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

universität wien
Faculty of Computer Science

OMiLAB
www.omilab.org

AGENDA

PART I:

- Motivation
- Foundations & Technologies
- Conceptualization & Development
- Best Practices

PART II:

- Hands-On Session

PART III:

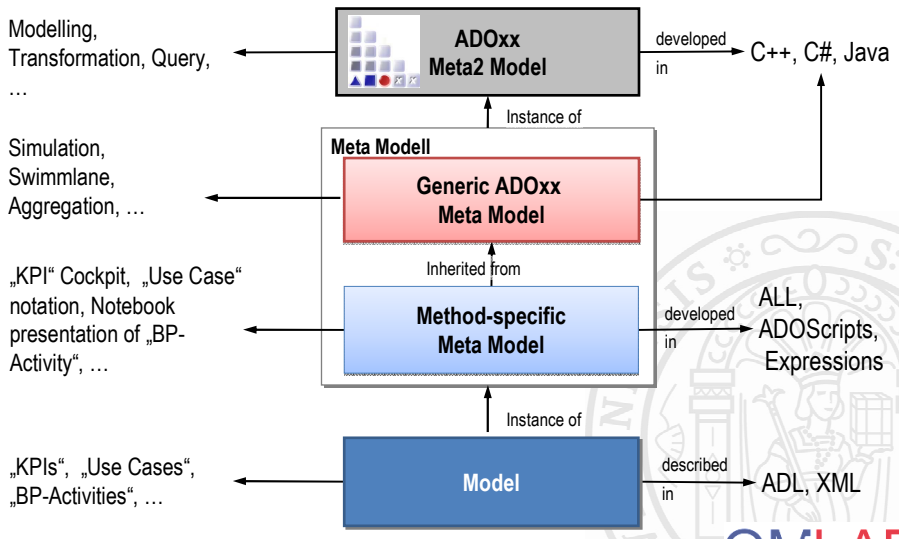
- Conclusion
- Outlook

Tutorial Specific Scenarios

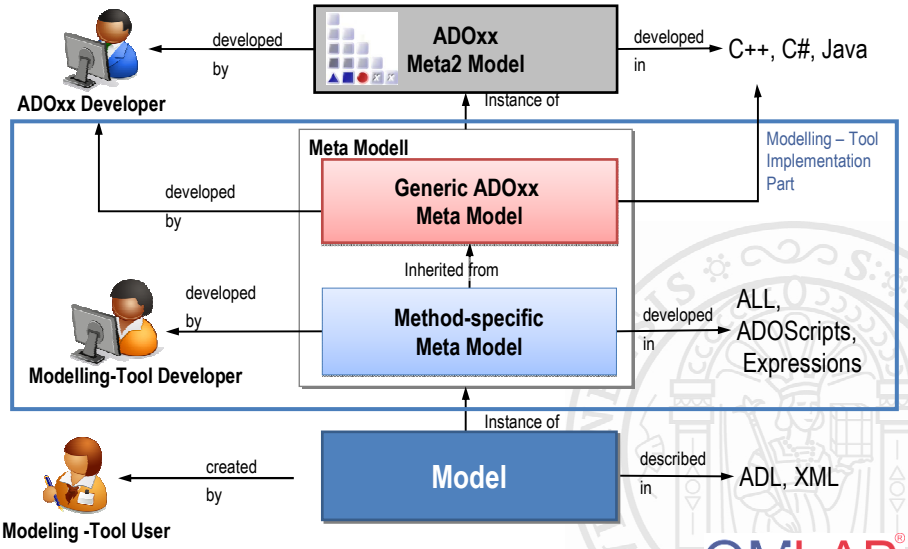
Selected Scenarios for Tutorial specific Hands-On:

