



# Meta-Modelling as a Concept: The Conceptualisation of Modelling Methods

Tutorial Team

Robert Woitsch, Hans-Georg Fill,  
Sabin Popescu, Vedran Hrgovic

## 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**

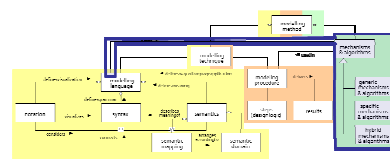
- Case: Structural Similarities of Business Processes

#### 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



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).

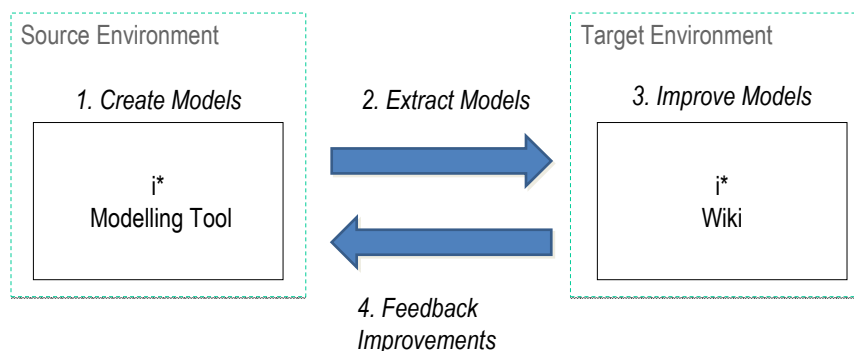
**GOAL:**

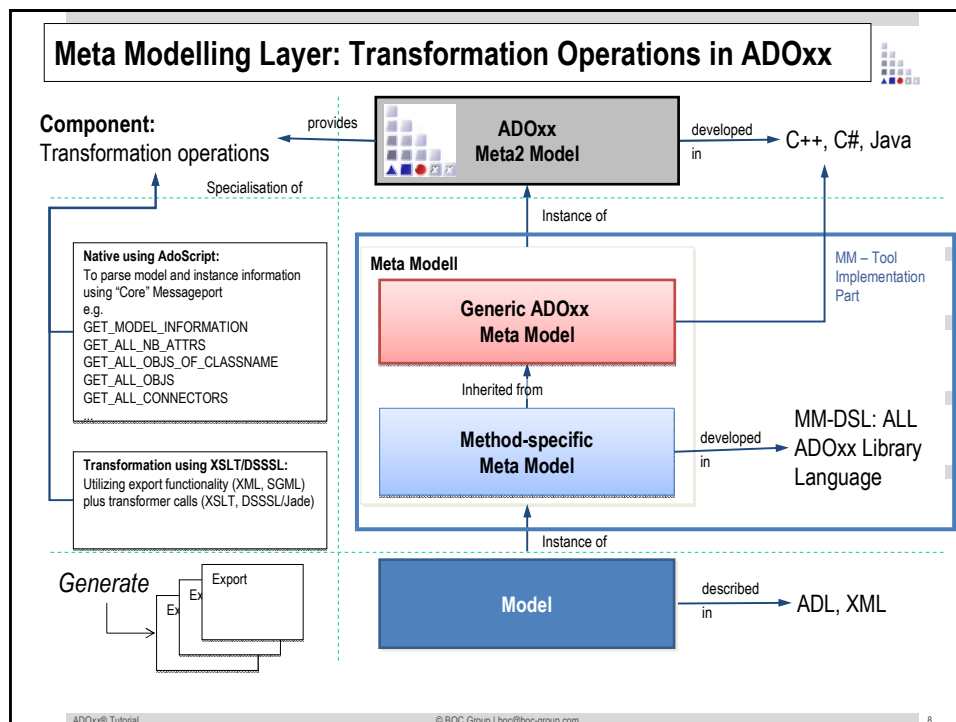
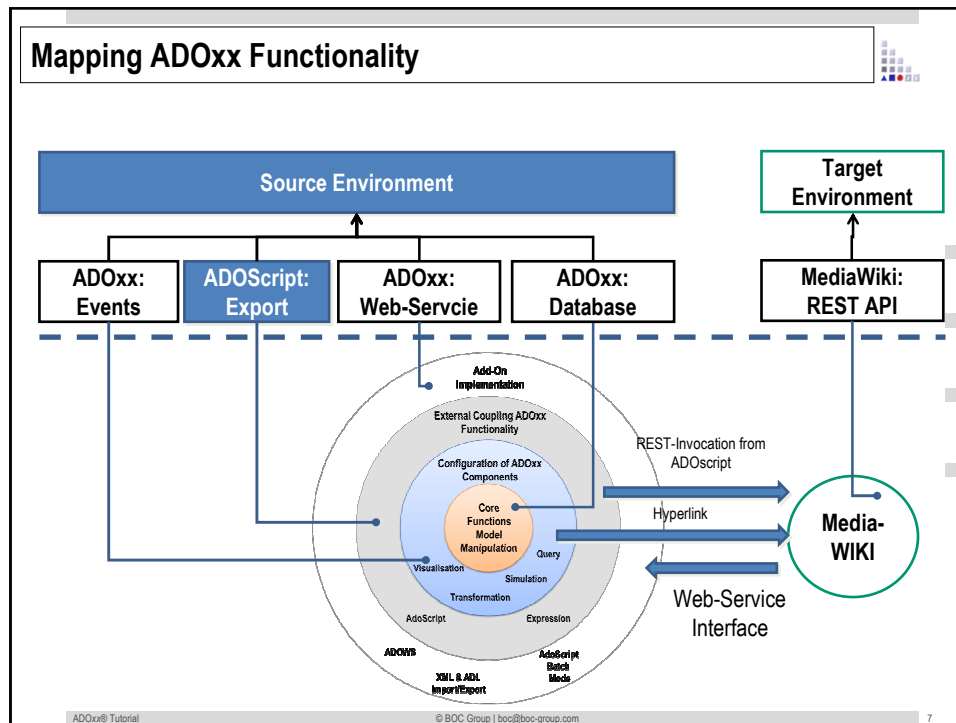
- Demonstrate usage of APIs in ADOxx to call external services
- Implement mechanisms for push and pull invocation to external services

