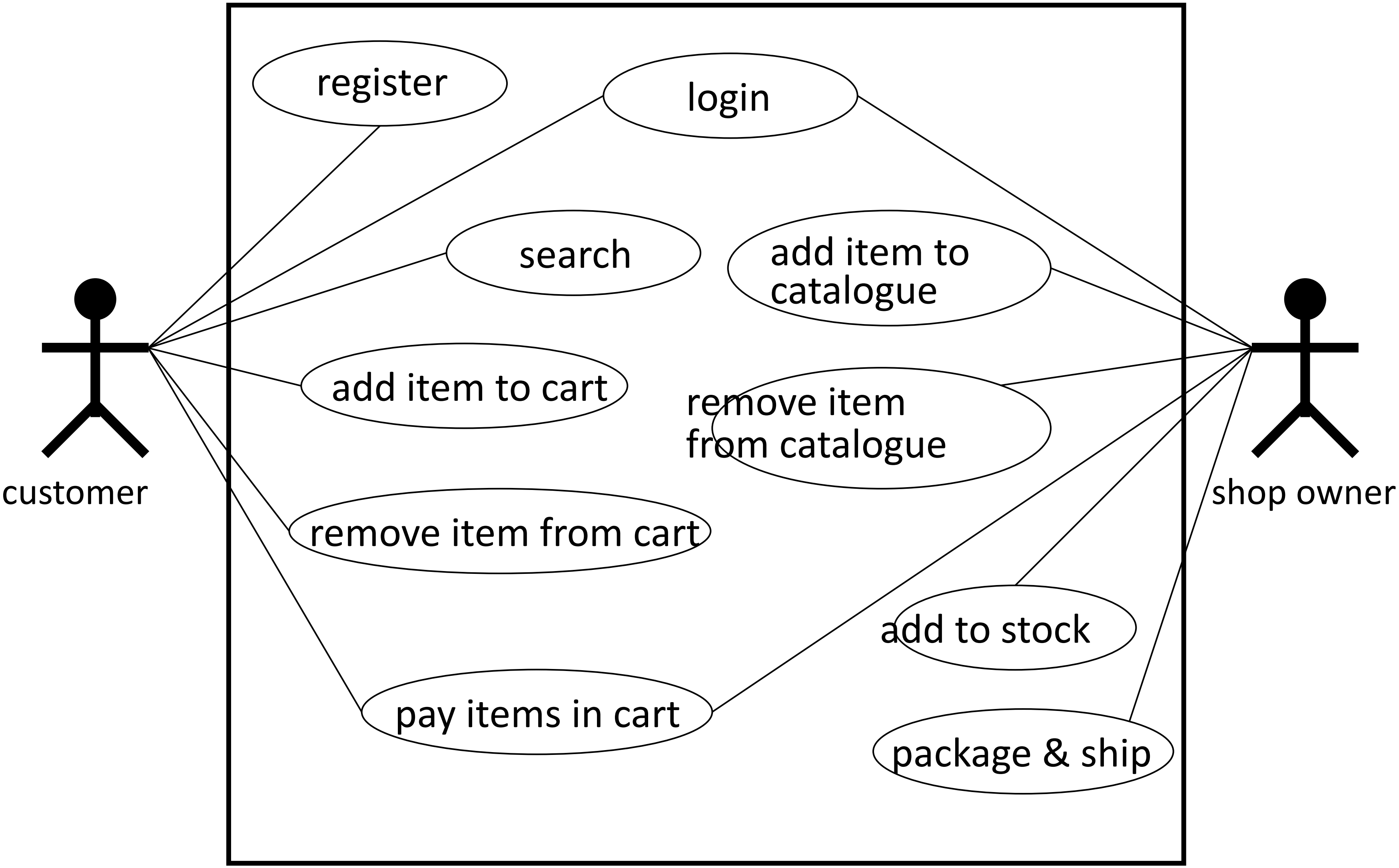
Autonomous Grass Mower
Sub-subsystem decomposition