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

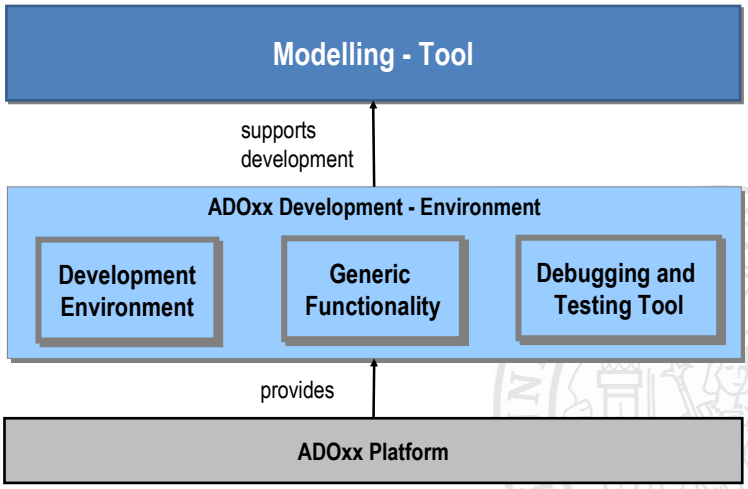


Development Roles per Meta Modelling Layer

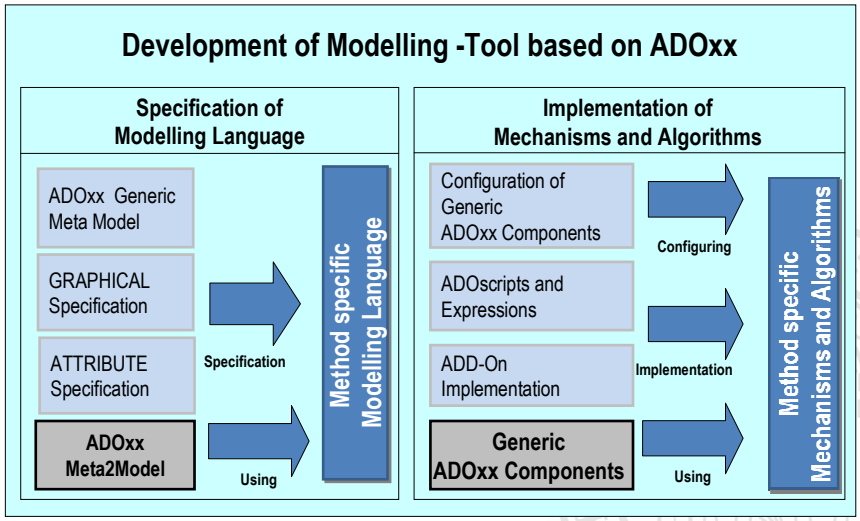


SETUP OF IMPLEMENTATION ENVIRONMENT

ADOxx Implementation - Infrastructure



ADOxx Development Environment



Developmnet Toolkit - STARTUP

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

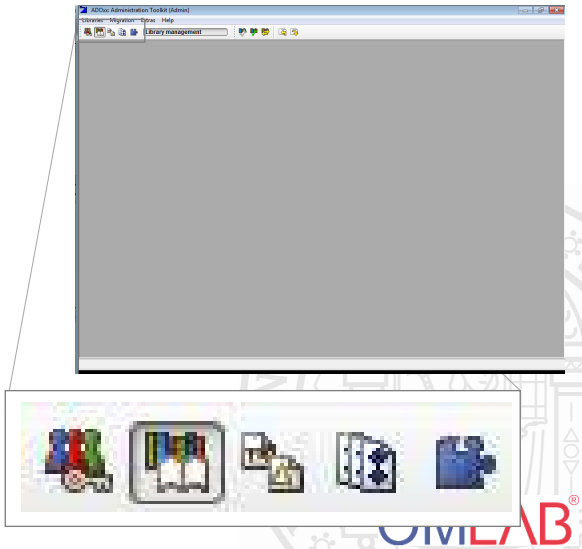
Development Toolkit - Components

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

U: debug
P: debug

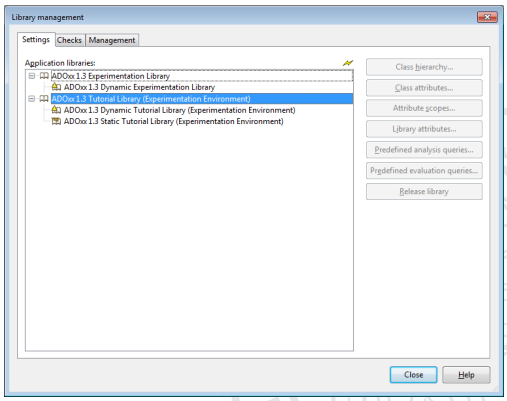
Create user in “User Management” component for testing purposes

Development Environment: Library Management Component



ADOxx (Experimentation) Library

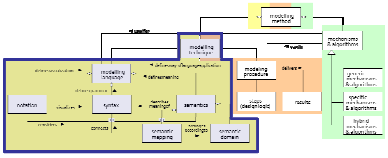
- Development aggregated in "Application Library" consisting of Static and Dynamic sub-library
  - **Dynamic:** ADOxx 1.3 Dynamic Library (Experimentation Environment)
  - **Static:** ADOxx 1.3 Static Library (Experimentation Environment)



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



CASE: Entity Relationship Model

1. SCENARIO: REALISING A MODELLING LANGUAGE

Scenario Description

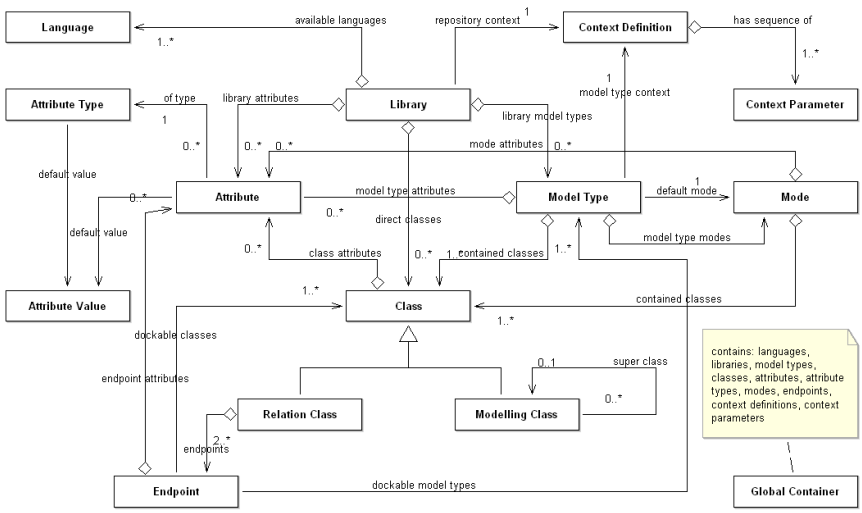
Case:

Realise a modelling tool for the Modelling Language "Entity Relationship Model".

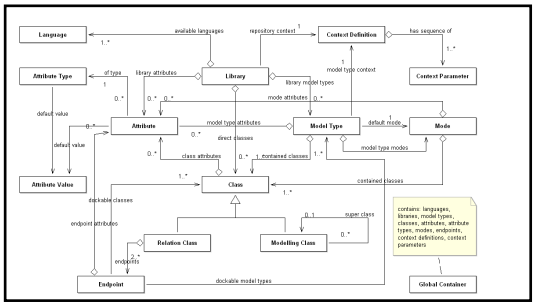
Goal:

Demonstrate the development of a model editor for a defined modelling languages using common constructs from ADOxx Meta^{2}Model.

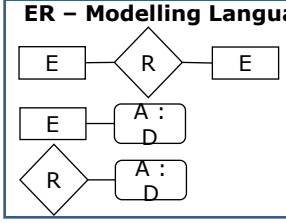
ADOxx Meta2Model



Mapping Meta2Model with ER-Meta Model



ER – Modelling Language Definition

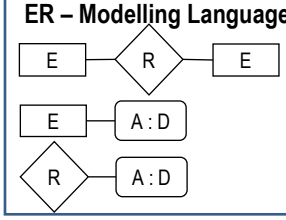


$E(A_1:D_1, \ldots, A_m:D_m)$
$E(A_1, \ldots, A_m)$
$R(A_1:D_1, \ldots, A_m:D_m)$
$R(A_1, \ldots, A_m)$

How to map generic Meta²Model to a concrete Modelling Language ?

Modelling Language Meta Model

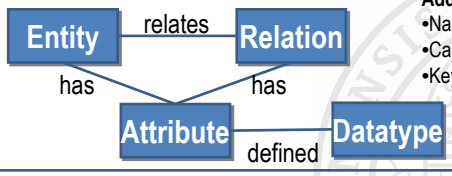
ER – Modelling Language Definition



$E(A_1{:}D_1, \ldots, A_m{:}D_m)$
$E(A_1, \ldots, A_m)$
$R(A_1{:}D_1, \ldots, A_m{:}D_m)$
$R(A_1, \ldots, A_m)$

Conceptualisation

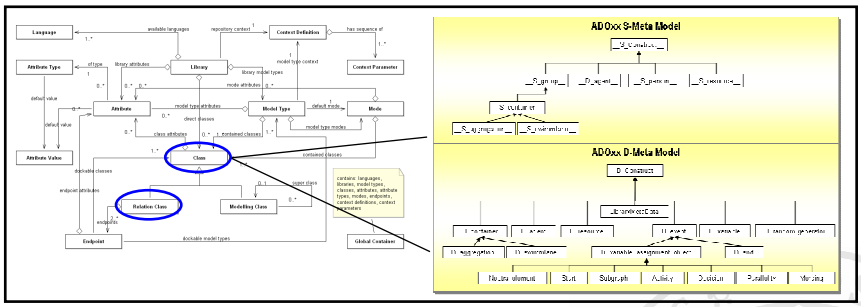
Conceptualised ER – Meta Model



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),
Attribute: name (STRING), cardinality (STRING),
key (INTEGER), datatype (ENUMERATION)

