

# Model Queries

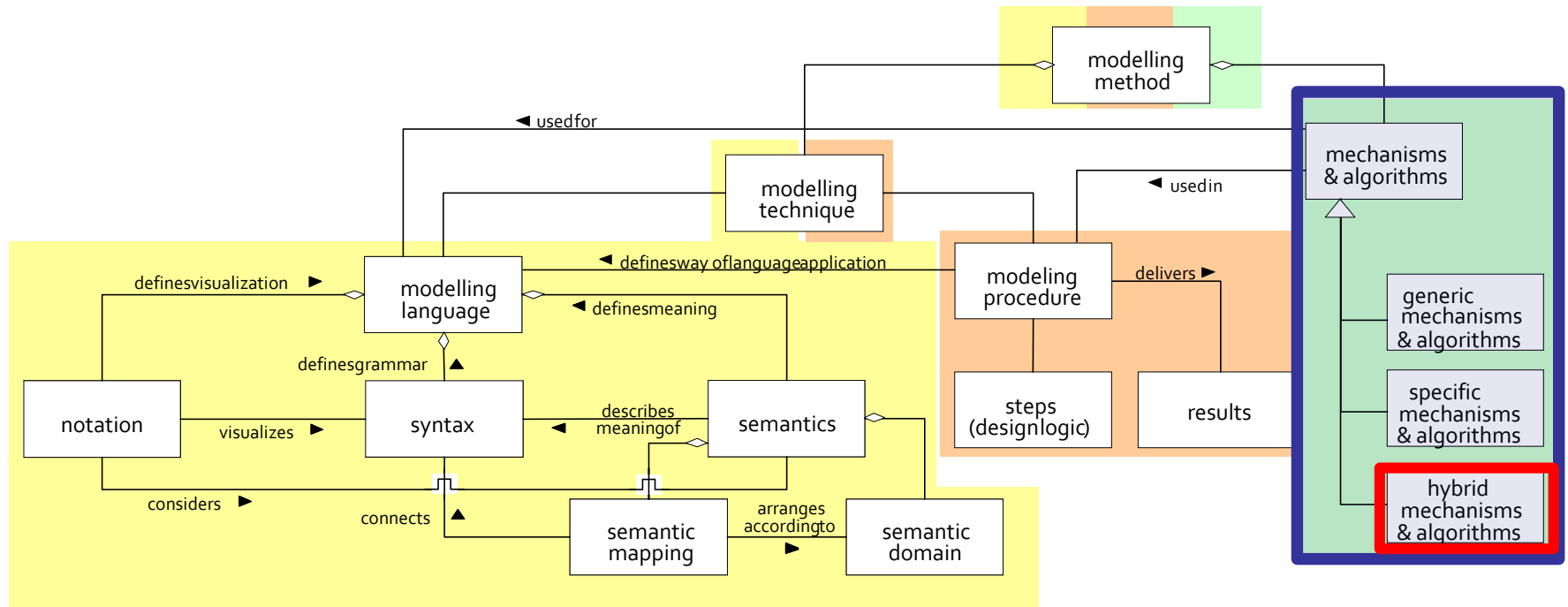
Prof. Dr. Dimitris Karagiannis  
Patrik Burzynski  
Victoria Döller

# Overview & Goals

1. Application Scenario: Model Queries for Smart Cities
2. Model Queries in ADOxx
3. Predefined Queries
4. Exercises: Smart City Model Queries
5. Lessons Learned

# 1. APPLICATION SCENARIO: MODEL QUERIES FOR SMART CITIES

# Generic Modelling Method Framework



Reference: Karagiannis, D., Kühn, H.: „Metamodelling Platforms“. In Bauknecht, K., Min Tjoa, A., Quirchmayer, G. (Eds.): Proceedings of the Third International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, September 2002, LNCS 2455, Springer, Berlin/Heidelberg, p. 182 ff.



# 1. Application Scenario: Model Queries for Smart Cities

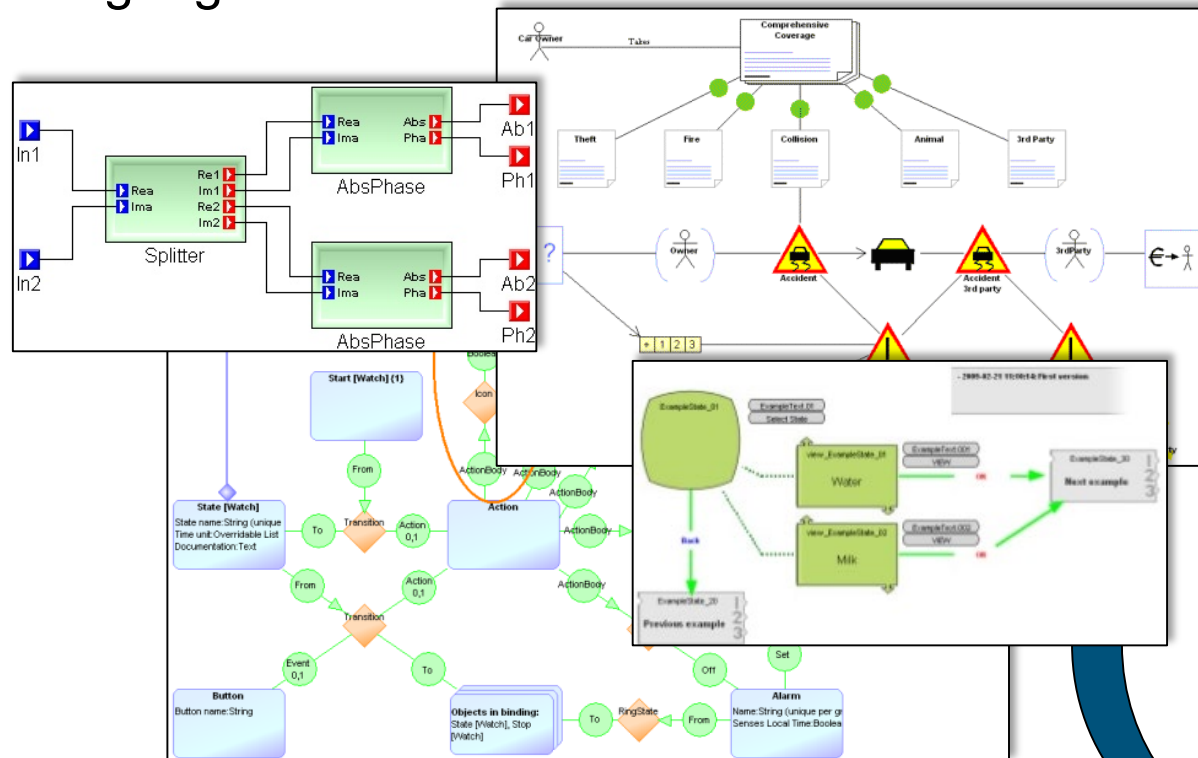
## Model Analysis e.g., Model Queries

- Analysis assists in utilizing the complex relationships within a model or between multiple models
- An analysis technique is chosen for a certain purpose
  - Analysis supports the decision making process
  - Analysis assists in understanding, visualizing and communicating structured and unstructured data
  - Contextualization, verification, validation, conformance, and consistency checking
- Manual model analysis can be costly

# 1. Application Scenario: Model Queries for Smart Cities

## Requirements for model queries

How to realize model analysis capabilities in domain-specific modeling languages?



