

Model-Driven SOA: The BPM–SOA–MDA Connection

Marc J. Balcer



What is SOA?

- “A system for linking resources *on demand*. In an SOA, *resources* are made available to other participants in the *network* as independent services that are accessed in a standardized way. This provides for more *flexible* loose coupling of resources than in traditional systems *architectures*.”

– Looselycoupled.com

Another View



- “SOA is concerned with the *independent* construction of services which can be *combined* to realize meaningful, higher level business processes within the *context of the enterprise.*”

—Mike Rosen
Cutter Consortium

Another View



- “We define a *service oriented architecture* as an software architecture for *building* applications that implement *business processes* or services by using a set of *loosely-coupled black-box* components to define a well-defined *level of service*”

—Judith Hurwitz, *et.al.*,
SOA for Dummies

Another View



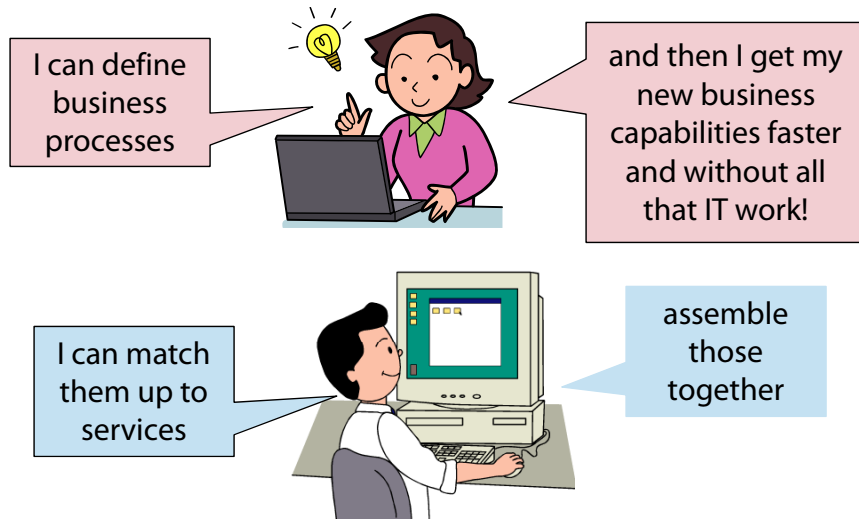
- SOA is an extension of the software as a service concept to the internal IT departments of companies. Both convert software from unfriendly, expensive lumps to easy-to-use services, available via a Web interface.
 - Fool.com (The Motley Fool)

SOA is...

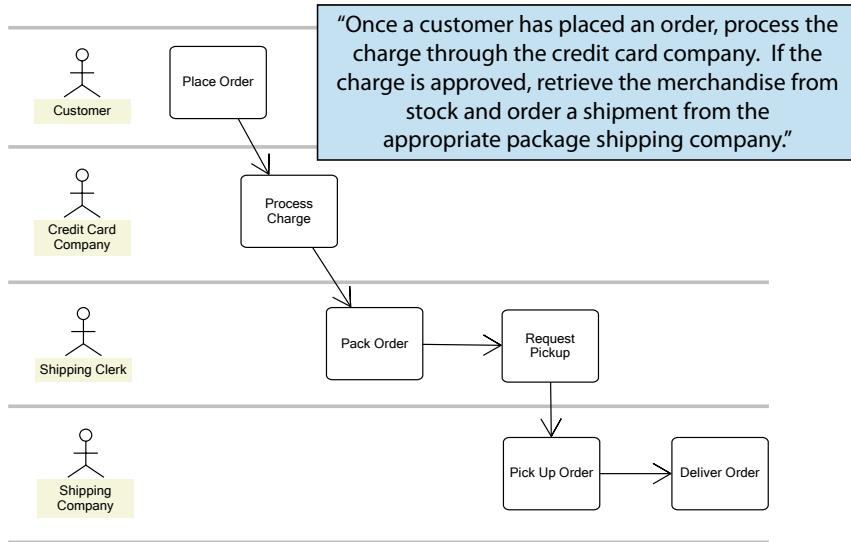


an *assembly/manufacturing* approach to creating systems for business in terms of *higher-level concepts* using *independent components*.

So what SOA means to business...

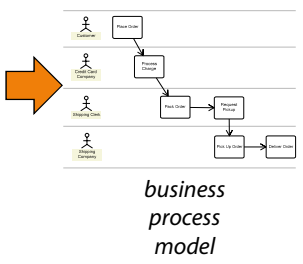
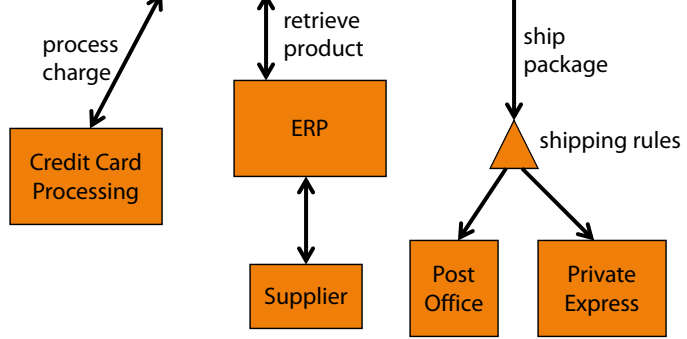


BPM



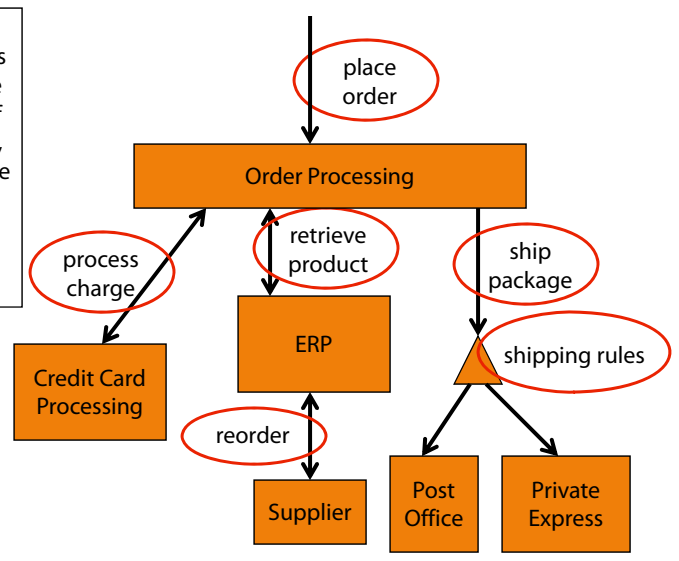
SOA and BPM (Business Process Modeling)

"Once a customer has placed an order, process the charge through the credit card company. If the charge is approved, retrieve the merchandise from stock and order a shipment from the appropriate package shipping company."

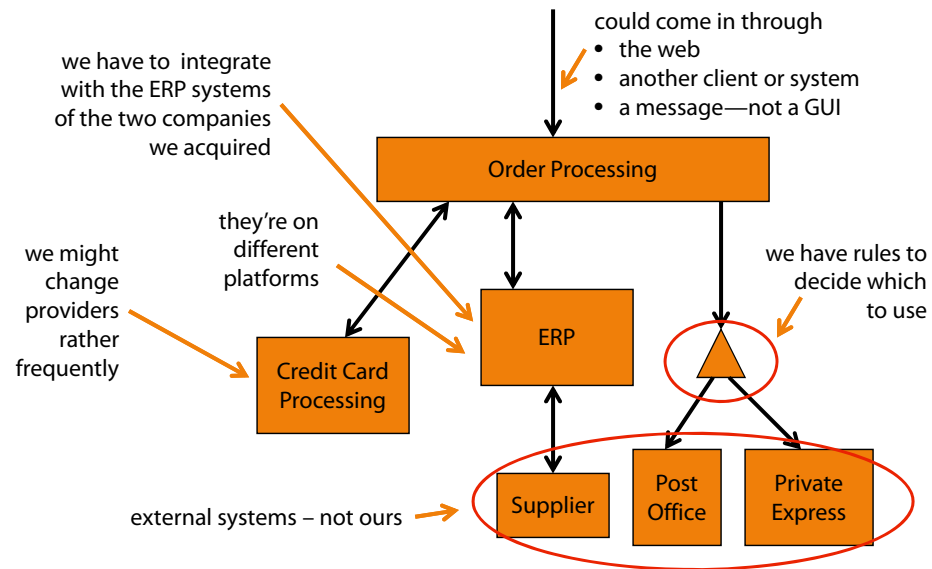


Service Example

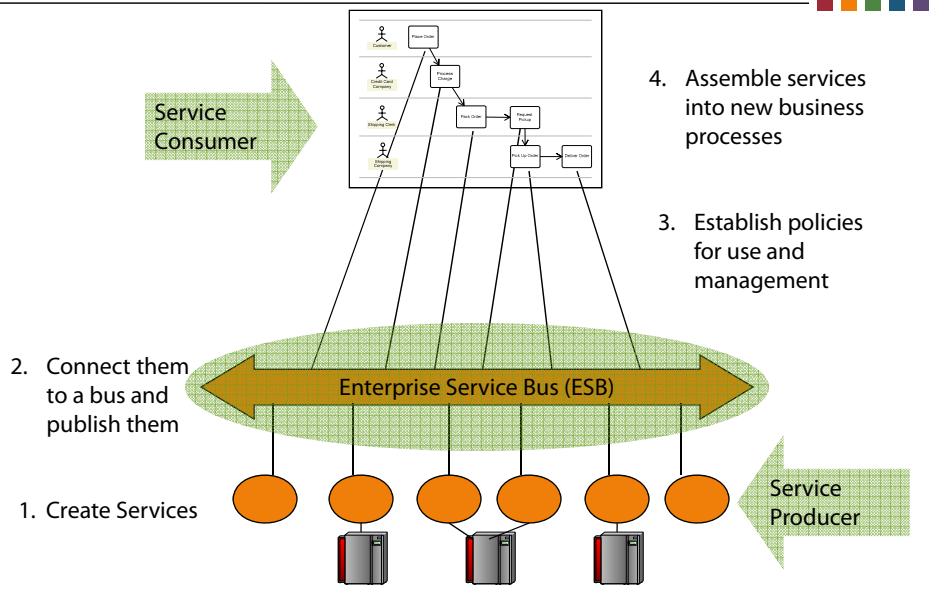
"Once a customer has placed an order, process the charge through the credit card company. If the charge is approved, retrieve the merchandise from stock and order a shipment from the appropriate package shipping company."



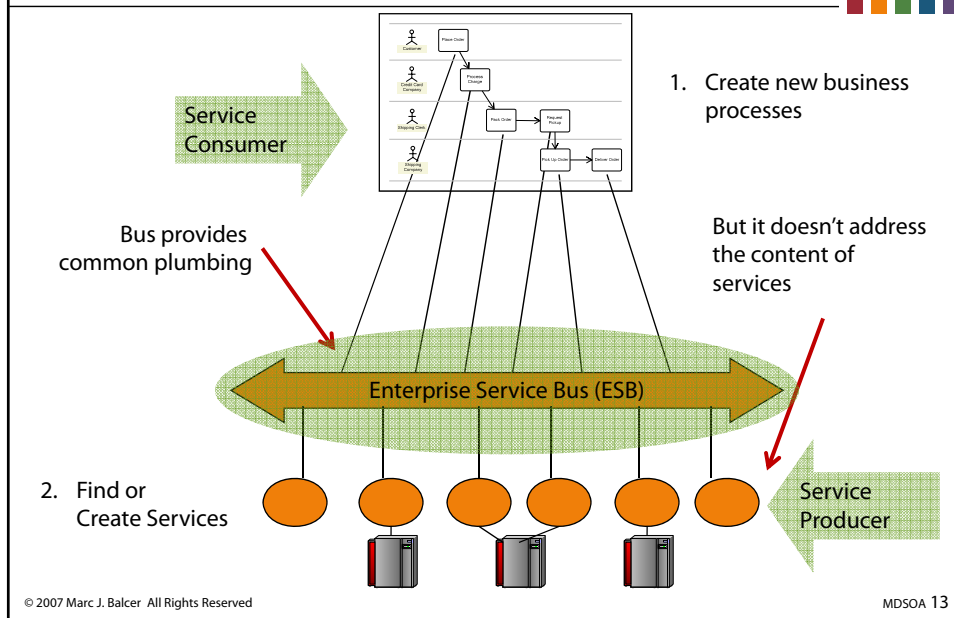
SOA What? (How is this different?)



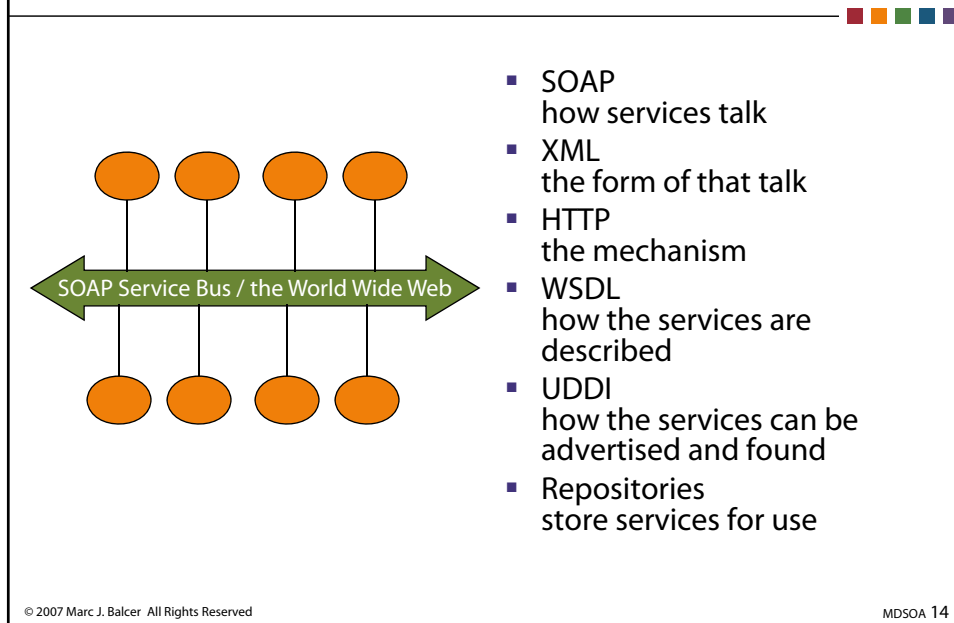
Technology to the Rescue?



"But really, it's so easy!"



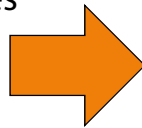
It's more than web services...



Technology ≠ SOA



But just having
these technologies
doesn't mean
you have SOA



- SOAP
how services talk
- XML
the form of that talk
- HTTP
the mechanism
- WSDL
how the services are described
- UDDI
how the services can be advertised and found
- Repositories
store services for use

"Architecture"



Abstractions that help
manage the complexity
inherent in system design

“Oriented”



- Object-Oriented
- Compliance-Oriented
- Performance-Oriented
- Search-Oriented
- Pattern-Oriented
- Delegation-Oriented
- Service-Oriented

Basic
organizing
principle
(of the architecture)

(put “Oriented Architecture” into Google)

“Service”



The encapsulation of a
cohesive set of capabilities,
expressed as an interface
decoupled from its
implementation

Service Example



Shipping Service
Shipment Confirmation
Request Quote Request Shipment Get Status Cancel Request

Interface Points

data
(documents)

operations

Good Services

Shipping Service
Shipment Confirmation
Request Quote Request Shipment Get Status Cancel Request

- Business –Focused
- Coarse-Grained
- Technology-Independent
- Interface does not imply implementation

Why Architecture for SOA?



