Meta-Modelling as a Concept: The Conceptualisation of Modelling Methods
Invited Tutorial

Tutorial Team
Dimitris Karagiannis, Hans-Georg Fill, Niksa Visic, Robert Woitsch, Wilfrid Utz, Srdjan Zivkovic, Elena Miron

AGENDA

PART I:
• Motivation
• Foundations & Technologies
• Conceptualization & Development
• Best Practices

PART II:
• Hands-On Session

PART III:
• Conclusion
• Outlook
Tutorial Specific Scenarios

Selected Scenarios for Tutorial specific Hands-On:

1. Realising a **Modelling Language**
   - Case: Entity Relationship Model

2. Implementing an **Algorithm**
   - Case: Structural Similarities of Business Processes

3. **API / Web-Service** Invocations
   - Case: WIKI Interaction
   - Case: Google Map Interaction

4. Coupling Modelling Languages to support **Modelling Procedures**
   - Case: Coupling BPMN and UML-Use Case Diagram

Implementation Environment per Meta Modelling Layer

Modeling, Transformation, Query, ...

Simulation, Swimlane, Aggregation, ...

„KPI“ Cockpit, „Use Case“ notation, Notebook presentation of „BP-Activity“, ...

„KPIs“, „Use Cases“, „BP-Activities“, ...

Developed in C++, C#, Java

Developed in ALL, ADOScripts, Expressions

Described in ADL, XML

Inherited from Method-specific Meta Model

Instance of ADOxx Meta2 Model

Instance of Generic ADOxx Meta Model

Instance of Model
Development Roles per Meta Modelling Layer

- ADOxx Developer
- Modelling-Tool Developer
- Model User

Generic ADOxx Meta Model
Method-specific Meta Model

Inherited from

Model

Developed in

C++, C#, Java

Inherited from

Expression Modelling - Tool

Developed in

ALL, ADOScripts, Expressions

Created by

Instance of

OML�

www.omilab.org

SETUP OF IMPLEMENTATION ENVIRONMENT
ADOxx Implementation - Infrastructure

Modelling - Tool

supports development

ADOxx Development - Environment

Development Environment

Generic Functionality

Debugging and Testing Tool

provides

ADOxx Platform

ADOxx Platform

Method specific Modelling Language

Faculty of Computer Science

Specification

Configuration of Generic ADOxx Components

Implementation

ADD-On Implementation

Testing Tool

ADOscripts and Expressions

Environment

Testing Tool

Configuring

Implementation

www.omilab.org

Graphical Specification

www.omilab.org

Development of Modelling - Tool based on ADOxx

Specification of Modelling Language

Method specific Modelling Language

ADOxx Meta2Model

Using

Generic ADOxx Components

Using

Method specific Mechanisms and Algorithms

Configuring

ADOxx Development Environment

Attribute Specification

Using

Implementation of Mechanisms and Algorithms
Developmnet Toolkit - STARTUP

1. Start Administration Toolkit
2. Login into Administration Toolkit
3. Default Development User
4. Username: Admin
5. Password: password
   DB: adoxdb
   as "ADOxx user"
6. BACKGROUND: connection to experimentation database hosted on a server platform

Development Toolkit - Components

Debug User needed in the database to start modelling toolkit for validation

U: debug
P: debug

Create user in “User Management” component for testing purposes

Development Environment: Library Management Component
**ADOxx (Experimentation) Library**

- Development aggregated in “Application Library” consisting of Static and Dynamic sub-library
  - **Dynamic**: ADOxx 1.3 Dynamic Library (Experimentation Environment)
  - **Static**: ADOxx 1.3 Static Library (Experimentation Environment)

---

**Tutorial Specific Scenarios**

**Selected Scenarios for Tutorial specific Hands-On:**

1. **Realising a Modelling Language**
   - Case: Entity Relationship Model

2. **Implementing an Algorithm**
   - Case: Structural Similarities of Business Processes

3. **API / Web-Service Invocations**
   - Case: WIKI Interaction
   - Case: Google Map Interaction

4. **Coupling Modelling Languages to support Modelling Procedures**
   - Case: Coupling BPMN and UML-Use Case Diagram
1. SCENARIO:
REALISING A MODELLING LANGUAGE

Scenario Description

Case:
Realise a modelling tool for the Modelling Language “Entity Relationship Model”.