Operationalizeable Meta Model

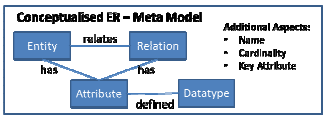
Meta²Model

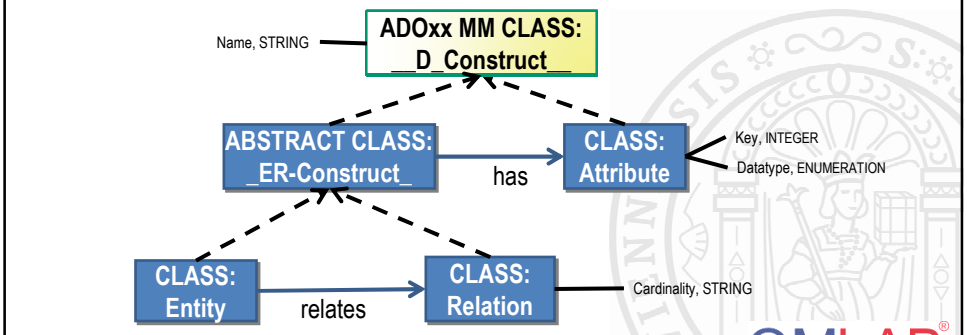
ABSTRACT CLASS
CLASS
RELATIONCLASS
ATTRIBUTE
ATTRIBUTETYPE
{STRING,...ENUMERATION}

Operationalisation MetaModel

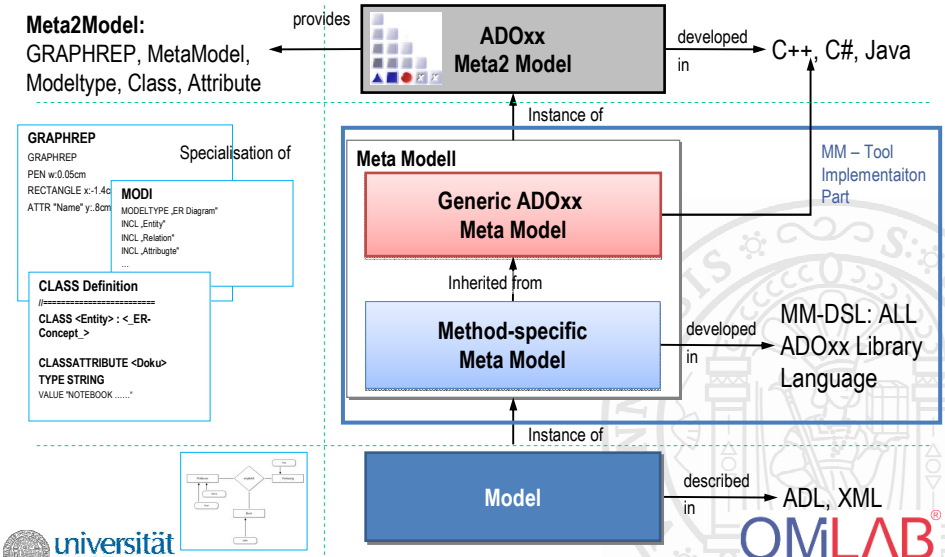
Class inheritance
_D_Construct_
GRAPHREP, ATTREP

ER-MetaModel





Meta Modelling Layer : Realising Modelling Language



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

Menubar

Actionbar

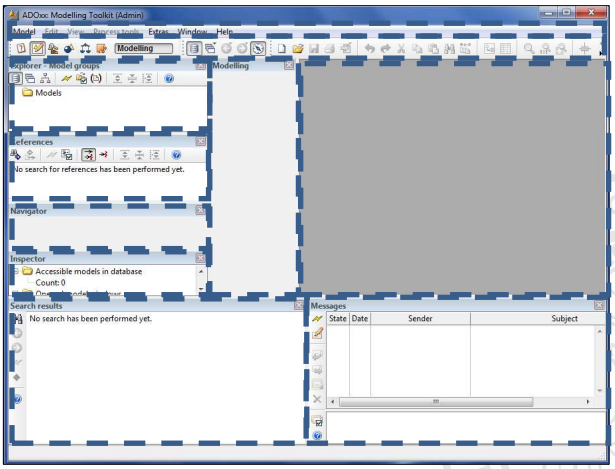
Explorer

References

Navigator

Inspector

Search Results



Modelling

Drawing Area

Messages

Used ADOxx Functionality: Realising a Modelling Language

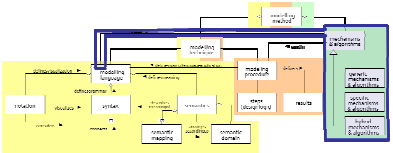
- Introduction
- Setup of Implementation Environment

Modelling Language Implementation

- **Classes** ✓
- **Relations** ✓
- **Class Attributes and Attributes** ✓
  - **GRAPHREP** ✓
  - **ATTRREP** ✓
  - CLASS Cardinality
  - **CONVERSION** ✓
  - **Model Pointer** ✓
- Attribute Facets
- **Model Types** ✓

Mechanisms & Algorithms Implementation

- Core Functions for Model Manipulation
  - Database
  - Visualisation
  - Query
  - Transformation
- Configuration of ADOxx Components
  - Visualisation
  - Query
- External Coupling ADOxx Functionality
  - ADOscript Triggers
  - ADOscript Language Constructs
  - Visualisation AdoScripts
  - Visualisation Expression
  - Query ADOscript
  - Transformation ADOscript
- ADD-ON Implementation
  - ADOxx Web-Service
  - XML / ADL Import – Export
  - ADOscript Batch Mode



Analysis of Structural Similarities

2. SCENARIO: IMPLEMENTING AN ALGORITHM

Scenario Description

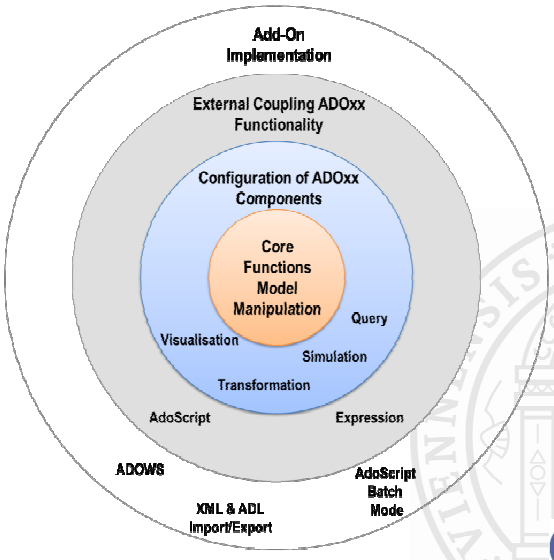
Case:

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

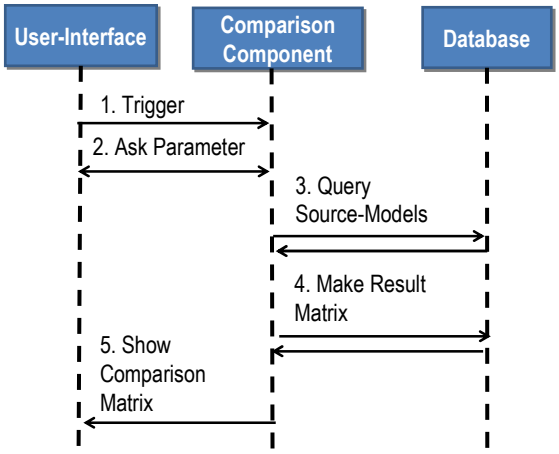
GOAL:

Demonstrate how models can be queried with AQL and ADOscripts, as well as indicate how to create and manipulate a model.