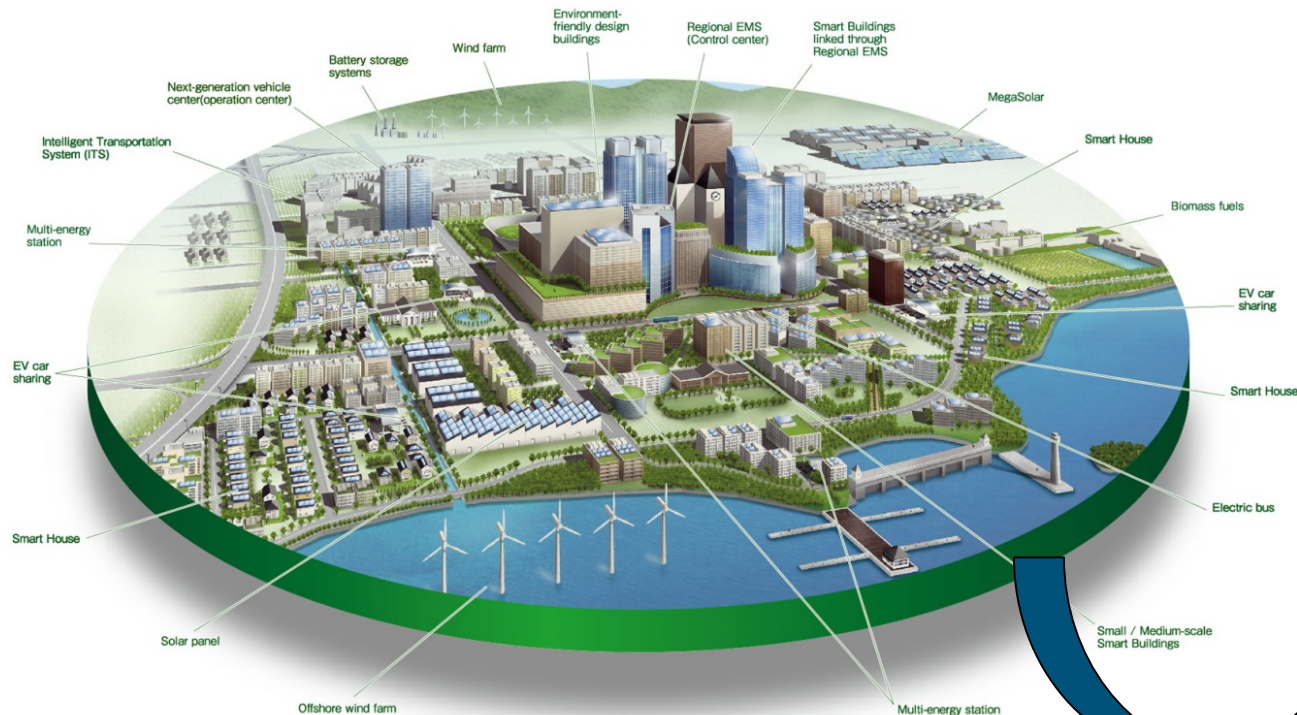
How would you use the domain-specific concepts to realize a model analysis? What would be the questions answered thereby?

Source: <http://www.slideshare.net/JuhaPekkaTolvanen/20-examples-on-domain-specific-modeling>  
<http://www.metacase.com/blogs/jpt/blogView?showComments=true&entry=3388831430>  
<http://www.isis.vanderbilt.edu/projects/gme/>

# 1. Application Scenario: Model Simulation for Smart Cities

## Requirements for model simulation

How to realize model analysis capabilities in a Smart City environment?



How would you use the domain-specific concepts to realize a model analysis? What would be the questions answered thereby?



Source: [http://www.districtoffuture.eu/uploads/imagenes/imagenes\\_meetinpoint\\_smart-city\\_2b637ab6.jpg](http://www.districtoffuture.eu/uploads/imagenes/imagenes_meetinpoint_smart-city_2b637ab6.jpg)

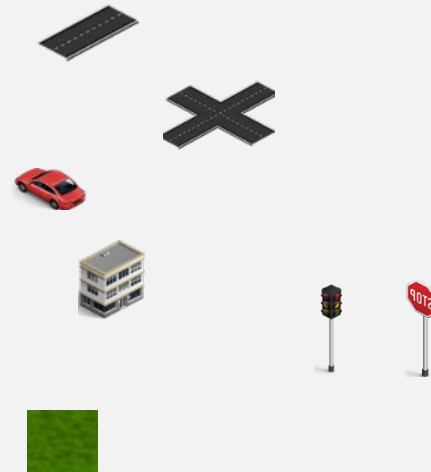
# 1. Application Scenario: Model Queries for Smart Cities

## Requirements for model queries

- What are questions we want to answer by model queries?
- How can we use the Smart City concepts and their attributes in a model query?

Requirements for new modeling concepts:

- Streets
- Crossings
- Cars, Citizens
- Buildings
- Traffic Lights and Traffic Signs
- Green areas





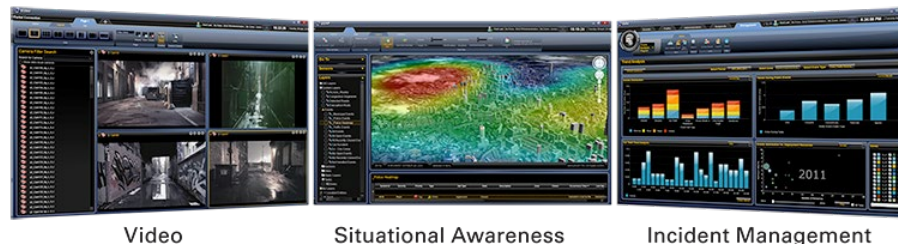
# 1. Application Scenario: Model Queries for Smart Cities

## Model Query Scenarios

- There is a building on fire!
- Smart City response:
  - Notify the emergency services
  - Block public traffic in the area
  - Find people in danger by locating their smart devices
  - Order smart buildings to guide people out of danger zones
  - Notify relevant people in proximity for quick first response (firefighters, doctors, policemen, ...)