Goal:
Demonstrate the development of a model editor for a defined modelling languages using common constructs from ADOxx Meta²-Model.
ADOxx Meta2Model

Mapping Meta2Model with ER-Meta Model

How to map generic Meta²Model to a concrete Modelling Language?
Modelling Language Meta Model

ER – Modelling Language Definition

Conceptual ER – Meta Model

Additional Aspects:
- Name
- Cardinality
- Key Attribute

Specification for Operationalization:
CLASS: Entity, Relation, Attribute, RELATIONCLASS: relates, has,
Attribute: name (String), cardinality (String),
key (Boolean), datatype (List)

Operationalization of “CLASS” Concept

Operationalization: ER Modelling Language

CLASS: Entity, Relation, Attribute, _ER-Concept_
RELATIONCLASS: relates (Entity->Relation),
has (_ER-Concept_ ->Attribute),
name (STRING), cardinality (STRING),
key (INTEGER), datatype (ENUMERATION)
Provided Functionality of Metamodelling Platform

Used meta-modelling functionality:

• Meta²Model: MODELTYPE, GRAPHREP, ATTREP, ATTRIBUTE
  TYPE, CLASS

• ADOxx Meta2Model Component:
  – Model Editor incl. Menubar
  – Query engine incl. AQL syntax
  – ADOscript interpreter and ADOscript syntax
  – Database

ADOxx Realisation HANDS-ON

1. Defining MODELTYPES

2. Inheriting CLASSES from ADOxx Meta Model

3. Implementing GRAPHREP

4. Inherit RELATIONCLASSES from ADOxx Meta Model

5. Defining ATTRIBUTES and ATTREP
GOAL: Development of Modelling Toolkit

Used ADOxx Functionality: Realising a Modelling Language

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Mechanics &amp; Algorithms Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup of Implementation Environment</td>
<td>Core Functions for Model Manipulation</td>
</tr>
<tr>
<td>Modelling Language Implementation</td>
<td>Database</td>
</tr>
<tr>
<td>Classes</td>
<td>Visualisation</td>
</tr>
<tr>
<td>Relations</td>
<td>Query</td>
</tr>
<tr>
<td>Class Attributes and Attributes</td>
<td>Transformation</td>
</tr>
<tr>
<td>GRAPHREP</td>
<td>Configuration of ADOxx Components</td>
</tr>
<tr>
<td>ATTRREP</td>
<td>Visualisation</td>
</tr>
<tr>
<td>CLASS Cardinality</td>
<td>Query</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>External Coupling ADOxx Functionality</td>
</tr>
<tr>
<td>Model Pointer</td>
<td>ADOscript Triggers</td>
</tr>
<tr>
<td>Attribute Facets</td>
<td>ADOscript Language Constructs</td>
</tr>
<tr>
<td>Model Types</td>
<td>Visualisation ADOscript</td>
</tr>
<tr>
<td></td>
<td>Visualisation Expression</td>
</tr>
<tr>
<td></td>
<td>Query ADOscript</td>
</tr>
<tr>
<td></td>
<td>Transformation ADOscript</td>
</tr>
<tr>
<td></td>
<td>ADD-ON Implementation</td>
</tr>
<tr>
<td></td>
<td>ADOxx Web-Service</td>
</tr>
<tr>
<td></td>
<td>XML / ADL Import – Export</td>
</tr>
<tr>
<td></td>
<td>ADOscript Batch Mode</td>
</tr>
</tbody>
</table>
2. SCENARIO: IMPLEMENTING AN ALGORITHM

Scenario Description

Case:
An algorithm for analysing structural similarities is implemented that queries business process models and creates a comparison matrix listing structural similarities.

