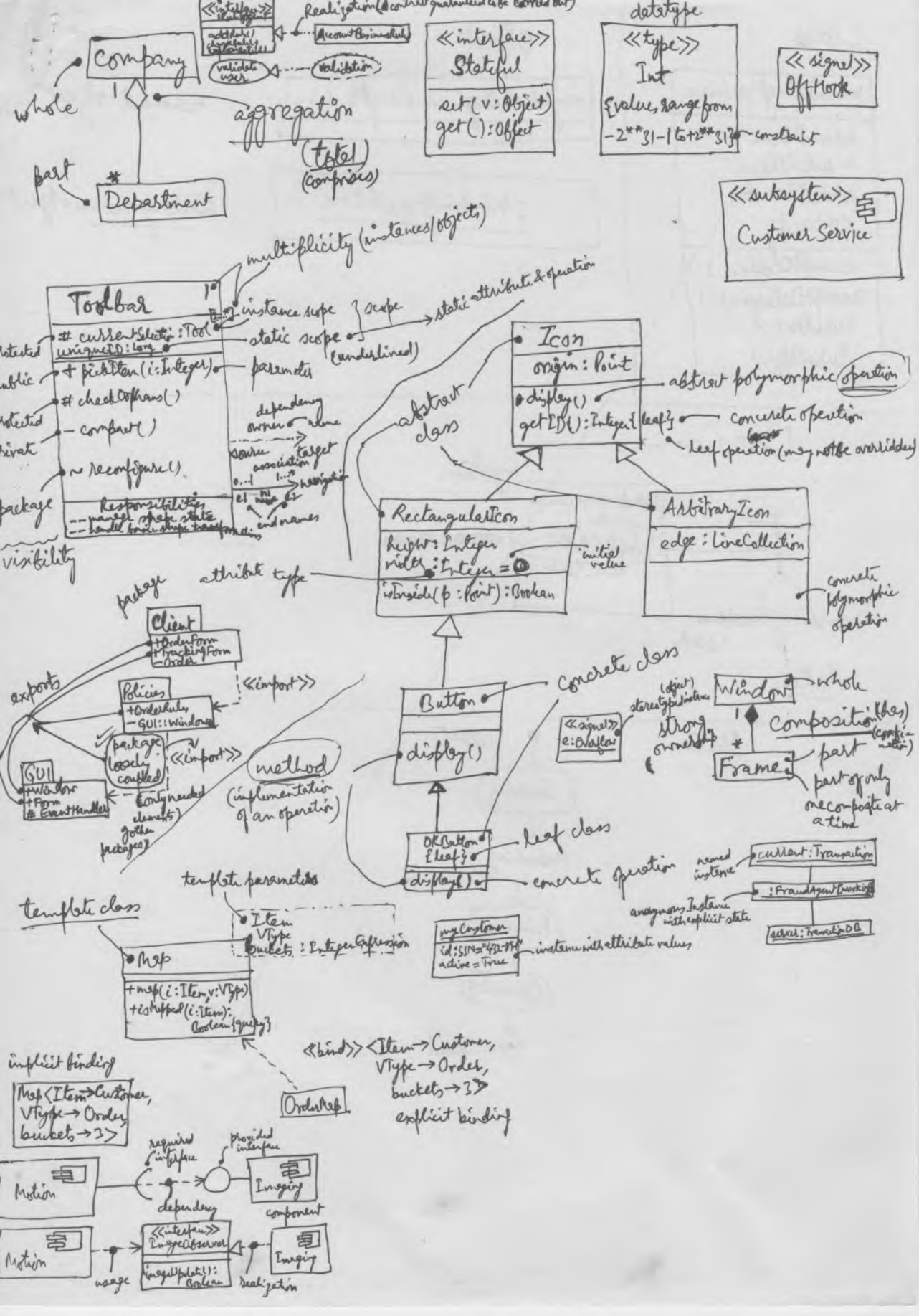
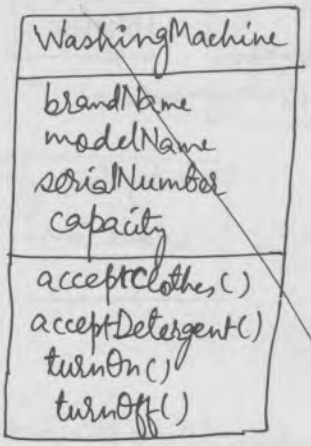
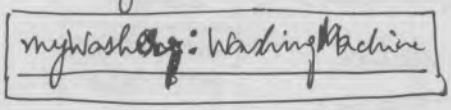