ADOxx Functionality on Meta Level



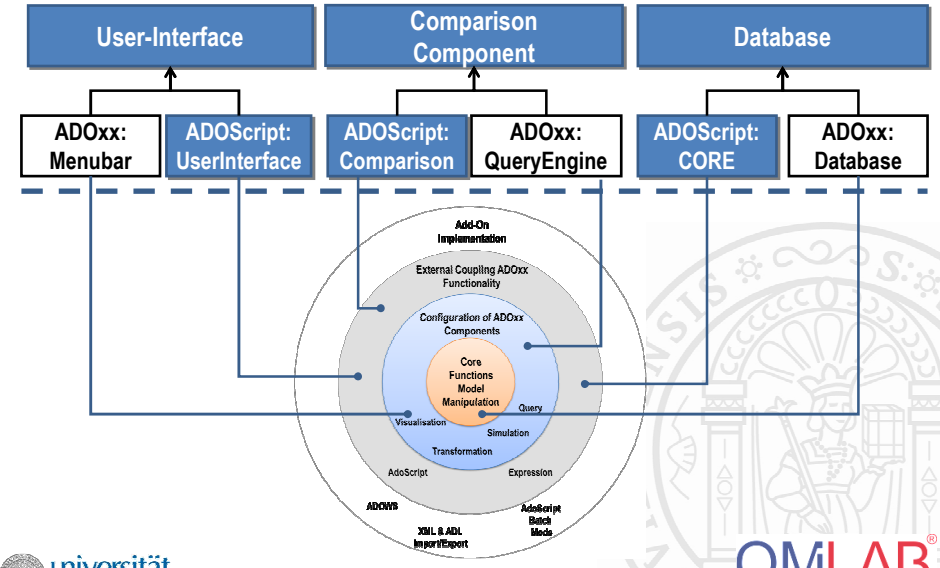
Description of Algorithm



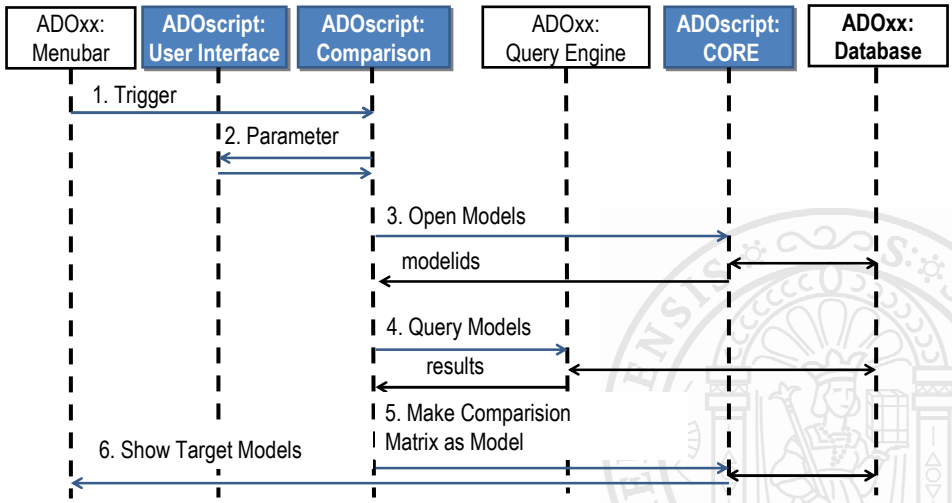
Additional Aspects:

- 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.

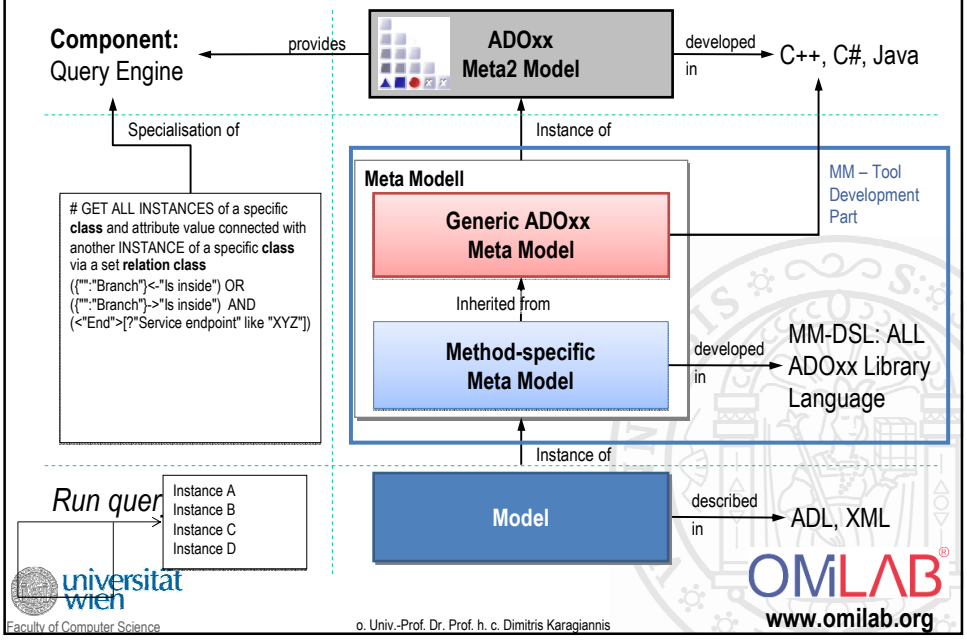
Mapping ADOxx Functionality



ADOxx Realisation Approach



Meta Modelling Layer: Implementing and Algorithm



Added Value of Metamodelling Platform

Used meta-modelling functionality for realisation of the scenario:

•**ADOScrip:** 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

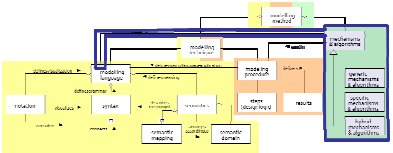
•**Hyperlinks and INTERREF:** Similar to the first scenario, the resulting model can use **INTERREFS** and **Hyperlink** for better navigation from the resulting "Comparison Matrix" to the originally compared business processes.