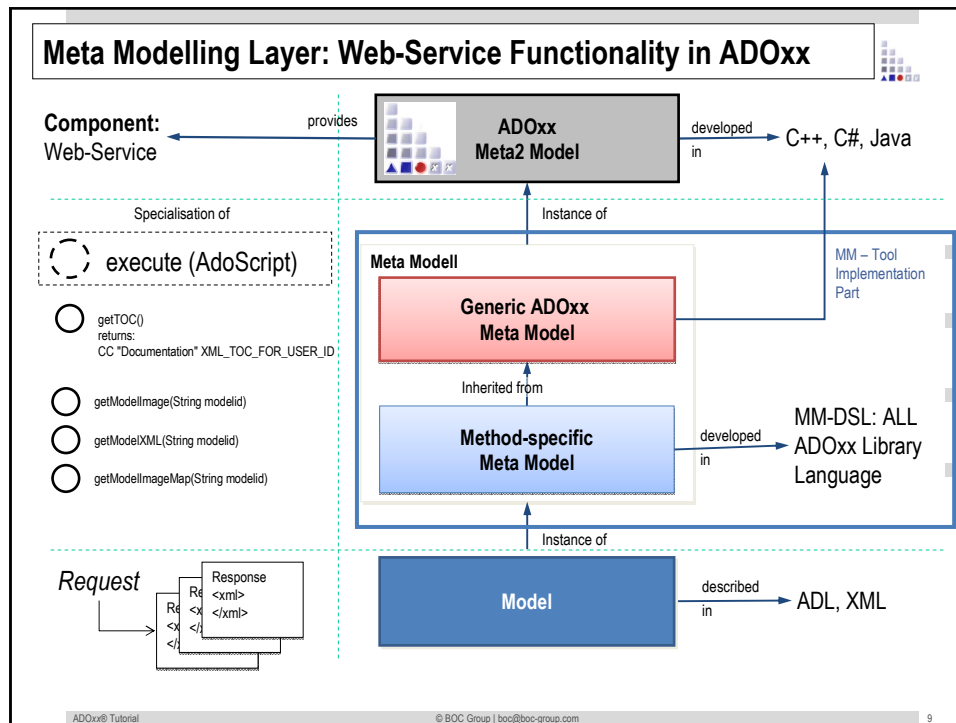
**Interaction Cases:**

- 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







### 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 Operations to enable feedback mechanisms via updating the model/instance**