«interface» Stateful
set(v:Object)
get():Object
datatype
«type» Int
{value range from -2**31-1 to +2**31}
constraints
«signal» OffHook
Company
whole
aggregation
{ordered}
(comprises)
part
Department
«subsystem» Customer Service
Toolbar
multiplicity (instances/objects)
instance scope
static scope
scope
static attribute & operation
(underlined)
currentSelection : Tool
- uniqueID : Long
+ pickItem(i:Integer)
parameter
checkOrphans()
- compact()
+ reconfigure()
Responsibilities
visibility
package
protected
public
private
Icon
origin : Point
display()
getID():Integer
abstract polymorphic operation
concrete operation
leaf operation (may not be overridden)
abstract class
RectangularIcon
height : Integer
width : Integer = 0
isInside(p:Point):Boolean
initial value
ArbitraryIcon
edge : LineCollection
concrete polymorphic operation
attribute type
dependency
owner
client
source
target
association
navigation
end names
Client
+OrderForm
+TrackingForm
-Order
Policies
+OrderRules
-GUI::Window
package (loosely coupled)
«import»
exports
GUI
+Window
+Form
EventHandler
(only needed elements)
(other packages)
method
(implementation of an operation)
Button
display()
concrete class
«signal» e:OnFire
(object) state type instance
strong ownership
Window
whole
Composition (has)
(configuration)
Frame
part
part of only one composite at a time
OKButton {leaf}
leaf class
display()
concrete operation
template parameters
template class
Item
VType
buckets : Integer{version}
Map
+map(i:Item, v:VType)
+isMapped(i:Item): Boolean{query}
myCustomer
id : SIN="421-111"
active = True
instance with attribute values
current : Transaction
named instance
: FraudAgentWorking
anonymous instance with explicit state
record : TransInDB
«bind» <Item→Customer, VType→Order, buckets→3>
explicit binding
implicit binding
Map<Item→Customer, VType→Order, buckets→3>
OrderMap
Motion
required interface
provided interface
dependency
Imaging
component
«interface» ImageObserver
imageUpdate():Boolean
usage
realization