GOAL:
Demonstrate how models can be queried with AQL and ADOscripts, as well as indicate how to create and manipulate a model.
**Description of Algorithm**

1. **Trigger**
2. **Ask Parameter**
3. **Query Source-Models**
4. **Make Result Matrix**
5. **Show Comparison Matrix**

**Additional Aspects:**

- Implementation as plug-in to be used in other modelling languages.
- Comparison queries should be adaptable but start with comparing used objects.
- Migration from modelling language without plug-in to modelling language with plug-in has to be considered.
**Added Value of Metamodelling Platform**

Used meta-modelling functionality for realisation of the scenario:

- **ADOscript**: ADOscript can generate a new model “Comparison Matrix” to present the results of the business process comparison. This technique can also be used for graph-rewriting.

- **AQL**: ADOxx Query Language
  - ADOxx query engine is provided by the platform and can analyze business process models.
  - ADOScripts can invoke the query engine and hence compare in a stepwise approach business processes.

- **Hyperlinks and INTERREF**: Similar to the first scenario, the resulting model can use INTERREFS and Hyperlink for better navigation from the resulting “Comparison Matrix” to the originally compared business processes.
ADOxx Realisation Hands-On

1. Modelling Language Extensions to enable this algorithms
   1. New model type “Comparison Model”
   2. New class “Box”, “Row Name” for Comparison Matrix Element

2. Configure ADOxx
   1. Configure Menubar
   2. Write AQL statements for query engine

3. Implement Algorithm with ADOscript
   1. ADOscript User Interface
   2. Invoking query engine with ADOscript
   3. Create target model “Comparison Matrix and place matrix elements according the results of the query.”

Used ADOxx Functionality: Implementing an Algorithm

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Mechanics &amp; Algorithms Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup of Implementation Environment</td>
<td>Core Functions for Model Manipulation</td>
</tr>
<tr>
<td>Modelling Language Implementation</td>
<td>Database</td>
</tr>
<tr>
<td>Classes</td>
<td>Visualisation</td>
</tr>
<tr>
<td>Relations</td>
<td>Query</td>
</tr>
<tr>
<td>Class Attributes and Attributes</td>
<td>Transformation</td>
</tr>
<tr>
<td>GRAPHREP</td>
<td>Configuration of ADOxx Components</td>
</tr>
<tr>
<td>ATTRREP</td>
<td>Visualisation</td>
</tr>
<tr>
<td>CLASS Cardinality</td>
<td>Query</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>ADOscript Triggers</td>
</tr>
<tr>
<td>Model Pointer</td>
<td>ADOscript Language Constructs</td>
</tr>
<tr>
<td>Attribute Facets</td>
<td>Visualisation ADOscript</td>
</tr>
<tr>
<td>Model Types</td>
<td>Visualisation Expression</td>
</tr>
<tr>
<td></td>
<td>Query ADOscript</td>
</tr>
<tr>
<td></td>
<td>Transformation ADOscript</td>
</tr>
<tr>
<td></td>
<td>ADO-ON Implementation</td>
</tr>
<tr>
<td></td>
<td>ADOxx Web-Service</td>
</tr>
<tr>
<td></td>
<td>XML / ADL Import – Export</td>
</tr>
<tr>
<td></td>
<td>ADOscriptBatch Mode</td>
</tr>
</tbody>
</table>
3. SCENARIO: API / WEB-SERVICE INVOCATIONS

Scenario Description

Case: An implementation of a modelling method is extended/enhanced by functionality external to the meta-modelling platform through API calls on WebServices (WS).

