Development Methodology

Version 061013
What shall we learn in school today?

- Why your java programs fail
  - The solution
    - thinking in a way corresponding to how your program will work
- Methodology
  - Some useful tips
- Dialectics – making conflicts drive the development
  - early conflicts are less dangerous
  - people with complementary competence is fruitful
  - complementary views help see the whole picture
- The need for harmonization
Why your normal Java program fails

- or how to think in correspondence with how the computer works
Agreeing on which movie to watch

- A group of persons are going to agree on which movie to watch this evening
- There is only a small number of movies (less than the number of persons). One can assume that the decision can be based on democratic principles: the movie with most votes win.
- We will use three different ways of communication:
  - (half-) duplex two-party telephony (synchronizing communication)
  - conference call (synchronous communication)
  - SMS (asynchronous communication)
Synchronizing communication

- pers1 is the master
- pers2 and pers3 are slaves
- pers1 cannot perform anything while pers2 and pers3 are trying to decide for themselves
Synchronous communication

- **pers1** is the central
- **pers2** and **pers2** are co-workers
- neither of the persons can do anything while the communication lasts

- (this is informal MSC since MSC-2000 have no mechanisms for synchronous
Asynchronous communication

- *pers1* is the central
- *pers2* and *pers2* are co-workers
- *pers1* can do other kinds of work while *pers2* and *pers3* decide their opinions
- *pers2* and *pers3* can make up their opinion in parallel
Threads

- Threads are flows of control
  - the metaphor is that the threads go through the web of objects like a thread in the fabric of a shirt that is sewn

- Threads are said to be “light weight processes”?! 
  - threads are not operating system tasks
  - threads refer to the same address space (object space)
  - threads must be considered concurrent

- What is the canonical mental model of threads?
  - this is a very hard question, and we shall try and look at this ....

- Are there simple ways to ensure thread-safe programming in Java?
  - there is no simple way, but some approaches are safer than others

- Threads can be used to enforce priority
  - but be conscious about what you can achieve through priority
Threads 1

- one thread
- in fact the whole system is sequential!
- anybody can program this in Java
Threads 2

- there are two independent threads of control
- in fact there could be even more since pers2 and pers3 could have had other business to attend to!
- as it is, it is a fairly simple “fork” / “join” and quite simple to program
- such a local fork and join is still almost sequential
Threads 2 (more)

- **Problems**
  - technical
  - conceptual

- If *pers1* following messages *movie* also updates the count for each movie, there is a concurrent update problem.

- Who are the threads? Are they concepts?
Threads 3 (JavaFrame / UML / SDL)

- `pers1`, `pers2` and `pers3` are all ActiveObject
- They are StateMachines
- `pers1` is Leader
- `pers2`, `pers3` are Followers
- There is one (or more Threads) controlled by Schedulers
- Schedulers are hidden for the programmer
Object Orientation

- The objects are the performers / executors
- They themselves perform their methods
- In Java in fact the Threads are executing the methods
- This means that the same object may be executed from different Threads, but conceptually being one active object in itself
Why we make errors with Threads in Java

- You use another Thread to achieve higher speed
  - usually wrong, if it is on the same machine, it will slow the machine down, not speed it up
- You use several Threads, but lose track of them because they are not associated closely with concepts
  - You use several Threads, but your concept of the ActiveObjects are not associated with them
- You are using a synchronizing approach and believe that the program is essentially sequential, but alas...
  - another programmer does the same, but your Threads interact without synchronization on some obscure common object
- You know about Thread problems and use synchronized methods to a large degree
  - either you run into deadlock, or very inefficient programs
Why use several Threads in Java?

- There are real external stimuli that should be handled according to interrupts
  - it would be better if all (or many) interrupts could be handled by the same Thread since Threads consume resources
- There are some parts of the system that requires better priority than the rest
  - Giving priority could give improved performance
    - duration of transitions vary considerably
  - Certain urgent operations are done in time,
  - but priorities should not be used in reasoning about the overall functionality
- The system is physically distributed over several machines
  - Then it is obvious that we need more than one JVM (Java Virtual Machine)
Why UML 2 / JavaFrame is different

- The predominant model of UML 2 State Machines / JavaFrame is that of telecom:
  - concurrency is an opportunity, not a mere threat
- Execution logic is tied to the programming concepts
- Execution performance discriminates between the programmers’ level and the execution platform
  - Threads are dealt with separately from the functional logic
- High degree of independence implies:
  - parallel design possible
  - modifiability / flexibility
  - early simulation / prototyping
  - known validation approaches
- In short: dependability with less efforts
Methodology

originally from Bræk&Haugen
"Engineering Real Time Systems” from 1993
made in the SISU project
The goals of the design

- **Readable** in the deep semantic sense that it supports collective understanding in a project team. It should support unambiguous communication among project members and in-depth understanding by the individual.

- **Analyzable** in the sense that properties can be derived and compared with requirements.

