Requirements Capture and Specification for Enterprise Applications: a UML Based Attempt

Christine Choppy (LIPN, Université Paris XIII, France)
Gianna Reggio (DISI, Università di Genova, Italy)
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Software development

Main characteristics
- Abstraction
- Structuration

Patterns

Reuse

Requirements Specification

Design

Code

Issues
- Motivation
- Introduction
- Problem Frame notation
- The Enterprise Frame
- Enterprise Applications: the Business (domain) Frame
- The EA Problem Frame
- Requirement Specification
- Conclusions
Software development

Main characteristics
- Abstraction
- Structuration

Patterns

Reuse

Requirements Specification

Design

Code

Use cases

Context diagram

Problem Frames

Architectural styles
- Design Patterns
- Code patterns

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Motivation

- a software development method for enterprise applications
Motivation

- a software development method for enterprise applications
  combines
  - the use of the structural concepts provided by problem frames
Motivation

- a software development method for enterprise applications combines
- the use of the structural concepts provided by problem frames
- the use of the UML notation
Motivation

- a software development method for enterprise applications
  combines
  - the use of the structural concepts provided by problem frames
  - the use of the UML notation

Method

1. Match the problem with a problem frame
2. Develop the UML description
Motivation

- a software development method for enterprise applications combines
  - the use of the structural concepts provided by problem frames
  - the use of the UML notation

Method

1. Match the problem with a problem frame
2. Develop the UML description

Guidelines to develop all the required artifacts dedicated choice of appropriate UML diagrams predefined schemas or skeletons for their contents
Motivation

- A software development method for enterprise applications combines:
  - The use of the structural concepts provided by problem frames
  - The use of the UML notation

Method:

1. Match the problem with a problem frame
2. Develop the UML description

Guidelines to develop all the required artifacts:

- Dedicated choice of appropriate UML diagrams
- Predefined schemas or skeletons for their contents

Modelling the domain, the requirements capture and specification, and their relationships
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications : the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
“Enterprise Applications are about the display, manipulation and storage of large amounts of often complex data and the support or automation of business processes with that data.” (M. Fowler)
“Enterprise Applications are about the display, manipulation and storage of large amounts of often complex data and the support or automation of business processes with that data.” (M. Fowler)

patterns: “ready-to-use” structures drawn from experience, granularity differs
problem frames $\rightarrow$ overall problem structure
architectural styles $\rightarrow$ overall system structure
design patterns $\rightarrow$ design structure
“Enterprise Applications are about the display, manipulation and storage of large amounts of often complex data and the support or automation of business processes with that data.” (M. Fowler)

patterns: “ready-to-use” structures drawn from experience, granularity differs

problem frames → overall problem structure
architectural styles → overall system structure
design patterns → design structure

UML notation, variety, consistency → guidance
Introduction

▶ “Enterprise Applications are about the display, manipulation and storage of large amounts of often complex data and the support or automation of business processes with that data.” (M. Fowler)

▶ patterns: “ready-to-use” structures drawn from experience, granularity differs

- problem frames → overall problem structure
- architectural styles → overall system structure
- design patterns → design structure

▶ UML notation, variety, consistency → guidance

▶ here: a new problem frame for Enterprise Applications, the Enterprise frame, composed of two parts (the Business Frame, devoted to the domain description, and the EA Frame), and UML based associated development method
a small e-commerce site, \( \mu EC \),

- clients buy products chosen in a browsable catalogue and pay using an external payment system
- the products are produced by an external factory
- stocked, and then delivered by a dedicated department of the company
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Problem Frames (M. Jackson)

- can be used by themselves or in combination
- a good tool to tackle with a first structuring of problems
- diagram: involved domains, requirements, design, interfaces
- five basic problem frames, variants

We propose associated development methods
A basic problem frame: transformation

```
Transform machine

Inputs

Outputs

IO relation

I!Y1

TM!Y2

Y3

X

Y4

X
```
A basic problem frame : required behaviour

There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control.
A basic problem frame : required behaviour

There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control.