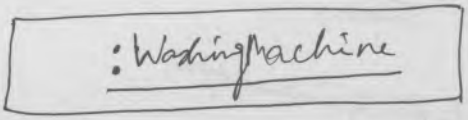
Class



Object

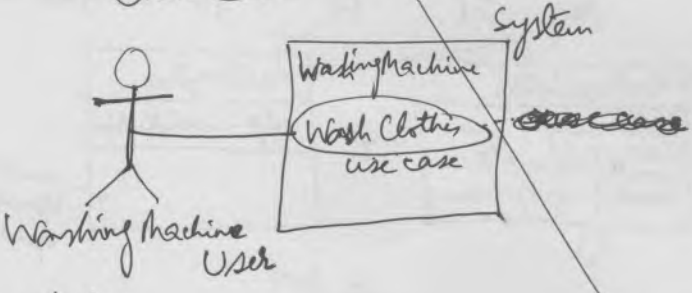


named object



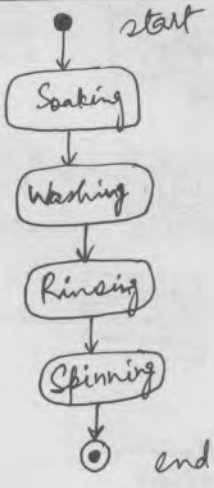
anonymous object

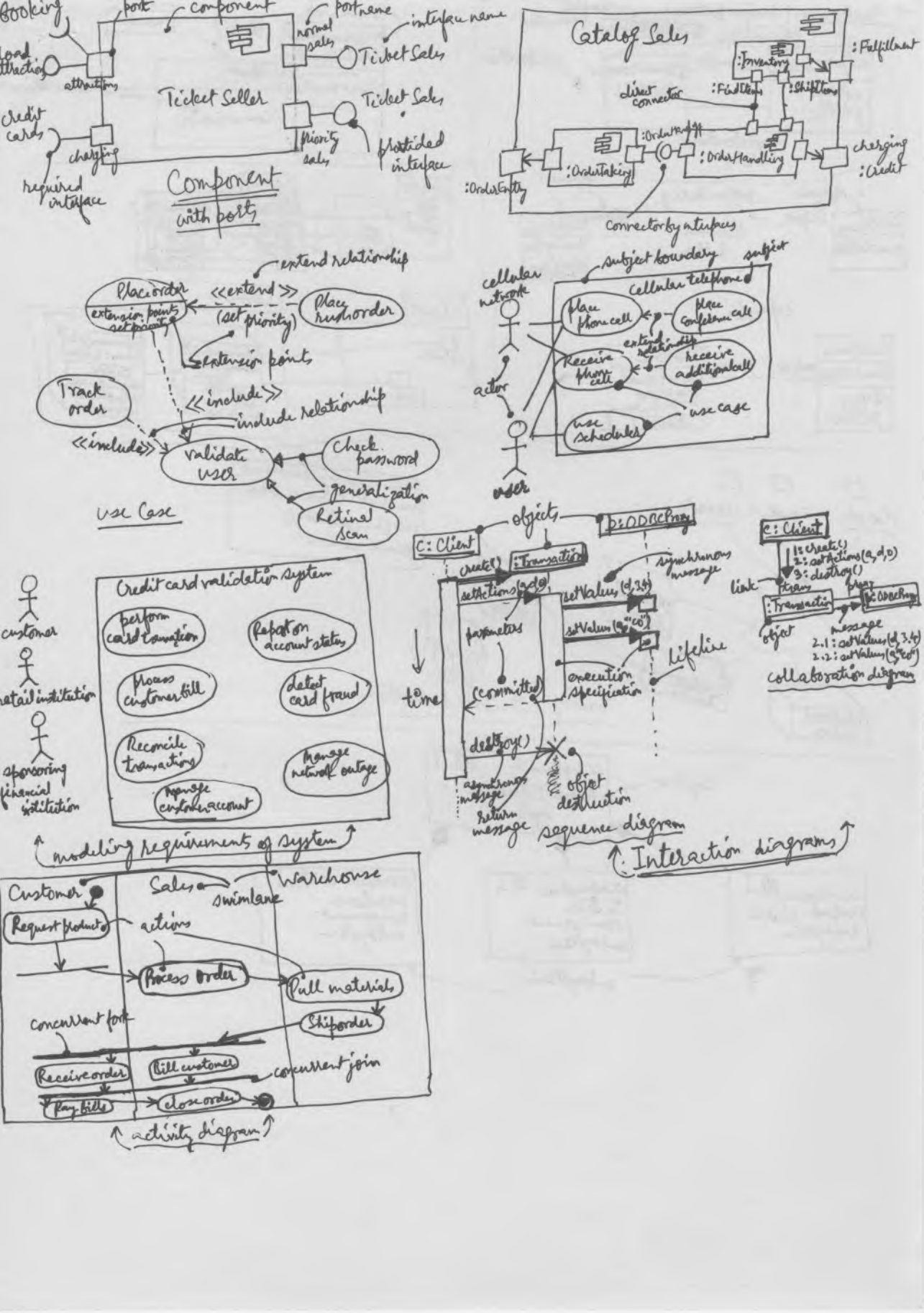
Use Case

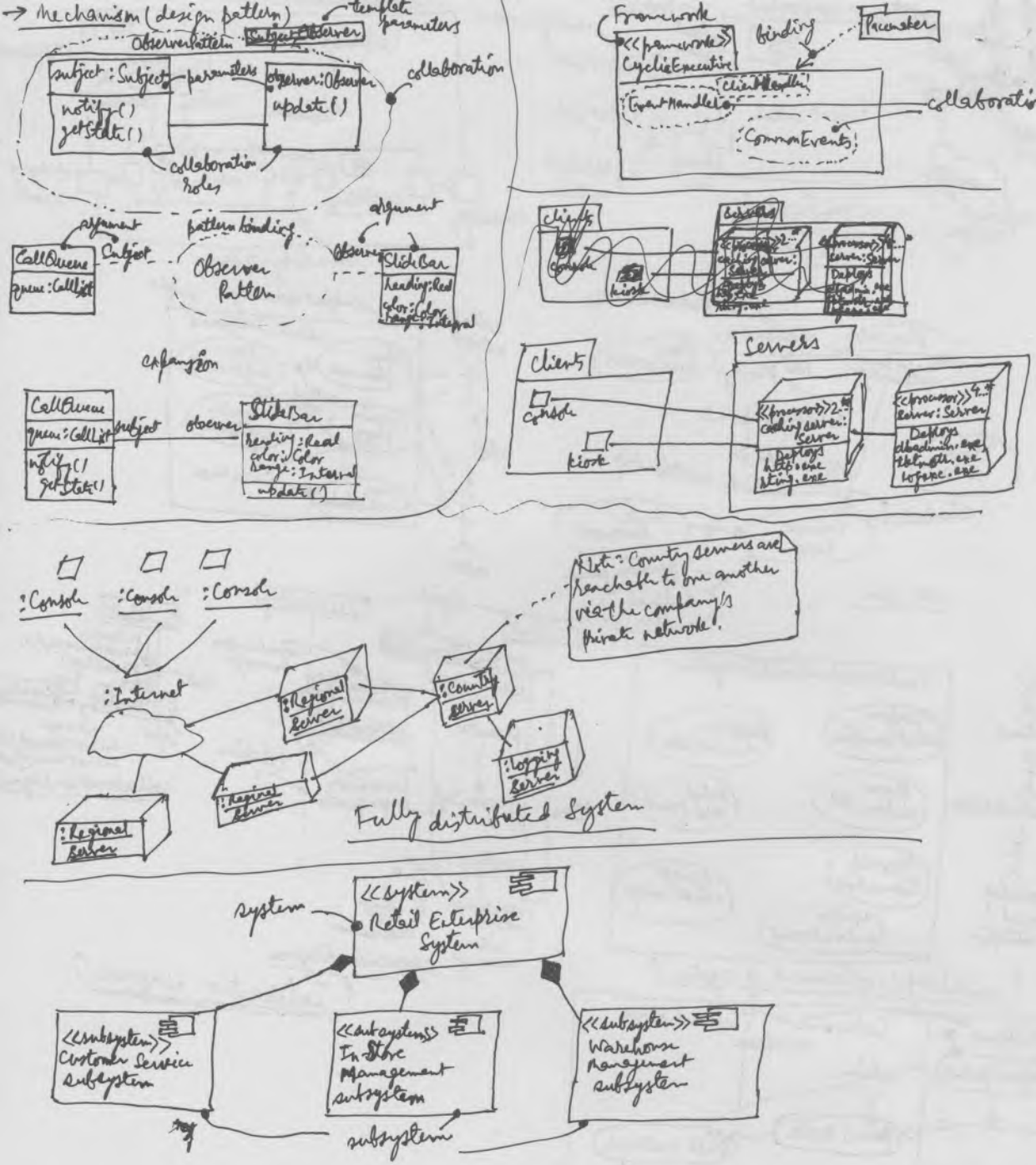


Actor

State

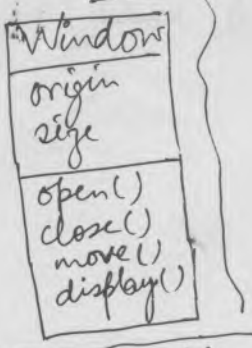



Booking
port
component
normal sales
port name
interface name
Ticket Sales
load attraction
attractions
Ticket Seller
Ticket Sales
credit cards
charging
priority sales
prioritized interface
required interface
Component with ports
Catalog Sales
:Inventory
:Fulfillment
direct connector
:FindItem
:ShipItem
:OrderProcessing
:OrderHandling
charging :Credit
:OrderEntry
:OrderTaking
Connector by interfaces
Place order
<<extend>>
extend relationship
extension point: set priority
Place rush order
extension point
Track order
<<include>>
include relationship
Validate user
Check password
generalization
Retinal Scan
use Case
cellular network
subject boundary
subject
Cellular telephone
Place phone call
Place conference call
extend relationship
Receive phone call
receive additional call
use case
use scheduled
actor
user
Credit card validation system
perform card transaction
Report on account status
process customer bill
detect card fraud
Reconcile transactions
Manage network outage
Manage customer account
customer
retail institution
sponsoring financial institution
modeling requirements of system
C: Client
object
D: ODBCProxy
:Transaction
create()
synchronous message
setActions(a,d,o)
setValues(d,3.4)
setValues(3,"CO")
parameters
lifeline
(committed)
execution specification
time
destroy()
asynchronous message
return message