Subsystems vs Layering



Example Design Case 3: Web Shop

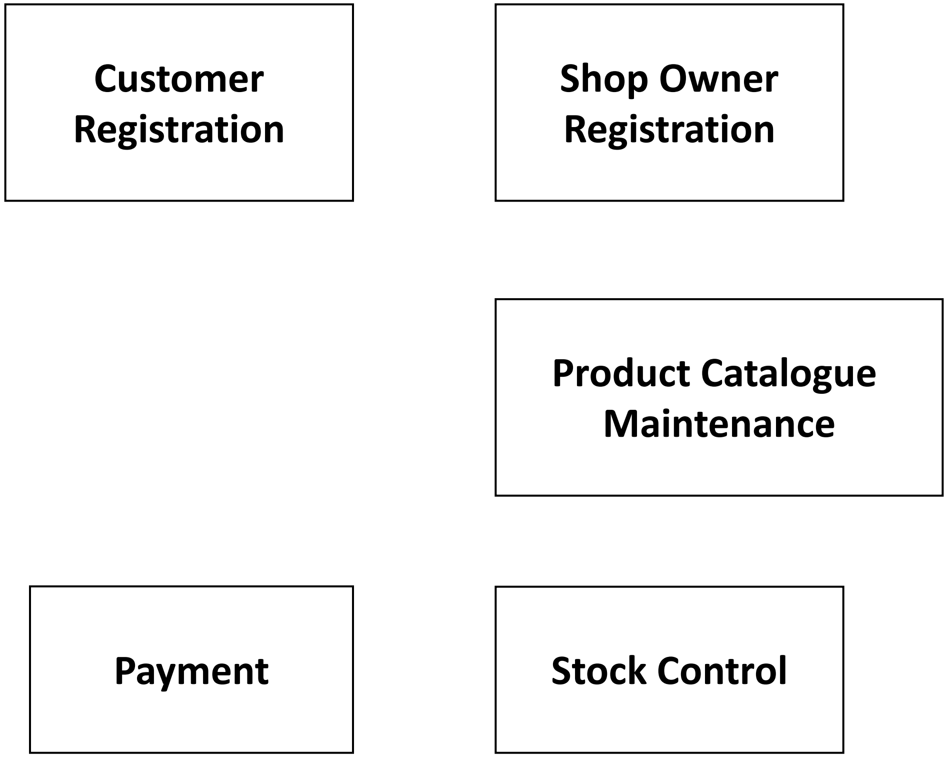


Structure/Group Functionality

- Defines subsystems of functionality
- Purpose
    - Define decomposition into subsystems
    - Provide support for use-cases
- Think in terms of responsibilities

- Use Component diagram

Web Shop: Functional Areas (V0.1)