~~What's a Service Oriented Architecture?~~

~~What technologies do I need to know?~~

the right usable and re-usable
How do I build services?
^

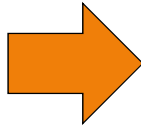
Architecture Goals



Shipping Service
Shipment Confirmation
Request Quote Request Shipment Get Status Cancel Request

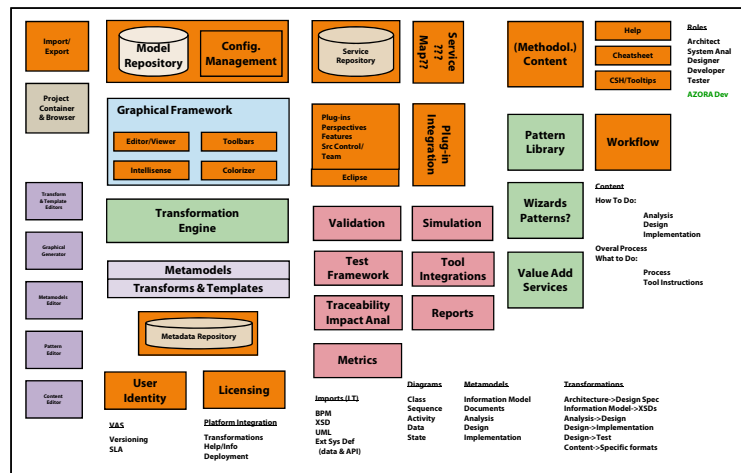
- How to design (create) services
- How to use services
- How to use services to create services

Five Key Practices

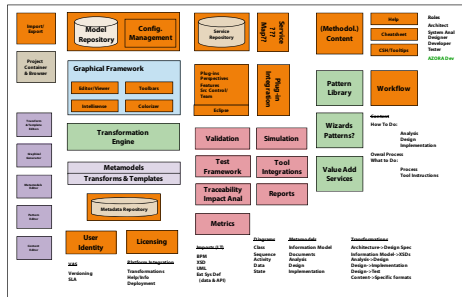


- Multidimensional Architecture
- Formal Modeling
- Subject-Matter Partitioning
- Translation of Models (MDA)
- Tolerance for Incompleteness

"Boxitecture?"

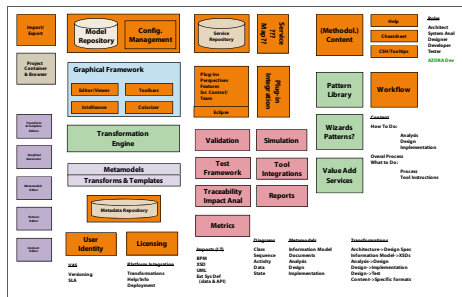


"Boxitecture"



- Tries to capture everything
- Tries not to miss anything
- Much more elegant than reality
- Creates veneer of organization
- Great slideware

"Boxitecture?"



- Can you build from this?
- Is this actionable?
- Can you get your
 - reusable services
 - vendor independence
 - complexity management
 - business-drives-IT

Architectural Views



Conceptual
View

Module
View

Execution
View

Code
View

from Hofmeister, Nord, Soni. *Applied Software Architecture*

Architectural Views



Logical
View

Development
View

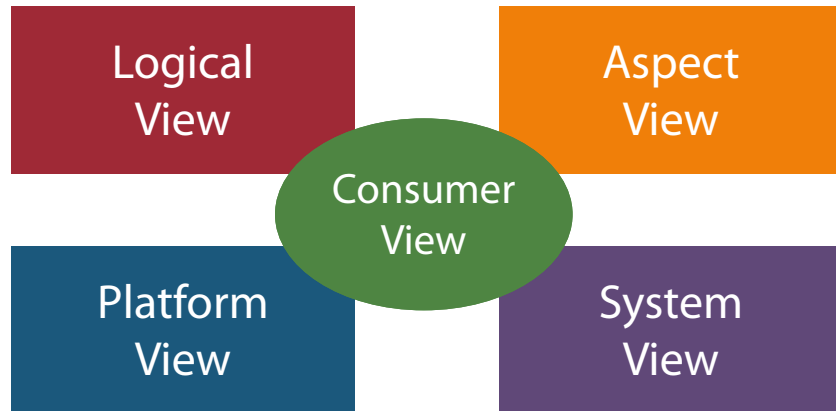
Scenario
(Use Case)
View

Process
View

Physical
View

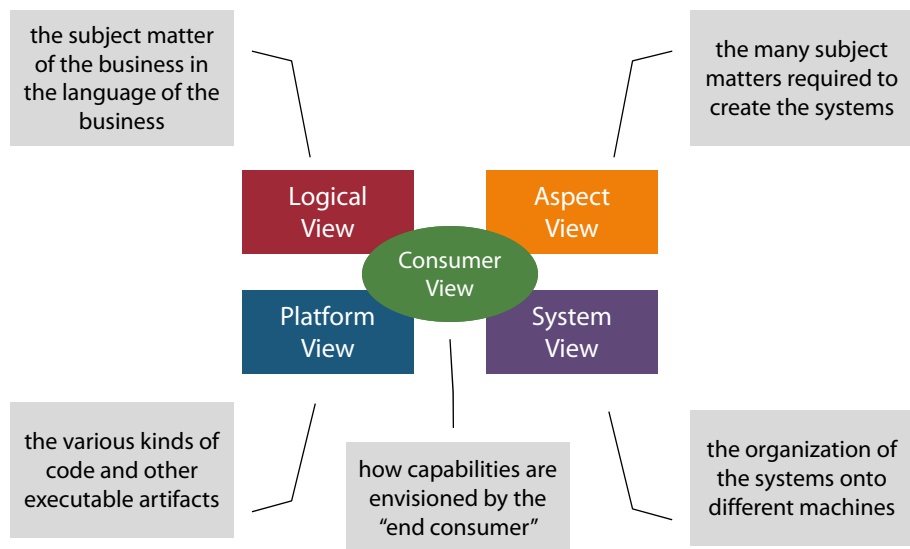
from Krutchen, "Architectural Blueprints: The 4+1 View..."

Architectural Views

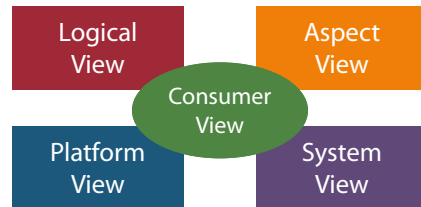


from Krutchen, "Architectural Blueprints: The 4+1 View..."

Multiple "4+1" Views

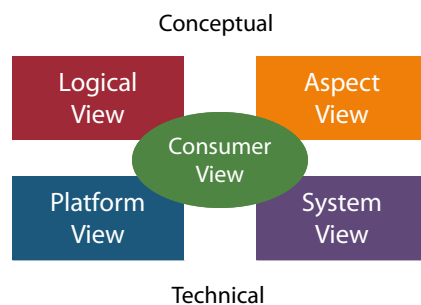


Which view is the architecture?

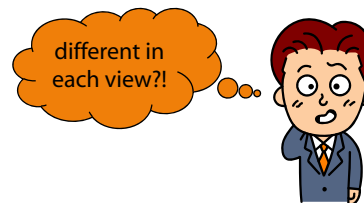


- All views are accurate
- Views are interrelated but
 - One view does not lead to another
 - One view is not an elaboration of another
 - No one view is primary

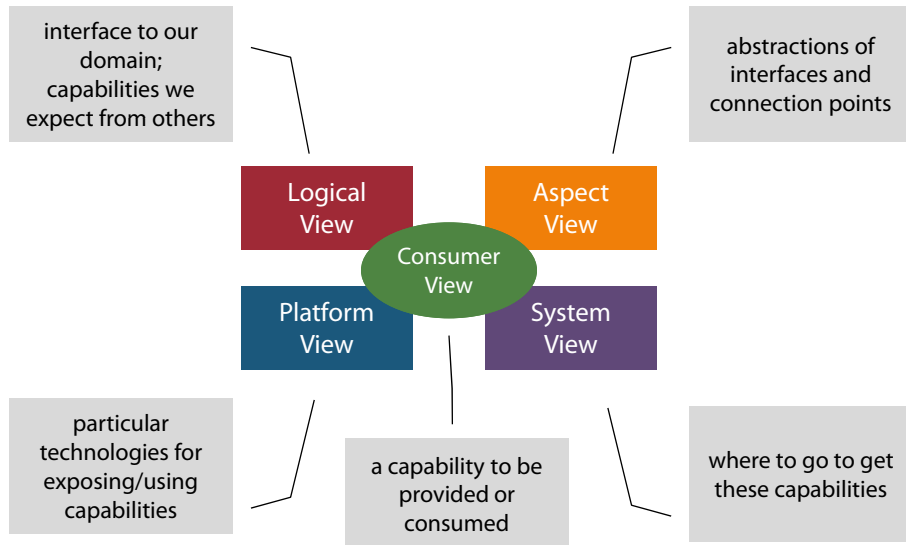
Multiple "4+1" Views



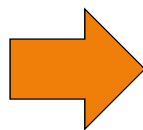
- Services are the basic organizing principle
- Services appear in each view
- Definition of "Service" is different in each view



Different Meanings of "Service"

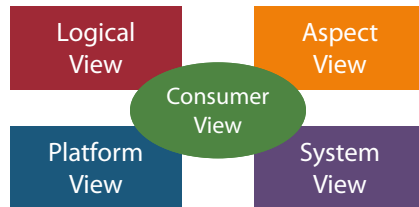


Five Techniques



- Multiple ("4+1") architectural views
- Formal Modeling
- Subject-Matter Partitioning
- Translation of Models (MDA)
- Tolerance for Incompleteness

Formal Modeling



- Each view has a distinct set of models
- Model each view independently
- Connect models to produce results

"Model"



Sketch	Blueprint	Executable
test of an idea	key properties of what's to be built	key properties of what's to be built
informal	advisory	exact
not delivered	delivered as a plan for construction	transformed into the product
not maintained	not maintained or maintained as historical record	maintained as source for the product

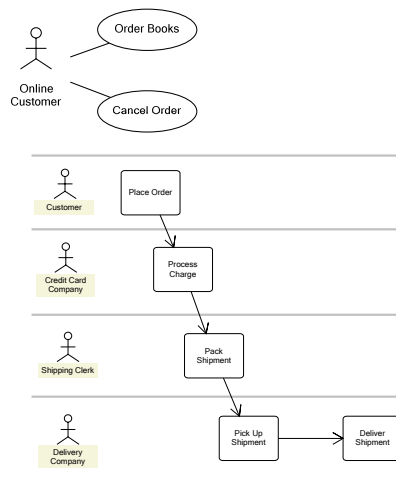
from Mellor, "The Bogus War"

Good Architectural Models



Meaningful	defined semantics
Usable	can do something with them
Actionable	they are part of a process
Reproducible	same problem leads to the same models
Evaluatable	can determine if they meet the needs
Verifiable	can test them

Consumer View



Content

- Use Cases
- Business Process Models
- Scenarios (single traces through the BPMs)

Produced by

- People with the requirements

Why it overlaps the other views



It expresses logical business functionality

It might express that functionality in terms of non-business things (e.g. GUI screens, data tables)

Consumer View

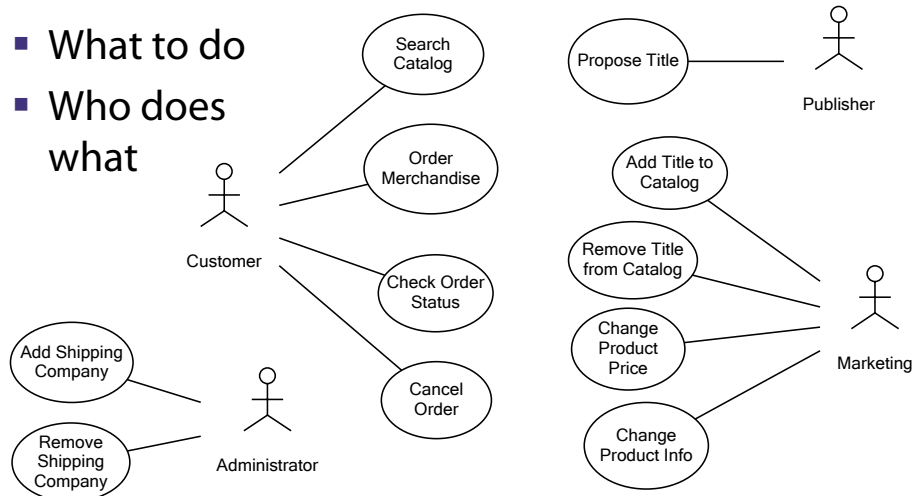
It might be expressed in terms of the software structure

It might be expressed in terms of different systems or system components (e.g. queues)

Use Cases

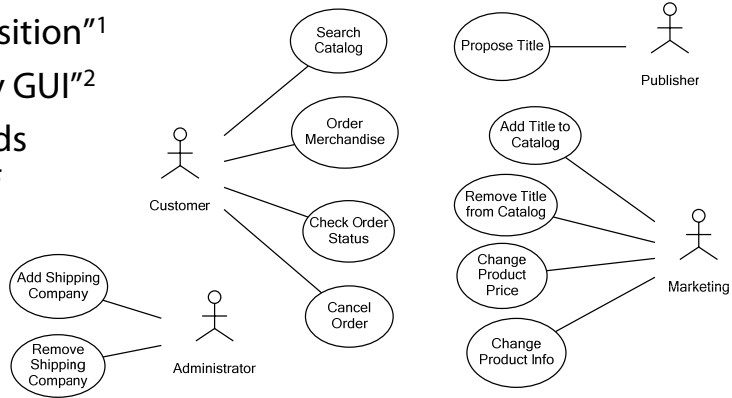


- What to do
- Who does what



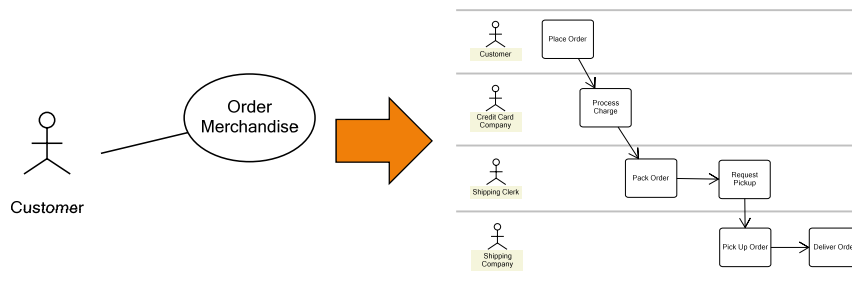
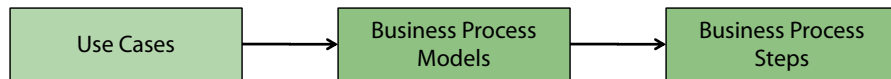
Use Case Traps

- “Abuse by decomposition”¹
- “Abuse by GUI”²
- 1000 words instead of a picture

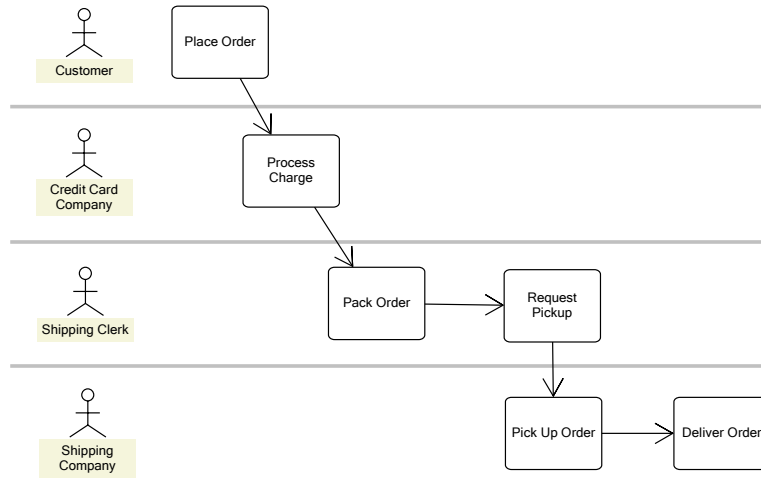


^{1,2}from Fowler, “Use and Abuse Cases...”