www.psfk.com



Video

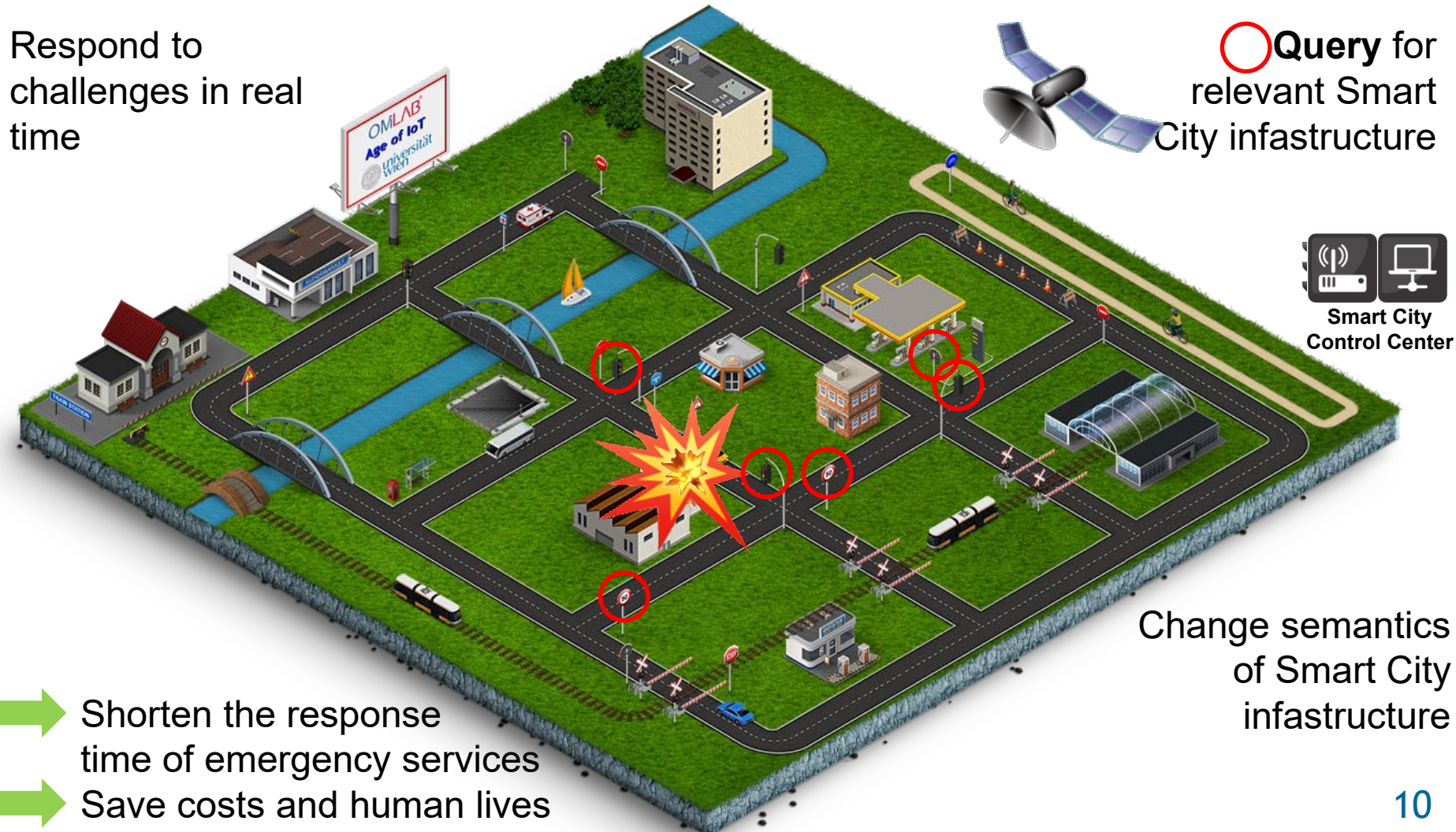
Situational Awareness

Incident Management

# 1. Application Scenario: Model Queries for Smart Cities

## Emergency Scenario: Block Public Traffic in the Area

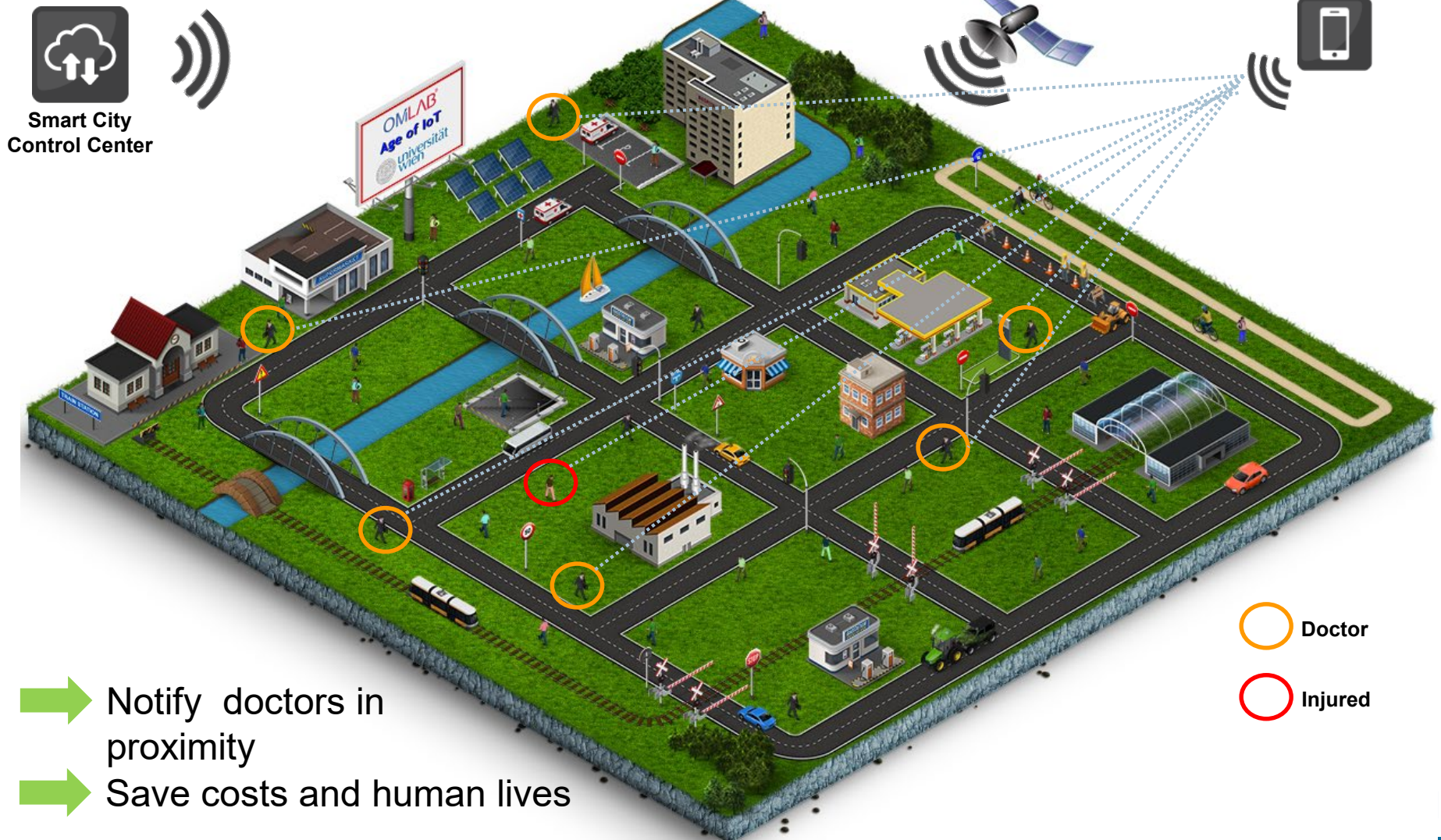
Respond to challenges in real time





# 1. Application Scenario: Model Queries for Smart Cities

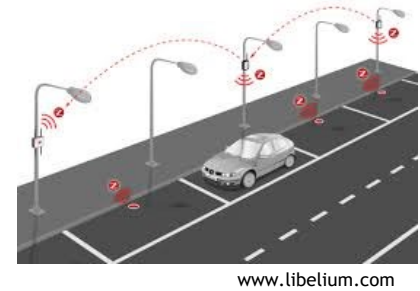
## Emergency Scenario: Medical Response



# 1. Application Scenario: Model Queries for Smart Cities

## Parking Scenario: Find a free Parking Spot

Searching for parking spots is an inefficient task. *“Many drivers spend on average 20 minutes while looking for a parking spot. This increases CO2 emissions and most important of all; people waste their time.”* [1]



- Smart Parking
  - Improves the urban environment
  - Reduces pollution
  - Generates revenue
  - Optimizes human resources
  - Creates happiness
  - Is vandalism proof
  - Easy to maintain
  - Reduces accidents

[1] Amsterdam Smart City : <http://amsterdamsmartcity.com/projects/detail/id/64/slug/smart-parking?lang=en>