object destruction
sequence diagram
c: Client
1: create()
2: setActions(3,d,0)
3: destroy()
link
:Transaction
D: ODBCProxy
object
message
2.1: setValues(d,3.4)
2.2: setValues(3,"CO")
collaboration diagram
Interaction diagrams
Customer
Sales
Warehouse
swimlane
Request product
actions
Process Order
Pull materials
Ship order
concurrent fork
Receive order
Bill customer
concurrent join
Pay bills
close order
activity diagram

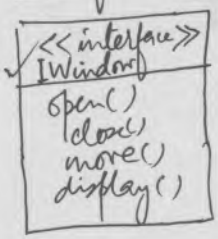

mechanism (design pattern)
template parameters
ObserverPattern
Subject
Observer
subject : Subject
notify()
getState()
parameters
observer : Observer
update()
collaboration
collaboration roles
CallQueue
queue : CallList
argument
Subject
pattern binding
Observer Pattern
Observer
SlideBar
leading : Real
color : Color
range : Interval
expansion
CallQueue
queue : CallList
notify()
getState()
subject
observer
SlideBar
leading : Real
color : Color
range : Interval
update()
Framework
<<framework>>
CyclicExecutive
binding
Presenter
ClientHandler
EventHandler
CommonEvents
collaboration
Clients
Console
kiosk
Servers
<<processor>> 2..*
CachingServer : Server
Deploys
HttpServer.exe
StringServer.exe
<<processor>> 4..*
Server : Server
Deploys
AdminServer.exe
HttpPathServer.exe
InfoServer.exe
:Console
:Console
:Console
:Internet
:Regional Server
:Regional Server
:Regional Server
:Country Server
:Logging Server
Note: Country servers are reachable from one another via the company's private network.
Fully distributed System
system
<<system>>
Retail Enterprise System
<<subsystem>>
Customer Service subsystem
<<subsystem>>
In-Store Management subsystem
<<subsystem>>
Warehouse Management Subsystem
subsystem