GOAL:
- Demonstrate usage of APIs in ADOxx to call external services
- Implement mechanisms for push and pull invocation to external services

Interaction Cases:
- **WIKI Interaction**: Models defined using the i* implementation in ADOxx are made available in a MediaWiki environment
- **Google Map Interaction**: Models defined for the design of supply chain distribution networks are enhanced with geolocation data using the Google Maps WS and OpenStreetMap WS
Description of MediaWiki Interaction

1. Create Models
   - i* Modelling Tool

2. Extract Models

3. Improve Models
   - i* Wiki

4. Feedback Improvements

Mapping ADOxx Functionality

Source Environment:
- ADOxx: Events
- ADOxx: Export
- ADOxx: Web-Service
- ADOxx: Database

Target Environment:
- MediaWiki: REST API

Adaptations and Enhancements:
- ADOxx: Web-Servcie
- ADOxx: Database
- MediaWiki: REST API
- Web-Service Interface
- Hyperlink
- REST Invocation from ADOscript
Meta Modelling Layer: Transformation Operations in ADOxx

Component: Transformation operations

Native using AdoScript:
- To parse model and instance information using "Core" Messageport
  - GET_MODEL_INFORMATION
  - GET_ALL_NB_ATTRS
  - GET_ALL_OBJS_OF_CLASSNAME
  - GET_ALL_OBJS
  - GET_ALL_CONNECTORS

Transformation using XSLT/DSSSL:
- Utilizing export functionality (XML, SGMIL) plus transformer calls (XSLT, DSSLS, Saxon)

Generate

Meta Modell

Generic ADOxx
Meta Model

Inherited from

Method-specific
Meta Model

Instance of

MM-DSL: ALL

MM – Tool
Implementation Part

Model

ADL, XML

Module

Part

Meta Modelling Layer: Web-Service Functionality in ADOxx

Component: Web-Service

execute (AdoScript)

get(TOC) returns:
- "Documentation" \{MML\_TOC\_FOR\_USER\_ID
- getModel(String modelID)
- getModelXML(String modelID)
- getModuleMap(String modelID)

Request

Response

MM-DSL: ALL
ADOxx Library Language

Module

Part

MM – Tool
Implementation Part

www.omilab.org
Applied ADOxx Functionality

- **ADOxx Constructs for Modelling Language Extensions**
  - Define a new attribute of type “PROGRAMCALL” to store/define the target URL of the wiki page
  - Update the interactive/independent graphical representation to show the link

- **ADOxx Constructs for Mechanism and Algorithms Development**
  - Define event handler “SaveModel” to trigger the export from the Modelling environment to the wiki system
  - Use AdoScript Core Operations to parse model
  - Use AdoScript External Call Operations to call and invoke the MediaWiki API for update of pages
  - Use AdoScript Core Operations to enable feedback mechanisms via updating the model/instance

### Attribute Type: PROGRAMCALL

A PROGRAMCALL attribute is characterized by a fixed set of items. These items are related to AdoScripts which can be called via the user interface. A PROGRAMCALL attribute value consists of at most one of the defined items and an optional parameter.

#### UI representation

<table>
<thead>
<tr>
<th>Item Definition</th>
<th>Item Text</th>
<th>Parameter Definition</th>
<th>FilterValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgramCallName</td>
<td>ITEM ITEM_TEXT [PARAMETER_DEFINITION] [FILTER_FILTER]</td>
<td>AdoScript</td>
<td></td>
</tr>
<tr>
<td>ParameterDefinition</td>
<td>para</td>
<td>paraITEM_TEXT [DEFAULT_TEXT_VALUE]</td>
<td></td>
</tr>
<tr>
<td>FilterFilter</td>
<td>fidg</td>
<td>fidgITEM_TEXT</td>
<td>filterDescriptionText</td>
</tr>
</tbody>
</table>

**Item Text**, **paraITEM_TEXT**, **DEFAULT_TEXT_VALUE**, **filterITEM_TEXT** and **filterDescriptionITEM_TEXT** are string values.
GRAPHREP WIKI Pointer for "Softgoal"