Footer: ADOxx® Tutorial | © BOC Group | boc@boc-group.com | 10

## 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

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

*itemText*, *paramText*, *defaultTextValue*, *filterText* and *filterDescriptionText* 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

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

Without LINK

With LINK

## 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
}
```

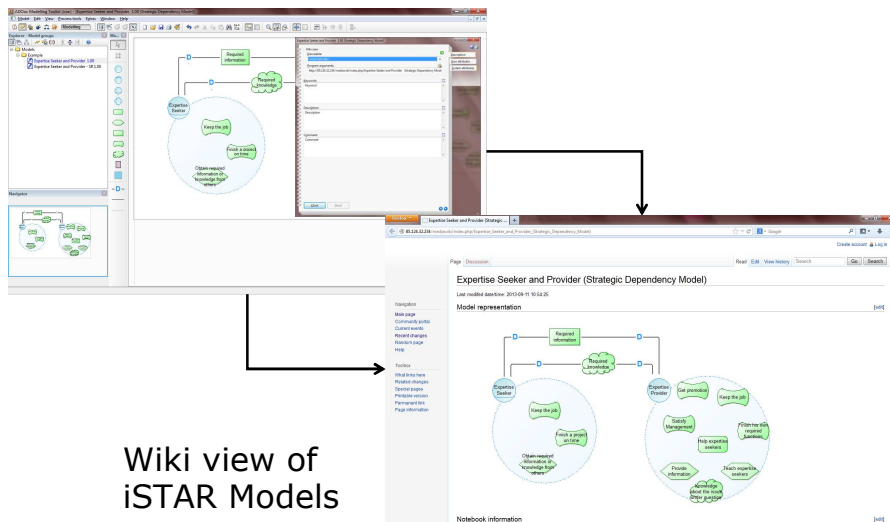
## Pseudo Code: PUSH Invocation

```
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



## 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:

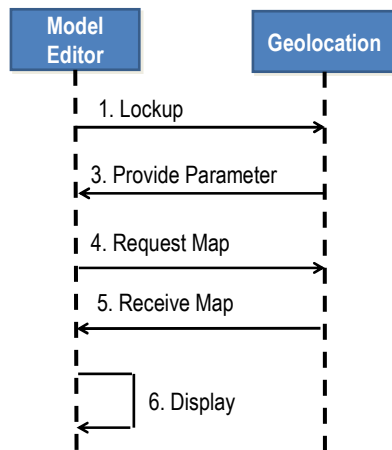
- Demonstrate usage of APIs in ADOxx to call external services
- Implement mechanisms for push and pull invocation to external services