Customer Registration

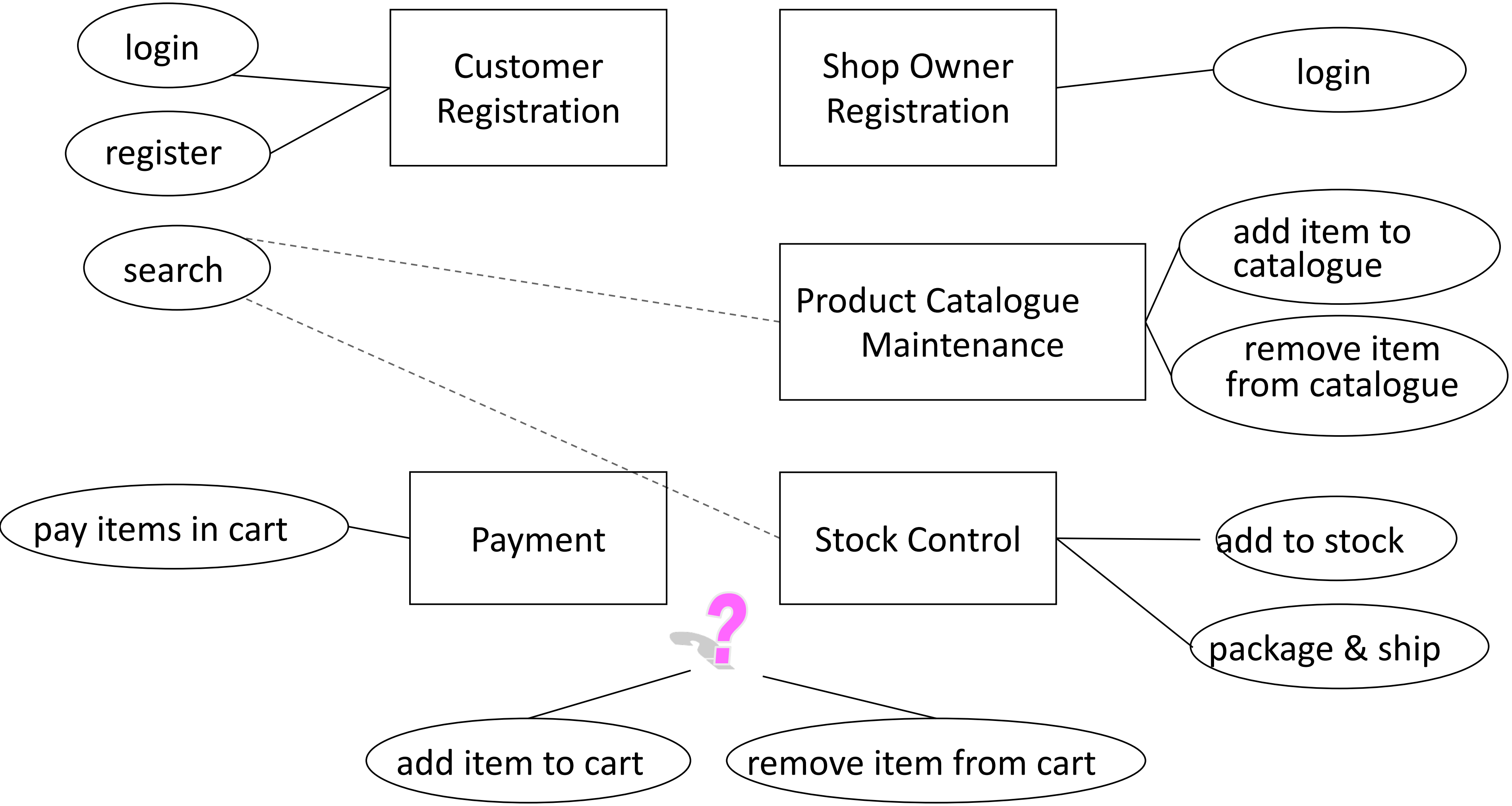
Shop Owner Registration

Product Catalogue Maintenance

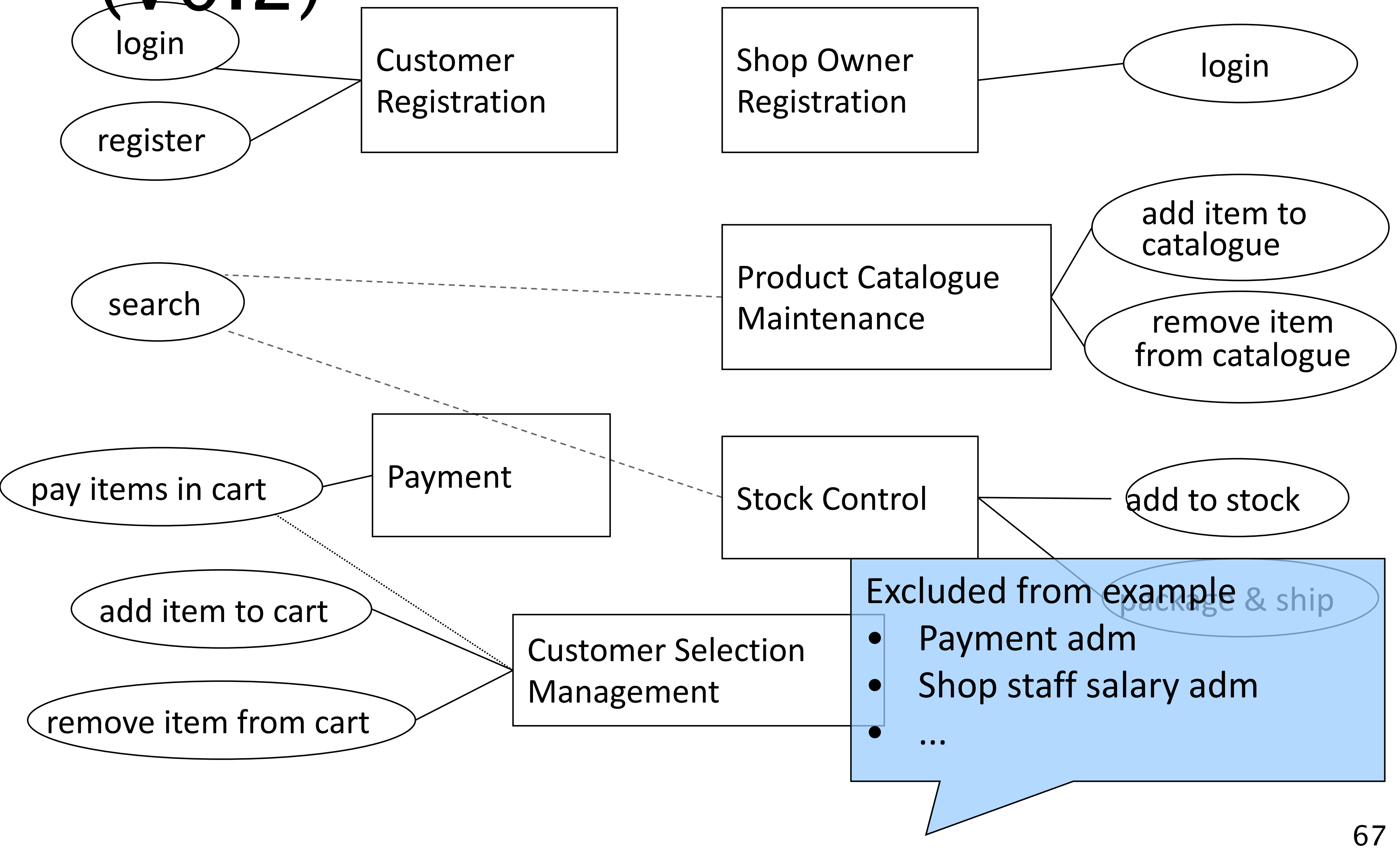
Payment

Stock Control

Check Use Cases Against Functional Areas



Web Shop: Functional Areas (V0.2)



login

register

Customer Registration

Shop Owner Registration

login

search

Product Catalogue Maintenance

add item to catalogue

remove item from catalogue

Payment

pay items in cart

Stock Control

add to stock

package & ship

add item to cart

remove item from cart

Customer Selection Management

Excluded from example

- Payment adm
- Shop staff salary adm
- ...