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

- Introduction
- Setup of Implementation Environment
- Modelling Language Implementation
  - Classes
  - Relations
  - Class Attributes and Attributes
    - GRAPHREP
    - ATTRREP
    - CLASS Cardinality
    - CONVERSION
    - Model Pointer
  - Attribute Facets
  - Model Types
- Mechanisms & Algorithms Implementation
  - Core Functions for Model Manipulation
    - Database
    - Visualisation
    - **Query**
    - Transformation
  - Configuration of ADOxx Components
    - Visualisation
    - Query
  - **External Coupling ADOxx Functionality**
    - **ADOscript Triggers**
    - **ADOscript Language Constructs**
    - Visualisation ADOscript
    - Visualisation Expression
    - **Query ADOscript**
    - Transformation ADOscript
  - ADD-ON Implementation
    - ADOxx Web-Service
    - XML / ADL Import – Export
    - ADOscriptBatch Mode



Interact with MediaWIKI and Google Maps

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

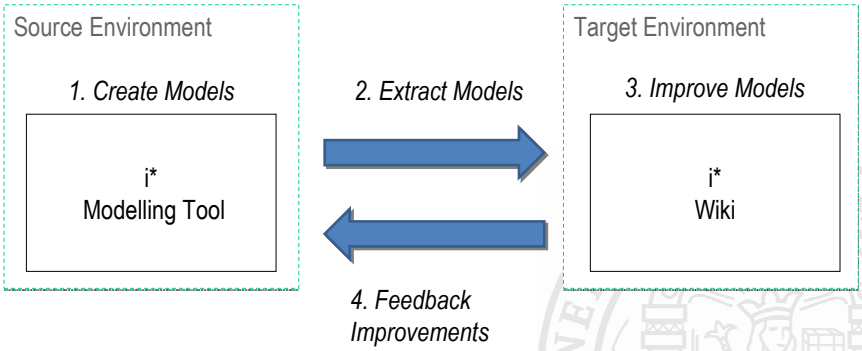
-Implement mechanisms for push and pull invocation to external services

Interaction Cases:

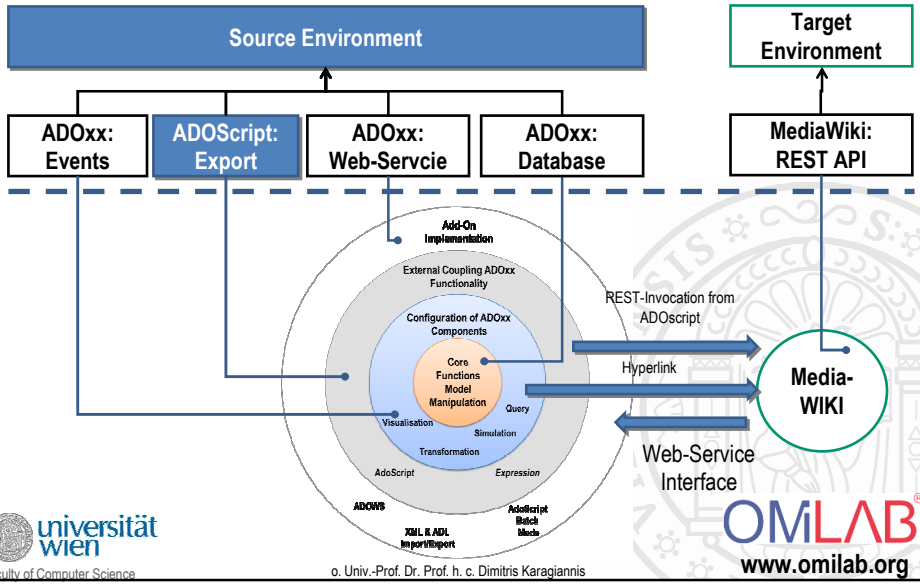
-*WIKI Interaction:* Models defined using the i* implementation in ADOxx are made available in a MediaWiki environment

-*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

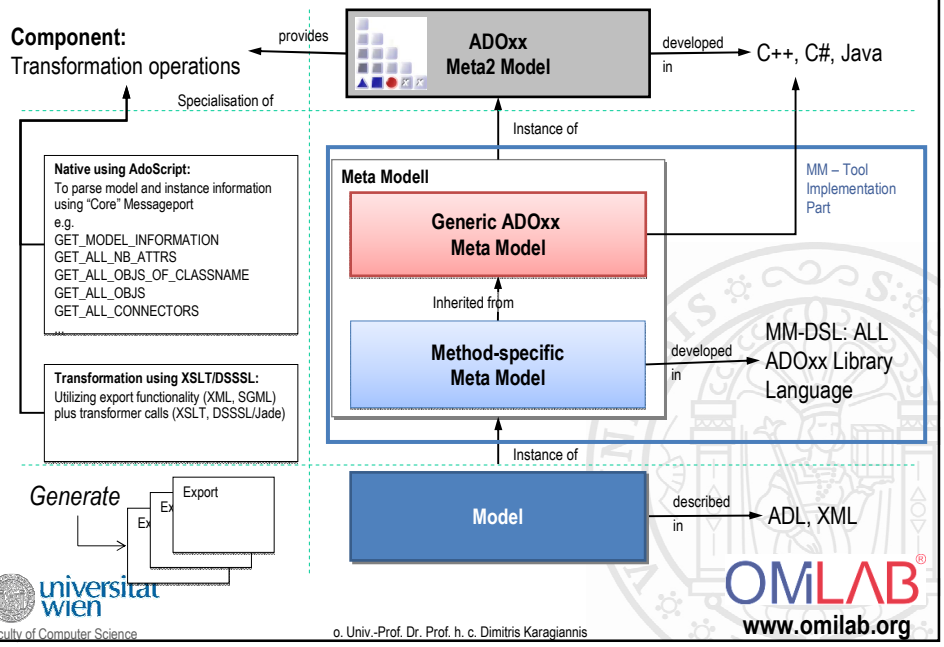
Description of MediaWiki Interaction



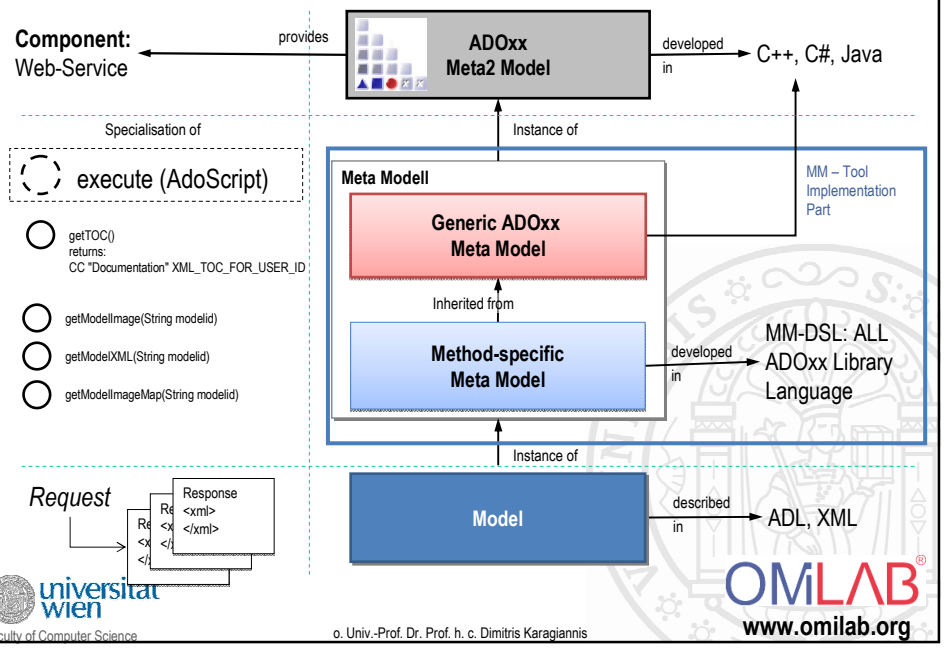
Mapping ADOxx Functionality



Meta Modelling Layer: Transformation Operations in ADOxx



Meta Modelling Layer: Web-Service Functionality in ADOxx



Applied ADOxx Functionality

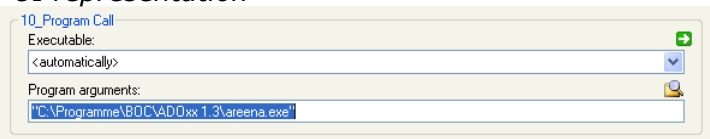
- **ADOxx Constructs for Modelling Language Extensions**
  - Define a new ***attribute of type "PROGRAMCALL"*** to store/define the target URL of the wiki page
  - Update the ***interactive/dependent graphical representation*** to show the link