- **Implementable** in the sense that the described functionality can be implemented in a way that satisfies non-functional requirements.
S-rules on concurrency

- Model independent and parallel behaviors as separate processes.
  - Parallelism is a real world fact
  - Parallel processes do not interfere with each other
  - Concurrent processes help to achieve encapsulation and modularity
  - Concurrency implies a logical separation of substance
S-rules on structuring

- **interconnections**
  - Use one channel and/or signal route to carry each independent and concurrent interaction dialogue.

- **system and environment**
  - For the elements at the periphery of your concern, place them inside the system if you wish to describe their behavior in detail.
  - If you are merely interested in their signal interface, place them in the environment which means they will not be identified explicitly in UML.
S-rules on purpose of composite structures

- Gradual approach to detail
- Units of reuse and repetition;
- Encapsulation of layering;
- Encapsulation of independent adaptation and change;
- Limited scope of process creation and communication;
- Correspondence with the physical system.
The Actor architecture

The controlling state machine – often performs routing

The inner actors – at the leaves, these are state machines, too

SimpleRouterMediator or MultiCastMediator
Does the pattern apply to the Basic Service?

- The controlling state machine – often performs routing

- The inner actors – at the leaves, these are state machines, too

- SimpleRouterMediator or MultiCastMediator
The purpose of Ports

- Ports represent interfaces
  - that are separate from their owner

- Ports may therefore remain while the owner is exchanged
  - change implementation during runtime without having to traverse the whole architecture to update the connectors

- Ports represent the only way to communicate with their owner
  - encapsulation

- Ports are often used for simple routing
  - while more complicated routing is done by state machines

- Use Ports always
  - even though UML 2 does not require them
S-rules on state machines

- **state orientation**
  - Represent what the environment may distinguish as control states of the process, as states in the process graph.

- **decisions**
  - Critically review all decisions to ensure that they are not symptoms of undesirable state hiding.

- **signal set**
  - Represent what the environment may distinguish as different control signals by different signal types

- **control flow**
  - Branch on input signals in states rather than on decisions.
Does this apply to GposController?

- State orientation
- Decisions
- Signal set
- Control flow

[Diagram of GposController state machine with labels and transition conditions]

The_sms_is_now: study konco system and my-client

[Code snippet for GposController operations]
S-rules for the use of data

- Proper use of data
  - non-decisive data
    - when the process graph structure is not dependent on the data values
  - context knowledge
    - to keep information about the situation and structure of the environment
  - loop control data
    - to control loops that are not terminated by specific signals

- shared data
  - Introduce special processes to encapsulate shared data.
  - Encapsulate data needing independent access in separate processes.
Does this apply to PositionUser?

- staticId
- visualId
- lastpos_time
- TimerMsg posreq
- validpos
- xcoord
- ycoord
Dialectic System Development

how to take advantage of conflicts
The language maturity staircase

- **SDL**  
  **Automatic semantics**: machine-oriented

- **MSC**  
  **Formal Semantics**: mathematical notation

- **UML**  
  **Semantics**: explained understanding

- **Visio**  
  **Syntax**: given syntax with illustrative add-ons

- **Doc. figs.**  
  **Illustrations**: one notation for each picture, natural language resemblance critical
German view

French view

Italian view

American view

press view

http://www.theregister.co.uk/2006/07/13/zidane_headbutt_outrage/
Access Control System
Domain Statement

- **Area of concern**
  - Access control has to do with controlling the access of users to access zones. Only a user with known identity and correct access right shall be allowed to enter into an access zone. Other users shall be denied access.

- **Stakeholders**
  - Users of the system, those responsible for the security of the access zones.

- **Services**
  - The user will enter an access zone through an access point.
  - A supervisor will have the ability to insert new users in the system.
  - Users shall be able to change their secret code.
    - The authentication of a user shall be established by some means for secret personal identification (code). The authorisation is based upon the user identity and access rights associated with the user.
Service: Change PIN

- Informal specification:
  - "Users shall be able to change their secret code"
Make More Precise

- **formalize**
  - move the description to a more formal language

- **refine**
  - narrow
    - add more properties to make it less ambiguous
  - supplement
    - add new aspects, consider supplementary scenarios
service PIN Change

- Users shall be able to change their personal identification
- The User shall be able to choose his new PIN
- The Card shall be validated by the old PIN before a new PIN can be given. The new PIN shall subsequently also be validated.
Supplementing

Interaction occurrence (use)

break_expression

continuation
The Access Control Context as UML Class

```
<table>
<thead>
<tr>
<th>ACContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd UserAccess</td>
</tr>
<tr>
<td>sd PINChange</td>
</tr>
<tr>
<td>sd OpenDoor</td>
</tr>
<tr>
<td>sd NewUser</td>
</tr>
<tr>
<td>sd EstablishAccess</td>
</tr>
<tr>
<td>sd GivePIN</td>
</tr>
</tbody>
</table>
```

- **class name**
- **defining interactions**
- **utility interactions**
- **composite structure**
System services

sd New_User

NewUser  Supervisor  ACSystem
ref AC NewUser

Idle

ref
EstablishAccess
("NotSupervisor")

alt

[Wrong PIN]

"Sorry()"

[PIN OK]

CardId()

ref
GivePIN

Card(Cid, PIN())

