Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Together. Free your energies

FP @ Capgemini

Betsy Pepels - CSD /Advanced Solutions

FP meeting, January 8, 2010

Agenda

- Introducing myself and our department
- Our approach: Functional Model Driven Development
- Challenges
  - Example real life challenge
  - FP inspired solution
  - More real life challenges asking FP approach
- Moving to FP languages and development tools
- Questions and discussion

Introducing myself and our department

- About me
- Model Driven Development (own method FMDD)
- Large, complex projects
- Mostly public sector
- Domain: compliancy
  - Social benefits, salaries, pensions, mortgages, insurances

Our approach:
Functional Model Driven Development

- Language Engineers define a Business Specific Language (BSL)
- Transformation Engineers build transformations that translate the BSL to the target code: they build the *software factory*
- Domain Engineers define the business of the customer using the BSL: they make *functional specifications*
- software factory generates the application
- whole process supported by dedicated tooling
  - also for testing on BSL level

BSL: Objects

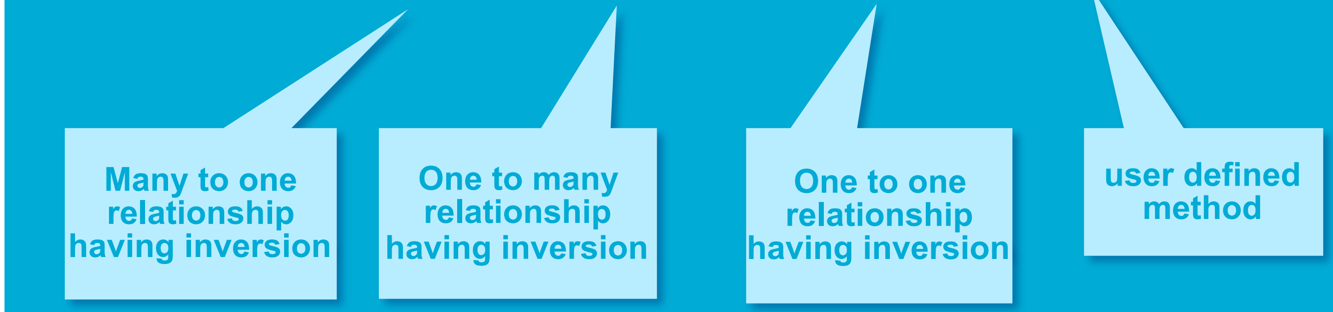
```
Citizen {
        partner :: Citizen
        address :: Address
        income :: Integer
}

Address {
        occupant:: [Citizen]
        street :: String
        zip:: String
}
```

Example parts of a BSL: object navigation, rule definition

- Example Social Benefit rule:
  - A citizen recieves an extra benefit if the total income of his/her household is below the threshold