- embedded systems
- sensors (C2) and actuators (C1)
A basic problem frame : required behaviour

Example: A machine that keeps the temperature of some liquid between given bounds. Then, the temperature of the liquid would be a shared phenomenon controlled by the environment. The corresponding sensor would be a thermometer. Another shared phenomenon would be the state of a burner. That state would be controlled by the machine, i.e., the machine is able to switch the burner on or off.
A basic problem frame: commanded behaviour

Controlled domain

Control machine

Operator

Commanded behaviour

Controlled

Commanding

Controlled

Commanded

Operator

Control machine
There is some part of the physical world about whose states and behaviour information is continually needed. The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form.
A basic problem frame: commanded information

C1 phenomena are referred to later as Events issued by the Real world
E1 are Enquiries from the Enquiry operator
E2 are Display Acts and Error Messages
displayed by the Answering Machine to the Enquiry Operator
Problem Frame notation

| Machine domain | Designed domain | Given domain |

Issues
Motivation
Introduction

Problem Frame notation

The Enterprise Frame
Enterprise Applications: the Business (domain) Frame
The EA Problem Frame
Requirement Specification
Conclusions
Problem Frame notation

- Machine domain
- Designed domain
- Given domain

- Lexical domain
- Biddable domain
- Causal domain
Problem Frame notation

Machine domain  Designed domain  Given domain

Lexical domain  Biddable domain  Causal domain

D1  D1!ph1(args)  D2
    D2!ph2(args)  D2

Interface and shared phenomena
Problem Frame notation and extensions

\[\text{Requirements} \rightarrow \text{D}1 \rightarrow \ldots \rightarrow \text{D}n\]

\[\text{ph}^{-1}(\text{args}) \quad \text{ph}^{-n}(\text{args})\]

A service S groups a set of phenomena internal (non sharable)/ external phenomena i-ph/e-ph.

\[S = \{\text{D}1!\text{ph}^{-i}(\text{...}), \ldots, \text{D}2!\text{ph}^{-k}(\text{...}), \ldots\}\]

A composite phenomena CPH

\[\text{D}3 \quad \text{D}1 \quad \text{D}2 \quad \text{D}4\]

\[\text{D}3!\text{ph}^{3}(\text{...}) \quad \text{D}1!\text{ph}^{1}(\text{...}) \quad \text{CPH}(\text{...}) \quad \text{D}4!\text{ph}^{4}(\text{...}) \quad \text{D}2!\text{ph}^{2}(\text{...})\]
Problem Frame notation and extensions

A service $S$ groups a set of phenomena internal (non sharable)/ external phenomena $i$-$ph$/$e$-$ph$.

\[ S = \{D1!ph1(...), ..., D2!phk(...), ...\} \]
Problem Frame notation and extensions

A service $S$ groups a set of phenomena internal (non sharable)/ external phenomena $i$-$ph$/e-$ph$.

$$S = \{D1!ph^{-i}(...), ..., D2!ph^{-k}(...), ...\}$$

A composite phenomena CPH
Overview

Issues
Motivation
Introduction
Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame
The EA Problem Frame

Requirement Specification

Conclusions
The Enterprise Frame

composed of 2 parts

- **(reuseable) Business (domain) Frame** describing the business (in large) to be managed by the EA business rules (subtle)

- **EA Frame**: “classical” problem frame, EA machine, domains and requirements.
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Enterprise Applications: the Business (domain) Frame

Business Frame domains (three categories):

- **BO** business objects (causal domains)
- **BW** business workers acting in the business (biddable domains)
  - no distinction between working for the business and business actors
- **EC** or **EB** external systems: entities external to the business used for outsourcing some activities (causal or biddable domains)

biddable $\rightarrow 1\text{+ internal phenomena}$
causal $\rightarrow 1\text{+ external phenomena (no internal)}$
Enterprise Applications : the Business (domain) Frame

Business Frame domains (three categories) :

- BO business objects (causal domains)
- BW business workers acting in the business (biddable domains)
  no distinction between working for the business and business actors
- EC or EB external systems : entities external to the business used for outsourcing some activities (causal or biddable domains)

biddable \(\longrightarrow\) 1+ internal phenomena
causal \(\longrightarrow\) 1+ external phenomena (no internal)

business case : cooperation for relevant procedures (composite phenomena)


\[ \mu \text{ECcase} : \text{Business Frame} \]

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram.png}
\caption{Diagram of \( \mu \text{ECcase} \) showing the business frame with modular decomposition.