BPMs are the “Flow of Events”

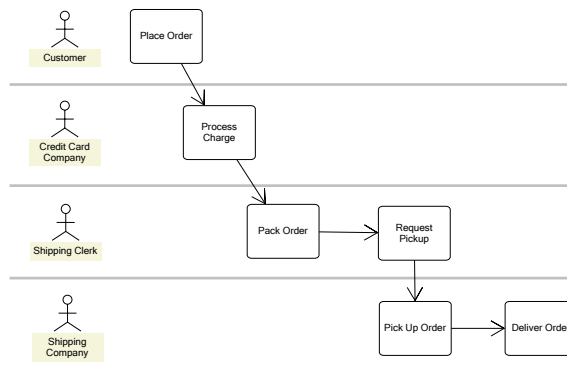


Business Process Model



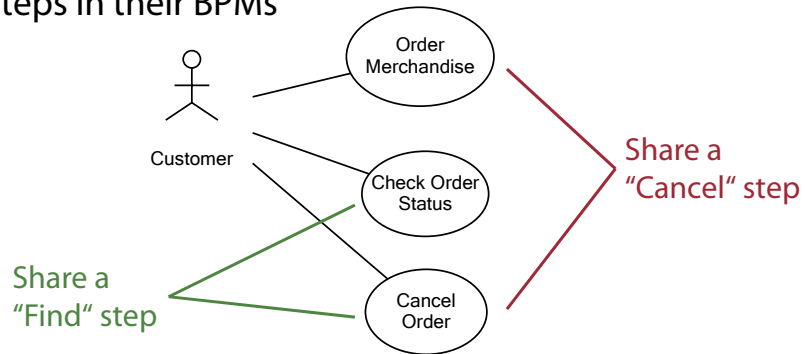
One BPM → Many Scenarios

- Coloring atop the BPM
- Coordinate the main and alternate flows

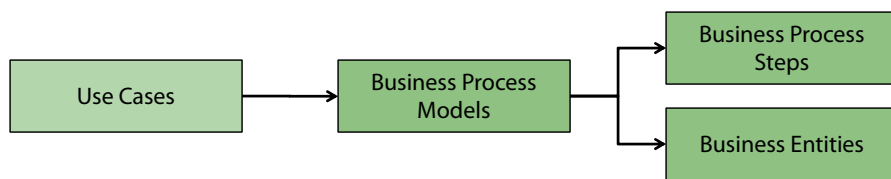


Process Reuse

- Multiple use cases often share process steps in their BPMs

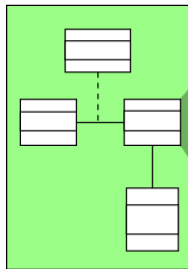
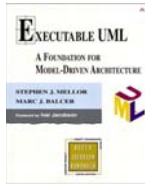


Process so far...

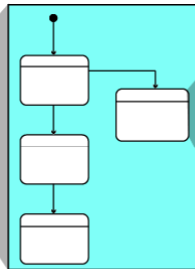


Now we need to actually create the services and documents to realize the business process steps and business entities.

Logical View



information model

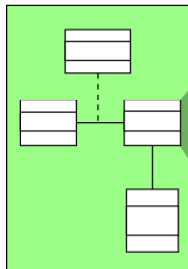


state model

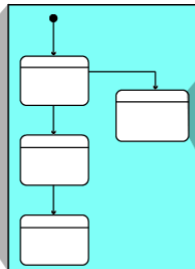
```
// Create a credit card charge and
// submit it to the credit card company
generate makeCharge (
  accountNumber :
    rcvd_evt.accountNumber,
  billingAddress: rcvd_evt.billingAddress,
  cardExpirationDate :
    rcvd_evt.cardExpirationDate,
  cardholderName :
    rcvd_evt.cardholderName,
  chargeAmount : self.totalValue,
  order: self)
to CreditCardCharge creator;
```

process model

Extended Logical View



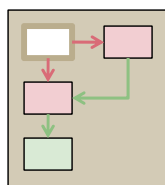
information model



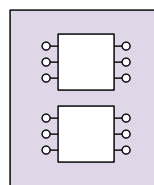
state model

```
// Create a credit card charge and
// submit it to the credit card company
generate makeCharge (
  accountNumber :
    rcvd_evt.accountNumber,
  billingAddress: rcvd_evt.billingAddress,
  cardExpirationDate :
    rcvd_evt.cardExpirationDate,
  cardholderName :
    rcvd_evt.cardholderName,
  chargeAmount : self.totalValue,
  order: self)
to CreditCardCharge creator;
```

process model

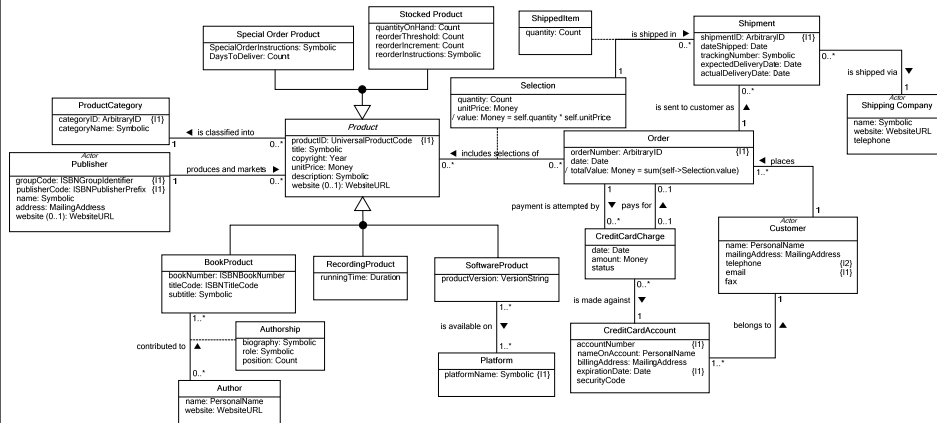


documents

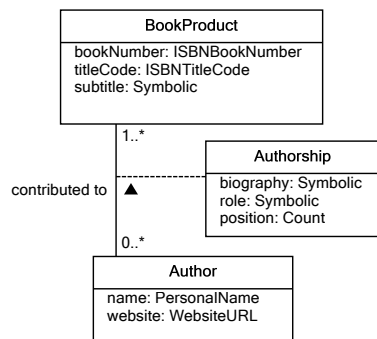


services

Information Model



Key Components



Classes

- abstractions of the real-world entities

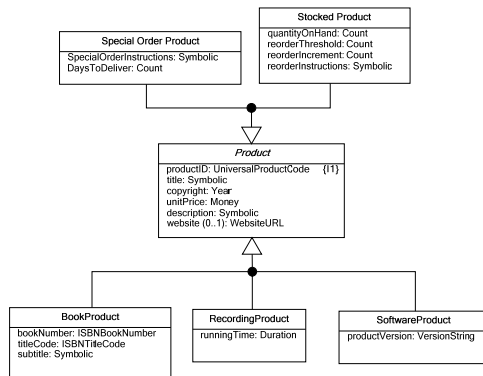
Attributes

- characteristics of those entities

Associations

- relationships between the entities

Specialization



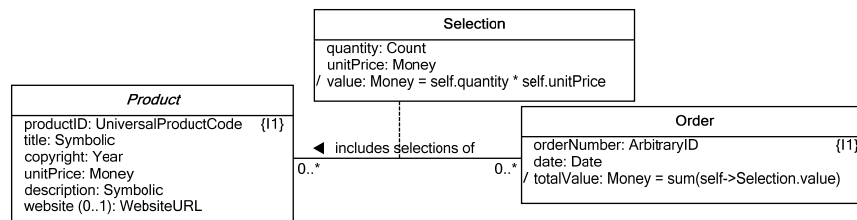
Explicitly show

- what's common
 - what's different
- “Close to “ inheritance
- show the real organization of things in the real world
 - not constrained by implementation technologies

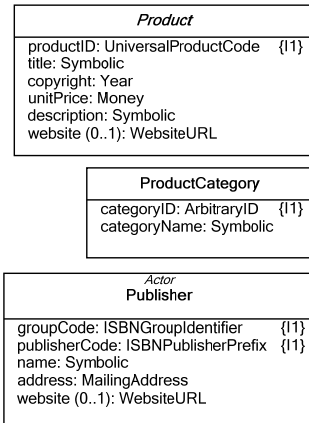
Derived Attributes

Calculations based upon values of other attributes

- in the same object
- in related objects

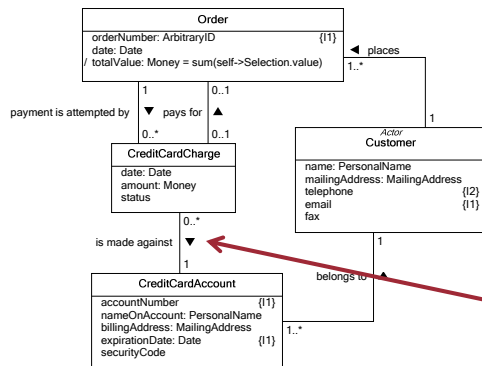


Identifiers



- Combinations of attributes and relationships that uniquely identify an instance
- Similar to relational concept of a “key”

Constraints

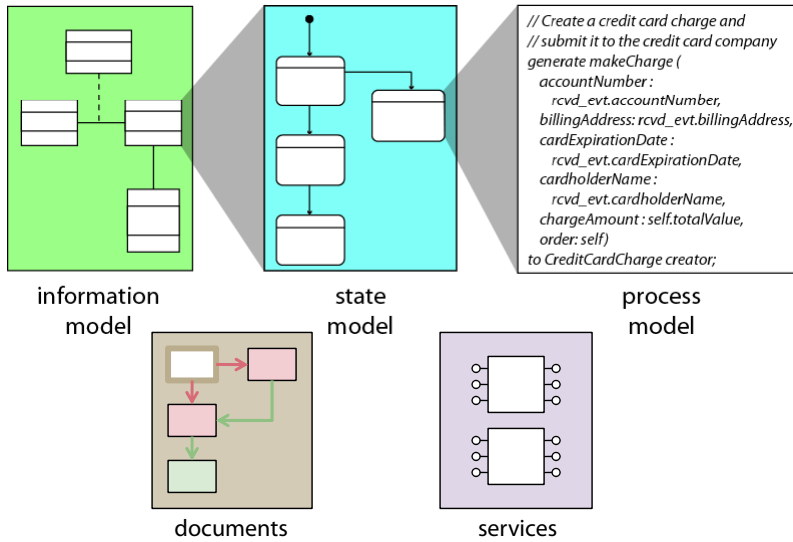


Rules that restrict

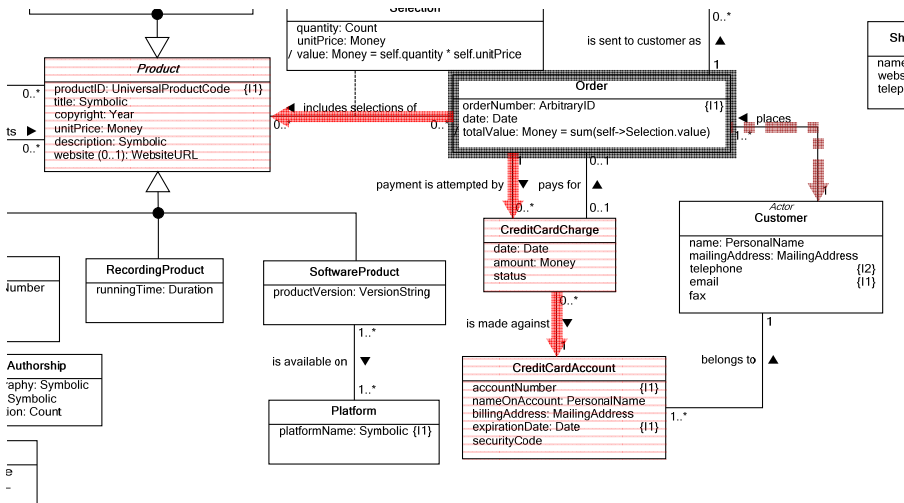
- values of attributes
- selections of related instances

a Customer can only pay using a Credit Card Account that belongs to the Customer

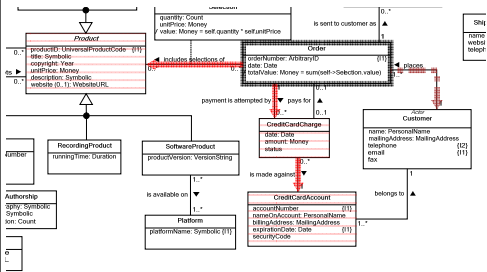
Extended Logical View



Documents



Document Behavior



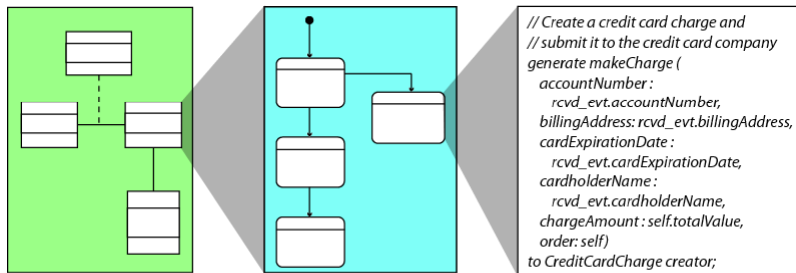
Receive a document

- Receive the contents
- Create new instances
- Update existing instances
- Link to existing instances

Produce documents

- Query parameters
- Select the root
- Build up the contents

Extended Logical View

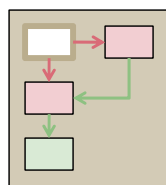


```
// Create a credit card charge and
// submit it to the credit card company
generate makeCharge (
  accountNumber :
    rcvd_evt.accountNumber,
  billingAddress: rcvd_evt.billingAddress,
  cardExpirationDate :
    rcvd_evt.cardExpirationDate,
  cardholderName :
    rcvd_evt.cardholderName,
  chargeAmount : self.totalValue,
  order: self)
to CreditCardCharge creator;
```

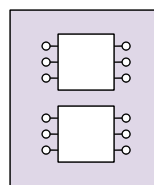
information model

state model

process model

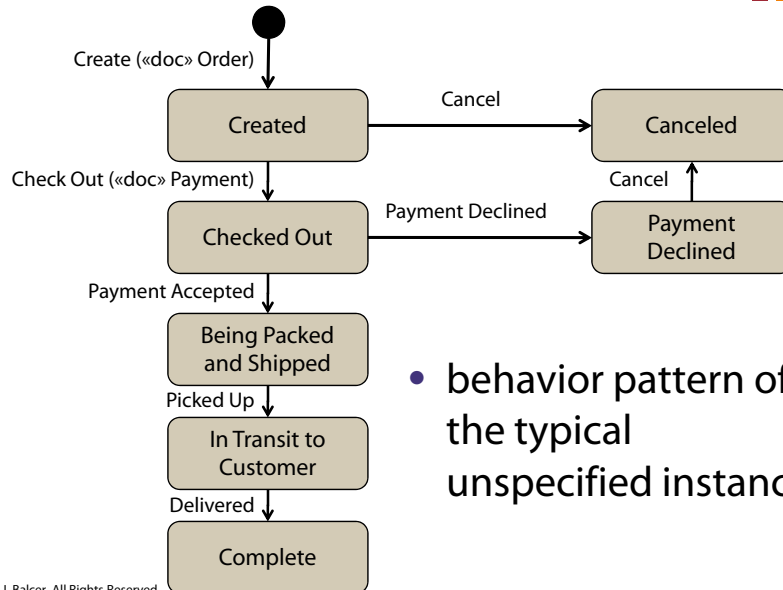


documents



services

Object Lifecycles

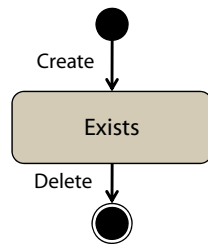


- behavior pattern of the typical unspecified instance

Stateless Services?