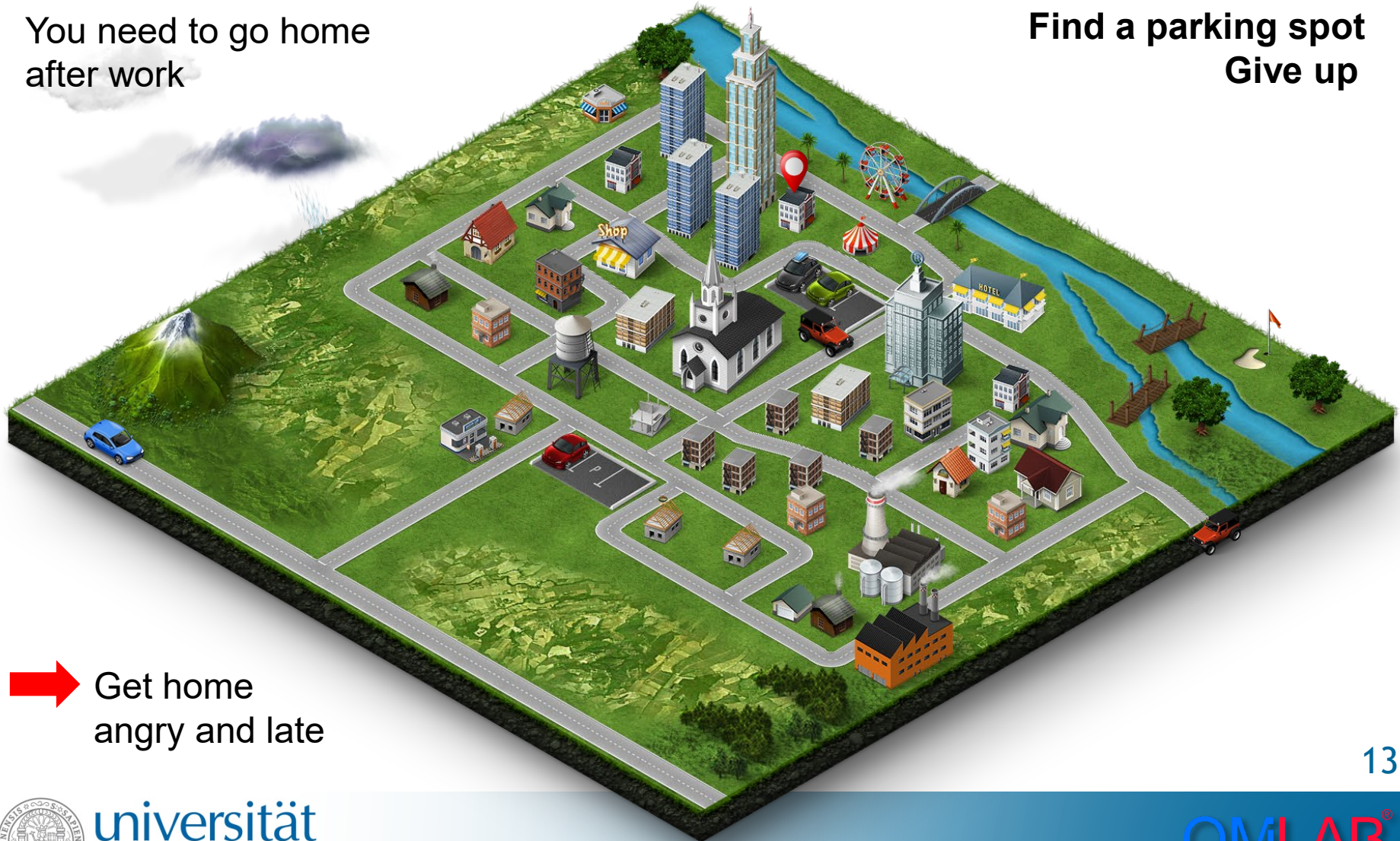


# 1. Application Scenario: Model Queries for Smart Cities

## Parking Scenario: Traditional Approach

You need to go home  
after work

Find a parking spot  
Give up



➔ Get home  
angry and late



# 1. Application Scenario: Model Queries for Smart Cities

## Parking Scenario: Smart City Approach

You need to go home after work

**Query** for a smart parking spot and book a reservation



→ Arrive at home happy and relaxed

# 1. Application Scenario: Model Queries for Smart Cities

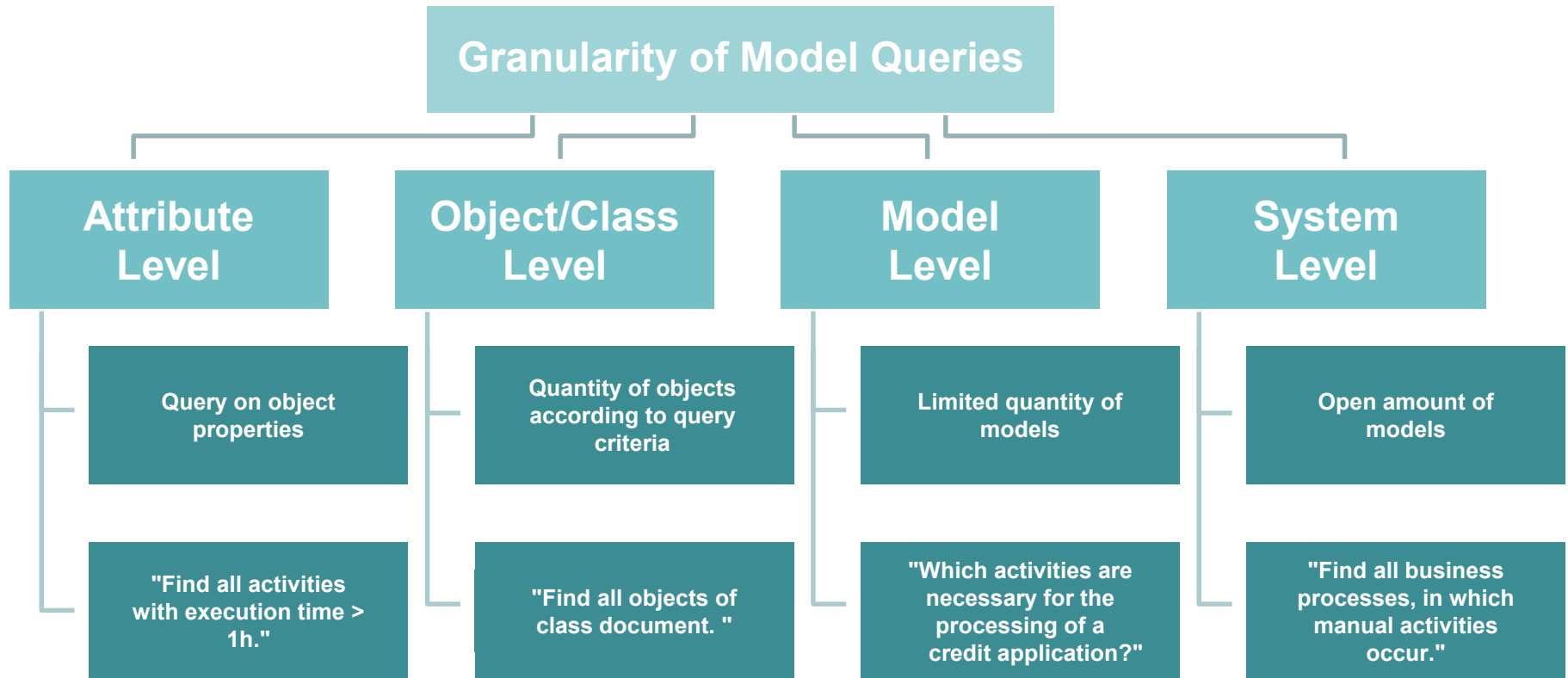
How can we realize  
model queries in ADOxx?



## 2. MODEL QUERIES IN ADOXX



## 2. Model Queries in ADOxx



## 2. Model Queries in ADOxx



ADOxx provides two types of queries:

- **Standard Queries**

- Standardized queries are utilized by configuration. Pre-defined building blocks of model queries can be customized with the domain-specific concepts by the user. No ADOxx Query Language (AQL) knowledge is required for definition and execution of these queries.

- **User-defined Queries**

- Queries which are defined by the user through combining elements of the AQL query syntax. AQL knowledge is required for definition and execution of these queries.