Implementation of
- **Attribute-dependent representation**: if a wiki link is available, the representation is changed
- **Interactive representation**: the wiki programcall is executed from the graphical view (hyperlink functionality) clickable on name representation

**PSEUDOCODE**
```plaintext
IF (attributeNotEmpty ('Wiki view')) {
    drawHyperlink (getCall('Wiki view'), name)
} ELSE {
    drawName()
}
```

**EVENT HANDLERS**

In ADOxx event handlers are used to:

a) Listen to events that result from the interaction with the modelling toolkit
b) Handle/Trigger operations based on the events

Event handlers are realized as an external coupling implementation in the platform, depending on the event, a certain set of parameters/variables are pre-set to be used during the implementation of the actual handler.

<table>
<thead>
<tr>
<th>Event Category</th>
<th>Number of Events Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>48</td>
</tr>
<tr>
<td>Application</td>
<td>3</td>
</tr>
<tr>
<td>Modelling</td>
<td>15</td>
</tr>
<tr>
<td>Simulation</td>
<td>2</td>
</tr>
<tr>
<td>Import/Export</td>
<td>2</td>
</tr>
<tr>
<td>Drawing</td>
<td>4</td>
</tr>
</tbody>
</table>

# Event implementation to prevent the deletion of instances of a certain class
```plaintext
ON_EVENT "BeforeDeleteInstance" {
    IF (attributeNotEmpty ('Wiki view')) {
        drawHyperlink (getCall('Wiki view'), name)
    }
    ELSE {
        drawName()
    }
}
```

The following statement is redundant, the EXIT means EXIT 0:
```plaintext
EXIT 0
```
Pseudo Code: PUSH Invocation

TRIGGER SaveModel {
    #preset by trigger: modelid
    modelinformation = getMSTModelInformation(modelid)
    wikiName = ConstructUniqueName(modelinformation)
    CallAPICreateWikiPage (wikiName)
    allInstances = getAllInstances(modelid)
    for instance in allInstances {
        instanceInformation = getInstanceInformation(instance)
        CallAPICreateWikiPage (instanceInformation)
        CallAPIAddTextToSection('Instances', instance)
        setTargetURL(instance)
    }
    setTargetURL(model)
    }

FUNCTION addAttributeValues(notebook) {
    CallAPICreateWikiSection('Notebook')
    List attributes = getAllAttributes()
    for attribute in attributes {
        CallAPIAddTextToSection('Notebook', attribute)
    }
}

Implementation Result

Wiki view of iSTAR Models
Scenario Description

Case: An implementation of a modelling method is extended/enhanced by functionality external to the meta-modelling platform through API calls on WebServices (WS).

GOAL:
- Demonstrate usage of APIs in ADOxx to call external services
- Implement mechanisms for push and pull invocation to external services

Interaction Cases:
- Wiki Interaction: Models defined using the i* implementation in ADOxx are made available in a MediaWiki environment
- Google Map Interaction: Models defined for the design of supply chain distribution networks are enhanced with geolocation data using the Google Maps WS and OpenStreetMap WS

Description of GeoCoding Invocation

<table>
<thead>
<tr>
<th>Model Editor</th>
<th>Geolocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Lockup</td>
<td>4. Request Map</td>
</tr>
<tr>
<td>3. Provide Parameter</td>
<td>5. Receive Map</td>
</tr>
<tr>
<td>4. Request Map</td>
<td>6. Display</td>
</tr>
</tbody>
</table>

Additional Details:
1. Define map (location by name, zoom factor)
2. Request LONG/LAT options by location name through ReST call
3. Select center location from options
4. Request map image through ReST call
Applied ADOxx Functionality

- **ADOxx Constructs for Modelling Language Extensions**
  - Define a new *attribute of type* “PROGRAMCALL” to invoke the map service calls
  - Update the *representation* of the modeltype to represent the map as a background