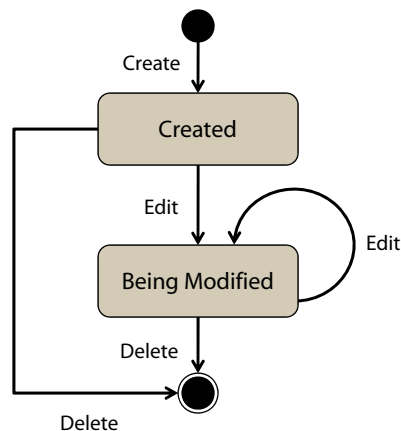
If good services are supposed to be stateless, how can you be using state machines?

When to Formalize Lifecycles



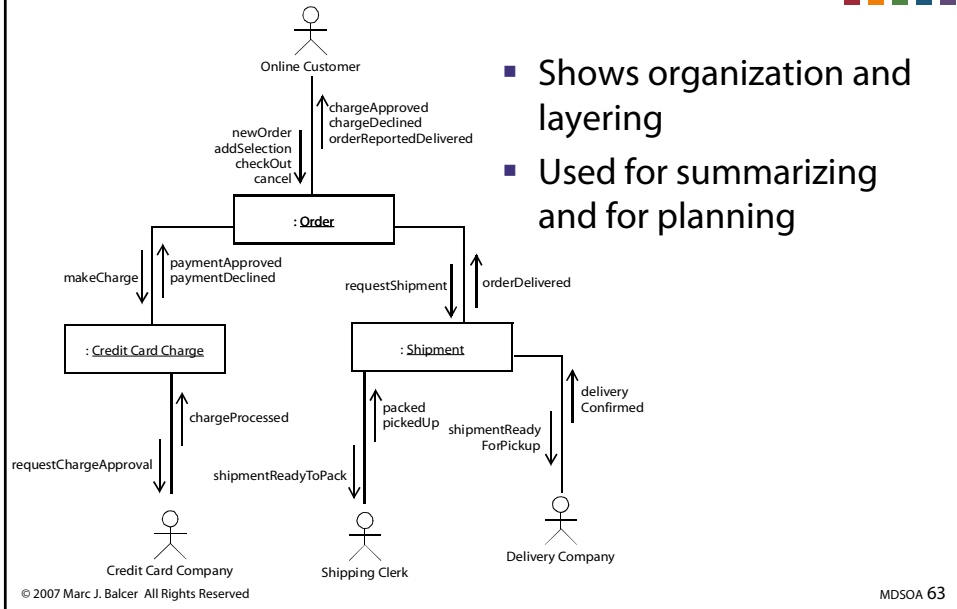
- Every object has a lifecycle
- Some lifecycles are more interesting than others
- Formalize interesting lifecycles with state machines

Events Change Object State



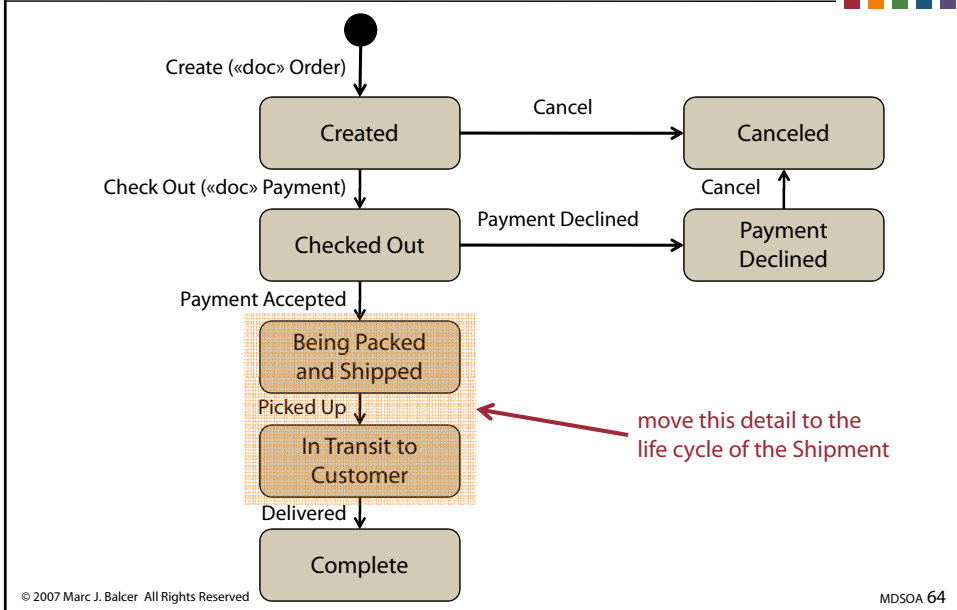
- Don't just create states and events as a place for behavior
- Not every interaction with a domain is through state machine events

Communication Model

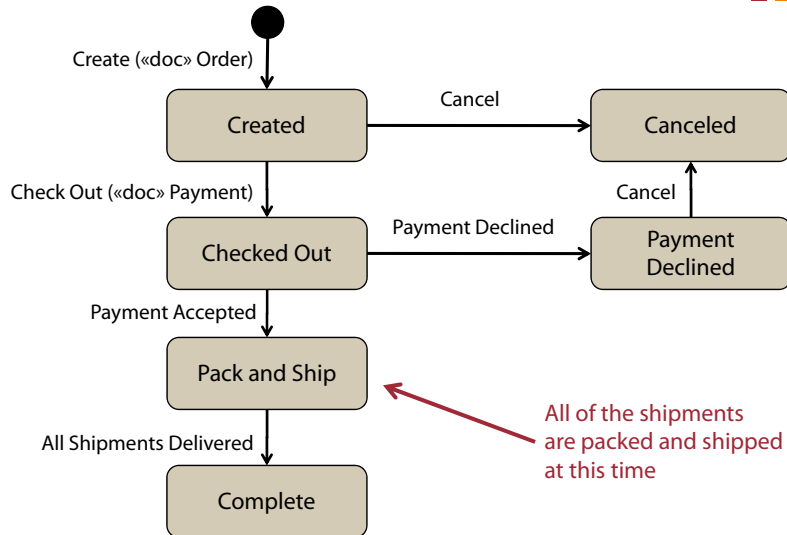


- Shows organization and layering
- Used for summarizing and for planning

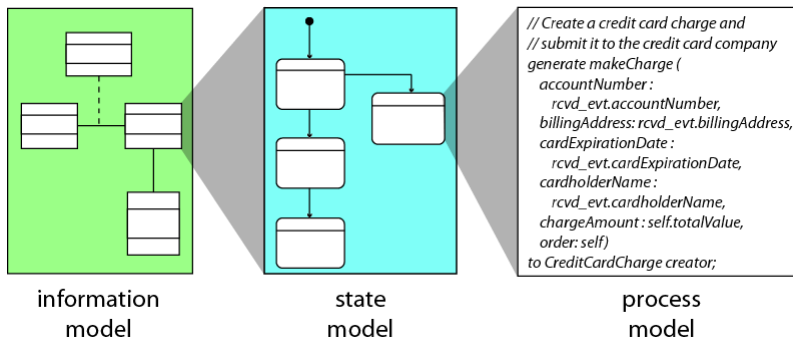
Dividing Responsibility



Dividing Responsibility

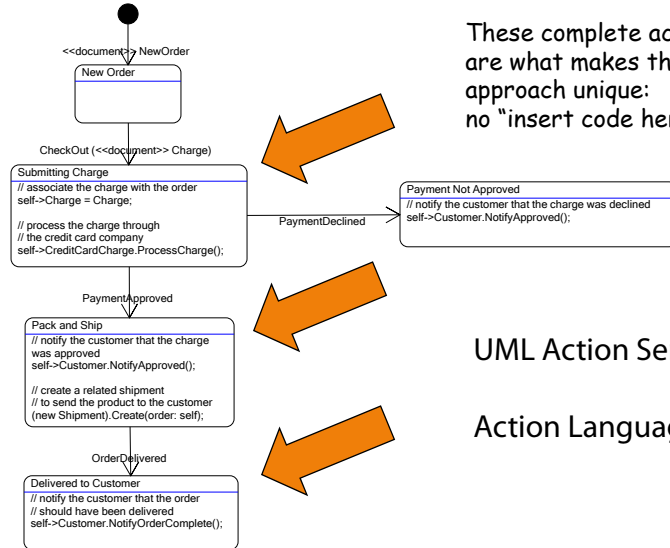


Process Models



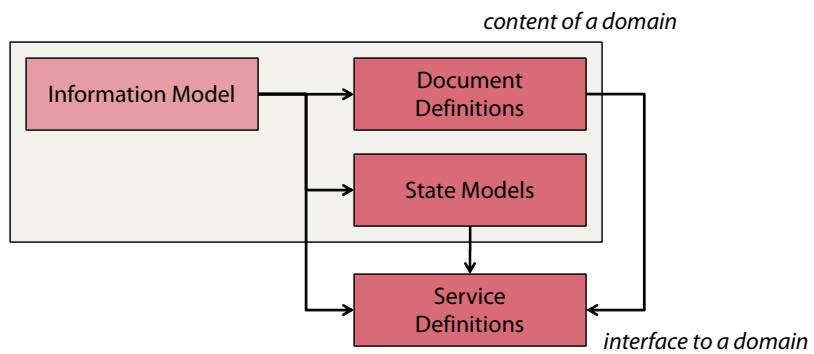
- State Procedures
- Derived Attributes
- Constraint Expressions

Process Models



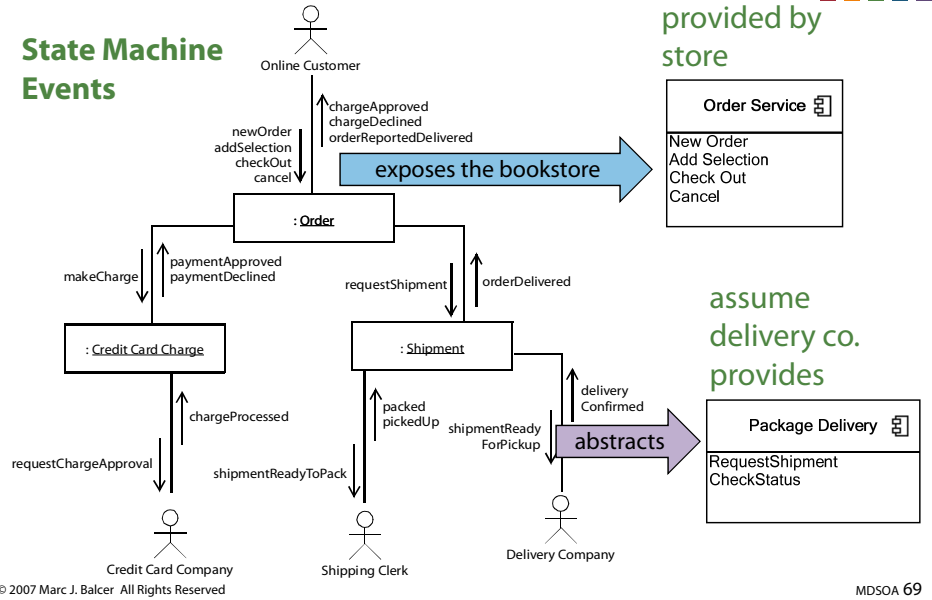
These complete actions are what makes this approach unique: no "insert code here."

Getting to Services



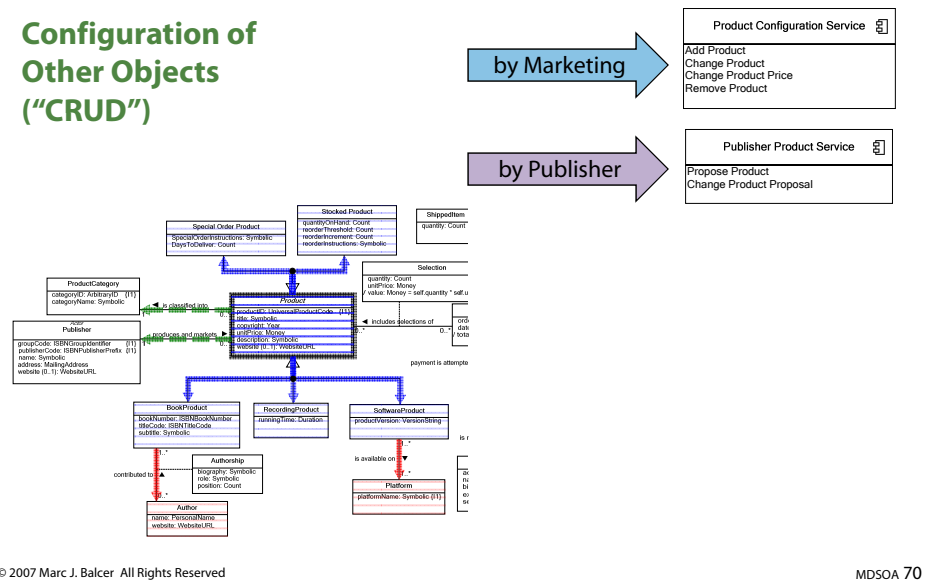
Service Definitions

State Machine Events

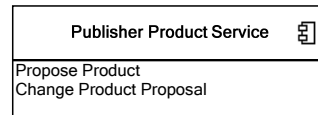
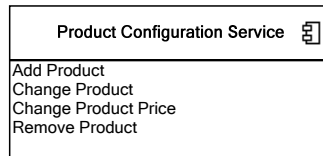
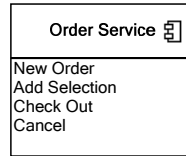


Service Definitions

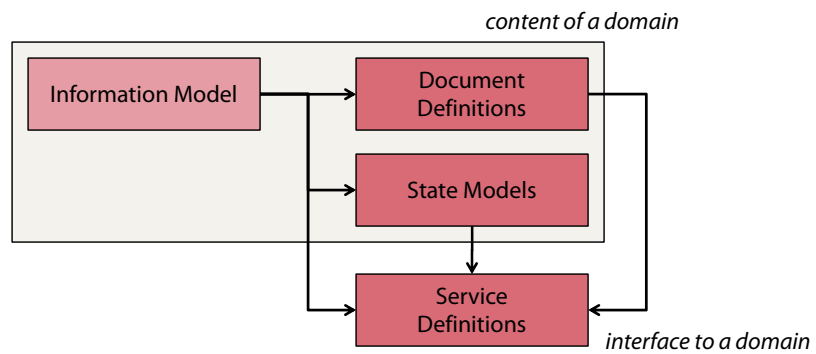
Configuration of Other Objects ("CRUD")



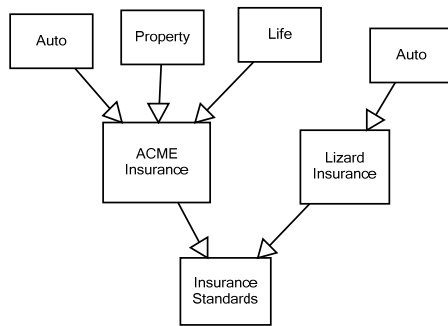
Service Definitions



Domain Models

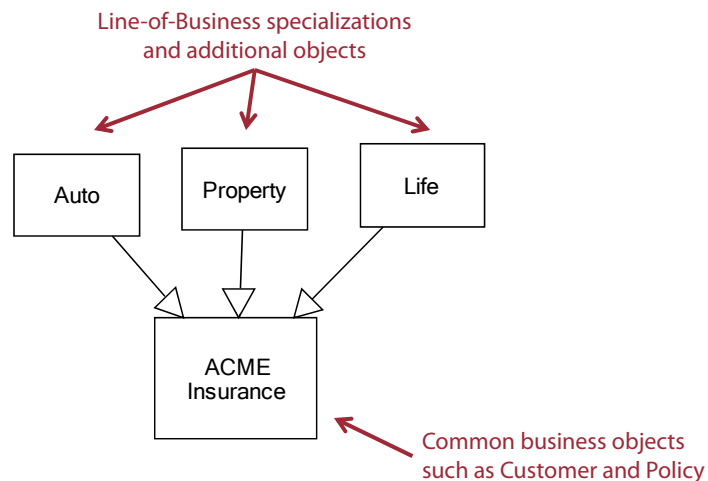


Subsystems



- Partition large domains into subsystems
- Subsystems define the business organization
 - Roles
 - Responsibilities
 - Information
 - Processes
- These are not the existing software entities