- **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 Opertions 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



Operations

ProgramCallDomain :	{ *ItemDefinition* } .
ItemDefinition :	**ITEM** *itemText* [*ParameterDef* inition] { *FDlgFilter* } *AdoScript* .
ParameterDefinition :	**param** : *paramText* [: *defaultTextValue*] .
FDlgFilter :	**fdlg-filter**<*i*> : *filterText* **fdlg-type**<*i*> : *filterDescriptionText* .

itemText, *paramText*, *defaultTextValue*, *filterText* and *filterDescriptionText* are string values.

GRAPHREP WIKI Pointer for "Softgoal"

Implementation of

-*Attibute-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

```
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 of 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.

Event Category	Number of Events Available
Core	48
Application	3
Modelling	15
Simulation	2
Import/Export	2
Drawing	4

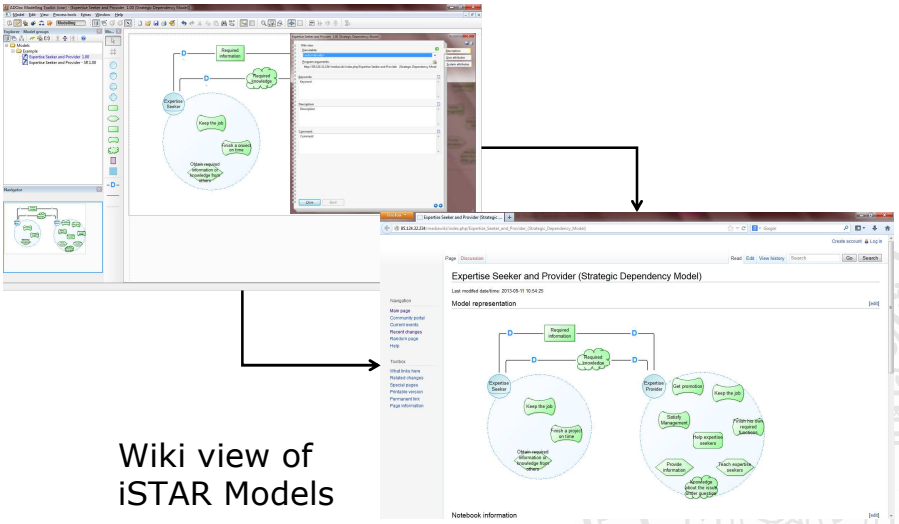
```
# Event implementation to prevent the deletion of instances of a
certain class
ON_EVENT "BeforeDeleteInstance" {
    CC "Core" GET_CLASS_NAME classid:(classid)
    IF (classname = "Information") {
        CC "Core" GET_ATTR_ID classid:(classid) attrname:"Allow
deletion"
        CC "Core" GET_ATTR_VAL objid:(instid) attrid:(attrid)
        IF (val = "no") {
            CC "AdoScript" ERRORBOX "Deletion not allowed!"
            EXIT -1
            # -1 means, that the deletion is aborted, but no error
            # message will appear. That's what we want here, as an
            # error box has already been shown by this event handler.
        }
    }
    # the following statement is redundant (no EXIT means EXIT 0)
    EXIT 0
}
```

Pseudo Code: PUSH Invocation

```
TRIGGER SaveModel {
  #preset by trigger: modelid
  modelinformation = getModelInformation(modelid)
  wikiName = ConstructUniqueName(modelinformation)
  CallAPICreateWikiPage (wikiName)
  addAttributeValues(getNotebook(model))
  List instances = getAllInstances(modelid)
  CallAPICreateWikiSection('Instances')
  for instance in:(instances) {
    instanceinformation = getInstanceInformation(instance)
    instancewikiName = ConstructUniqueName(instanceinformation)
    CallAPICreateWikiPage (instancewikiName)
    addAttributeValues(getNotebook(instance))
    CallAPIaddTextToSection('Instances', instance)
    setTargetURL(instance)
  }
  setTargetURL(model)
}
FUNCTION addAttributeValues(notebook) {
  CallAPICreateWikiSection('Notebook')
  List attributes = getAllAttributes()
  for attribute in:(attributes) {
    CallAPIaddTextToSection('Notebook', attribute)
  }
}
```

...Core Operations
...Invocation Operations

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

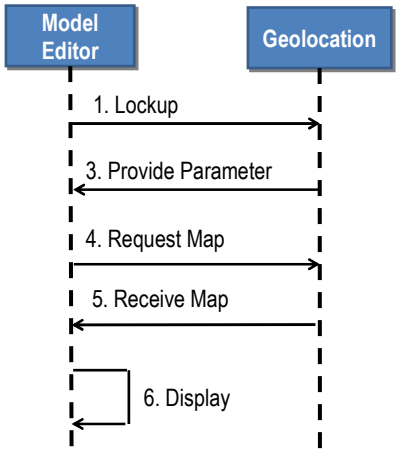
-Implement mechanisms for push and pull invocation to external services

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

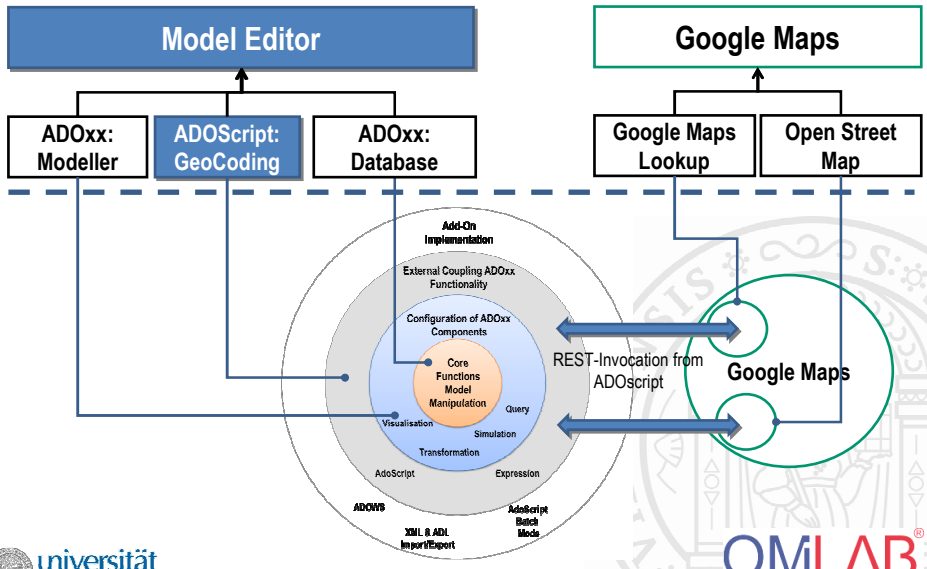
Description of GeoCoding Invocation



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

Mapping ADOxx Functionality



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

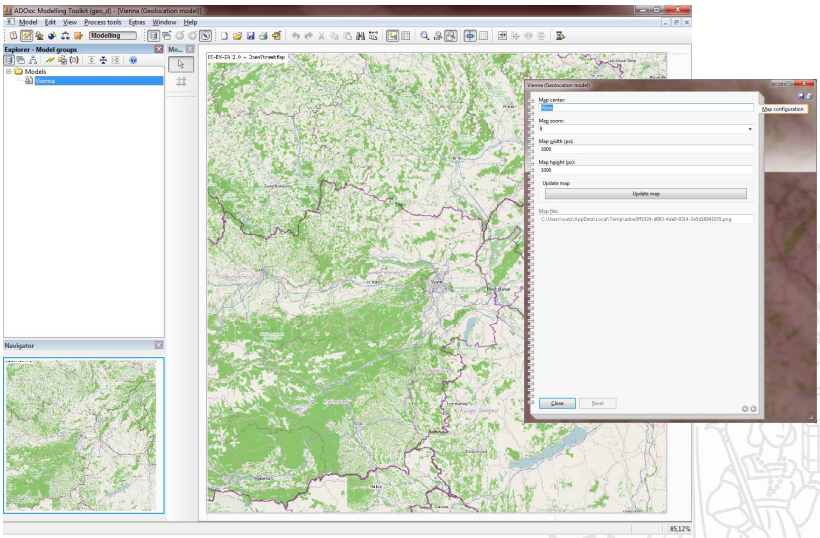
Pseudo Code: PULL Invocation