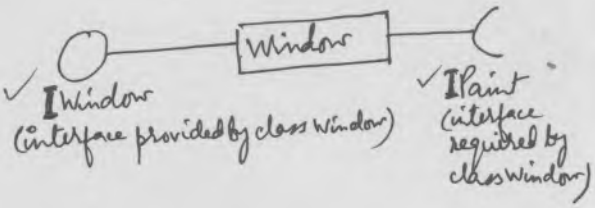
UML

Class

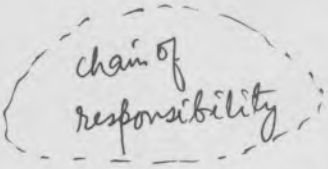


✓Interfaces



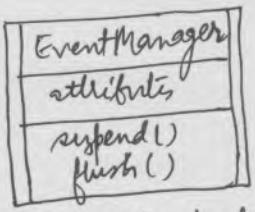


collaboration defines an interaction and is society of roles and other elements that work together to provide some cooperative behavior

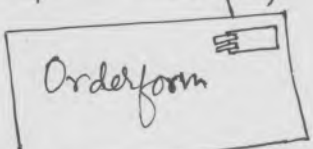


Use case is a description of sequences of actions that a system performs that yield observable results of value to a particular actor.

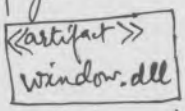
Active class whose objects own one or more processes or threads and therefore can initiate control activity.



Component (modular part of system design that hides its implementation behind a set of external interfaces)



artifact (physical and replaceable part of a system that contains physical information ("bits").



node (physical element that exists at run time and represents a computational resource, generally, having at least some memory and, often, processing capability)

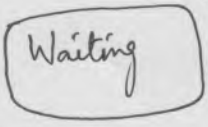
structural things (nouns of UML models)

Behavioral things (dynamic parts, verbs of models)

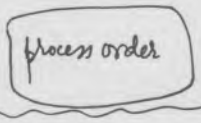
interaction (behavior that comprises a set of messages exchanged among a set of objects or roles within a particular context to accomplish a specific purpose)
(the focus is on set of objects interacting)

display operation →

state machine (behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events)
(focus is on life cycle of one object at a time)

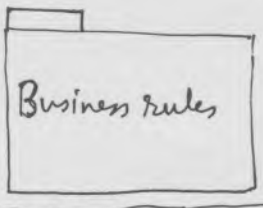


activity (behavior that specify the sequence of steps a computational process performs)
(focus is on flows among steps without regard to which object perform each step/action)



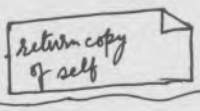
grouping things (organizational parts of UML models)
(boxes into which a model can be decomposed)

package (mechanism for organizing the design itself, as opposed to classes, which organize implementation constructs.) (purely purely conceptual, i.e., it exists only at development time.)



Annotational things (explanatory parts of UML models.)
(comments/notes)

note (attached to elements as remark/comment)



Relationships

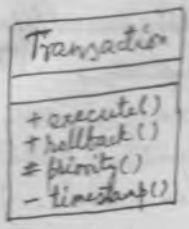
- - - - - - -> **dependency** (between two model elements / independent-dependent)

0.1 employer —— * employee **association** (among classes) **aggregation** (a special association, between a whole and its parts)

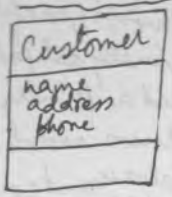
——▷ **generalization** (generalized element - specialized element relationship)

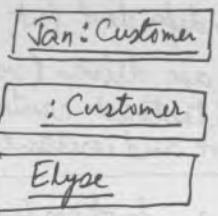
- - - - - - -▷ **realization** (between classifiers i.e. contract-guarantee

Adornments

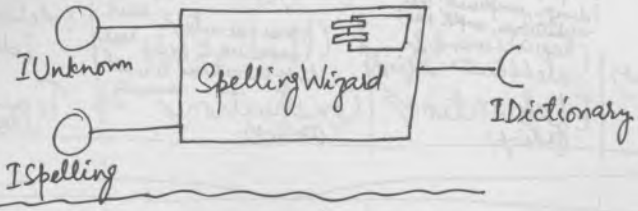


Classes and objects



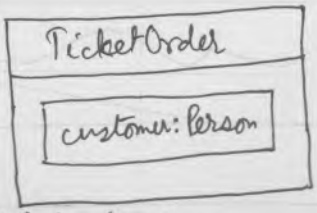


Interfaces and Implementations



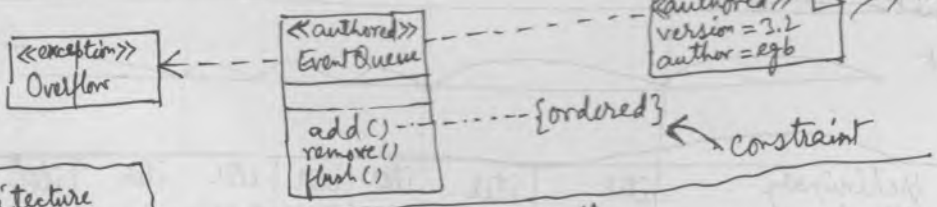
type and role

(type = declares the class of an entity, such as an object, an attribute, or a parameter)
(role = describes the meaning of an entity within its context, such as a class, component, or collaboration.)