## 2. Model Queries in ADOxx

### The ADOxx Query Interface

- **Query scope**
  - Select the scope of the query, i.e., on model or on model content basis
- **Standardized Queries**
  - Select out of the provided standardized queries the one you want to customize by parameterization
- **Query Options**
  - Queries can be **added** in order to create combined queries or **evaluated** by execution
- **User-Defined Queries**
  - This sections show the ADOxx Query Language (AQL) representation of the query. Users can also edit the AQL code manually and combine new elements with **logical operators AND, OR, DIFF**

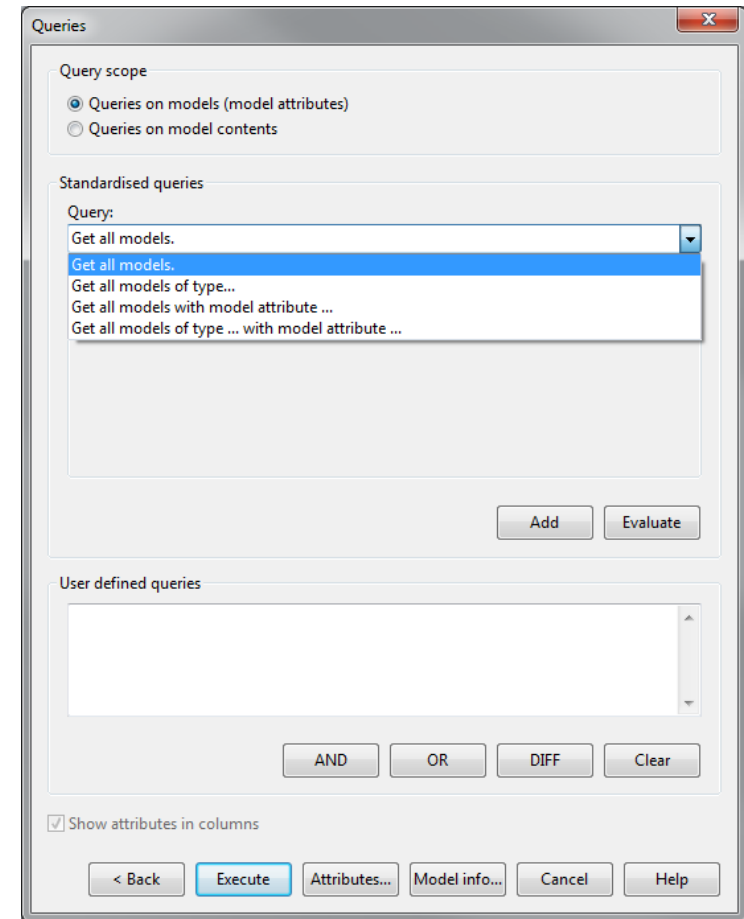
The screenshot shows the 'Queries' window with the following sections:

- Query scope:** Two radio buttons. The first, 'Queries on models (model attributes)', is selected. The second is 'Queries on model contents'.
- Standardised queries:** A 'Query:' dropdown menu showing 'Get all models.'. Below it is an 'Input field' containing the text 'Get all models.'.
- Buttons:** 'Add' and 'Evaluate' buttons are located below the standardized queries section.
- User defined queries:** A large text area for manual query input. Below it are buttons for 'AND', 'OR', 'DIFF', and 'Clear'.
- Options:** A checkbox labeled 'Show attributes in columns' is checked.
- Footer:** A row of buttons: '< Back', 'Execute', 'Attributes...', 'Model info...', 'Cancel', and 'Help'.

## 2. Model Queries in ADOxx

### Standard Queries – Queries based on Models

- Get all models
- Get all models of type ...
  - Returns all models of a defined **modeltype**
- Get all models with model attribute ...
  - Returns all models with a defined **value** of a defined **model attribute**
- Get all models of type ... with model attribute ...
  - ...
  - Returns all models of a defined **modeltype** with a defined **value** of a defined **model attribute**

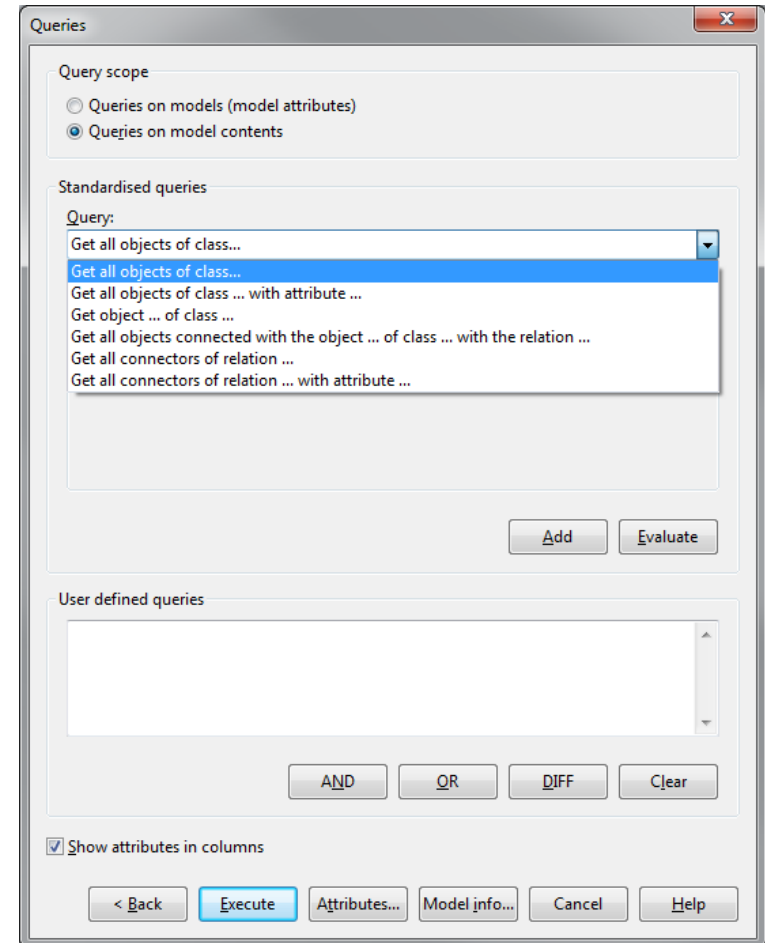




## 2. Model Queries in ADOxx

### Standard Queries – Queries based on Model Content I

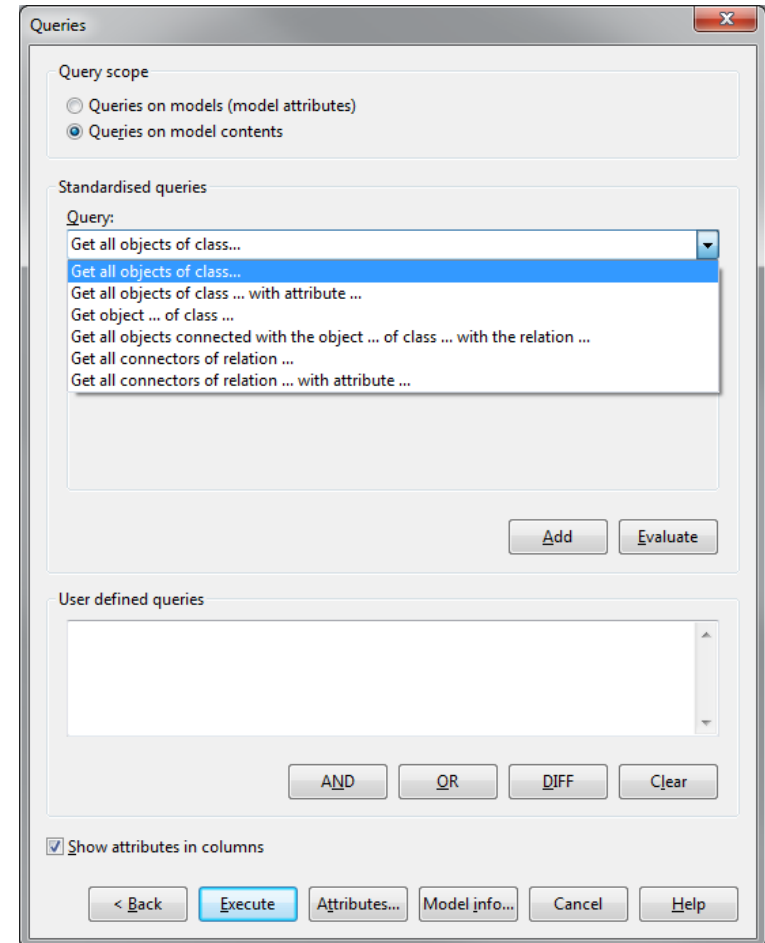
- **Get all objects of class ...**
  - Returns all objects of a defined **class**
- **Get all objects of class ... with attribute ...**
  - Returns all objects of a defined **class** with a defined **value** of a defined **class attribute**
- **Get object ... of class ...**
  - Returns the object of a defined **class** that has the defined **name**
- **Get all objects connected with the object ... of class ... with the relation ...**
  - Returns all objects that have a defined **relationship** with an object defined by **class** and **name**.