\textbf{Client}: BW

\textbf{Payment System}: EC

\textbf{Browse}: BW

\textbf{Put money}: BW

\textbf{Get money}: BW

\textbf{Put order}: BW

\textbf{Catalogue}: BO

\textbf{Orders}: BO

\textbf{Manager}: BW

\textbf{Delivery Dept}: EB

\textbf{Factory}: EB

\textbf{Stock}: BO

\textbf{Refill}: BW

\textbf{Delivery}: BW

\textbf{Update}: BW

\textbf{Put order}: BW

\textbf{Client}: BW

\textbf{Put money}: BW

\textbf{Get money}: BW

\textbf{Put order}: BW

\textbf{Catalogue}: BO

\textbf{Orders}: BO

\textbf{Manager}: BW

\textbf{Delivery Dept}: EB

\textbf{Factory}: EB

\textbf{Stock}: BO

\textbf{Refill}: BW

\textbf{Delivery}: BW

\textbf{Update}: BW

\textbf{Put order}: BW

\end{figure}

modular decomposition ...
μEC Business Frame: Put order Business Case

Client

\textit{putOrder}\textsuperscript{BW}

getMess(…)
getProd(…)

Catalogue\textsuperscript{BO}

addOrder(…)

Orders\textsuperscript{BO}

pay(…)
takeProd(…)

Stock\textsuperscript{BO}

Payment System\textsuperscript{EC}

putOrder

putOrder

putOrder

putOrder

putOrder

putOrder
BF associated (UML) Business Model

- precise method of Astesiano-Reggio
  - A class diagram: a class for each BF domain
    - biddable → active
    - causal → passive
    - stereotypes: "business object", "business worker" and "external system", cf. domain markings BO, BW and EB, EC.
  - internal phenomenon → private operation
  - autonomous acts (stereotype "A")
  - external phenomena: operations of the corresponding class
BF associated (UML) Business Model

precise method of Astesiano-Reggio

- A class diagram: a class for each BF domain
  - biddable $\rightarrow$ active
  - causal $\rightarrow$ passive
  - stereotypes: ≪bo≫ ("Business Object"), ≪bw≫ ("Business Worker") and ≪es≫ ("External System"), cf. domain markings BO, BW and EB, EC.

- internal phenomenon $\rightarrow$ private operation
- autonomous acts (stereotype ≪A≫)

- external phenomena: operations of the corresponding class

- behaviour: active class $\rightarrow$ statechart
- passive class $\rightarrow$ operations behaviour
precise method of Astesiano-Reggio

- A class diagram: a class for each BF domain
  - biddable $\rightarrow$ active
  - causal $\rightarrow$ passive
  - stereotypes: $\ll bo \gg$ (“Business Object”), $\ll bw \gg$ (“Business Worker”) and $\ll es \gg$ (“External System”), cf. domain markings BO, BW and EB, EC.
- internal phenomenon $\rightarrow$ private operation
- autonomous acts (stereotype $\ll A \gg$ )
- external phenomena: operations of the corresponding class

- behaviour: active class $\rightarrow$ statechart
  - passive class $\rightarrow$ operations behaviour

- A description of each business case: UML collaboration and activity diagram
**μECBusiness Model**: class diagram

---

**Client**
- `getMess(String)`
- `shown(Set(Product))`
- `lookCatalogue()`
- `putOrder(ProdID,Int,Date,ClientID)`

**Manager**

**Orders**
- `os: Set(Order)`
- `addOrder(ClientID,ProdID,Int,Date)`
- `delivered(OrdID)`
- `first(): Order`

**Factory**
- `deliverProd(ProdID,Int)`

**Catalogue**
- `ps: Set(Product)`
- `getProd(ProdID): Product`
- `contents : Set(Product)`

**Product**
- `id: ProdID`
- `price: Euro`
- `descr: String`

**Order**
- `client: ClientId`
- `prod: ProdId`
- `quant: Int`
- `date: Date`
- `id: OrderId`
- `status: {wait,deliv}`

**DeliveryDept**
- `deliver(Order)`
- `delivered(OrdID)`

**PaymentSystem**
- `pay(Euro,ClientID)`

**Stock**
- `receiveProd(ProdID,Int)`
- `availableProd(ProdID): Int`
- `takeProd(ProdID,Int): Bool`
μECBusiness Model: Catalogue Behaviour View