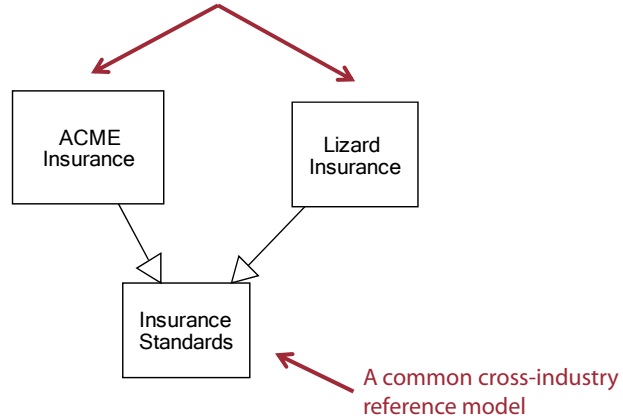
Business Organization



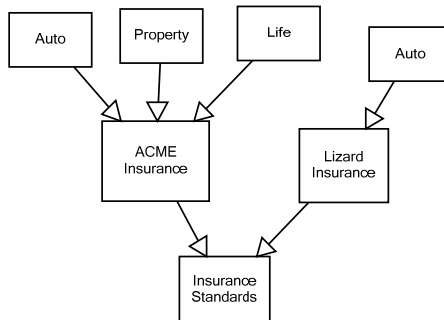
Industry Standards



Each subscribing company's
instantiation of the reference model

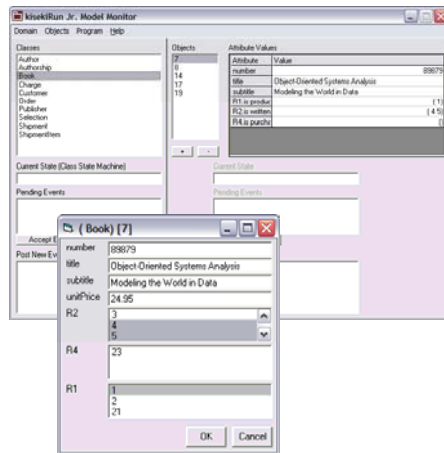


Subsystems and Governance



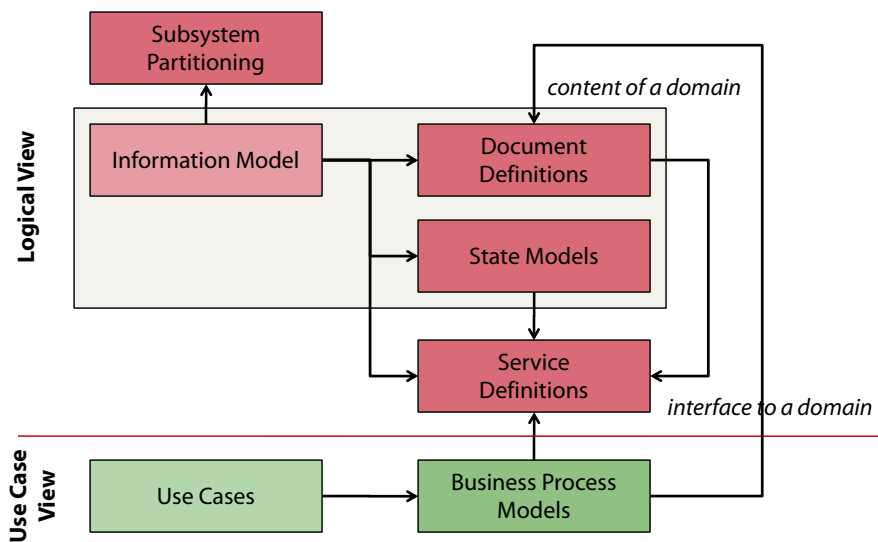
- Each subsystem represents a company, division, department
- That company, division, department can be responsible for the design of the services
- The links define who can build upon which subsystems

Testable, Executable, ...

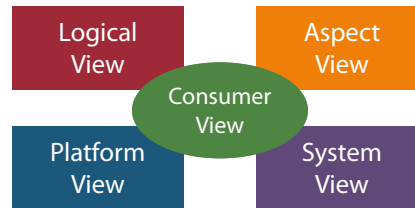


- The pictures run
 - They are executable
 - They are testable
- Test models to
 - verify business processes
 - evaluate alternate scenarios
 - make educated tradeoffs
- Business problems are solved by business people

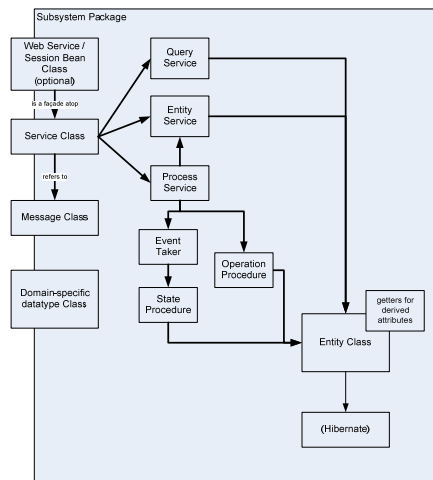
Connections



Other Views

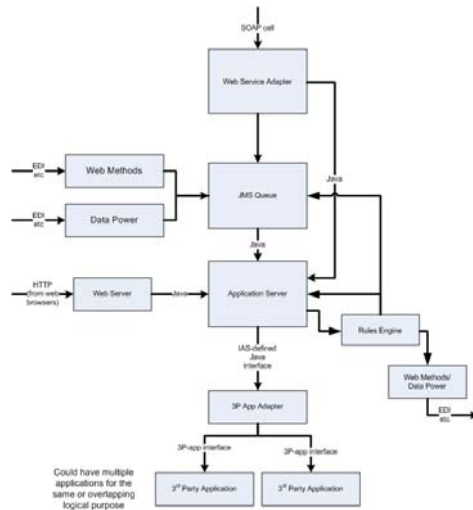


Platform View



- Basic design policy for software construction
- Creates a vocabulary for talking about the software structure
- What has to be built?
- Think “Patterns”

System View

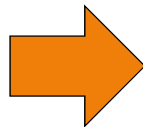


© 2007 Marc J. Balcer All Rights Reserved

MDSOA 81

- Physical structure of servers and machines
- Legacy systems
- 3rd Party Systems
- Message flow and organization

Five Techniques

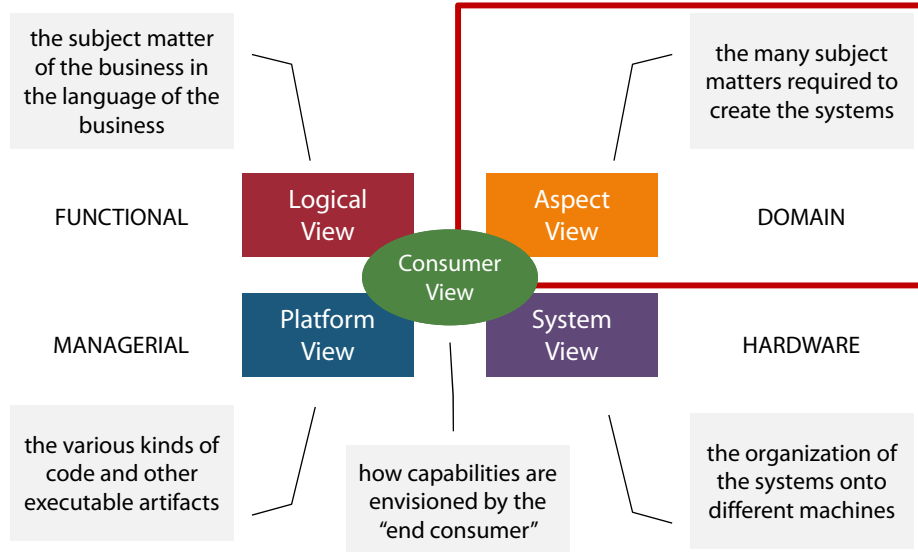


- Multiple ("4+1") architectural views
- Formal Modeling
- Subject-Matter Partitioning
- Translation of Models (MDA)
- Tolerance for Incompleteness

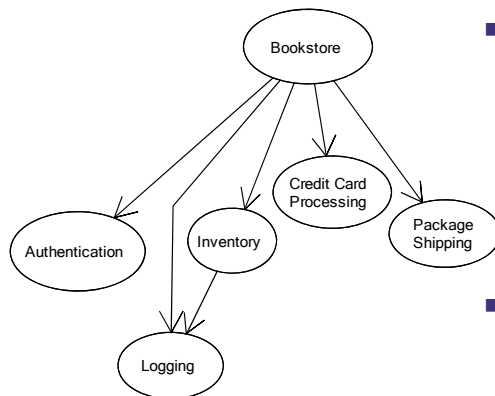
© 2007 Marc J. Balcer All Rights Reserved

MDSOA 82

Subject Matter Partitioning

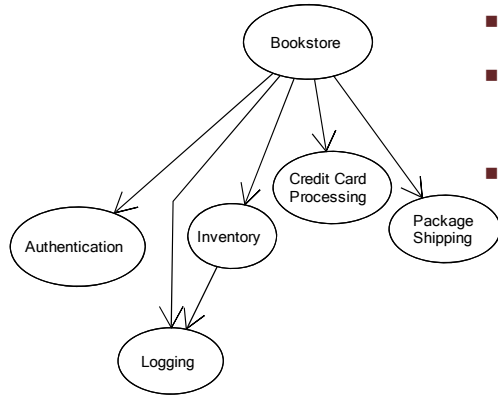


Aspect View



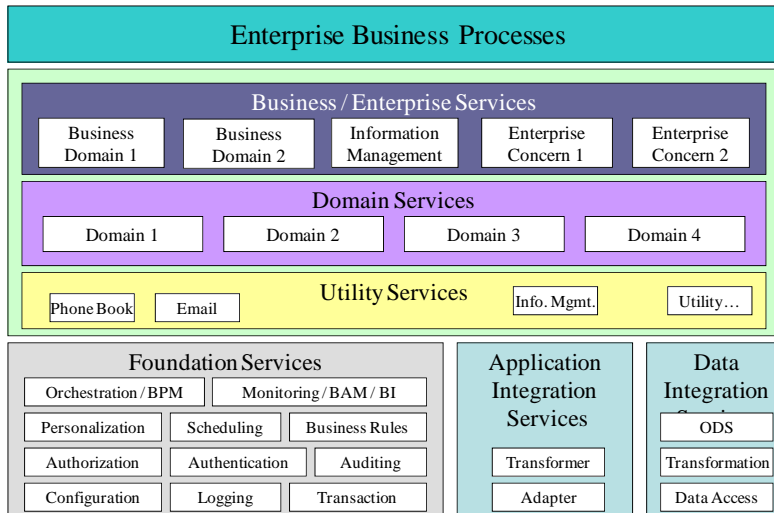
- To build systems we need to deal with multiple subject matters
- We call these "domains"

Aspect View



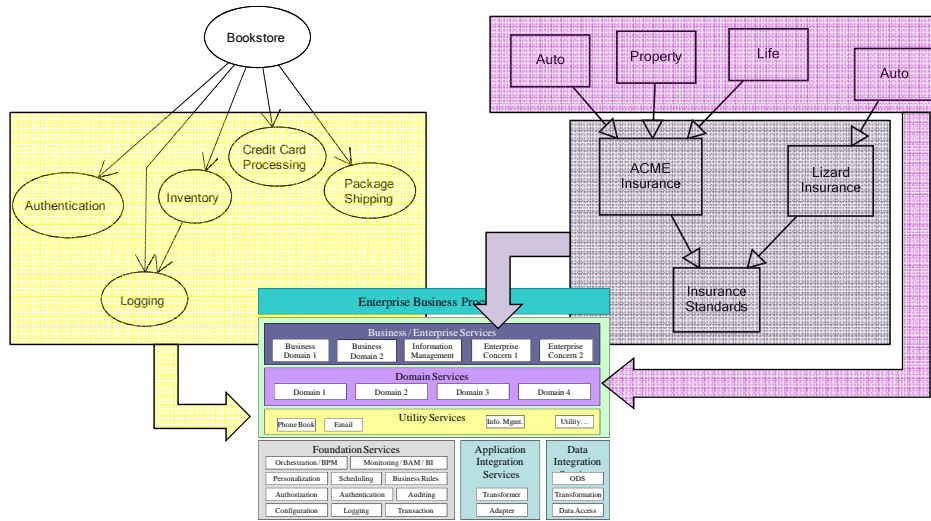
- Say “how it’s built”
- Separate app from implementation
- Identify generic utilities