Idle

sd UserAccess

NewUser  ACSystem
ref AC UserAccess

Idle

ref
EstablishAccess
("Illegal PIN")

opt

"Please Enter()"

P IN OK

ref
OpenDoor

Idle

Similarities
Need for generalization: Entry

- On what connectors is EstablishAccess applied?
  - between the AccessPoint and a normal User
  - between the Console and the Supervisor user
Harmonizing: Entry, AccessPoint and Console

**Entry**
- Panel: Lifeline
- Controller: Lifeline
+ Entry_EstablishAccess(): sd
+ Entry_GivePIN(): sd

**AccessPoint**
- Door: Lifeline
+ AP_UserAccess(): sd

**Console**
- Panel: Lifeline
- Controller: Lifeline
+ Console_NewUser(): sd
+ Console_PINChange(): sd

**ACSystem**
- AccessPoint: Lifeline
- Console: Lifeline
- Authorizer: Lifeline
+ AC_UserAccess(): sd
+ AC_PINChange(): sd
+ AC_NewUser(): sd
+ AC_EstablishAccess(): sd
+ AC_OpenDoor(): sd
+ AC_GivePIN(): sd

**ACContext**
- ACSystem: Lifeline
- User: Lifeline
- Supervisor: Lifeline
- NewUser: Lifeline
+ UserAccess(): sd
+ PINChange(): sd
+ NewUser(): sd
+ EstablishAccess(): sd
+ OpenDoor(): sd
+ GivePIN(): sd

**ACSystem**
- AccessPoint: Lifeline
- Console: Lifeline
- Authorizer: Lifeline
+ AC_UserAccess(): sd
+ AC_PINChange(): sd
+ AC_NewUser(): sd
+ AC_EstablishAccess(): sd
+ AC_OpenDoor(): sd
+ AC_GivePIN(): sd

**Authorizer**
- CardId():
- Digit():
- Code(Cld, PIN)()

**AC_GivePIN**
- Code(Cld, PIN)()

**Loop<0,3>**
- “Try Again”()
- AccLevel(m)()
- AccLevel(n)()
- CardOut()

**Console**
- msg(x):()

**Entry**
- Entry_EstablishAccess(x):()
- Entry_GivePIN():

**Idle**
- PIN OK
- UserAccess() : sd
- PINChange() : sd
- NewUser() : sd
- EstablishAccess() : sd
- OpenDoor() : sd
- GivePIN() : sd

**ACSystem**
- Lifeline
- ACSystem
- AccessPoint
- Console
- Authorizer
- User
- Supervisor
- NewUser

**ACContext**
- ACSystem
- User
- Supervisor
- NewUser
- Entry
- AccessPoint
- Console
- Authorizer
- Panel
- Controller
The Entry class hierarchy
Detailing through commutative decomposition
Change PIN

Decomposition
Commutative Decomposition
Verification 1: Model checking PIN Change in Panel
Panel: UML State Machine, GivePIN as a method

```
sm Panel

AllPanel

NoCard

OneCard

H

msg(t)/"t"
cardout / cardout
cardid(cid) / GivePIN, ^code(cid,pin)
givePIN / GivePIN, ^code(cid,pin)
```
Model checking continued....

sd Entry_EstablishAccess(String txt)

Panel

Controller

Idie

Cardid()

Digit()

Entry_GivePIN

ref

Code(Cid,PIN()) Code(Cid,PIN())

loop<0,3>

"Try Again"()

AccLevel(m())

GivePIN()

Digit()

Entry_GivePIN

ref

Code(Cid,PIN()) Code(Cid,PIN())

CardOut()

AccLevel(n())

alt

msg(bt())

msg(bt())

Idle

PIN OK

sm Panel

AllPanel

NoCard

cardout / cardout
cardid(cid) / givePIN, ^code(cid,pin)

OneCard

givePIN / givePIN, ^code(cid,pin)
Model checking continued.... ....

When `EstablishAccess` has elapsed, `Panel` is in state `NoCard`, but it receives `GivePIN`!
Harmonizing

We decide to move CardOut from EstablishAccess to the end of PIN_Change
Verification 2: AccessPoint’s Controller

sd: User Access vs sm: Controller = OK!

Are we then certain that AccessPoint’s Controller is perfect?

The User opens the door exactly when the timer expires. door+opened in input port.
Verification 3: Detecting default transitions

- Sequence Diagrams are not suited to uncover all possible variants of interaction
- State Machines (JavaFrame or UML 2) supported by automatic techniques can find unwanted signaling combinations
- There are several techniques to evaluate projections of processes to uncover the complexity of the software
Dialectic Software Development

- Software Development is a process of learning
  - once you have totally understood the system you are building, it is done
- Learning is best achieved through conflict, not harmony
  - discussions reveal problematic points
  - silence hides critical errors
- By applying different perspectives to the system to be designed
  - inconsistencies may appear
  - and they must be harmonized
- Inconsistencies are not always errors!
  - difference of opinion
  - difference of understanding
  - misunderstanding each other
  - a result of partial knowledge
- Reliable systems are those that have already met challenges