method newProd(PI,E,S)
    {P = create(Product); P.id = PI; P.price = E;
     P.descr = S; ps = ps->including(P)}
context deleteProd(PI) post :
    not ps.id ->includes(PI)
context changePrice(PI,E) post :
    ps->select(id = PI).price = E
μECBusiness Model: Put Order business case

collaboration

Client

CI: ClientID
PI: ProdID
Q: Int

PaymentSystem

CT

Catalogue

ST

Stock

ORS

Orders

Put Order

collaboration
μECBusiness Model: Put Order business case

activity diagram

- C.putOrder(PI,Q,CI,now)
  - False: ST.takeProd(PI,Q)
    - False: C.getMess("Refused")
    - True: PS.pay(Q*CT.getProd(PI).price,CI)
      - False: C.getMess("Refused")
      - True: ORS.addOrder(PI,Q,CI,now)
        - C.getMess("Accepted")

- True
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
The EA Problem Frame
A BO-D is a designed domain with a full model of the business object BO that EA works with. An ES-I/BW-I domain corresponds to some limited information about ES/BW that EA needs to interact with it (e.g., its name and the way to access it).
some checks

- Each enclosed domain must be connected by a chain of business cases having a common participant with an outside domain
- There should be at least one domain outside
- ...
Issues
Motivation
Introduction
Problem Frame notation
The Enterprise Frame
Enterprise Applications: the Business (domain) Frame
The EA Problem Frame
Requirement Specification
Conclusions
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
precise method of Astesiano-Reggio

- each frame domain $\rightarrow$ a class diagram +
  (biddable $\rightarrow$ active, causal $\rightarrow$ passive)
- stereotypes ($\ll bo \gg$, $\ll bw \gg$ and $\ll es \gg$)
- shared phenomena operations
- a class for the EA :
  shared external phenomena $\rightarrow$ operations
precise method of Astesiano-Reggio

- each frame domain \(\rightarrow\) a class diagram +
  (biddable \(\rightarrow\) active, causal \(\rightarrow\) passive)
  stereotypes (≪bo≫, ≪bw≫and ≪es≫)
  shared phenomena operations
  a class for the EA:
  shared external phenomena \(\rightarrow\) operations

- A complete behaviour definition of all the ≪bo≫ classes and possibly others
Requirement Specification

precise method of Astesiano-Reggio

► each frame domain \(\rightarrow\) a class diagram +
  (biddable \(\rightarrow\) active, causal \(\rightarrow\) passive)
  stereotypes (\(\ll bo\), \(\ll bw\) and \(\ll es\))
shared phenomena operations
a class for the EA :
shared external phenomena \(\rightarrow\) operations

► A complete behaviour definition of all the
  \(\ll bo\) classes and possibly others

► A use case diagram, use cases description (actors :
  domains connected with EA), use case behaviour :
  a statechart associated with the class EA, such that
  – events : timed or call events,
  – conditions w.r.t. its attributes only
  – actions : attribute upd or actors operations calls
μEC

Requirements Capture and Specification for Enterprise Applications: a UML Based Attempt
Christine Choppy (LIPN, Université Paris XIII, France)
Gianna Reggio (DISI, Università di Genova, Italy)

Issues
Motivation
Introduction
Problem Frame notation
The Enterprise Frame
Enterprise Applications: the Business (domain) Frame
The EA Problem Frame
Requirement Specification
Conclusions

Product
id: ProdID
price: Euro
descr: String

ClientRecords
correct(ClientID, String): Bool
newPsw(): String

Manager
...

<<system>> microEC
cls: ClientRecords
putOrder(ProdID, Int, ClientID, Date)
login(ClientID, String)
getOrderToDeliver()
delivered(OrdID)
register(ClientID)
refilledStock(ProdID, Int) ...