```
ITEM Notebook Button "Update map" {
  #preset by button: modelid
  locationName = getModelAttribute('locationame')
  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)
  }

}
```

...Core Operations
...Invocation Operations
...Basic UI Operations

Implementation Result: Maps in Modelling Editor

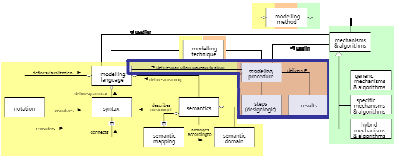


Used ADOxx Functionality: API / Web-Service Invocation

- Introduction
- Setup of Implementation Environment

- Modelling Language Implementation
  - Classes
  - Relations
  - Class Attributes and Attributes
    - **GRAPHREP**
    - **ATTRREP**
    - CLASS Cardinality
    - CONVERSION
    - Model Pointer
  - Attribute Facets
  - Model Types

- Mechanisms & Algorithms Implementation
  - Core Functions for Model Manipulation
    - Database
    - Visualisation
    - Query
    - Transformation
  - Configuration of ADOxx Components
    - Visualisation
    - Query
  - External Coupling ADOxx Functionality
    - ADOscript Triggers
    - **ADOscript Language Constructs**
    - **Visualisation ADOscript**
    - Visualisation Expression
    - Query ADOscript
    - Transformation ADOscript
  - ADD-ON Implementation
    - **ADOxx Web-Service**
    - XML / ADL Import – Export
    - ADOscript Batch Mode



Coupling BPMN and UML Use Case Diagrams

4. SCENARIO: COUPLING MODELLING LANGUAGES TO SUPPORT MODELLING PROCEDURE

Scenario Description

Case:

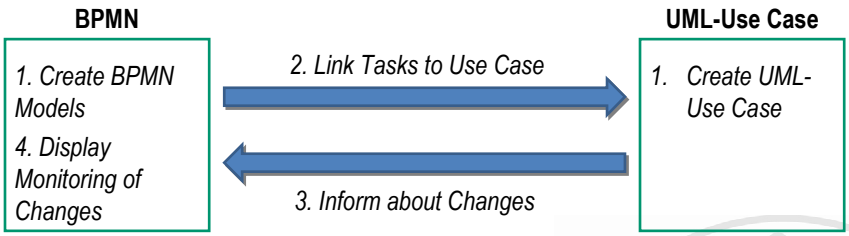
Two implementations of different modelling approaches are combined to support a common modelling procedure ("Vorgehensmodell"). The modelling approach BPMN and UML - use case models are implemented in a coupled way to enable an integrated view from processes to use cases.

GOAL:

-Demonstrate how a combined usage of two modelling approaches is realized

-Develop functionality based on combined view

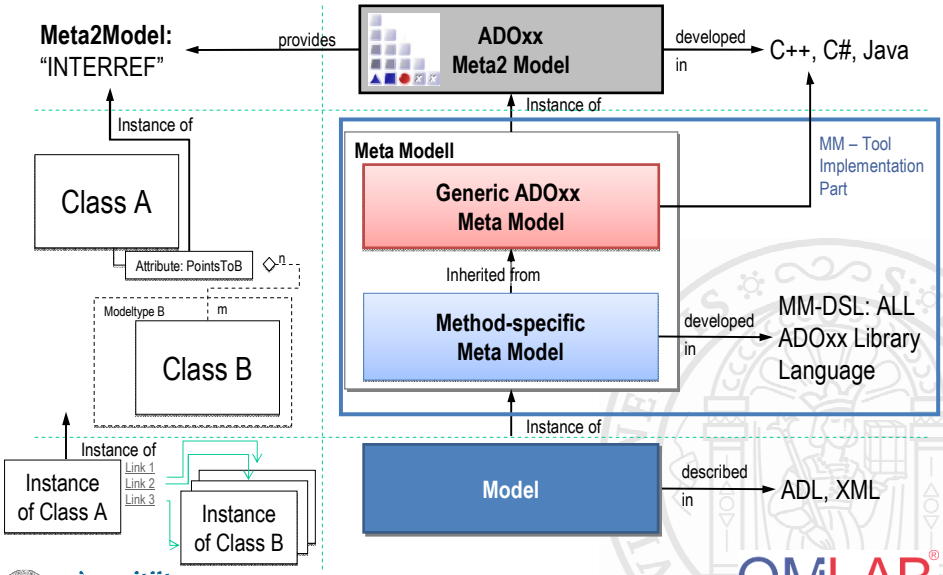
Coupling of BPMN and UML-Use Case



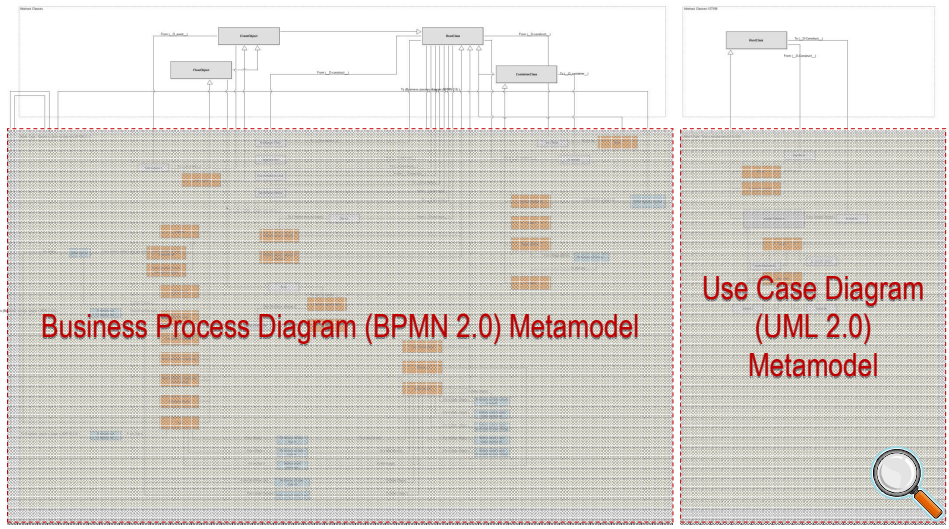
1. Interlink/Integrate BPMN task class with UML use case model to establish an integrated view