## 2. Model Queries in ADOxx

### Standard Queries – Queries based on Model Content II

- Get all connectors of relation ...
  - Returns all connectors of a defined relation class
- Get all connectors of relation ... with attribute ...
  - Returns all connectors of a defined relation class with a defined value of a defined class attribute



## 2. Model Queries in ADOxx

### User-defined Queries I

- ‘<’ and ‘>’ in this order are used to represent a **class**, a **relation**, a **model**, a **model type**, etc. (e.g. <"A">, <"My Model 01">)
- ‘:’ is used for specifying the class of a certain object, or the model where a specific class is included (e.g. <"A": "My Model 01": "My First Model Type">)
- Example of SmartParking Model

(<"Road": "1\_SmartParking": "Analysis-SmartParking">)

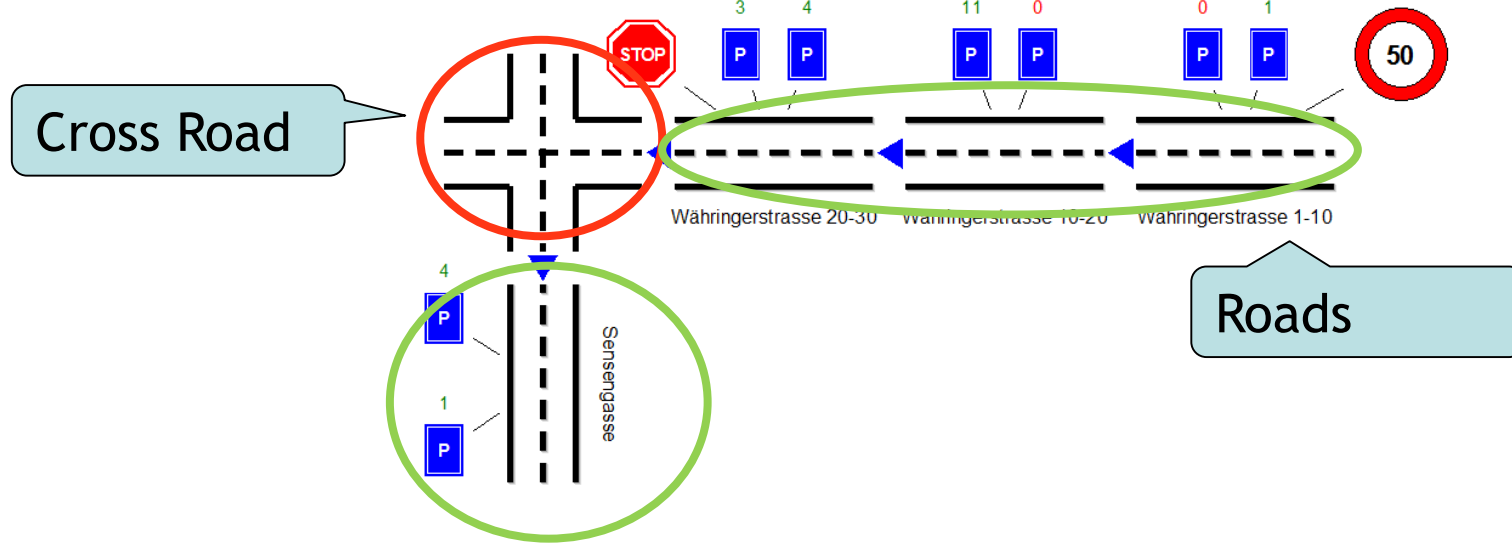
Class

Model Name

Model Type

- ‘{’ and ‘}’ are used to represent the object with the specified name (e.g.: {"A01"})

# Example



({"CrossRoad-12628"})



(<"Road">)



## 2. Model Queries in ADOxx

### User-defined Queries II

- **Queries on classes**

- the result is all objects of the specified class

- **Examples (generic):**

- `<"class_name">`
  - `{"obj_name":"class_name"}`
  - `<"class_name":"Model001":"MyModelType">`
  - `{"obj_name":"class_name":"Model001":"MyModelType"}`

- **Examples (concrete):**

- `<"Road">`
    - `→ (Sensengasse, Währingerstrasse 1-10,10-20,20-30)`
  - `{"Währingerstrasse 20-30":"Road"}`
    - `→ (Währingerstrasse 20-30)`
  - `<"Road":"1_SmartParking":"Analysis-SmartParking">`
    - `→ (Sensengasse, Währingerstrasse 1-10,10-20,20-30)`
  - `{"Währingerstrasse 20-30":"Road":"1_SmartParking":"Analysis-SmartParking"}`
    - `→ (Währingerstrasse 20-30)`

**Attention: Make sure that you  
use " instead of “**

## 2. Model Queries in ADOxx

### User-defined Queries III

- ‘(’ and ‘)’ are used for deciding the order in which logical operators are evaluated i.e. ‘a OR b AND c’ and ‘(a OR b) AND c’ return different results.
- ‘[’ and ‘]’ are used to introduce a **criteria to an AQL expression** ([?"Radius">"10"])
- ‘?’ is used for imposing a condition on an **attribute** in the query criteria  
(e.g.: <"A">[?"Description" like "\*OK\*"] returns all objects of class A, whose **attribute** „Description" contains the word „OK")
- ‘!’ is used for imposing a condition on a **variable during the simulation**  
(e.g.: (<"A">[!"objectCount">"10"]) OR (<"B">[!"objectCount"<="10"])) returns all objects of class A, if the variable objectCount is higher than 10 and all objects of class B otherwise.
- AND: The result is the intersections set of the two expressions result sets
- OR: The result is the union set of the two expressions result sets
- DIFF: The result is the difference set of the two expression result sets

## 2. Model Queries in ADOxx

### User-defined Queries IV

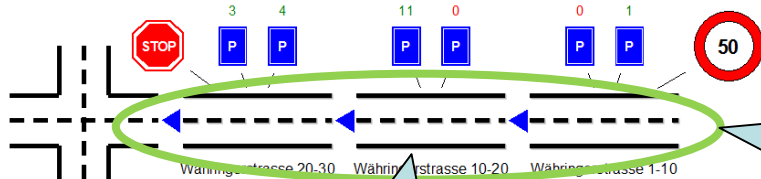
- **Relation Classes** `<AQL expression> '->' | '<-' | '->>' | '<<-'`  
`<Relation>`
  - Result contains all objects which are linked through the given relation with at least one object from the AQL expression
- **Examples (generic):**
  - `'->'` returns all direct targets of the relation
  - `'<-'` returns all direct start objects of the relation
  - `'->>'` returns all transitive targets of the relation
  - `'<<-'` returns all transitive start objects of the relation