<<bo>>
Catalogue
ps: Set(Product)
getProd(ProdID): Product
contents: Set(Product)

<<bo>>
Orders
os: Set(Order)
addOrder(ClientID, ProdID, Int, Date)
first(): Order

<<bo>>
Stock
receiveProd(ProdID, Int)
availableProd(ProdID): Int
takeProd(ProdID, Int): Bool

<<es>>
PaymentSystem
pay(Euro, ClientID)

<<es>>
DeliveryDept
deliver(Order)

<<es>>
Factory
getMess(String)
show(Set(Product))
passWdIs(String)

<<bw>>
Client
getMess(String)
show(Set(Product))
passWdIs(String)

<<bw>>
Stock
receiveProd(ProdID, Int)
availableProd(ProdID): Int
takeProd(ProdID, Int): Bool

<<bw>>
Manager
correct(ClientID, String): Bool
newPsw(): String

<<bw>>
Catalogue
ps: Set(Product)
getProd(ProdID): Product
contents: Set(Product)
os: ...   ...
microEC

<<es>>
PaymentSystem
pay(Euro, ClientID)

<<es>>
DeliveryDept
deliver(Order)

<<es>>
Factory
getMess(String)
show(Set(Product))
passWdIs(String)

<<bw>>
Client
getMess(String)
show(Set(Product))
passWdIs(String)

<<bw>>
Stock
receiveProd(ProdID, Int)
availableProd(ProdID): Int
takeProd(ProdID, Int): Bool
μEC Requirement Specification: Use Case Diagram

C: Client
- Register
- Deliver
- Put Order
- Browse
- Update
- Refill

M: Manager
- microEC
- DP: Delivery Dept.
- ORS: Orders
- ST: Stock
- PS: Payment System
- CT: Catalogue
- F: Factory
Use Case Description

\mu ECR

Requirement Specification: Put Order

**Use Case Description**

- **login(Cl, x)**
  - [not cls.correct(Cl, x)]
  - /C.getMess("wrong psw")

- **putOrder(PI, Q, CI, now)**
  - [ST.get(Prod(PI, Q) and PS.pay(Q*CT.getProd(PI).price))]
  - /C.getMess("Accepted")

- **putOrder(PI, Q, CI, now)**
  - [not ST.getProd(PI, Q)]
  - /C.getMess("Refused")

- **putOrder(PI, Q, CI, now)**
  - [not ST.getProd(PI, Q) and not PS.pay(Q*CT.getProd(PI).price)]
  - /C.getMess("Refused")
Overview

Issues

Motivation

Introduction

Problem Frame notation

The Enterprise Frame

Enterprise Applications: the Business (domain) Frame

The EA Problem Frame

Requirement Specification

Conclusions
Conclusions

- a software development approach for Enterprise Applications
  1. match the Business Frame and the EA Frame with the problem
  2. model the various frames parts following the proposed UML diagrams
Conclusions

- a software development approach for Enterprise Applications
  1. match the Business Frame and the EA Frame with the problem
  2. model the various frames parts following the proposed UML diagrams

- problem frames are very good at providing a first requirement structure that is invaluable to start the analysis of a problem and understand its nature. A means to reuse experience helpful to start a complex problem analysis with some structuring concepts in mind.
Conclusions

▶ a software development approach for Enterprise Applications
  1. match the Business Frame and the EA Frame with the problem
  2. model the various frames parts following the proposed UML diagrams

▶ problem frames are very good at providing a first requirement structure that is invaluable to start the analysis of a problem and understand its nature. A means to reuse experience helpful to start a complex problem analysis with some structuring concepts in mind.

▶ reduce the time spent to decide which UML constructs to use and how to model the domain and the requirements
Conclusions (follwd)

• a more direct path to the UML models, improved models quality (relevant issues are addressed, a uniform style is offered)
Conclusions (follwd)

- a more direct path to the UML models, improved models quality (relevant issues are addressed, a uniform style is offered)
- requiring that the modelling of the domain and of the problem is accompanied by their "framing" helps the developer to manage complex things, and offers a support to navigate the complex UML models produced
Conclusions (follwd)

- a more direct path to the UML models, improved models quality (relevant issues are addressed, a uniform style is offered)
- requiring that the modelling of the domain and of the problem is accompanied by their "framing" helps the developer to manage complex things, and offers a support to navigate the complex UML models produced
- essence: use and combination of the relevant underlying concepts, possible different notations (e.g., a graphical language for a different level of abstraction, UML, formal specification language, etc).