2. Provide graphical representation of change status (notification mechanism if a use case changes)

1. Provide same structure and operations as for regular use case
2. Additional attribute to hold a point to an external file or URL
3. Updated graphical representation using a dashed outline and to visualize the attribute of 2) and provide a hyperlink
4. Allow conversion between regular use case and use case (documentation)

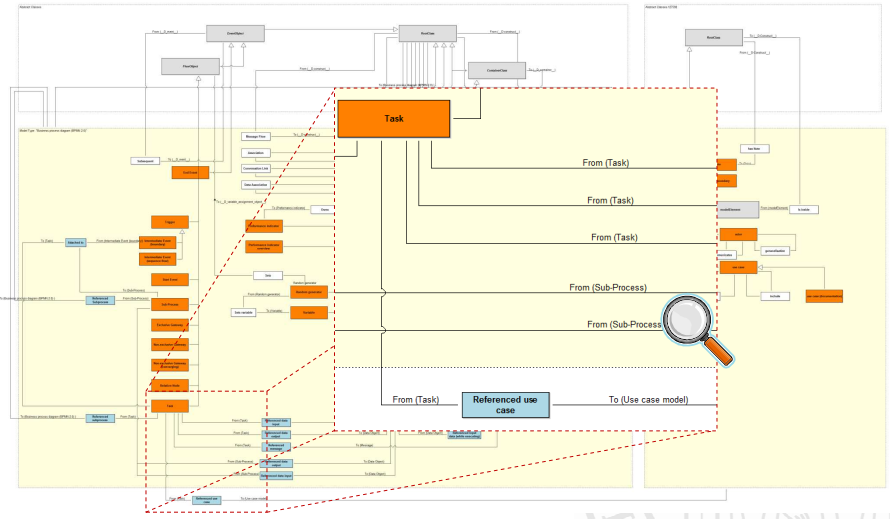
Meta Modelling Layer: Coupling Modelling Languages



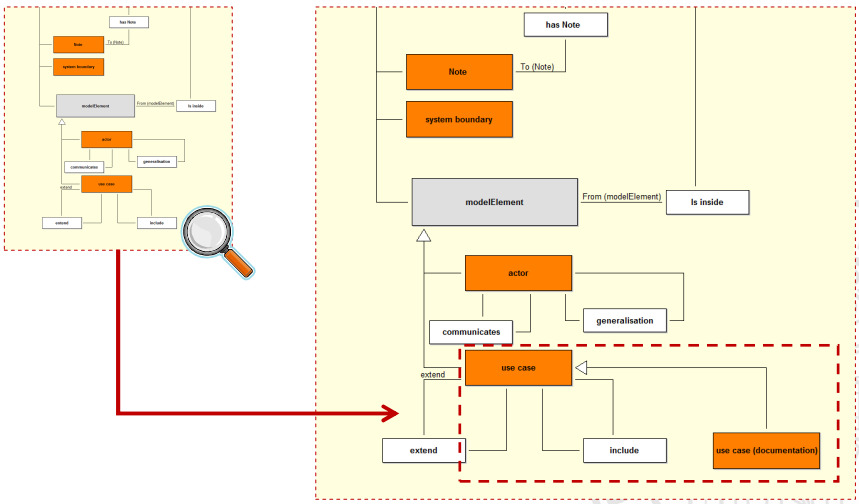
Meta Model of BPMN and UML Use Case



Integrated View of BPMN and UML Use Case Meta Model



Specialisation: CLASS "USE CASE (Documentation)"



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

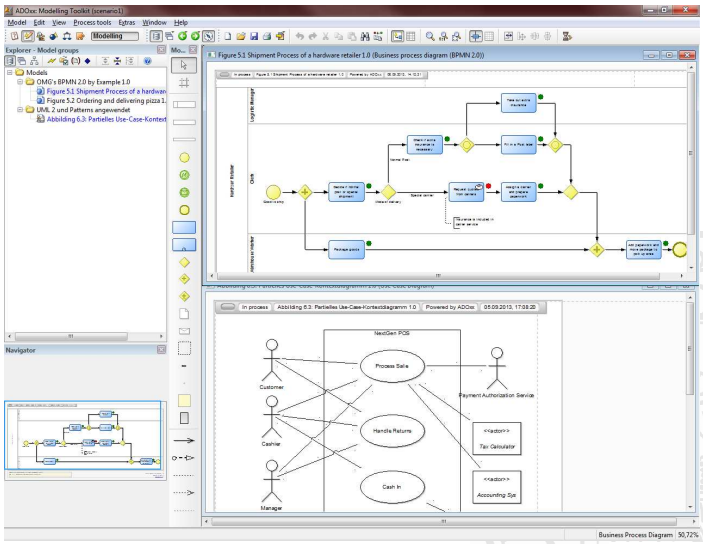
- **ADOxx Constructs for Modelling Language Extensions**
  - INTERREF attribute: the **attribute type INTERREF** enables to combine two modelling classes using a pointer within one meta model.
  - Hyperlink: the **hyperlink and model-pointer** functionality enables to navigate better within models.
  - **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

- Introduction
- Setup of Implementation Environment
- **Modelling Language Implementation**
  - **Classes**
  - Relations
  - Class Attributes and Attributes
    - **GRAPHREP**
    - **ATTRREP**
    - CLASS Cardinality
    - **CONVERSION**
    - Model Pointer
  - Attribute Facets
  - Model Types
- Mechanisms & Algorithms Implementation
  - Core Functions for Model Manipulation
    - Database
    - **Visualisation**
    - Query
    - Transformation
  - Configuration of ADOxx Components
    - Visualisation
    - Query
  - External Coupling ADOxx Functionality
    - **ADOscript Triggers**
    - ADOscript Language Constructs
    - Visualisation AdoScripts
    - Visualisation Expression
    - Query ADOscript
    - Transformation ADOscript
  - ADD-ON Implementation
    - ADOxx Web-Service
    - XML / ADL Import – Export
    - ADOscript Batch Mode

Thank you for your attention!

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