### 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



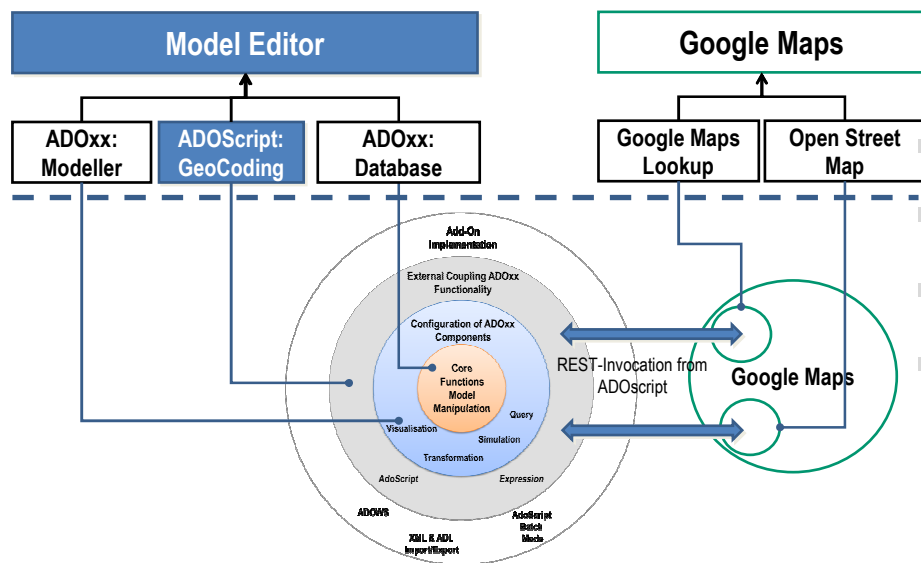
## Description of GeoCoding Invocation



### 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



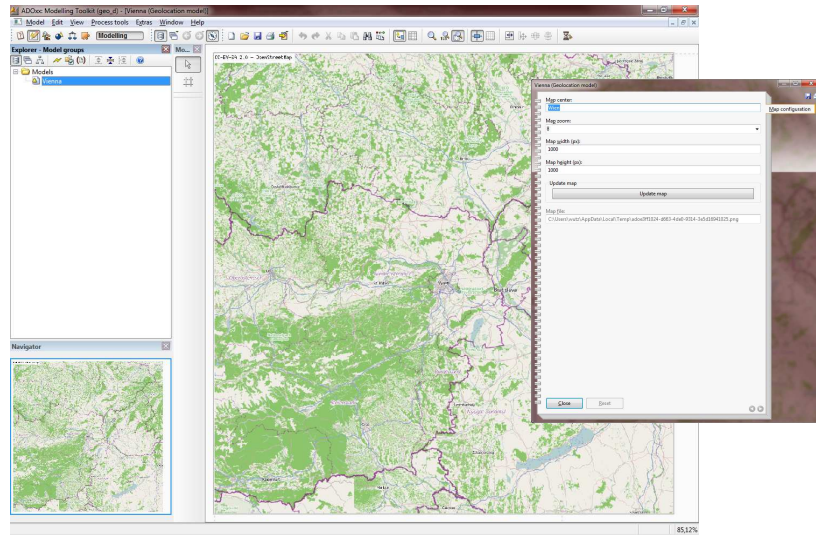
## Applied ADOxx Functionality

- **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('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)          ...Core Operations
    triggerModelRepresentationUpdate(modelid)  ...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





# HANDS-ON

Interact with MediaWiki and Google Maps

## 3. SCENARIO: MECHANISM IMPLEMENTATION FOR API / WEB-SERVICE INVOCATIONS

### How to implement the Wiki integration



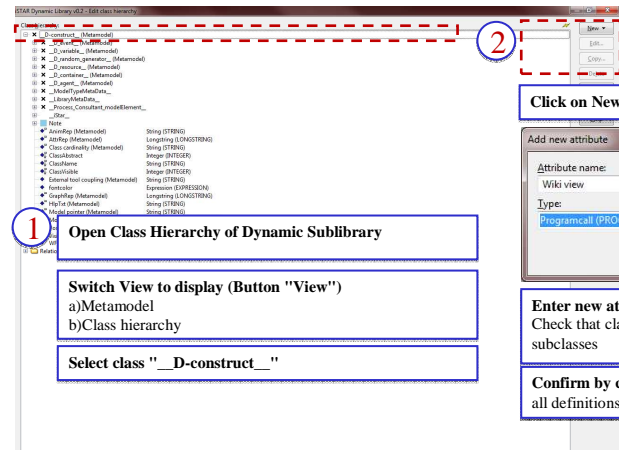
#### Pre-Condition:

MediaWiki is configured to answer ADOxx

#### Implementation Steps:

1. Add Attribute „Wiki view“ in \_D-construct\_
2. Change ATTREP of „Note, Actor, Agent, Role, Position, Goal, Task, Resource, Softgoal, Belief and Aggregation“
3. Change GRAPHREP of „Softgoal“
4. Add Save Event
5. CHANGE Modeltype ATTREP
6. Copy curl.exe, wget.exe and run\_wiki\_export.asc in directory. Set file directory accordingly.

## NEW ATTRIBUTE "Wiki view" for all classes



**1** Open Class Hierarchy of Dynamic Sublibrary

Switch View to display (Button "View")

- a) Metamodel
- b) Class hierarchy

Select class " \_D-construct "

**2** Click on New -> New attribute...

Add new attribute

Attribute name: Wiki view

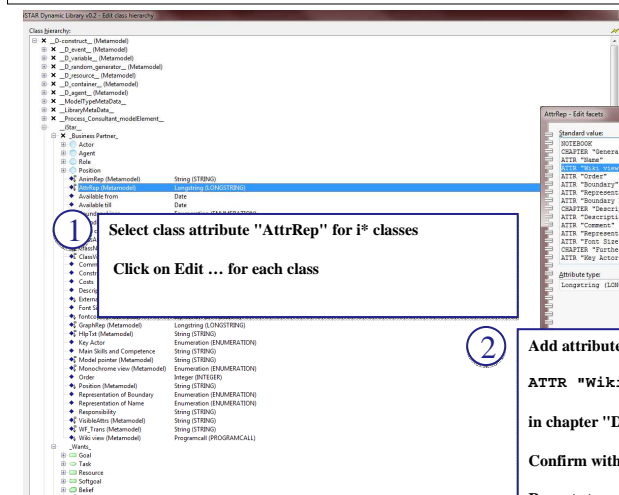
Type: Programcall (PROGRAMCALL)

OK Edit Cancel Help

Enter new attribute name "Wiki view"  
Check that class is defined for root class and inherited to all subclasses

Confirm by clicking on OK -> Attribute has been created, all definitions/setup have been inherited to subclasses.