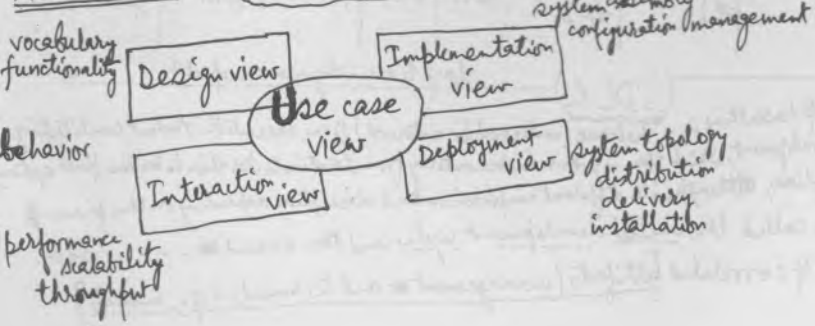


Part with role and type

Extensibility Mechanisms



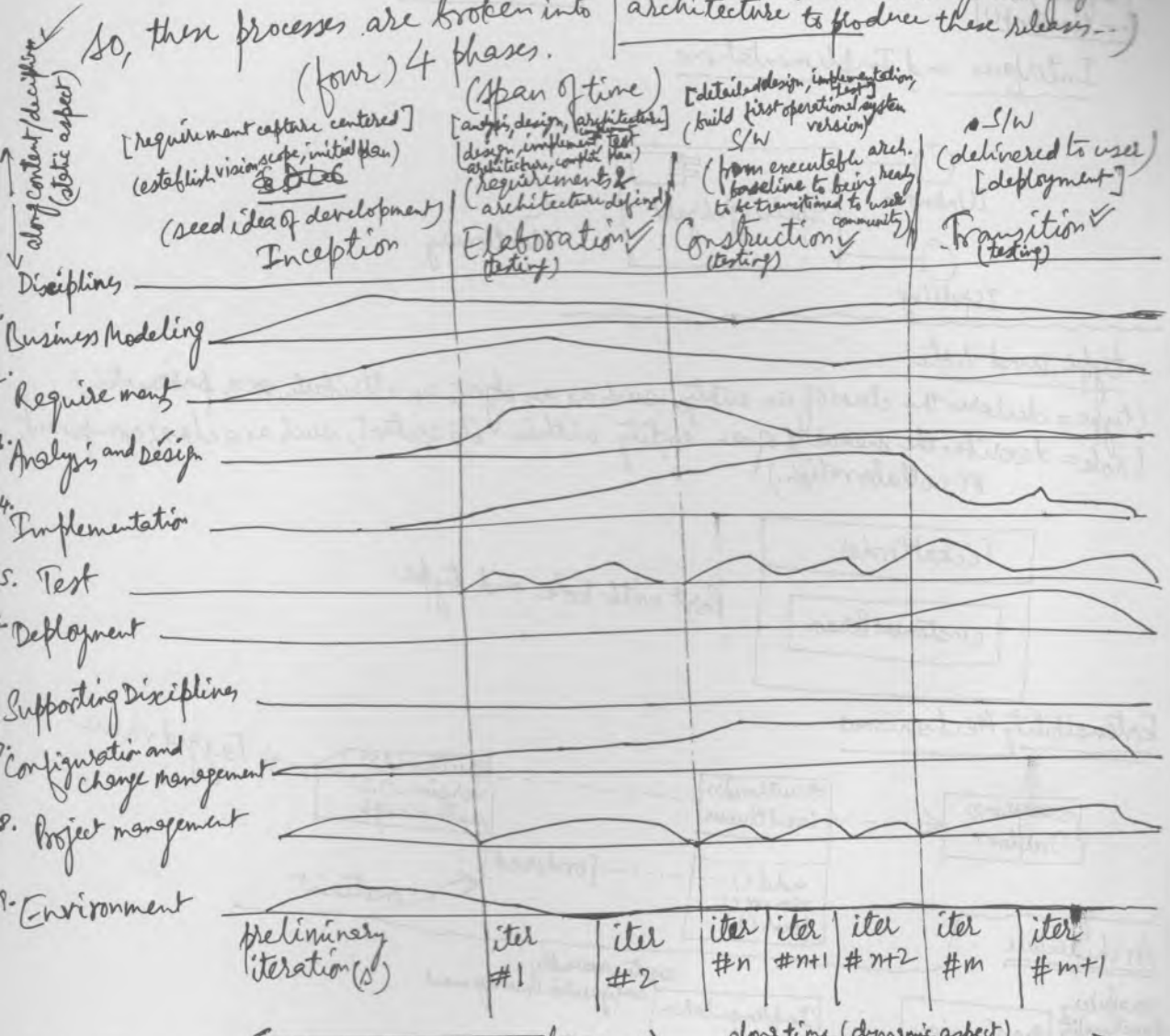
Architecture



SDLC (Software Development Life Cycle)

UML is largely process-independent but, to get most benefit from UML, a process should be
= use case driven (used as primary artifact for establishing desired behavior)
= architectural-centric (used as primary artifact for conceptualizing ...)
= iterative and incremental (process involves managing stream of executable releases and continuous integration of system architecture to produce these releases ...)

So, these processes are broken into (four) 4 phases.



(SDLC)

* A iteration is a complete development loop resulting in a release (internal/or external) of an executable product constituting a subset of the final product under development, which then is grown incrementally from iteration to iteration to become final system.
* each iteration goes through the various disciplines, although with different emphasis on each discipline depending on the phase.
* The first pass through the four phases is called the initial development cycle, and the second ---- soon.
* captured within each discipline is a set of correlated artifacts (management and technical) e.g., models and activities.

Inside UML and Rational Rose (UML seminar)

① Activity diagram – (modeling business workflow)

open (use case view) → ~~use/class diagram~~ → new/activity diagram/--