- Specification

IF citizen.address.occupant.income.sum < threshold THEN <computation>



Many to one relationship having inversion

One to many relationship having inversion

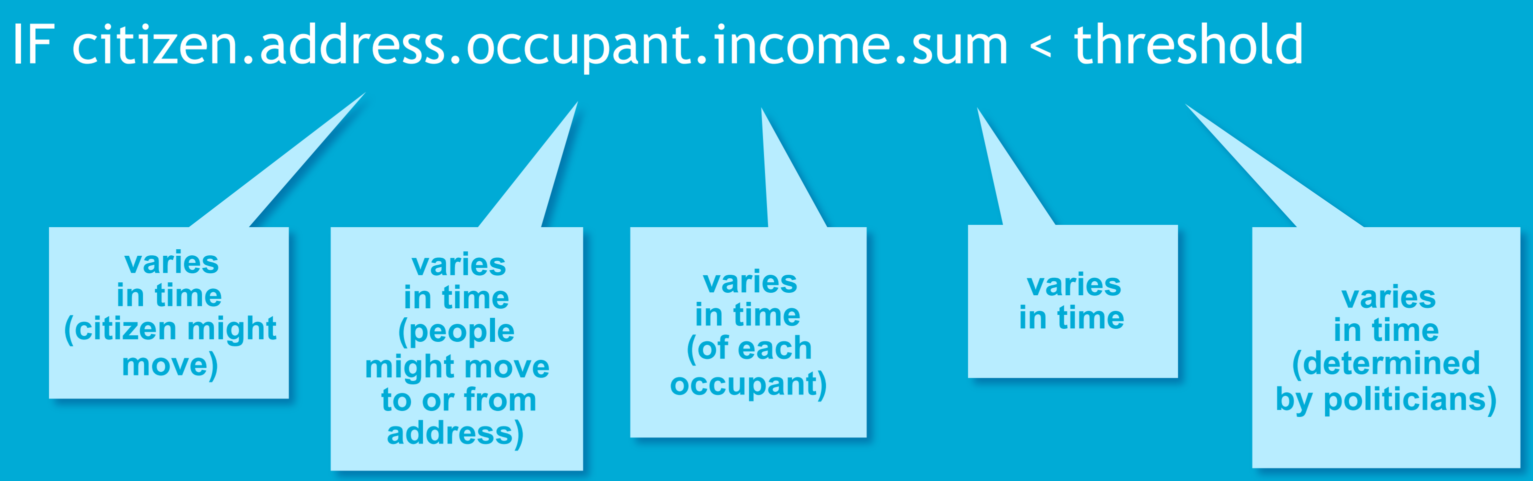
One to one relationship having inversion

user defined method

Example real life challenge

- Time dependent computing
- SOA/EDA architecture
- Consequences of SOA/EDA for time dependent computing

Time dependent computing introduction



IF citizen.address.occupant.income.sum < threshold

varies in time (citizen might move)

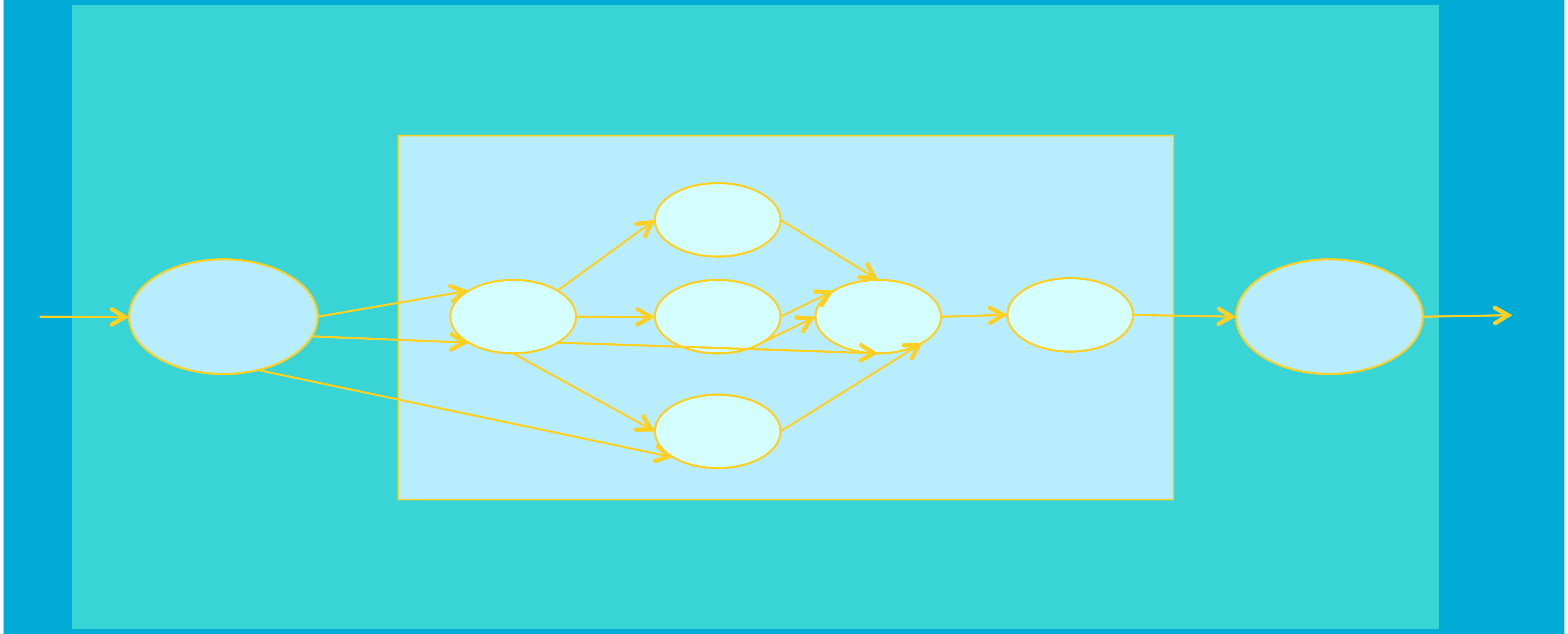
varies in time (people might move to or from address)