## ADD NEW ATTRIBUTE TO NOTEBOOK OF CLASSES



**1** Select class attribute "AttrRep" for i\* classes

Click on Edit ... for each class

**2** Add attribute to notebook by adding the line

ATTR "Wiki view"

in chapter "Description"

Confirm with Close to save the change.

Repeat steps for other classes

## ADD NEW ATTRIBUTE TO NOTEBOOK OF MODELTYPE

1 Select class attribute "ISTAR Model Attributes" of class " \_ModelTypeMetaData\_"

Click on Edit ...

2 Add attribute to notebook by adding the line

ATTR "Wiki view"

in chapter "Description"

Confirm with Close to save the change.

## 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

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

Without LINK

With LINK

## UPDATE OF GRAPHREP FOR "Softgoal"

1 Select class attribute „GraphRep“

Click on Edit ...

2 Open Development Wizard/Support

## UPDATE OF GRAPHREP FOR "Softgoal"

3 Update GRAPHREP code to provide an attribute dependent representation of the new attribute

```

AVAL set-default:@" wv:"Wiki view"

IF (r = "inside")
  IF (LEN wv > 1)
    AVAL name:"Name"
    ATTR "Wiki view" text:(name) w:c:2.6cm h:c line-break:
words
  ELSE
    ATTR "Name" w:c:2.6cm h:c line-break: words
  ENDIF
ELSE
  IF (LEN wv > 1)
    AVAL name:"Name"
    ATTR "Wiki view" text:(name) w:c:2.6cm h:t y:0.8cm line-
break: words
  ELSE
    ATTR "Name" w:c:2.6cm h:t y:0.8cm line-break: words
  ENDIF
ENDIF
  
```

## EVENT HANDLER: "SaveModel"

1

### Event handler for save model

Define the event listener/handler in the ExternalCoupling Attribute

„SaveModel“ listens to the save operation for an updated model

2

```
ON_EVENT "SaveModel" {
#preset: modelid, origin ["new", "save", "saveas-new" or "saveas-save"]

# Only for strategic dependency models that are saved by the modeller
IF (origin = "save") {
  CC "Core" GET_MODEL_INFO modelid:(modelid)
  IF (modeltype = "Strategic Dependency Model") {
    SETG nProcessedModel:(modelid)
    # Update logic (implemented in an external file)
    EXECUTE file:("d:\\run_wiki_export.asc")
  }
}
```

## AdoScript Implementation run\_wiki\_export.asc

```
# Update mechanism for model/instance information into a collaborative
environment, such as wiki
# Preset: nProcessedModel
# 1. Parse model information
# Get information on the model, to construct a unique name for the wiki page:
# ModelName + ModelVersion + Modeltype
# location of the AdoScript and curl/wget extensions - needs to be a file
location and not db:\\
SETG sLocation:("d:\\")
CC "Core" GET_MODEL_INFO modelid:(nProcessedModel)
# RESULT: modelname:strValue ver:strValue version:strValue threadid:id
modeltype:strValue libid:id libtype:LibType libname:strValue access:Access
ecode:intValue
SETL sUniqueWikiPageName: (modelname + " " + version + " (" + modeltype + ")")
CC "AdoScript" MSGWIN ("Wiki view is generated for Model \" +
sUniqueWikiPageName + "\")
# get a temp file handler to store result
CC "AdoScript" GET_TEMP_FILENAME
# RESULT filename:strValue
# API calls are encoded in base64 to ensure special characters are treated
correctly
SYSTEM (sLocation + "wget.exe -O "+filename+"
http://85.124.32.234/mediawiki_adoxx/createpage.php?pagename="+.base64encode
(sUniqueWikiPageName)) #with-console-window
...
```





Any Questions?