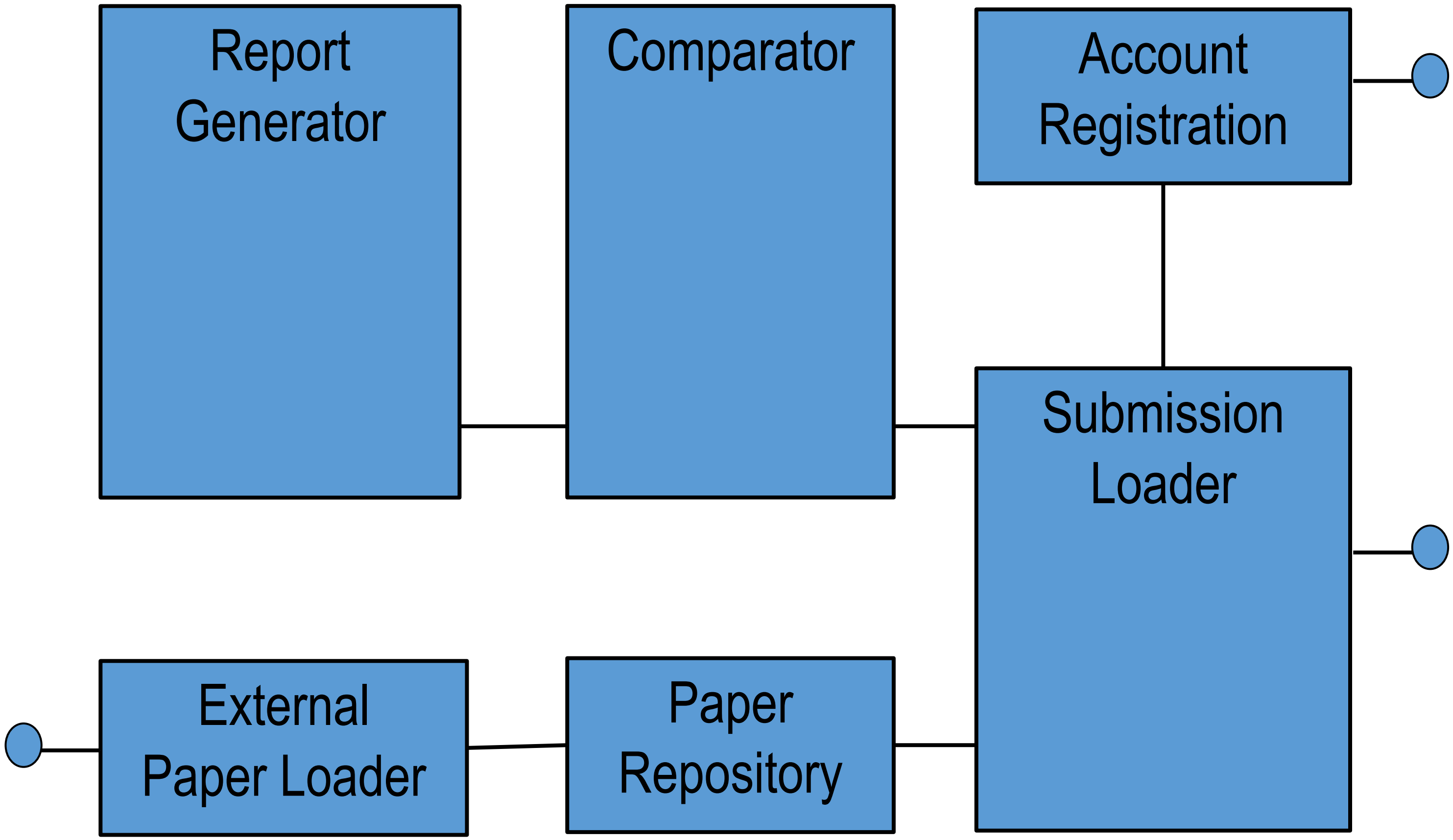
Web Shop: Responsabilities

Customer Registration	Maintain customer accounts
Shop Owner Registration	Maintain staff accounts
Product Catalog Maint.	Maintain product data
Cust. Selection Mngmt.	Maintain customer product selection
Payment	Handle payment between customer, shop & bank
Stock Control	Maintain availability of products in stock

Example 4:
Automated Plagiarism Checking System

- University can have subscriptions
- University-faculty can make accounts
- Faculty can send in documents for checking
    - Documents are turned into a standard internal format
    - The document is segmented (chapters, section, sentences, …)
    - Document is compared on a sentence by sentence basis.
    - A plagiarism score is produced
    - A report is sent to the person that sent in the document
- The system keeps records of use for producing yearly accounting reports

Decomposition into Subsystems



Where do these sub-subsystems go?