Service Taxonomy

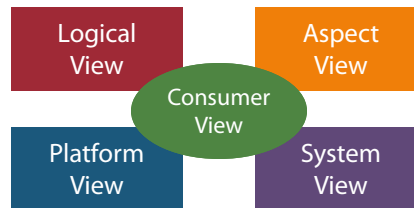


Generic Service Taxonomy courtesy Mike Rosen

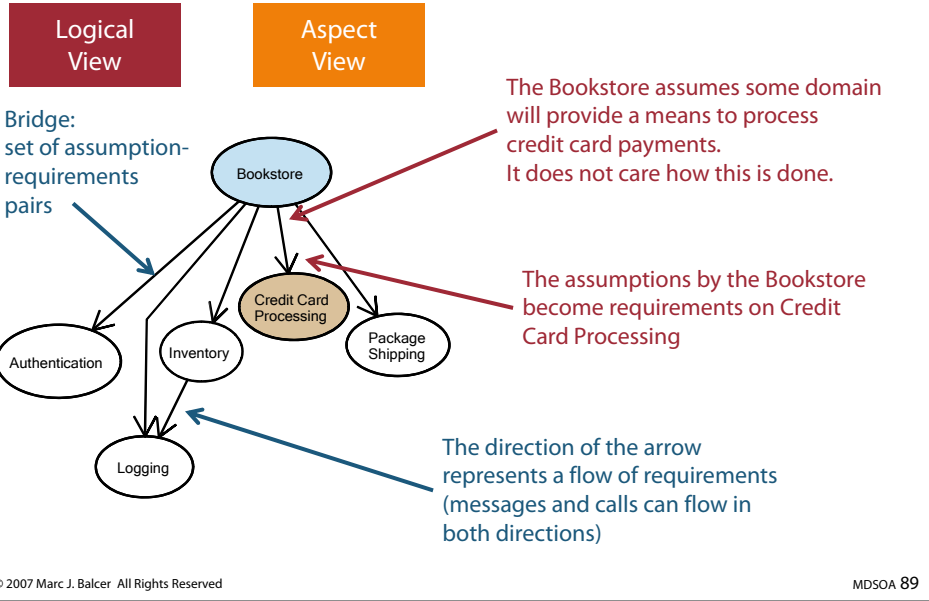
Partitioning the Taxonomy



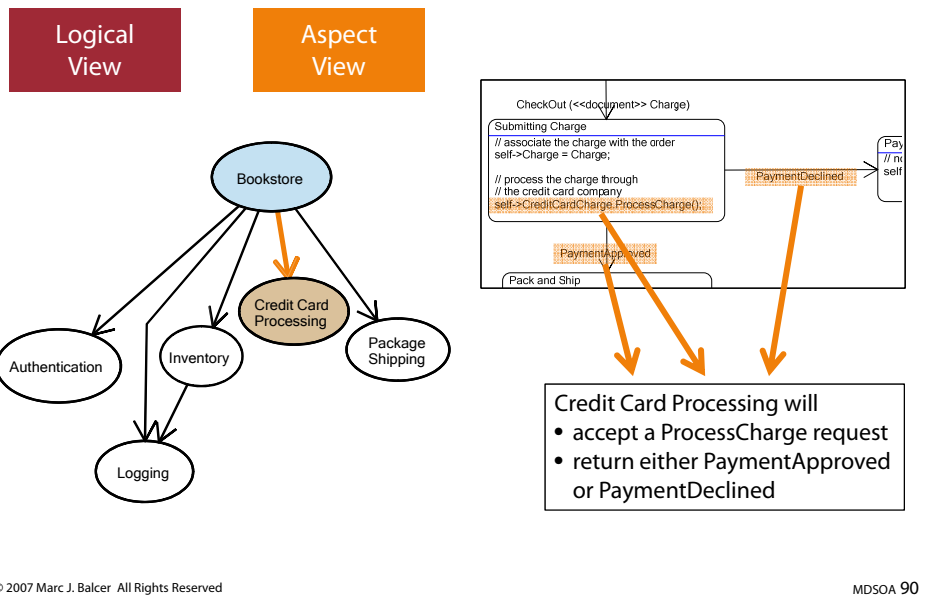
Connect Views Together



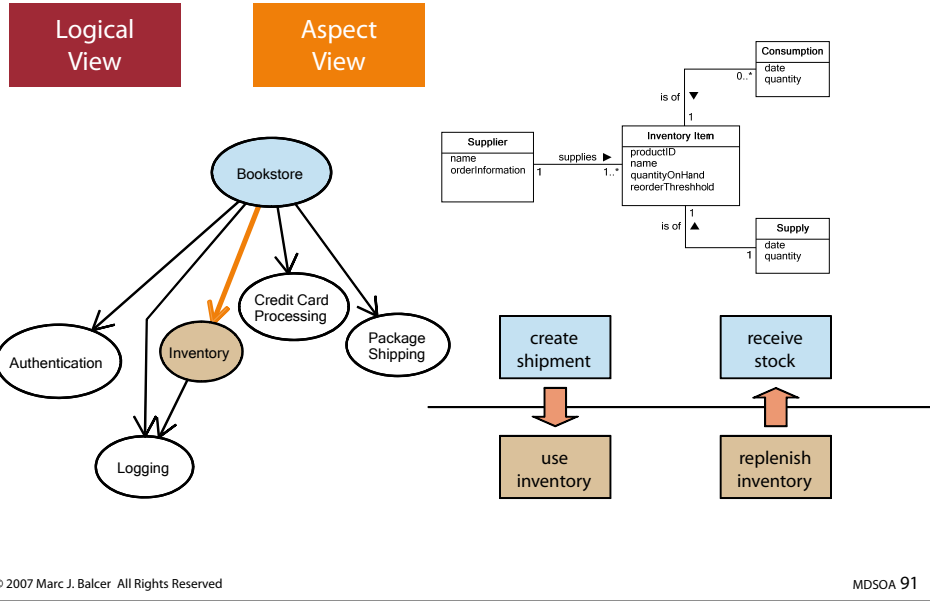
Assumptions and Requirements



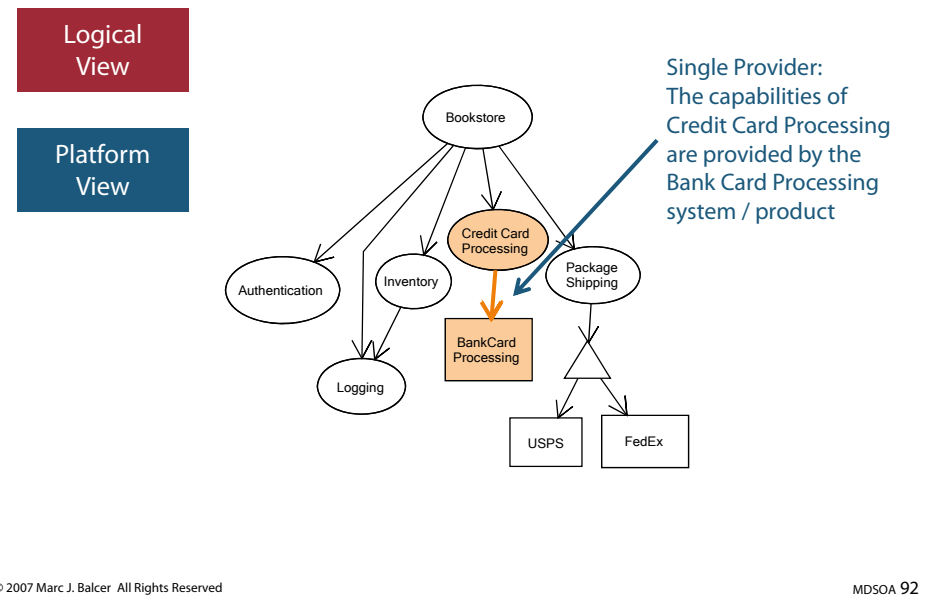
Explicit Bridging



Implicit Bridging



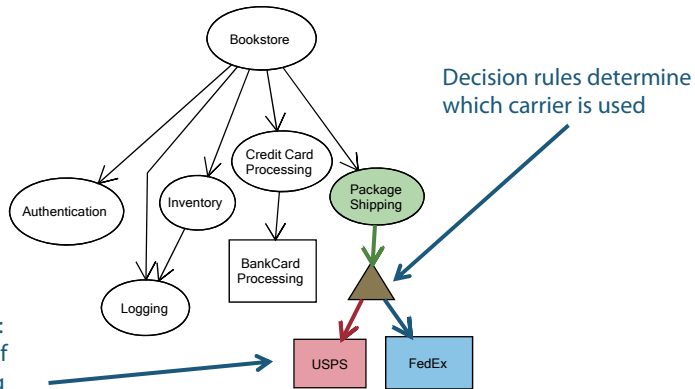
Domains and Systems



Domains and Systems

Logical View

Platform View



Multiple Provider:
The capabilities of
Package Shipping
are provided by both
USPS and FedEx

Decision rules determine
which carrier is used

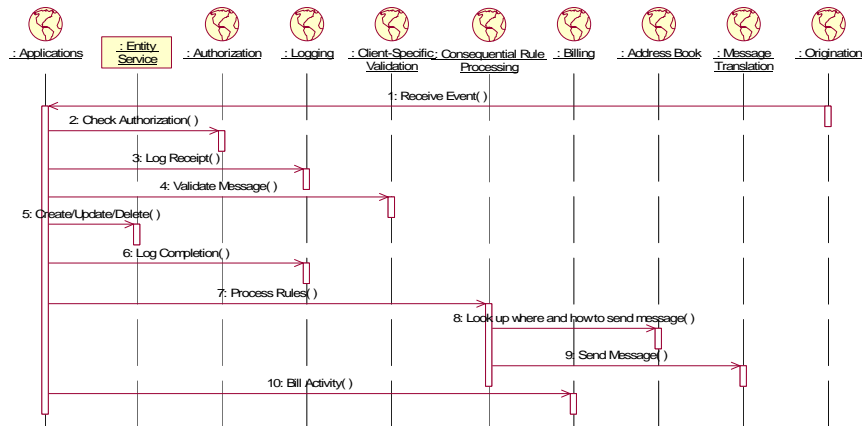
Other View Connections

Connection	Description
Platform - System	Assignments of process types to machines
Logical - Aspect	Populated archetypes
Consumer - all others	Requirement Matrix: Assignment of use case content to views
Consumer - Logical	Actors → Classes Documents → Classes + document definitions

Domains and Components

Platform View

Aspect View

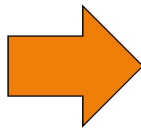


© 2007 Marc J. Balcer All Rights Reserved

MDSOA 95

Five Techniques

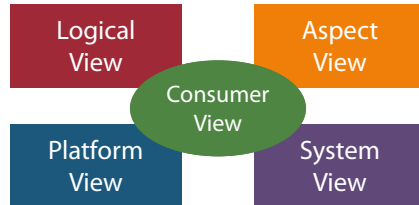
- Multiple ("4+1") architectural views
- Formal Modeling
- Subject-Matter Partitioning
- Translation of Models (MDA)
- Tolerance for Incompleteness



© 2007 Marc J. Balcer All Rights Reserved

MDSOA 96

Translation of Models (MDA)



- So far we have
 - Models for the four views
 - Connections between those models
- How to turn these models into implementations?

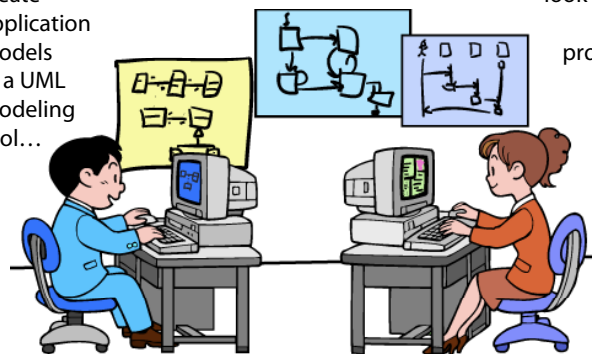
What's the Problem?



Architects create application models in a UML modeling tool...

...print them out...

... then the developers look at the models and write the production code

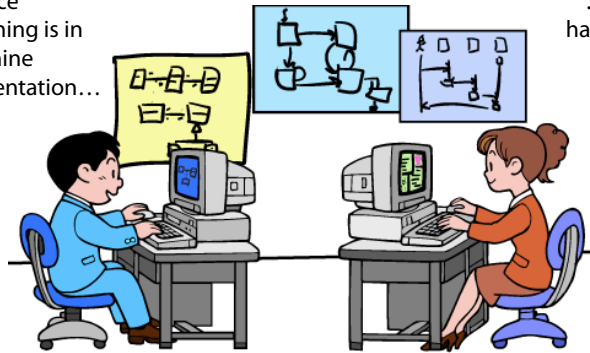


What's the Problem!



But once something is in a machine representation...

...we shouldn't have to manually reinterpret it ("re-code it") into another.

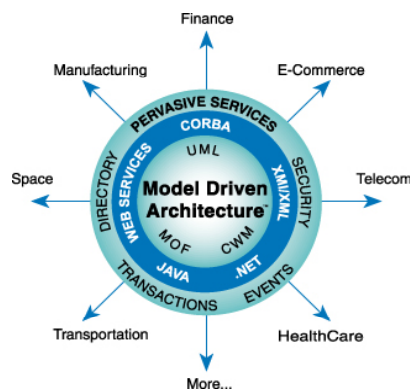


The architect's diagrams don't have to be complete, correct, or consistent models

Keeping the diagrams in sync with the code is a daunting task.

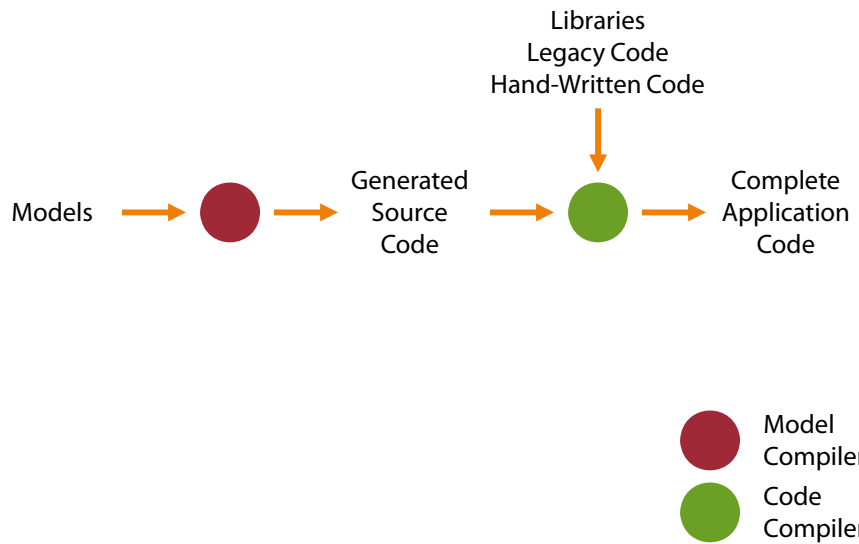
The re-coding introduces development errors and obscures analyst errors

MDA – Model Driven Architecture

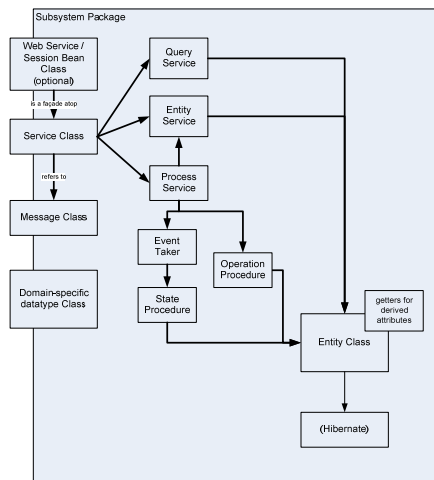


- Formal Models
- Metamodel
- Translation

Naive Translation



What to Generate



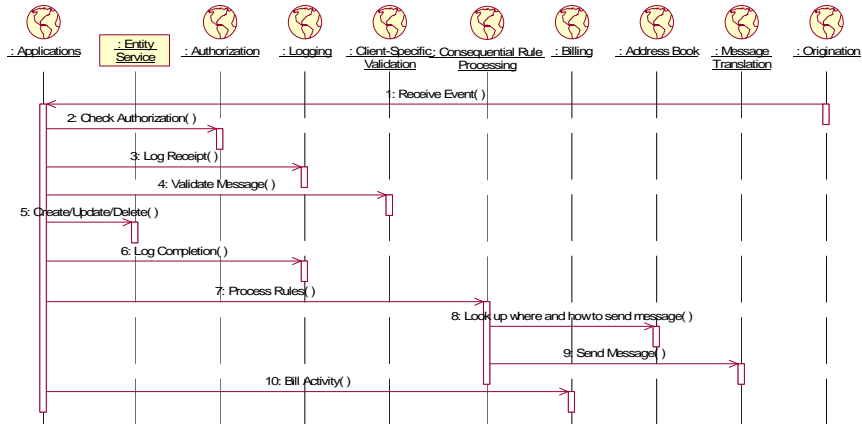
- Create instances of all of the things in the Process View
- Each has an archetype (template) populated from the logical view models

What Form to Generate

Platform View

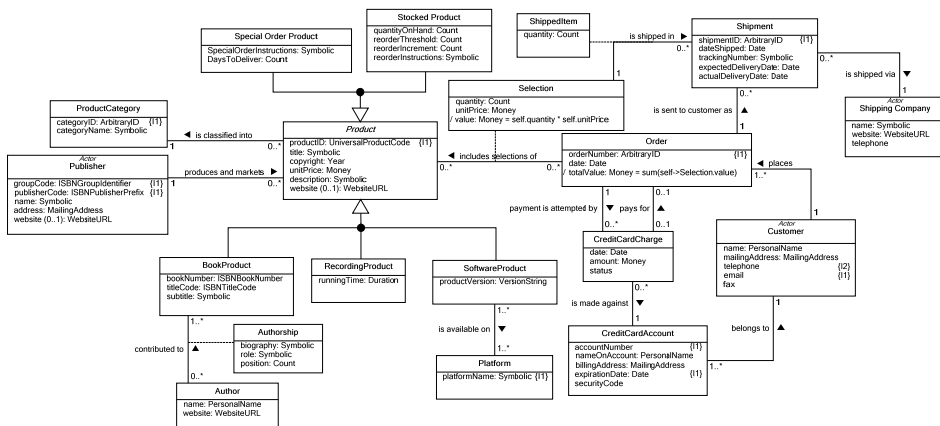
Aspect View

- Basic patterns for the archetype contents



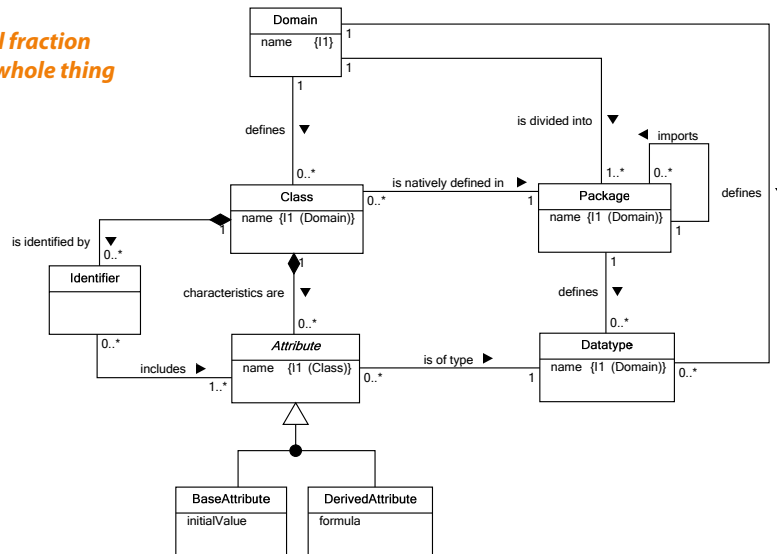
Models...in what form?