## 2. Model Queries in ADOxx

### User-defined Queries V

- ‘->’, ‘<-’, ‘->>’, ‘<<-’, ‘-->’, ‘-->>’, ‘<--’ are used for creating AQL expressions that involve relations in the query criteria
- ‘>’ and ‘<’ **in this order** are used for filtering the results of a query by a specified class (e.g.: <"A"> <<- "Relation" >"B"< has as results all objects that fulfill the query criteria <"A"><<- "Relation" AND are of class B)

(<"CrossRoad"><<- "Subsequent" >"Road"<)



CrossRoad

Road

Result

all road objects  
that have a  
subsequent  
relation to  
crossRoad



## 2. Model Queries in ADOxx

### User-defined Queries VI

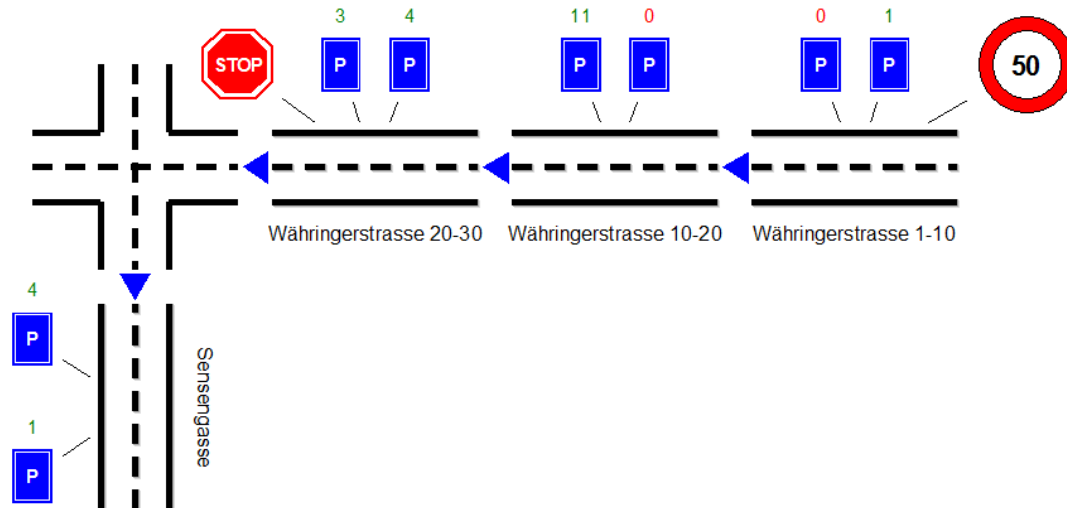
'->' returns all direct targets of the relation  
'<-' returns all direct start objects of the relation

#### Examples (concrete):

- {"Währingerstrasse 10-20"}-> "Subsequent"
- {"Währingerstrasse 10-20"}<- "Subsequent"
- <"Road">-> "Subsequent"
- <"Road"><- "Subsequent"

Target objects

All start objects  
which have a  
subsequent relation  
to a road element



## 2. Model Queries in ADOxx

### User-defined Queries VI

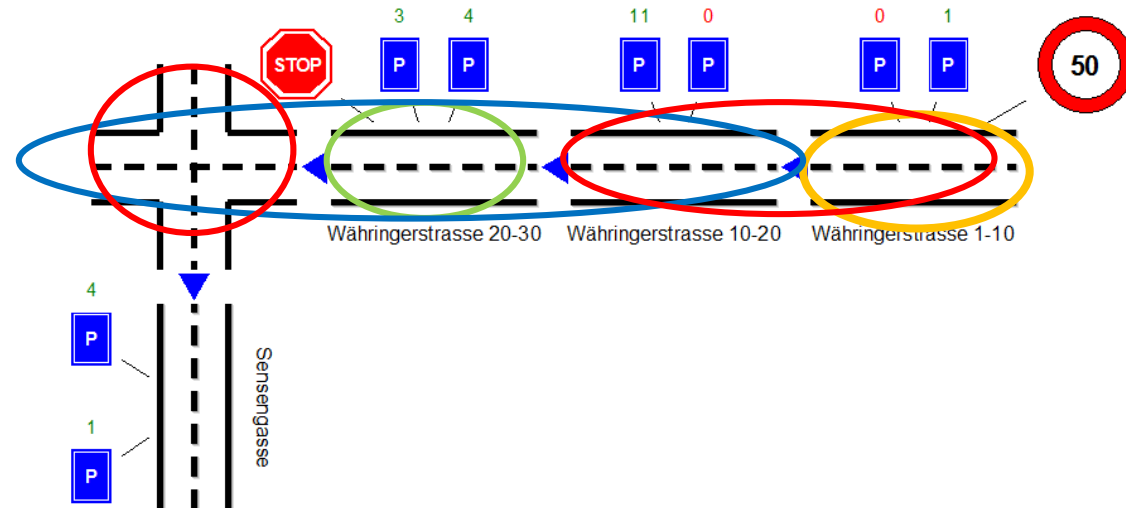
'->' returns all direct targets of the relation  
'<-' returns all direct start objects of the relation

#### Examples (concrete):

- {"Währingerstrasse 10-20"}-> "Subsequent"
- {"Währingerstrasse 10-20"}<- "Subsequent"
- <"Road">-> "Subsequent"
- <"Road"><- "Subsequent"

Target objects

All start objects which have a subsequent relation to a road element



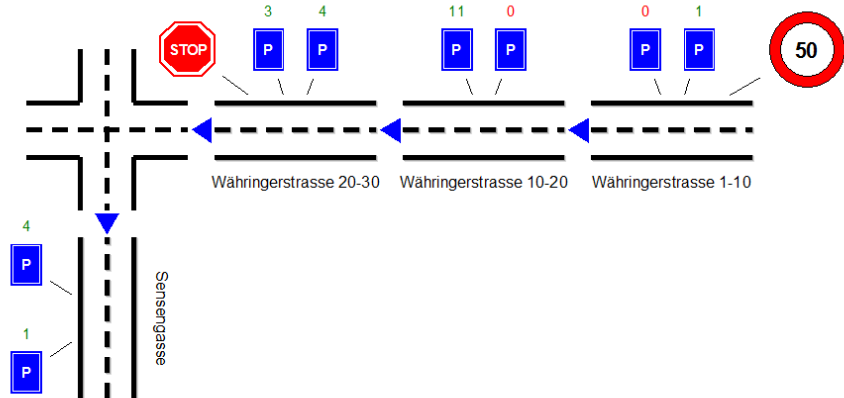
## 2. Model Queries in ADOxx User-defined Queries V

'->' returns all direct targets of the relation  
'<-' returns all direct start objects of the relation

All target elements of W10-20  
which are connected with  
subsequent

### ■ Examples (concrete):

- {"Währingerstrasse 10-20"}<<- "Subsequent"
- ({"Währingerstrasse 10-20"}->>"Subsequent") -> "Subsequent"
- <"Road"><<- "Subsequent"
- <"Road">-> "Subsequent" >"Road"<