Paper Reporter

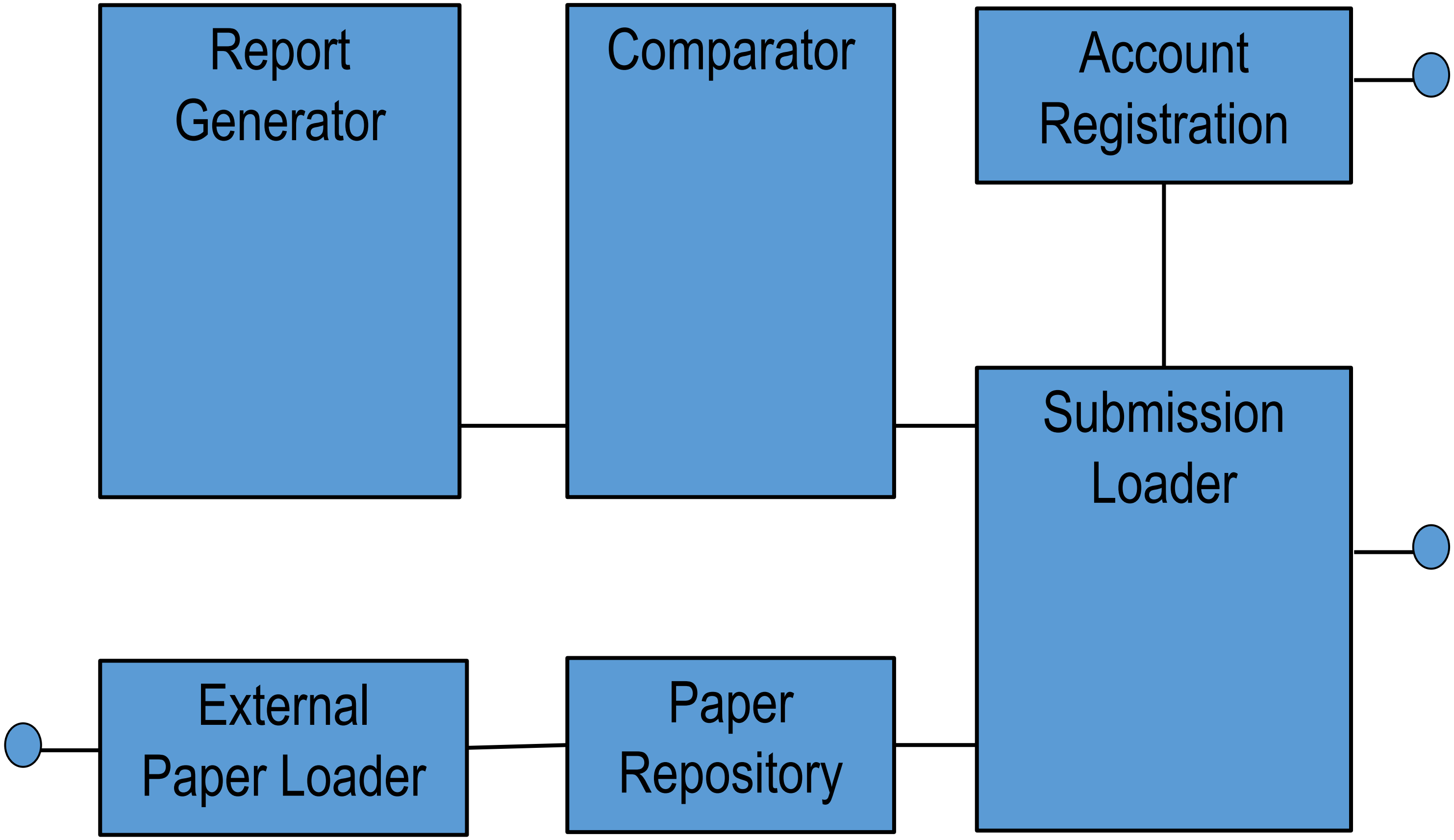
University Reporting

Compute Score

Comparator

File Segmenter

File Converter



Report Generator

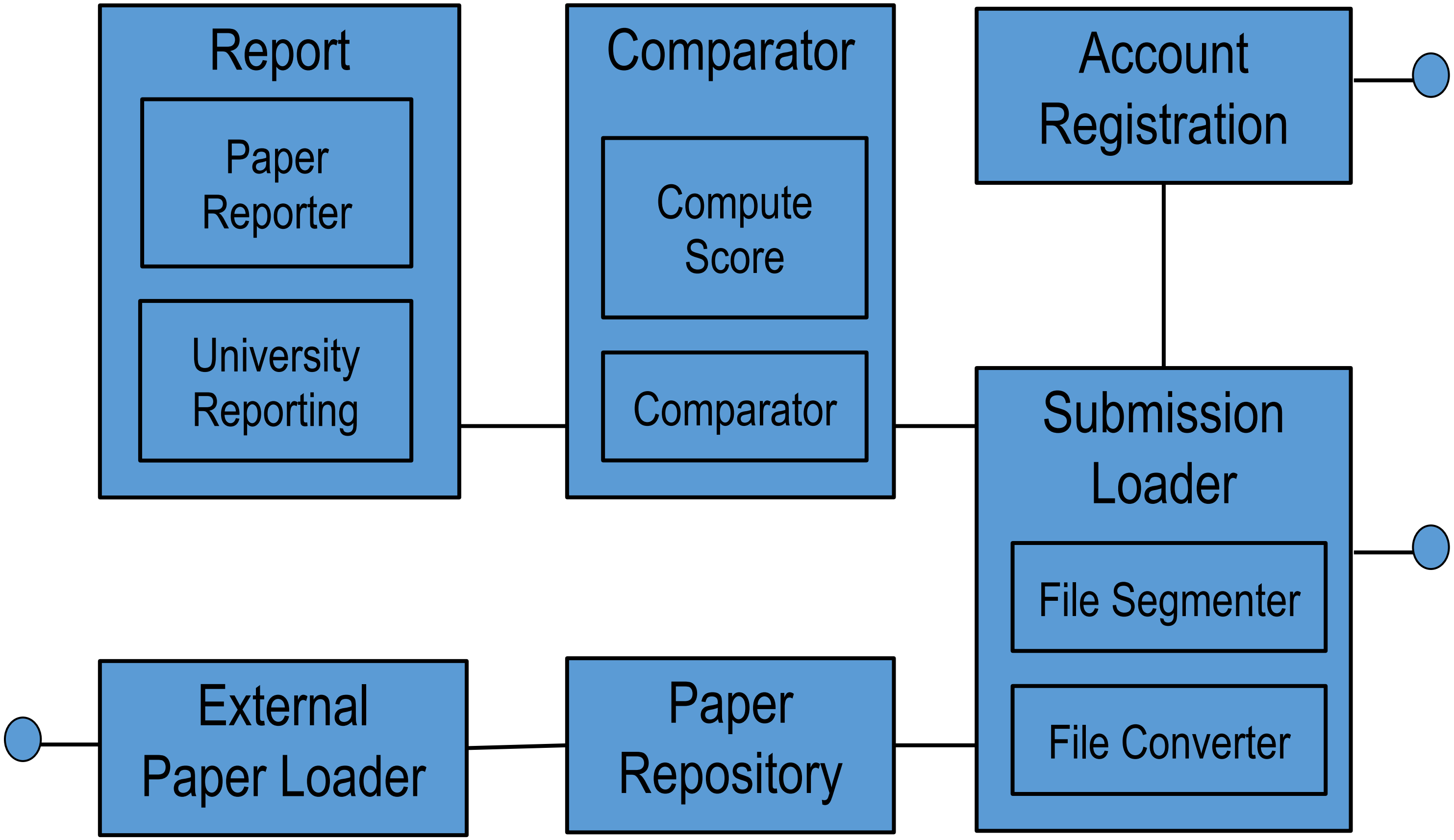
Comparator

Account Registration

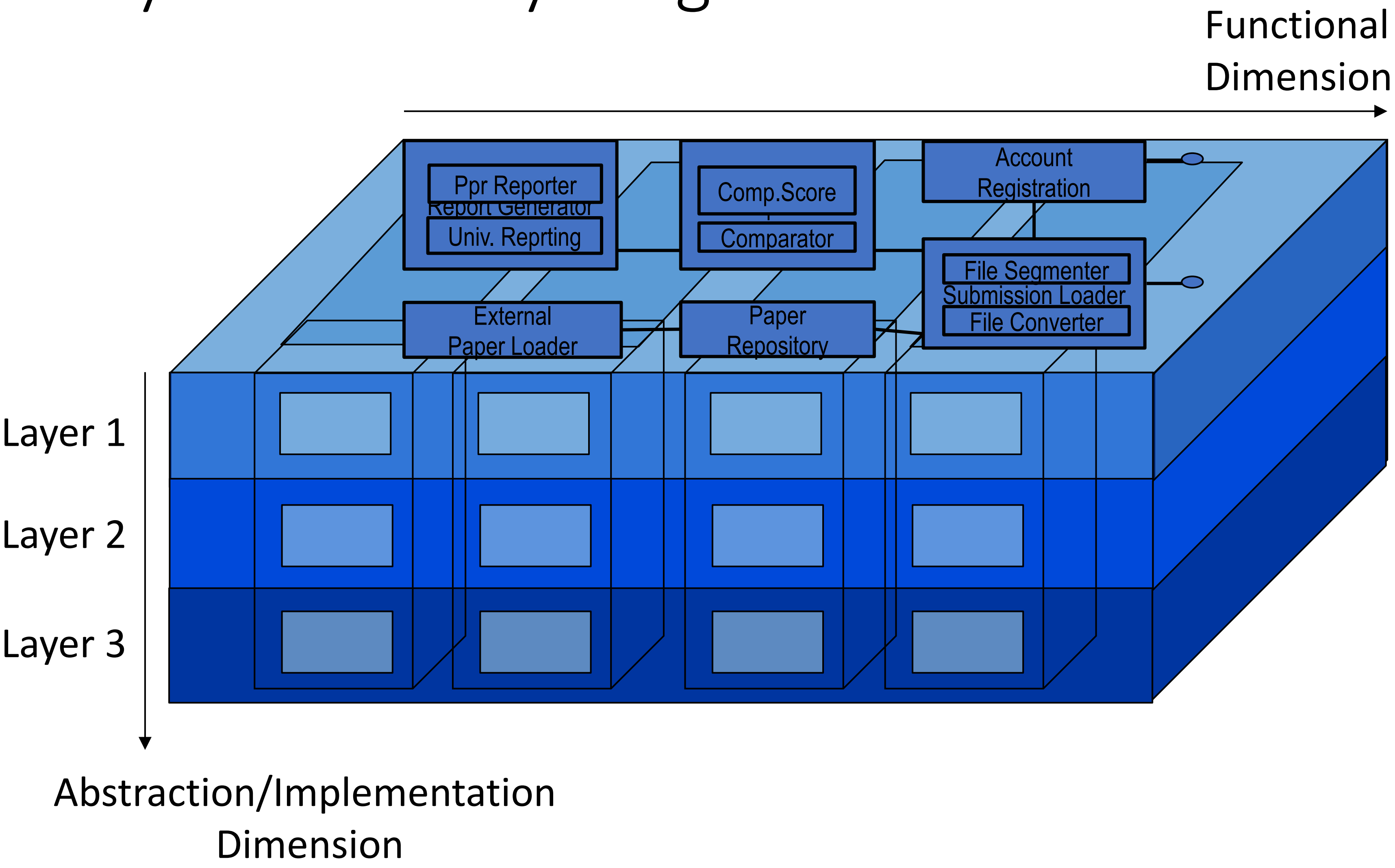
Submission Loader

External Paper Loader

Paper Repository


Report
Paper Reporter
University Reporting
Comparator
Compute Score
Comparator
Account Registration
Submission Loader
File Segmenter
File Converter
External Paper Loader
Paper Repository