- More than diagrams...real semantic content

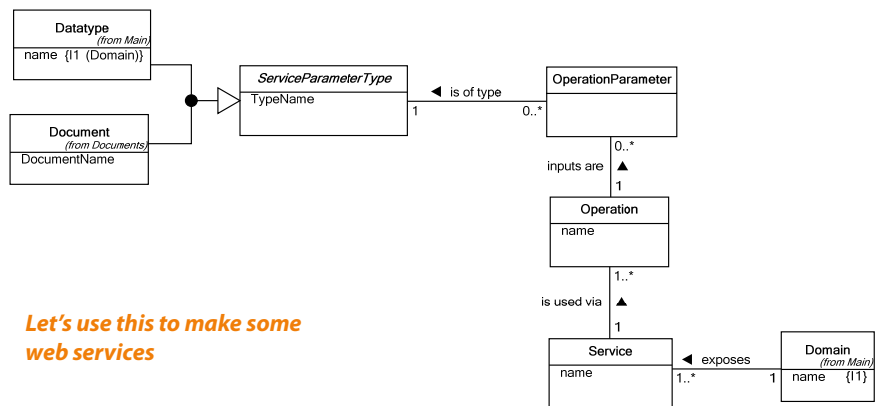


Metamodel

A small fraction of the whole thing

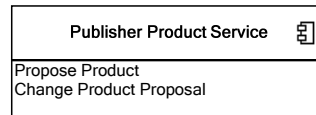
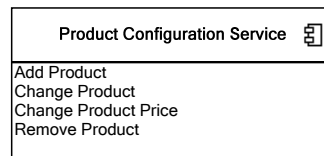
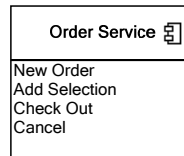


Metamodel (Services Part)



Let's use this to make some web services

Model elements become...

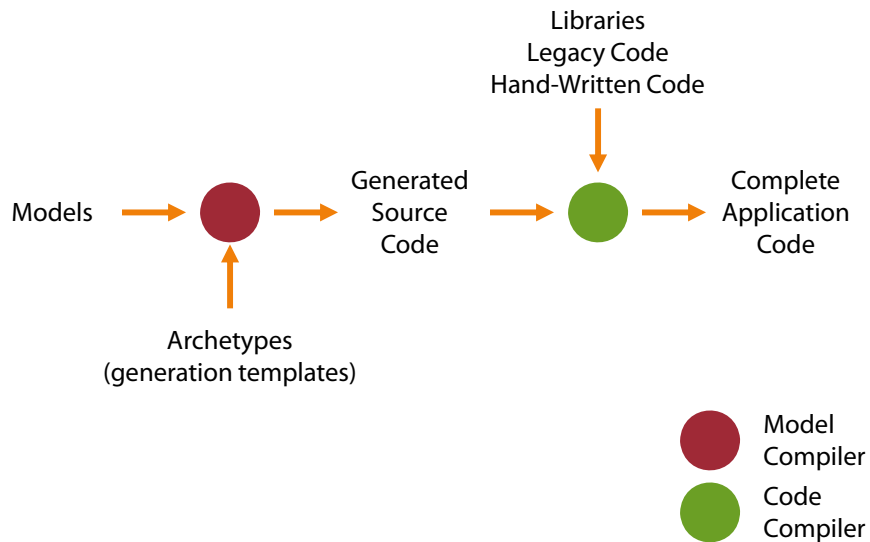


...Metamodel Instances



Service	Operation	Parameter	Type
Order	NewOrder	Order	«Document» Order
Order	AddSelection	ProductID	UniversalProductCode
Order	AddSelection	Quantity	Count
Order	CheckOut	Payment	«Document» CreditCardCharge
Order	Cancel	OrderID	ArbitraryID
Product Config	AddProduct	Product	«Document»Product
Product Config	ChangeProduct	Product	«Document»Product
Product Config	ChangeProductPrice	ProductID	UniversalProductCode
ProductConfig	ChangeProductPrice	NewPrice	Money
ProductConfig	RemoveProduct	ProductID	UniversalProductCode

Archetypes



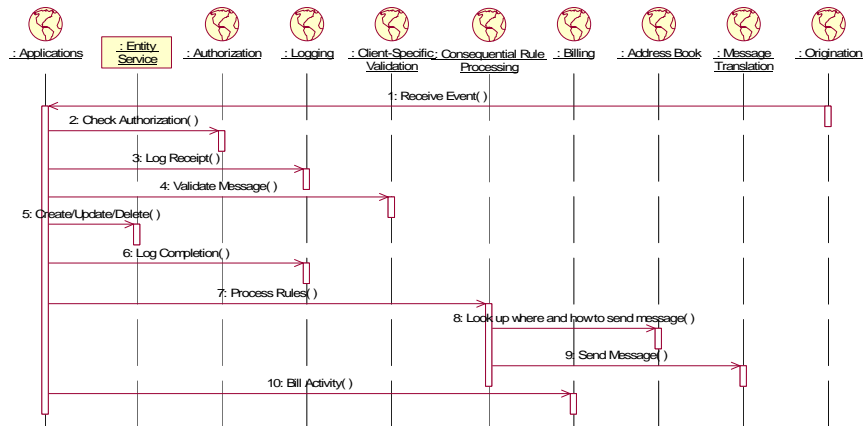
Archetype Example

```
public class <<=VBNameMapping(ClassName)>>
    inherits NamedItem
    Public Shared XPathSelector As String = GetType(<<=VBNameMapping(ClassName)>>).Name

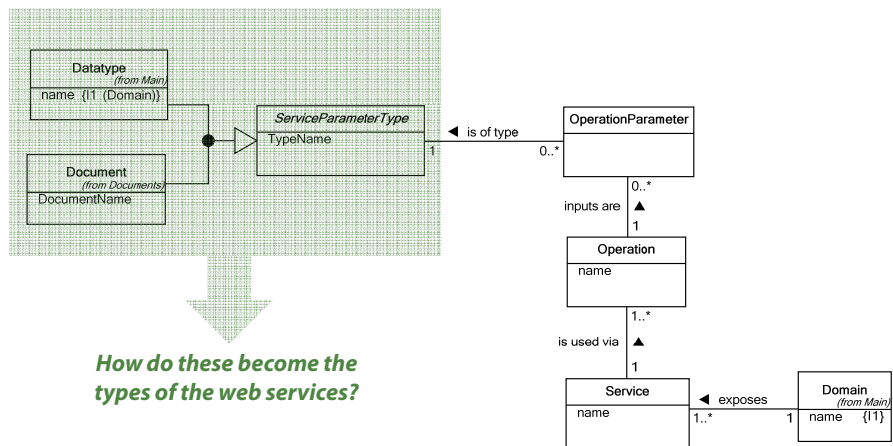
    ' Include the attributes
    <<=for each attr as ModelCompilers.ExecUML.Attribute in theClass.Attributes
    dim attrTypeName as string
    if attr.Type is nothing then
        attrTypeName = "symbolic"
    else
        attrTypeName = attr.Type.Name
    end if
    if attr.Name.ToLower << "name" then
    Private <<=attr.Name>> As <<=TypeNameMapping(attrTypeName)>>
    Public Property <<=attr.Name>> As <<=TypeNameMapping(attrTypeName)>>
        Get
            Return <<=attr.Name>>
        End Get
        Set(ByVal value As <<=TypeNameMapping(attrTypeName)>>
            <<=attr.Name>> = value
        End Set
    End Property
    end if
    next
    ' Include the associations. Do these in two groups:
    ' First: associations where this class is the origin (container)
    <<=for each assn as Association in Model.Associations
    dim typeName as string ' typeName is a reserved word
    dim memberName as string
    dim privateMemberName as string
    if theClass is assn.Origin then
        if assn.DestinationMultiplicity = MultiplicityString.One or assn.DestinationMultiplicity = MultiplicitySt
```

Component Patterns

The archetypes ensure that all of the utility services are used and used properly



Mappings



Mappings

Map each Datatype to an implementation type

Datatype	Impl Type
symbolic	System.String
numeric	System.Integer
count	System.Integer
Money	System.Currency
Date	System.DateTime

Create a class for each Document

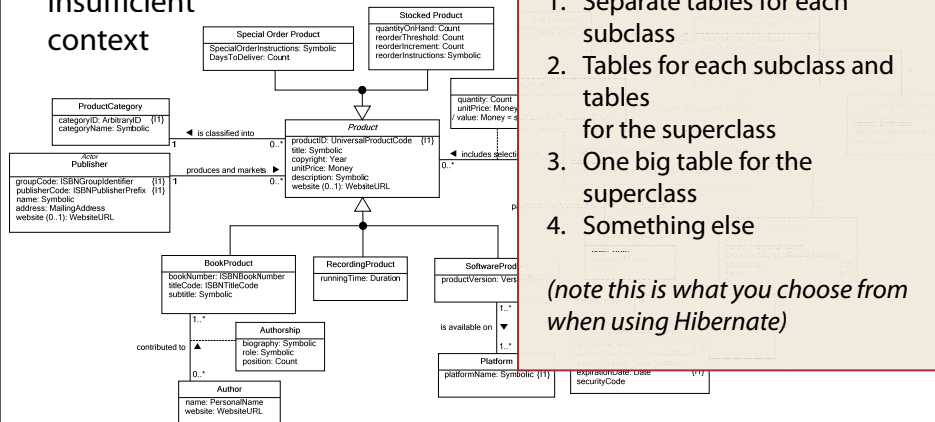
(another archetype that reads the document definitions)

class <%Document.Name%>

...

Marks

Use marks to direct generation when the models contain insufficient context

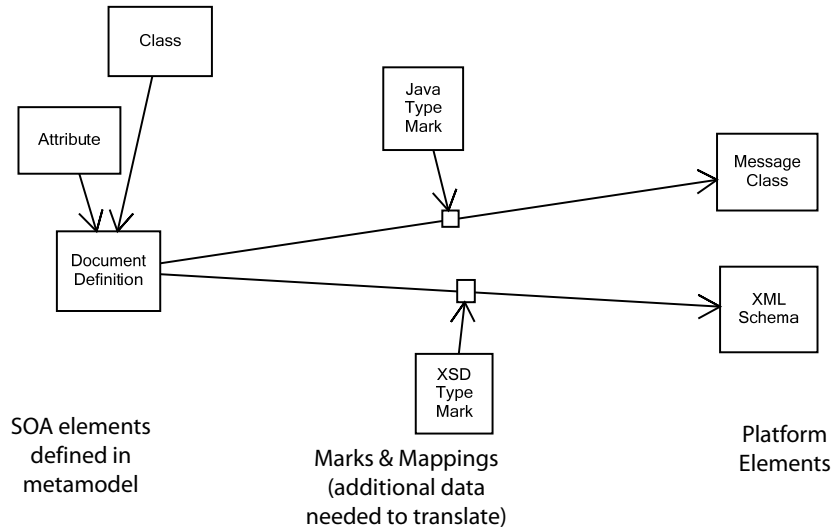


Schemes for generating generalization:

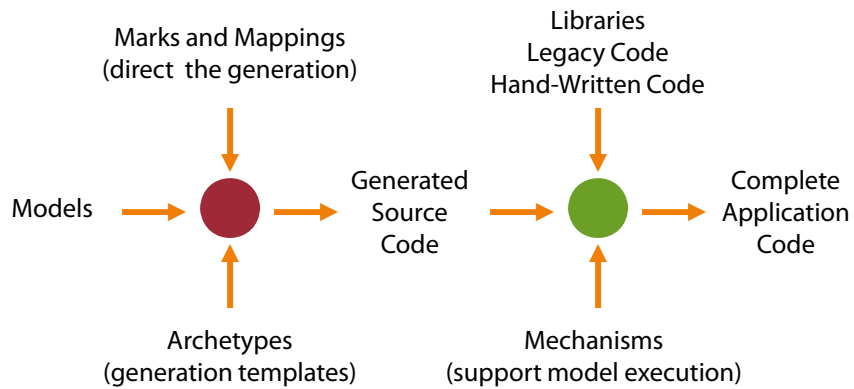
1. Separate tables for each subclass
2. Tables for each subclass and tables for the superclass
3. One big table for the superclass
4. Something else

(note this is what you choose from when using Hibernate)