varies in time (of each occupant)

varies in time

varies in time (determined by politicians)

Basic SOA/EDA Architecture

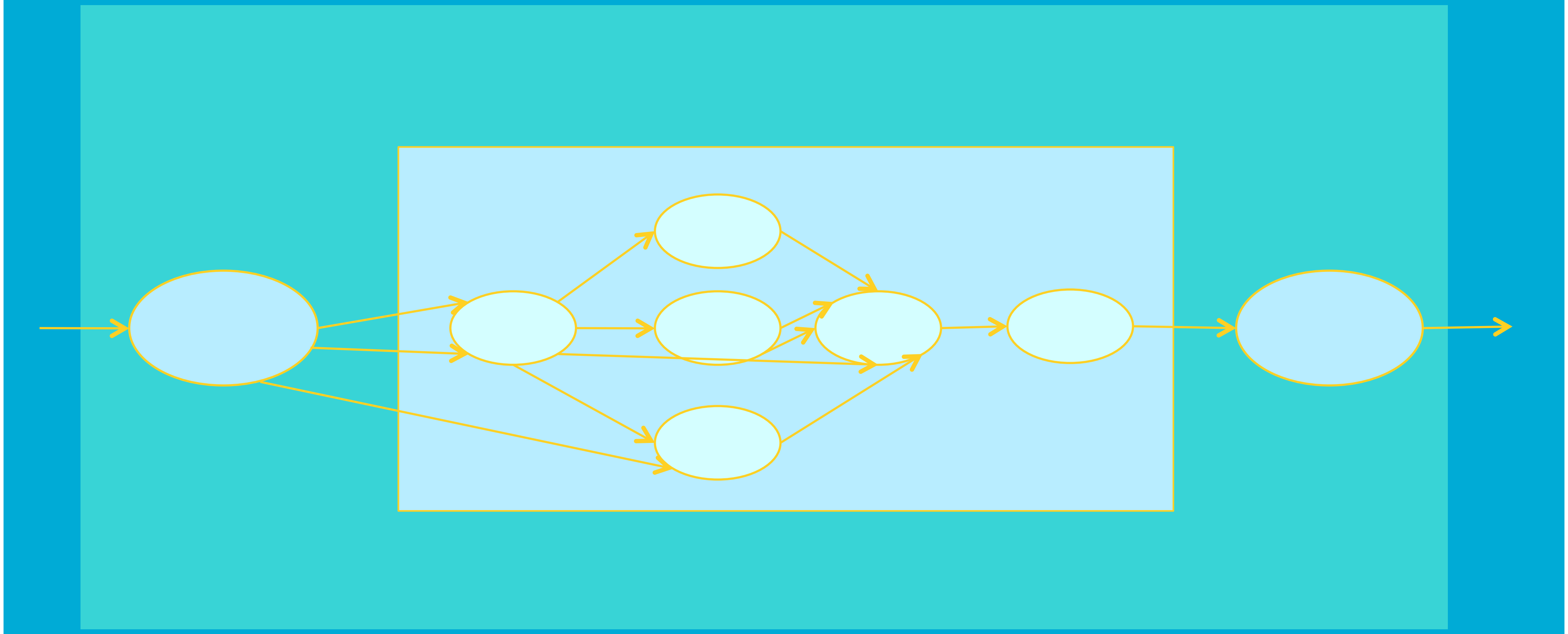


SOA/EDA Architecture cont'd

- designed by customer
- important aim: scalability
- distributed computation
- 1 incoming event in a service can lead to 0, 1, 2 or even more outgoing events
- flow is strictly from *in* to *out*, no feedback
- final result should be independent from actual event flow
- highly parallel: two physical locations, many cores, many instances of each service

SOA/EDA Architecture cont’d

- Example event flow



SOA/EDA Architecture contn'd

- Example incoming events
  - Citizen C moves to address X1 on 30-12-2009, reported on 03-01-2010
  - Citizen A passes away on 05-01-2010, reported on 06-01-2010
  - Citizens A and B marry on 29-12-2009, reported on 07-01-2010
  - Citizen C moves to address X2 on 12-07-2009, reported on 08-01-2010
- Example internal events
  - A and B stop to be partners, from 05-01-2010, reported on 06-01-2010
  - The rent of C is € 441, from 30-12-2009, reported on 03-01-2010
  - The rent of C is € 368, from 01-08-2009, reported on 08-01-2010

A closer look at time dependency: 3 time axes

- Valid time: used to record the actual value
  - Sheila moves to address P on 12-01-2010
    - actual date of move
- Reporting time: used to record when the system could have known the value
  - move of Sheila is reported to the system on 07-01-2010
    - from 07-01-2010 the system could have known this
- Transaction time: used to record when it is actually registered in the system
  - move of Sheila is registered in the system on 08-01-2010
    - from 08-01-2010 the system actually computes with this value

Non solution

- Domain Engineer is responsible for timing aspects
  - to little expertise