(to create activity diagram) ⇒ swimlane for activities

② Create Use-Cases (to model business process)

↳ use case view/main (use can diagram)/

| you may enter informations in the documentation and use can specification

③ move to Logical view : (to create object/class)

new/class/---

④ use case view/(project "trade")/new/sequence diagram (to the trade class)/

logical view/(project "trade")/new/operation "newtrade"

⑤ Component view (physical modeling):

main/---

we can do round-trip engineering:

select the class/Java/Generate Java

↓then

select the class/Java/Browse code

↓then

tools/Java/Browse segment Java

choose Java reverse Engineer

⑥ Deployment view (nodes)

⑦ tools/model integrator/Contributors/(select models to compare) x.mdl y.mdl

compare(mods) → switch to merge(mods)

options

merge/contributors

⑧ tools/web Publisher/(choose views + documents)/---

VC++: (ATL, MFC)

① tools/Visual C++/update model from code/add VC++ component---/

② logical view/("trade" project)/

component view/("trade" project)/new/add method---/(parameters [in]---, [in] long---, [out, retval] long* y)---/

for ATL

" " " " " " /(interface) → expand/new ATL object/---

* model assistant is an important help

Rose ~~xxxxxxxx~~: data modeler demo:

① Component view / data modeler / new / database / (choose target of database)

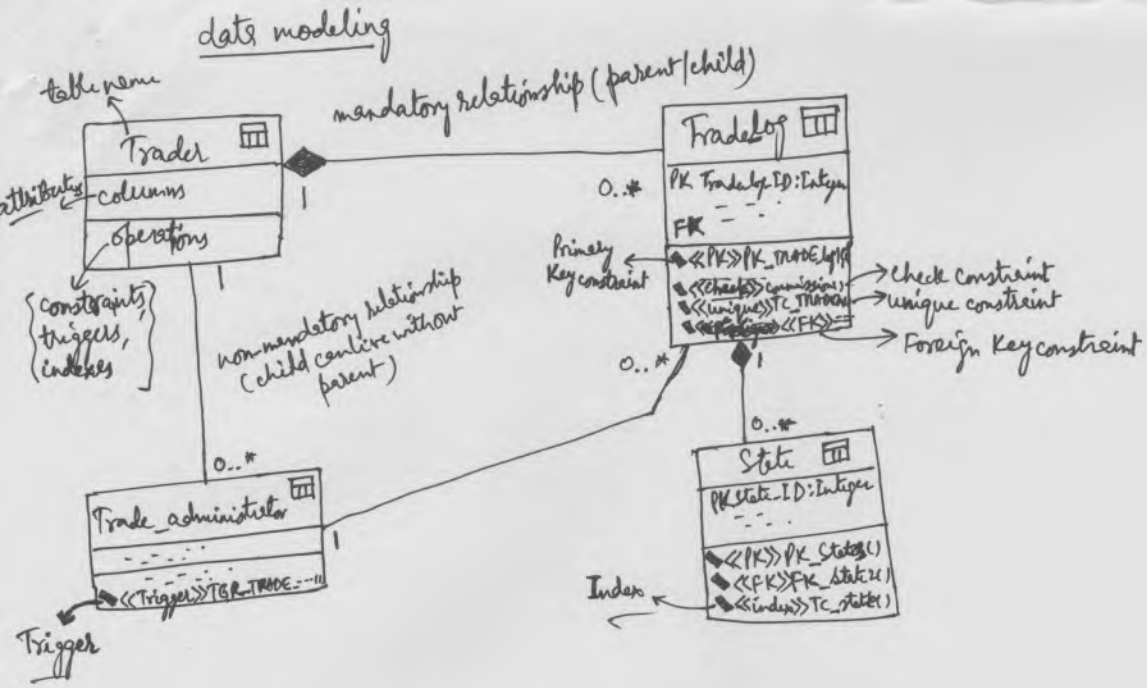
② select the package to transform / (right click) Data Modeler / transform to data modeler /