- **ADOxx Constructs for Mechanism and Algorithms Development**
  - Use AdoScript *External Call Operations to call and invoke* the GoogleMaps/OpenStreetMap API for map information
  - Establish *basic UI elements* for selection of LONG/LAT options of model
Pseudo Code: PULL Invocation

```
ITEM Notebook Button "Update map" {  #preset by button: modelid
  locationName = getModelAttribute('locationName')
  mapZoom = getModelAttribute('zoom')
  List locations = CallAPIGeoLookUpLocation (locationName )
  locationSelectionBox = buildListBox (parse(locations))
  locationSelectionBox.show(modal)
  If (endbutton = cancel) {
    EXIT
  } Else {
    File map = CallAPIGeoStaticMapService(selectedLonLat)
    setModelAttribute('mapfile', map)
    triggerModelRepresentationUpdate(modelid)
  }
}
```

Implementation Result: Maps in Modelling Editor
Used ADOxx Functionality: API / Web-Service Invocation

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Mechanisms &amp; Algorithms Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup of Implementation Environment</td>
<td>Core Functions for Model Manipulation</td>
</tr>
<tr>
<td>Modelling Language Implementation</td>
<td>Database</td>
</tr>
<tr>
<td>Classes</td>
<td>Visualisation</td>
</tr>
<tr>
<td>Relations</td>
<td>Query</td>
</tr>
<tr>
<td>Class Attributes and Attributes</td>
<td>Transformation</td>
</tr>
<tr>
<td>GRAPHREP</td>
<td>Configuration of ADOxx Components</td>
</tr>
<tr>
<td>ATTRREP</td>
<td>Visualisation</td>
</tr>
<tr>
<td>CLASS Cardinality</td>
<td>Query</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>Transformation ADOscript</td>
</tr>
<tr>
<td>Model Pointer</td>
<td>ADOscript Triggers</td>
</tr>
<tr>
<td>Attribute Facets</td>
<td>ADOscript Language Constructs</td>
</tr>
<tr>
<td>Model Types</td>
<td>Visualisation ADOscript</td>
</tr>
<tr>
<td></td>
<td>ADOscript Triggers</td>
</tr>
<tr>
<td></td>
<td>XML / ADL Import – Export</td>
</tr>
<tr>
<td></td>
<td>ADD-OX Implementation</td>
</tr>
<tr>
<td></td>
<td>ADOxx Web-Service</td>
</tr>
<tr>
<td></td>
<td>ADOscript Batch Mode</td>
</tr>
</tbody>
</table>

Coupling BPMN and UML Use Case Diagrams

4. SCENARIO: COUPLING MODELLING LANGUAGES TO SUPPORT MODELLING PROCEDURE
Scenario Description

Case:
Two implementations of different modelling approaches are combined to support a common modelling procedure ("Vorgehensmodell"). The modelling approach BPMN and UML - use case models are implemented in a coupled way to enable an integrated view from processes to use cases.

GOAL:
- Demonstrate how a combined usage of two modelling approaches is realized
- Develop functionality based on combined view

Coupling of BPMN and UML-Use Case

<table>
<thead>
<tr>
<th>BPMN</th>
<th>UML-Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create BPMN Models</td>
<td>1. Create UML-Use Case</td>
</tr>
<tr>
<td>2. Link Tasks to Use Case</td>
<td>3. Inform about Changes</td>
</tr>
<tr>
<td>4. Display Monitoring of Changes</td>
<td></td>
</tr>
</tbody>
</table>

1. Interlink/Integrate BPMN task class with UML use case model to establish an integrated view
2. Provide graphical representation of change status (notification mechanism if a use case changes)
3. Provide same structure and operations as for regular use case
4. Additional attribute to hold a point to an external file or URL
5. Updated graphical representation using a dashed outline and to visualize the attribute of 2) and provide a hyperlink
6. Allow conversion between regular use case and use case (documentation)
Meta Modelling Layer: Coupling Modelling Languages

Meta2Model: “INTERREF”