Subsystems vs Layering



Analysis & Design

In this course

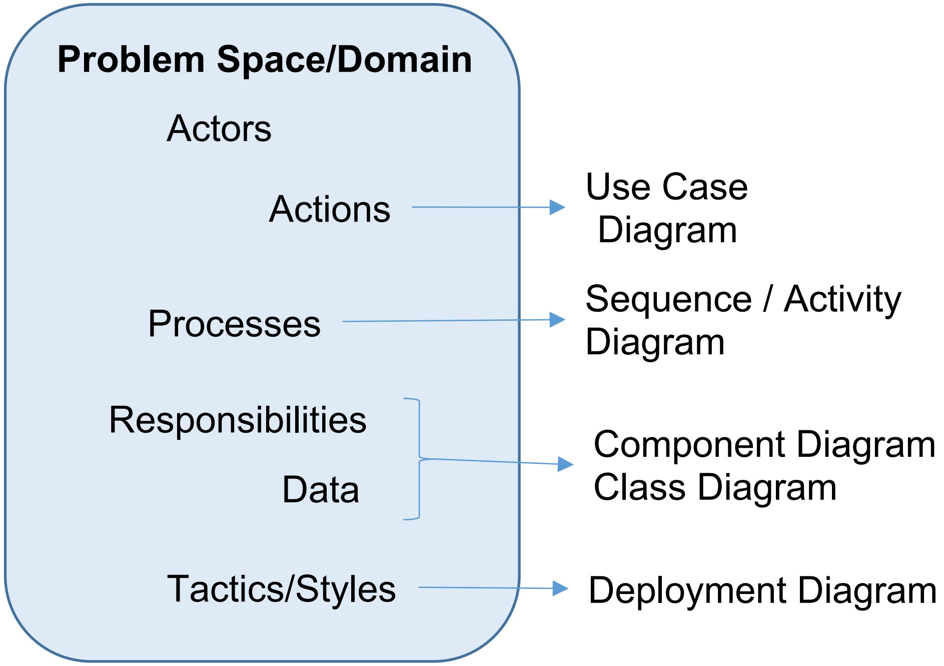
Analysis is for: understanding & describing the domain

describes ***what*** : main concept & their relations

Design is for: synthesis of executable solution

describes ***how*** a construction of a solution should work

From Analysis to Design



Problem Space/Domain

Actors

Actions → Use Case Diagram

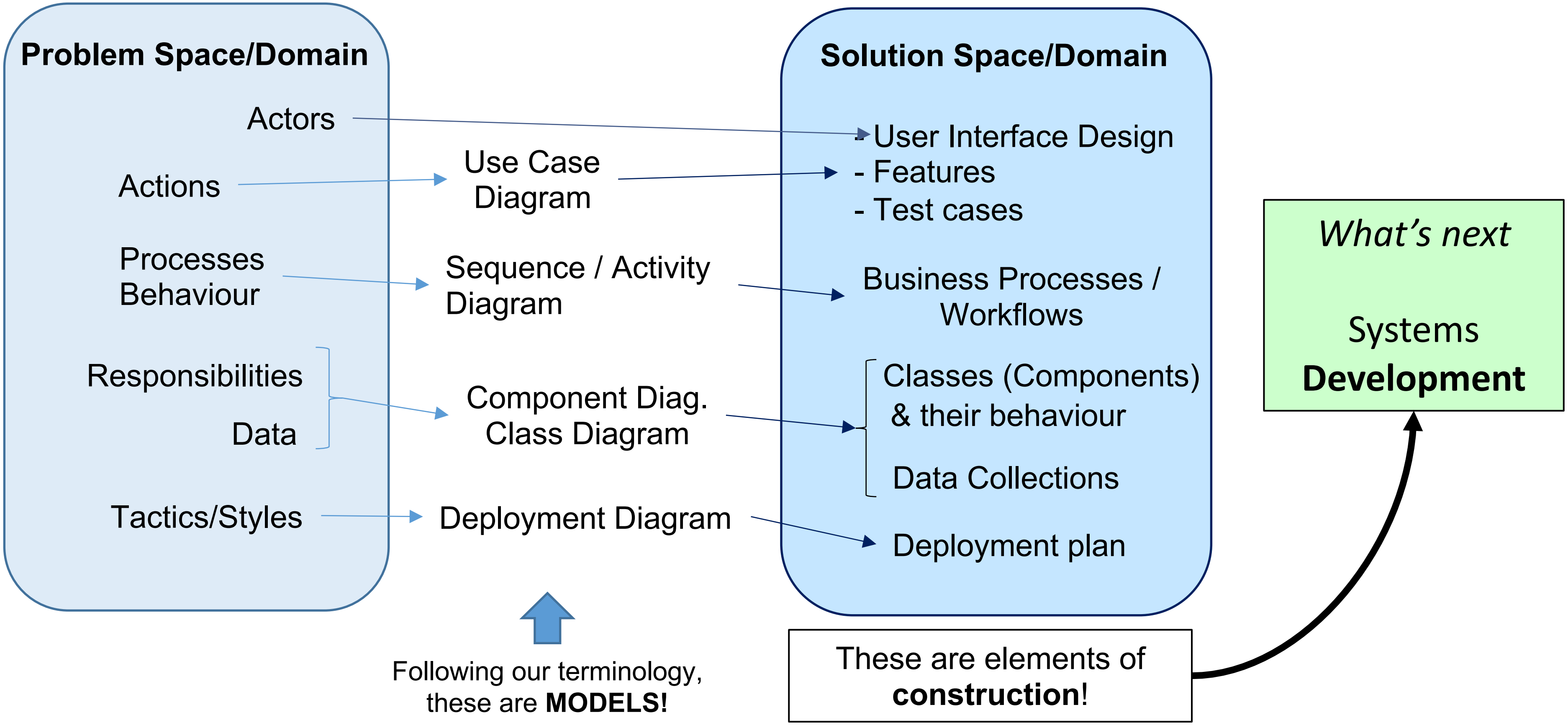
Processes → Sequence / Activity Diagram

Responsibilities, Data → Component Diagram, Class Diagram

Tactics/Styles → Deployment Diagram

These are elements of **analysis:**
What is there?