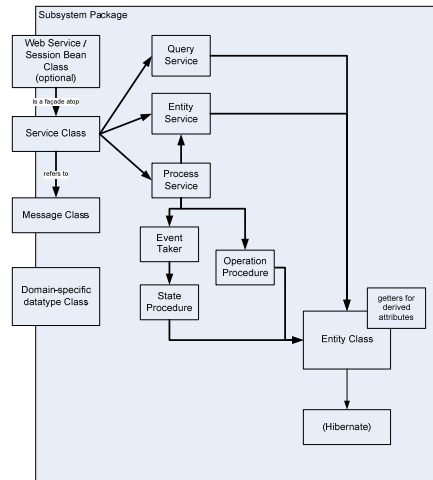
Translation



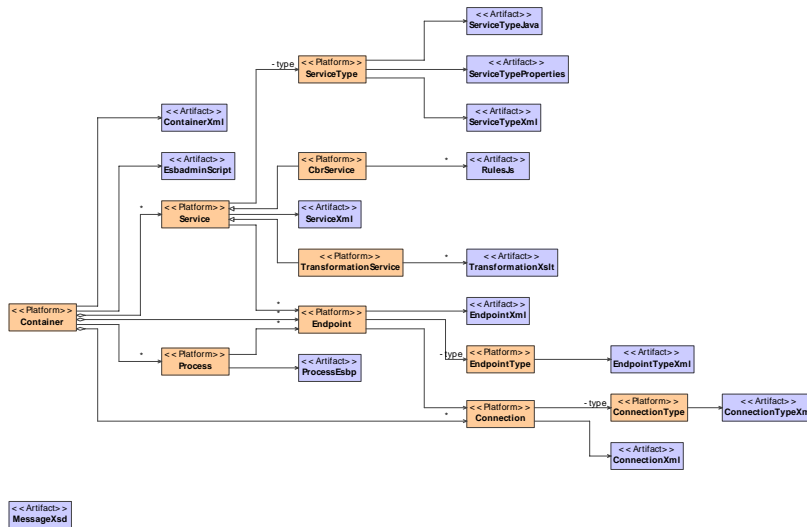
Translation Components



A "Traditional" Platform



An ESB Platform

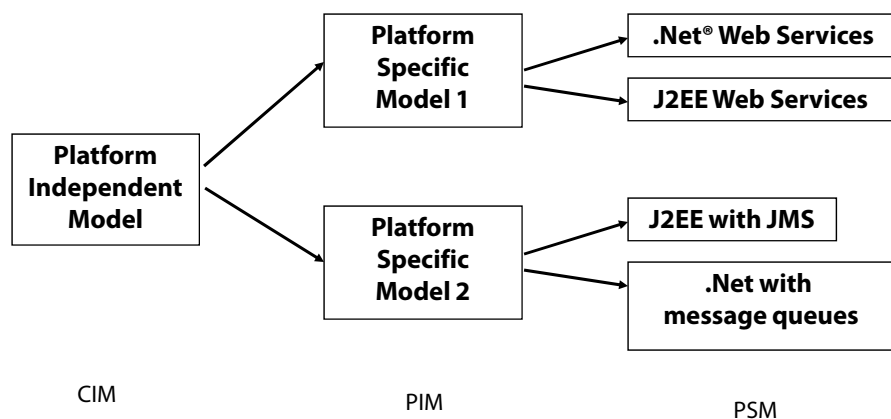


Software Architecture

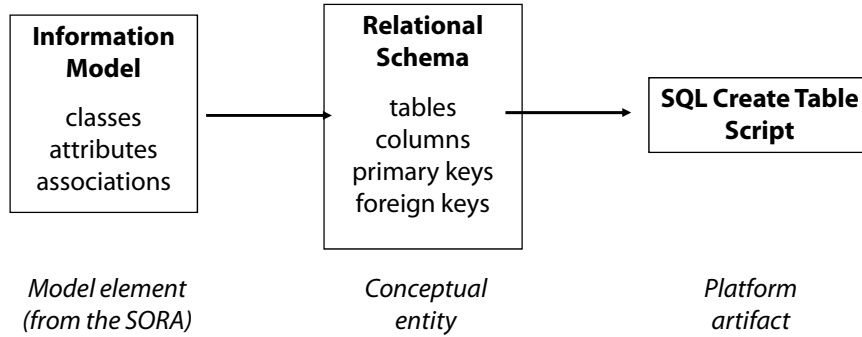


- Basic design policy
- Conceptual Architecture
 - Platform-independent representation
 - “Character” of the architecture
- Platform Architecture
 - Platform-specific representation
 - Select technologies
 - Define actual artifacts
- Once conceptual architecture can have many platform architectures

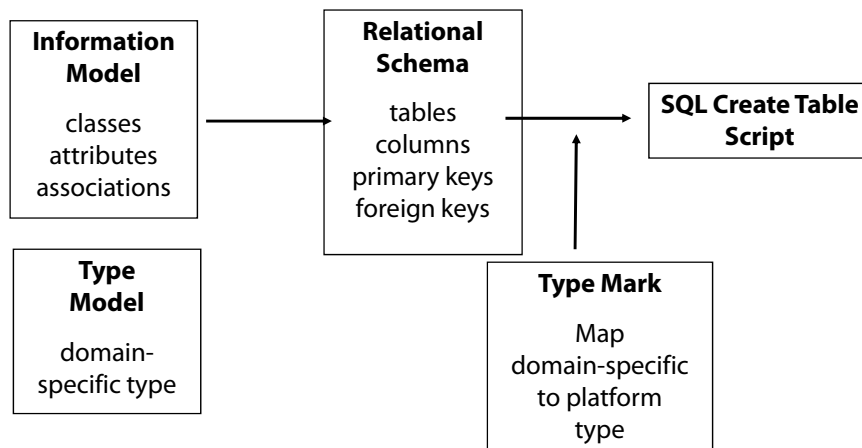
“Platform” is relative



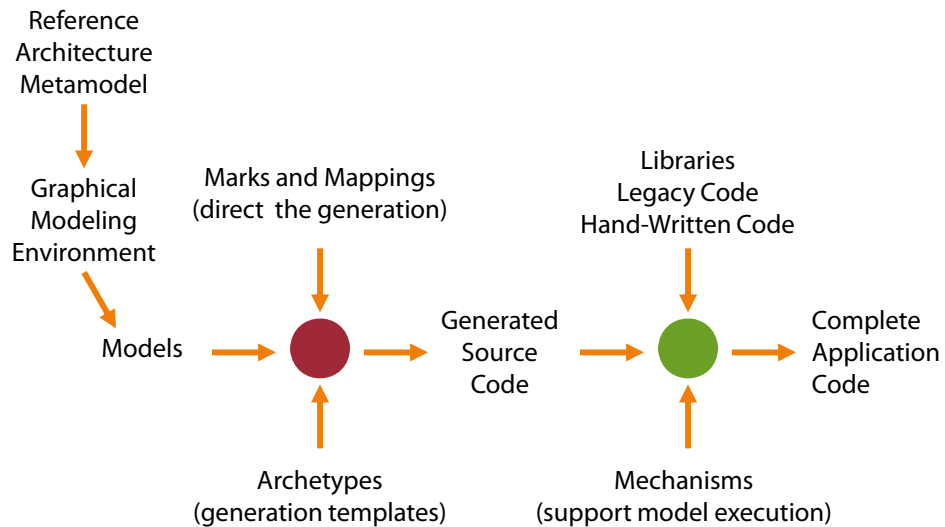
Example



Additional Information



Automation Overview



© 2007 Marc J. Balcer All Rights Reserved

MDSOA 123

Model-Driven Myths

- “This approach is good for top-down but we’re service-enabling existing applications”
- “We can’t wait to figure out the whole problem before starting development”
- “We do Agile”
- “Do we really have to do all of this?”

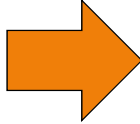
© 2007 Marc J. Balcer All Rights Reserved

MDSOA 124

Five Techniques



- Multiple (“4+1”) architectural views
- Formal Modeling
- Subject-Matter Partitioning
- Translation of Models (MDA)
- Tolerance for Incompleteness

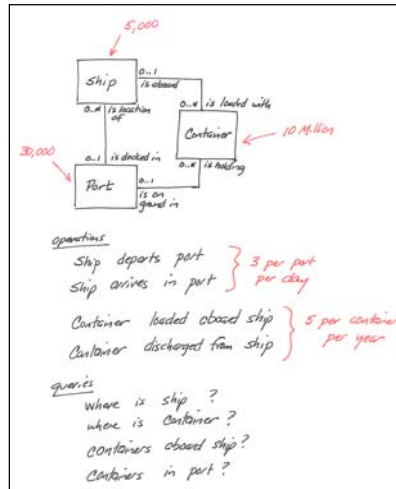


Building a Model Compiler



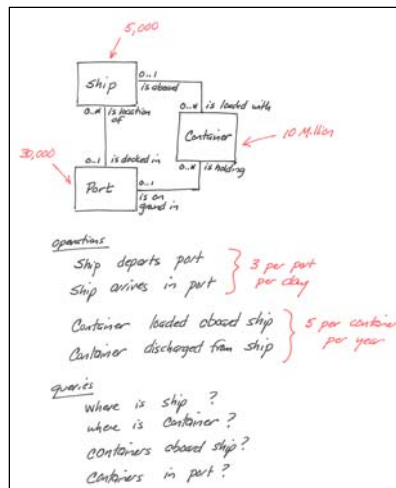
- First, don't build a model compiler
- Prototype the platform using a simple model
- Model the platform
- Create archetypes
- Understand and use marks and mappings

Code This!



- Why?
 - How do we build this thing?
 - What is a model compiler?
 - What is our software architecture?
 - What is our domain chart?
- Prototyping helps to figure these out

Code This!



- A super-simple guess—real app still horribly undefined
- Intended to get the architecture & practices in place
 - Test “-abilities”
 - Development foundation required for incremental development
 - “Working Software over ... Documents”

Prototype Architecture Immediately



- Development infrastructure must be in place
- Team members must be trained in technologies
- Key performance issues need to be demonstrated
- Team needs to learn to work together
- Reinforce notion of “deliver value quickly”

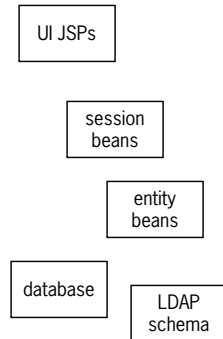
Two weeks, two tracks



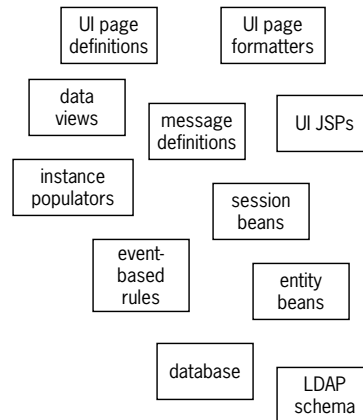
- | | |
|---|---|
| <ul style="list-style-type: none">▪ Prototype Architecture<ul style="list-style-type: none">- establish development & deployment environments- establish automated build, test, and deployment processes- test key performance objectives- code, test, deploy super-simple application | <ul style="list-style-type: none">▪ Application Modeling<ul style="list-style-type: none">- identify priorities- set scope- sketch solutions- formalize knowledge- define the vocabulary- build models |
|---|---|

New domains = New kinds of sources

*for the pilot,
we created...*



*by increment 3
we were creating...*



Tolerance for Incompleteness

- Strike a balance
 - don't try to boil the ocean
 - don't be afraid to start
- Do what you can, and plan to do more
 - incremental development
 - incremental improvement

Incremental improvement

- Pilot project
- Develop internal expertise
- Don't need to do all the techniques

Start Simple, Increase Complexity

Increment 1	Increment 2	Increment 3	Increment 4
Single-item order	Multi-item order		
Each order is made up individually		Allow reordering from a "shopping cart"	Allow sharing of shopping carts (e.g. a book list for a course)
Books only		Movies, videos, and software	
All items assumed to be in stock		Items may not be in stock	
All items assumed to be stocked by and shipped from bookstore			Some items may be special-order from the publisher; we submit order to publisher and publisher ships directly
Ship when payment (credit card charge) approved		Ship only when all product in stock	Partial shipments as product is restocked
Order closed when shipped	Order closed when shipment received by customer		
Do not handle returns		Customer can produce a "return authorization" from a shipment	No return authorizations for items not yet shipped
Credit card payments only			House accounts for pre-approved customers; billing done by separate system
Credit card payment must be approved or order canceled	Allow charge resubmitting against declined charges		
Full payment when order checked out			
Can cancel only orders not yet checked out		Can cancel orders not yet shipped	Can cancel parts of orders not yet shipped
Cannot cancel paid orders		Canceling paid orders requires issuing a credit only when returned items received	Canceling parts requires issuing partial credit

Baseline versions of features

Cell = "user story"

A complete trace through the whole system

Two weeks, two tracks



- **Prototype Architecture**
 - establish development & deployment environments
 - establish automated build, test, and deployment processes
 - test key performance objectives
 - code, test, deploy super-simple application
- **Application Modeling**
 - identify priorities
 - set scope
 - sketch solutions
 - formalize knowledge
 - define the vocabulary
 - build models

Application Models



- **Develop collaboratively with the customer**
 - Ensure understanding of the “big picture”
 - Time-box the process
 - Make simplifying assumptions
- **Deliver understanding, not documents**
 - Little thinking tools to get the big picture
 - Build what’s testable
 - Expose detail – formalize knowledge

Five Techniques



- Multiple (“4+1”) architectural views
architecture ≠ boxitecture
- Formal Modeling
make it relevant
- Subject-Matter Partitioning
create effective reuse
- Translation of Models (MDA)
produce results
- Tolerance for Incompleteness
don't get stuck

Meaning for SOA?



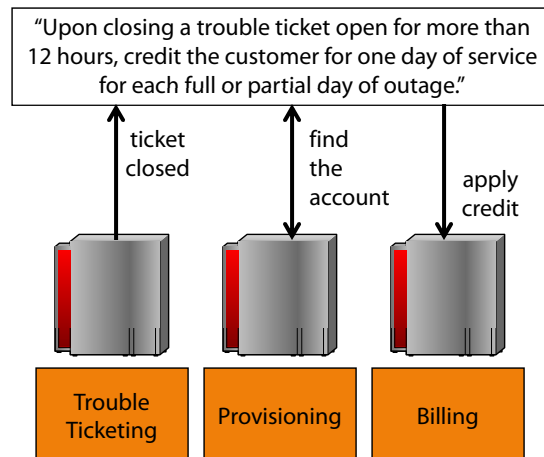
- SOA promises faster, better, cheaper
- Need to build the right services
- Need to build the right service consumers
- Building SOA is still coding
- Code smart by “one fact in one place”
- Don't forget architecture

Addendum



Sample Problems

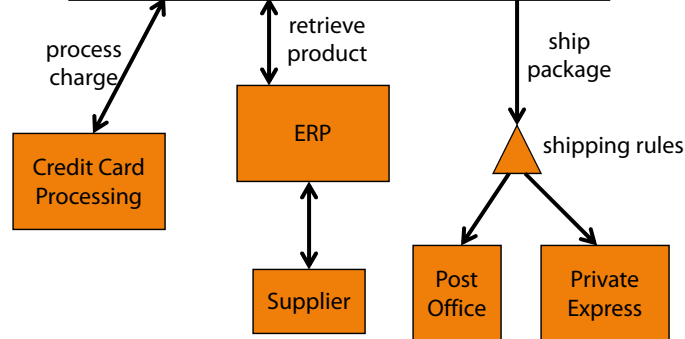
Business Unification



Web Services



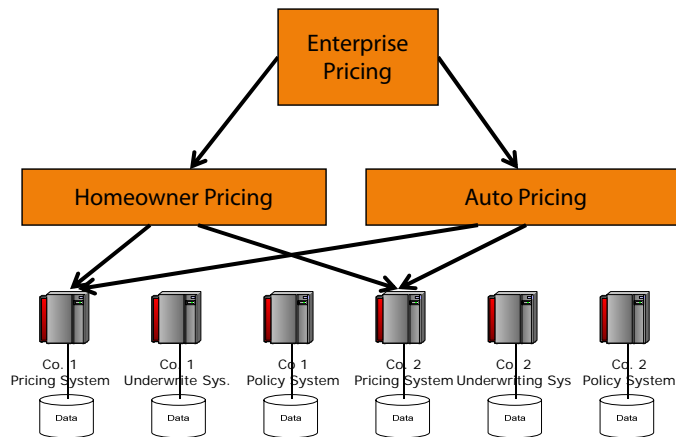
“Once a customer has placed an order, process the charge through the credit card company. If the charge is approved, retrieve the merchandise from stock and order a shipment from the appropriate package shipping company.”



Company Mergers



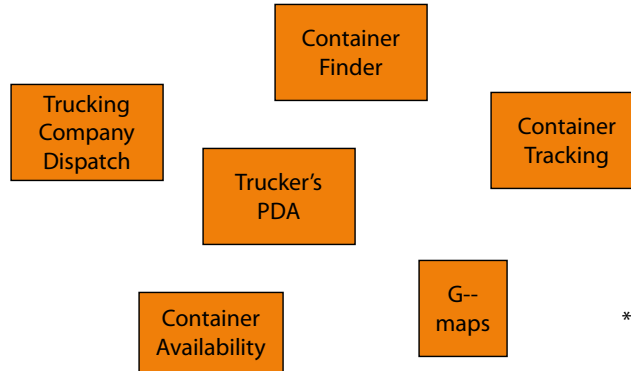
“When pricing policy renewals, apply discounts for multiple policy renewals.”



Web 2.0 Mashup / SAAS*



"Find an empty, available 40-foot dry container owned by ABC Lines that's in a 25-mile radius of Fresno, send directions to the trucker, and notify the shipping line that it will arrive in Oakland on the 20th."



*SAAS: Software
As A Service