③ Scheme (project) / data modeler / new / data model diagram / ⟹ (data model diagram)

④ Scheme (project) / data modeler / transform to object model / ⟹ (changing the model)

⑤ " / " / forward engineering ⟹ (new database)

data modeling



SQL –
- DML (changing rows)
- DDL (changing DB structures (= metadata)

ODQL – (Object Database Query Language)
- like C++ or Java
- queries { <object> FROM <object> ALONE WHERE
  <object>.<column> == <expression> ; }
- change data { <object>.new(<expression list>);
  <object>.change(<expression list>);
  <object>.delete(<expression list>); }

* By creating index, we make much smaller view of a table (physically smaller table)
  (a few columns)
* avoid too many indexes, because (problem of maintenance, DML, locking issues)
  * = Do not remove FK indexes

Composite indexes :–
— fewer columns the better
— not small enough relative to table
— don't have to index all columns in a composite

* use integers, short, fixed length strings for indexing

* Indexing type = –
- BTrees (OLTP, warehouse,
- Bitmaps (...
- Clustering (joins, ...
- Materialized views (

— Good & Bad Tricks —
- copying columns
- summary columns to parents
- separate active from inactive (data warehouses)
- separate heavily and lightly accessed columns (similar to 4NF)
- Focus tuning on heavily used entities
- Avoid views (logical overlays)
- use materialized views (data warehouse, replication)
- application level caching (static data,)

* striping = splitting file into pieces smaller
* mirroring = simply a copy of Data
* RAID array = (most common configuration = RAID 0, RAID 1, RAID 0+1, RAID 5)
  striping, mirroring, striping & mirroring, mirroring minimized with parity only

* Replication of DB – (among masters and slaves DBs)
* clustering –

Data modeling (=design)

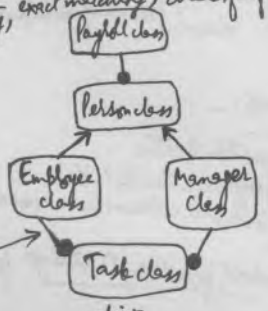
- create data structures
- from business process
- and raw data
- as efficient as possible

www.oracledbaexpert.com/oracle/papers/Normalization.html
/DeNormalization.html
/The Very Basics of Datawarehouse...
/Object Vs Relational...

Data Modeling SW ⇒ ERwin, Normalizing, Modelija

Object Database – modeling objects (=classes, attributes, method, deals complexity, exact matching, not suitable for reporting, dreadful for reporting)

- three dimensional (spherical structure)
- links between objects (objects can point to other objects by pointers, parent-child relationship) = collection inclusion
- High speed (=highly complex data sets.) (= individual object access. * Not data set groupings)
- exists object concepts (=like class, object, inheritance, multiple inheritance, data abstraction, encapsulation, reuse code, black boxing, object DB Query language like SQL)

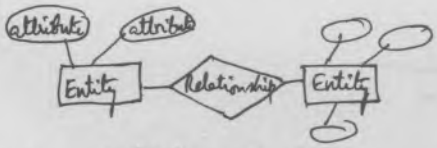


Object-Relational Database =

= Combine relations and objects (binary objects = large text items, multimedia, unstructured storage)

– does allow some object design

E-R diagram –



UML database model –

Relational Data model –

– Normalization (1NF, 2NF, 3NF, 4NF, 5NF) = duplication removal, granularization, reduction of complexity, (not many-to-many → many-to-one → many-to-many → separate NULL values)

– Referential integrity (Primary & foreign keys) by triggers, DB events

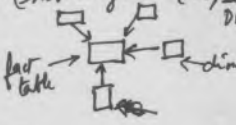
– keys (surrogate keys, alternate keys)

– granularity high ≠ SQL (code runs too slow)

Normalization = granularity

Datawarehouse data model – (Denormalized) = (SQL code runs faster) = reduces no. of joins,

– star schema



– snowflake

* Client server DB
* OLTP (DB for intranet + internet internal users + outside users)