  - error prone
  - repeating work for every object/attribute/event

Implemented solution: lifting Timed Object Model

- borrowed from FP philosophy

basic idea:

- transform BSL objects to timed objects
  - every atrribute is separately timed (requires 7 time stamps for each)
  - leads to 3-dimensional time “cubes”
- make timed counterpart for each basic operation of BSL
  - if-then-else, dotting, operators, …
  - leads to loops over time cubes
- transform BSL elements to timed counterparts
  - including strategy for transforming user defined methods

Technology

- .NET framework obligatory
- C# 3.0 with LINQ, for implementation of
  - timed operations
  - persistence (Object Relational Mapping)
- we managed to do it, but …
  - we missed the expressiveness of pure lazy functional languages

More real life challenges asking FP approach

New FMDD tooling:

- Executable semantics of BSL
- Evolution of BSL
- Domain Specific Language for transformation specification
- Automated statistical testing (like GAST)
- …

Moving to FP languages and development tools

- A pure lazy language just isn't enough

Imagine a small scale MDD/FP group (~50 developers).

To get and keep it, we need:

- some guarantee of continuity
- some IDE
  - straightforward installing, updating, …
  - code completion, refactoring, debugging, …
- integration with other languages and to data bases
- some delivery process
- introductory books, working conferences, courses, education, …

Questions and discussion

- Our question!
  - we think cooperation between industry and academia is necessary and fruitful
  - Capgemini wants to take responsibility – do you have suggestions?

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

COLLABORATIVE BUSINESS EXPERIENCE

Together. Free your energies

www.capgemini.com