Meta Modell

Generic ADOxx Meta Model

Inherited from

Method-specific Meta Model

MM – Tool Implementation Part

MM-DSL: ALL ADOxx Library Language

Model

described in

ADL, XML

MM – Tool Implementation Part

Inherited from

MM – Tool Implementation Part

MM – Tool Implementation Part

Class A

Class B

Attribute: PointsToB

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of

Instance of
Update Listener

1. Enable debugging facility for AdoScript Development
2. Listen on change events in Use case diagram
3. Update automatically related BPMN activities and their state

Applied ADOxx Functionality

• ADOxx Constructs for Modelling Language Extensions
  – INTERREF attribute: the attribute type INTERREF enables to combine two modelling classes using a pointer within one meta model.
  – Hyperlink: the hyperlink and model-pointer functionality enables to navigate better within models.
  – Attribute Depended Graphical Representation: depending on the value of an attribute, the graphical representation changes and hence a status can be presented in the model.

• ADOxx Constructs for Mechanism and Algorithms Development
  – Event-Listener: the platform provides a set of event-listener and hence changes in the model can be identified.
  – Use AdoScript Core Operations to parse model and update the corresponding model accordingly.
ADOxx Realisation Approach Overview

• Modelling Language
  – New class for UML Use Case
  – Change GRAPHREP, ATTRREP of new UML Use Case
  – Define CONVERSION of UML Use Case to new UML Use Case
  – Change GRAPHREP, ATTREP from BPMN Activity
  – Attribute dependent representation for status

• Mechanism and Algorithms
  – AdoScript identifies changes in specification and changes status
  – Outlook on ADD-ON: Document changes can be tracked and AdoScript can be invoked from outside.

Implementation Result
Used ADOxx Functionality: Coupling Modelling Languages

<table>
<thead>
<tr>
<th>Modelling Language Implementation</th>
<th>Mechanisms &amp; Algorithms Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>Core Functions for Model Manipulation</td>
</tr>
<tr>
<td>Relations</td>
<td>Database</td>
</tr>
<tr>
<td>Class Attributes and Attributes</td>
<td>Visualisation</td>
</tr>
<tr>
<td>GRAPHREP</td>
<td>Query</td>
</tr>
<tr>
<td>ATTRREP</td>
<td>Transformation</td>
</tr>
<tr>
<td>CLASS Cardinality</td>
<td>Configuration of ADOxx Components</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>Visualisation</td>
</tr>
<tr>
<td>Model Pointer</td>
<td>Query</td>
</tr>
<tr>
<td>Attribute Facets</td>
<td>Transformation</td>
</tr>
<tr>
<td>Model Types</td>
<td>ADOscript Triggers</td>
</tr>
<tr>
<td></td>
<td>ADOscript Language Constructs</td>
</tr>
<tr>
<td></td>
<td>Visualisation ADOscript</td>
</tr>
<tr>
<td></td>
<td>Visualisation Expression</td>
</tr>
<tr>
<td></td>
<td>Query ADOscript</td>
</tr>
<tr>
<td></td>
<td>Transformation ADOscript</td>
</tr>
<tr>
<td></td>
<td>ADD-ON Implementation</td>
</tr>
<tr>
<td></td>
<td>ADOxx Web-Service</td>
</tr>
<tr>
<td></td>
<td>XML / ADL Import – Export</td>
</tr>
<tr>
<td></td>
<td>ADOscript Batch Mode</td>
</tr>
</tbody>
</table>

In case of any questions, please contact:

**tutorial@adoxx.org**

For any questions please contact:

Prof. Dr. Dimitris Karagiannis

University of Vienna
Faculty of Computer Science
Research Group Knowledge Engineering
Währinger Str. 29
A-1090 Vienna
Tel.: +43-1-4277-789 10
Fax: +43-1-4277-8789 10
Email: dk@dke.univie.ac.at
Web: http://www.dke.univie.ac.at