From Analysis to Design



Problem Space/Domain

Actors

Actions

Processes
Behaviour

Responsibilities

Data

Tactics/Styles

Use Case
Diagram

Sequence / Activity
Diagram

Component Diag.
Class Diagram

Deployment Diagram

Solution Space/Domain

- User Interface Design
- Features
- Test cases

Business Processes /
Workflows

Classes (Components)
& their behaviour

Data Collections

Deployment plan

What's next

Systems
Development

Following our terminology,
these are **MODELS!**

These are elements of
construction!

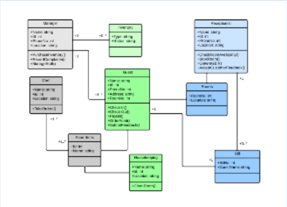
From Analysis to Initial Design

The OO paradigm has been designed such that Analysis and Design models look ‘alike’: Classes that appear in a domain model, can also appear in a design model

Descriptive Models

Problem Space/Domain

Component/Class Diagram

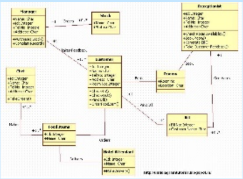


These are elements of **analysis:**
What is there?

Prescriptive Models

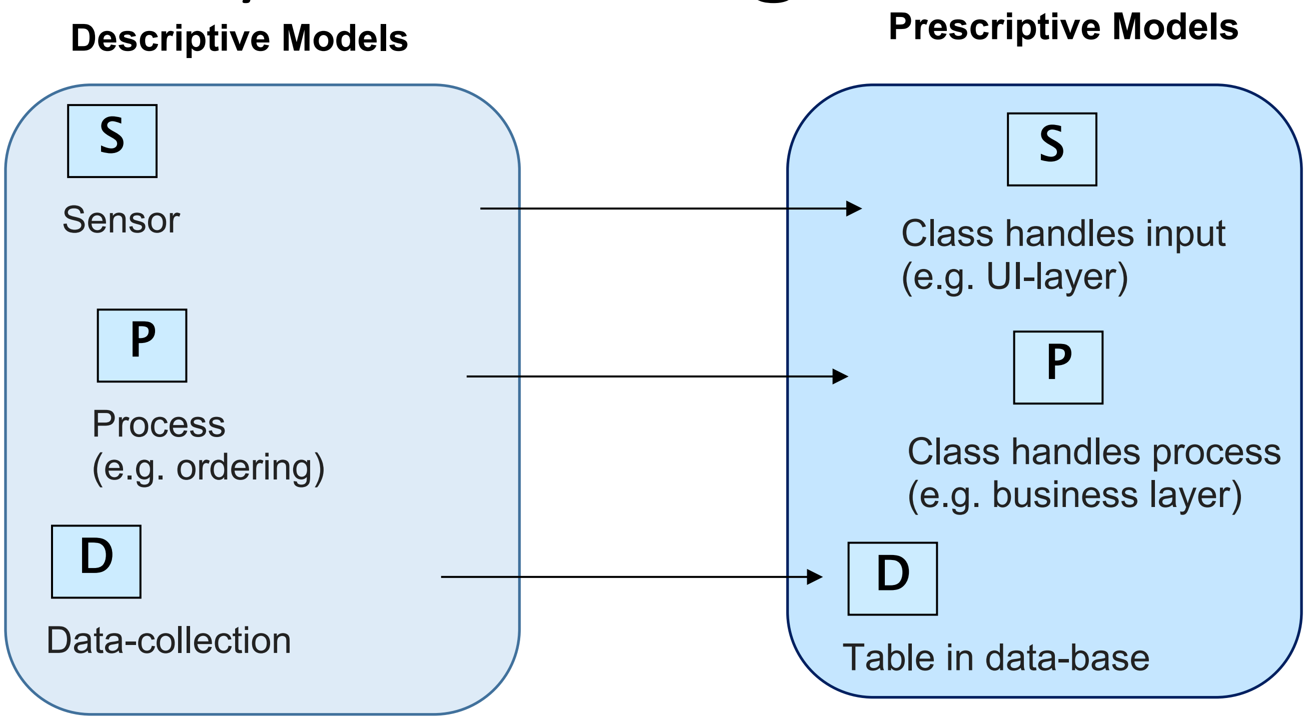
Solution Space/Domain

Components/Classes & their behaviour



These are elements of **construction**:
How will it work?

From Analysis to Design



Descriptive Models

S

Sensor

P

Process
(e.g. ordering)

D

Data-collection

Prescriptive Models

S

Class handles input
(e.g. UI-layer)

P

Class handles process
(e.g. business layer)

D

Table in data-base

These are elements of **analysis:**
What is there?

These are elements of **construction**:
How will it work?

Summary Part II

- Functional Decomposition vs Implementation Decomposition

- Functional Decomposition as the first step to analyse the system from the problem spaces

- Transition from Analysis (Problem Domain/Space) to Design (Solution Domain/Space)