Recent achievements in automated database design based on business process models

D. Banjac, D. Brdjanin and G. Banjac
University of Banja Luka, Bosnia & Herzegovina
About this presentation

- Introduction
- Domain specific languages
  - Examples of DSLs
- Business modeling language
  - Implementation
  - Approach
  - Metamodel
- Example of automated generation of CDM
- Conclusion
Introduction

• There are papers that present automated generation of the initial conceptual database model (CDM) based on different business process modeling notations

• **Source**: Business process model (BPM) represented by different business process modeling notations (UML activity diagram, BPMN, etc.)

• **Target**: CDM represented by UML class diagram

• In order to achieve metamodel independency we implemented **Domain specific language** (DSL) as intermediate layer, between source and target model
Domain specific languages

• “Domain Specific Language (DSL) is a computer programming language of limited expressiveness focused on a particular domain.” (Martin Fowler)

• **Domain** - an area or sphere of knowledge, influence, or activity

• Two main forms:
  – **External DSL** (*free standing*) – language parsed independently of the host general purpose language (GPL), e.g. regular expressions, CSS
  – **Internal DSL** (*embedded*) – designed and implemented using GPL, particular form of API in a host GPL, e.g. JMock

• Each DSL consists of:
  – **Abstract syntax** – defines domain concepts, their attributes and relations
  – **Concrete syntax** – language syntax in its representation that we see
  – **Semantics** – usually added with interpretation or code generation
Example of DSLs

- Regular expressions, SQL, CSS, make, rake, ant, BPMN etc.

```sql
SELECT * FROM Course c WHERE c.ECTS > 6;
```

```regex
/^[a-z0-9_\.\-\+]+@[\da-z\-\.]+\.(\[a-z\.]\{2,6}\)$/
```

```xml
<project name="AnExampleProject" default="jarit" basedir=".">
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="distrib" location="distrib"/>
  <target name="compile" description="compile your Java code from src into build">
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
  <target name="jarit" depends="compile" description="jar it up">
    <jar jarfile="${distrib}/AnExampleProject.jar" basedir="${build}"/>
  </target>
</project>
```

```bash
hellomake: hellomake.c hellofunc.c
gcc -o hellomake hellomake.c hellofunc.c -l.
```
Implementation of BML

- DSL called **business modeling language** (BML) is developed
- BML describes (so far identified) **semantic capacity** of the business process models
- BML provides **independency** from different business process modeling notations used for modeling the source model
- Identified **transformation rules** were used to implement generator which transforms BML to initial CDM
- We used **Xtext** framework for implementation of DSL
- We used **Xtend** for code generation
- We, also, implemented generators to transform BPM represented by BPMN and UML activity diagram to BML (**Acceleo** implementation)
Approach

Source → Target

Source → BML → Target
Metamodel of BML

- Abstract syntax tree (AST) is represented by Ecore model
Rules for mapping BPM → CDM

**Classes**

$T_1$

$T_2$

$T_3$
Rules for mapping BPM → CDM

Associations

$T_4$

$T_5$

$T_6$

$T_7$

$T_8$

$T_9$
Source model

- Order processing - BPMN
BML

• Order processing - BPMN

/* Participants and roles */
participant Supplier
role Stockman of Supplier
role Driver of Supplier
role Commercial of Supplier
participant Customer

/* Objects */
object OrderHeader
object Response
object CatalogItem
object Delivery
object Request
object StockItem
object OrderDetail

/* ObjectReference */
objectReference ExistingStockItem_ references StockItem existing
objectReference ExistingCatalogItem_ references CatalogItem existing
objectReference OrderHeader_ references OrderHeader
objectReference Delivery_New references Delivery[New]
objectReference OrderDetail_New references OrderDetail[New]
objectReference StockItem_Prepared references StockItem[Prepared]
objectReference OrderHeader_Accepted references OrderHeader[Accepted]
objectReference OrderHeader_New references OrderHeader[New]
objectReference OrderHeader_Canceled references OrderHeader[Canceled]
objectReference OrderDetail_Confirmed references OrderDetail[Confirmed]

/* Tasks */

// Tasks */
task SendOrderSpecification_1 {
  actor: Customer
  input {
    OrderDetail_New multiplicity -1
  }
  output {
    OrderDetail_ multiplicity -1
  }
}

task Receive_2 {
  actor: Customer
  input {
    Delivery_ multiplicity 1
  }
  output {
  }
}

task AcceptOrder_3 {
  actor: Commercial
  input {
    OrderHeader_ multiplicity 1
  }
  output {
    OrderHeader_Accepted multiplicity 1
  }
}

...
Target model

- Order processing - BPMN
Order processing – Activity diagram
Conclusion

- BML provides source model **independency**
- Potential changes in transformation rules will affect BML $\rightarrow$ CD generator, while generators which transform concrete business process modeling notations will remain unchanged
- Generation of the target conceptual model with a high percentage of **completeness** and **precision**
- In the future we plan to:
  - further identify the semantic capacity of BPMs for automated CDM design
  - improve the BML accordingly to identification of the semantic capacity of BPMs
  - implement generators for some other business process modeling notations
Recent achievements in automated database design based on business process models

Thank You!