## 2. Model Queries in ADOxx

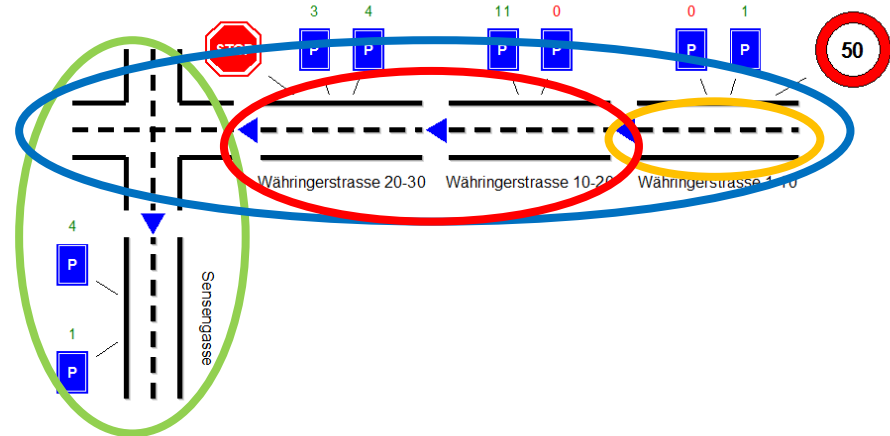
### User-defined Queries V

'->' returns all direct targets of the relation  
'<-' returns all direct start objects of the relation

#### Examples (concrete):

- {"Währingerstrasse 10-20"}<<- "Subsequent" 
- ({ "Währingerstrasse 10-20" }->>"Subsequent") -> "Subsequent" 
- <"Road"><<- "Subsequent" 
- <"Road">-> "Subsequent" >"Road"< 

All target elements of W10-20  
which are connected with  
subsequent



## 2. Model Queries in ADOxx

### User-defined Queries VI

<AQL expression>   '->' | '<-'   '<' <Relation> '>'

- **Result contains all relations** which have as start or target object one of the objects in the AQL expression

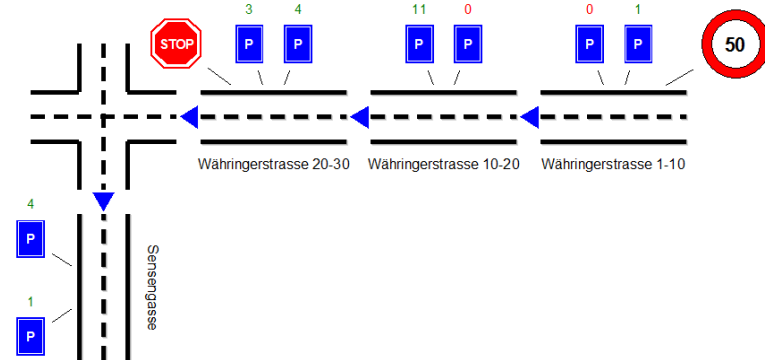
- **Examples (generic):**

- '->' returns all connectors originating from the objects of the AQL expression
- '<-' returns all connectors ending in the objects of the AQL expression

**Please note similarities and differences with before: if you use the '<' and '>' symbols, the result contains the connectors and if you don't use them, it contains the objects!**

- **Examples (concrete):**

- <"Road"> -> <"Subsequent">
- <"Road"> <- <"Subsequent">
- {"Sensengasse"} <- <"Subsequent">
- {"Währingerstrasse 20-30"} -> <"Subsequent">



## 2. Model Queries in ADOxx

### User-defined Queries VI

<AQL expression> '->' | '<-' '<' <Relation> '>'

- **Result contains all relations** which have as start or target object one of the objects in the AQL expression

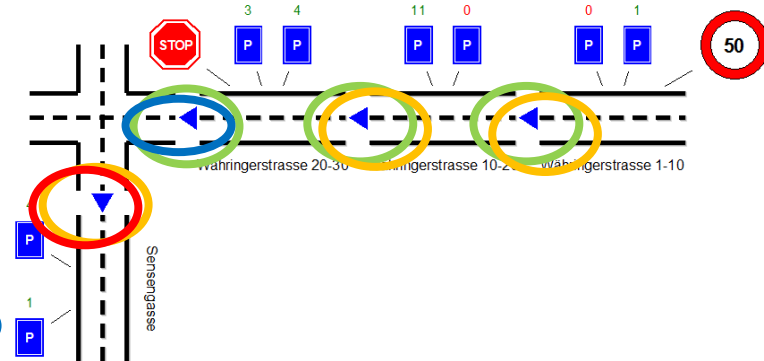
#### ■ Examples (generic):

- '->' returns all connectors originating from the objects of the AQL expression
- '<-' returns all connectors ending in the objects of the AQL expression

**Please note similarities and differences with before: if you use the '<' and '>' symbols, the result contains the connectors and if you don't use them, it contains the objects!**

#### ■ Examples (concrete):

- <"Road"> -> <"Subsequent">
- <"Road"> <- <"Subsequent">
- {"Sensengasse"} <- <"Subsequent">
- {"Währingerstrasse 20-30"} -> <"Subsequent">



## 2. Model Queries in ADOxx

### User-defined Queries VII

#### Interrefs: <AQL expression> '-->' | '-->>' <InterRef>

- The result contains all objects which are referenced in the specified attribute of any of the objects in the AQL expression
- The '-->>' operator returns all objects which are transitively referenced in the specified attribute of any of the objects in the AQL expression
- The '<--' operator returns all objects which refer any of the objects in the AQL expression
- **Examples (generic):**
  - <"class\_name"> --> "interref\_name"

## 2. Model Queries in ADOxx

### User-defined Queries VIII

**Attributes: <AQL expression> '[' <Value> <Operator> <Value> ']**

- The result contains all objects whose attributes fulfill the criteria
- Constants (numbers, strings) can only be on the right hand side of the operator
- On the left hand side of the operator only attributes or variable references are allowed
- **Examples (concrete):**
  - (<"Road">[? "Name" like "Währingerstrasse\*"]) AND (<"Road">[?"Max Speed" > 30])
  - (<"Smart\_Parking">[? "Parking\_Status" = "Open"])



## 2. Model Queries in ADOxx

### User-defined Queries IX

**Record Classes and Attribute Profiles:** <AQL expression> '['<Value>']' '['<Value>  
<Operator> <Value>']'

- The result contains all objects of the start query where their record attribute or attribute profile fulfills the defined criteria
- The first value specifies the name of the record attribute or attribute profile.
- See above the rules for the second expression

Note: In case of a record attribute, the criteria is always fulfilled, if at least one table row of the record attribute meets the defined criteria.

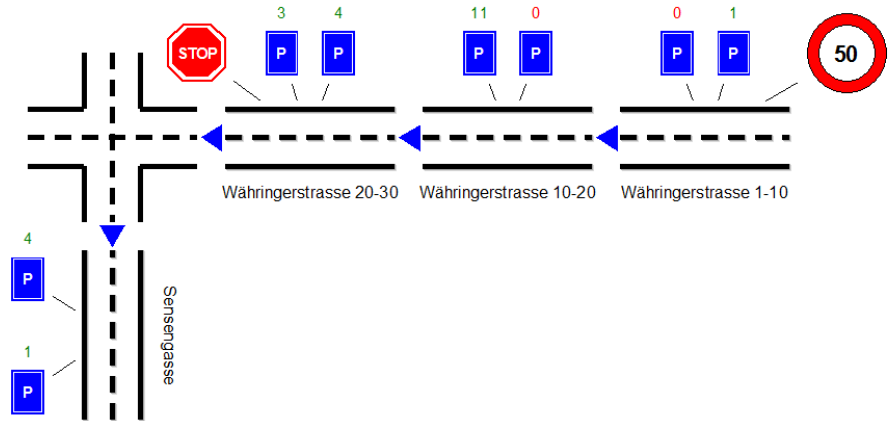
#### ■ Examples:

- |                                  |                              |
|----------------------------------|------------------------------|
| – record attribute:              | – attribute profile:         |
| – <"class_name">["attr_name"]    | – <"classname">              |
| ["column_name" = "column_value"] | ["attr_name"]["AP_attr_name" |
|                                  | = "attr_value" ]             |

## 2. Model Queries in ADOxx

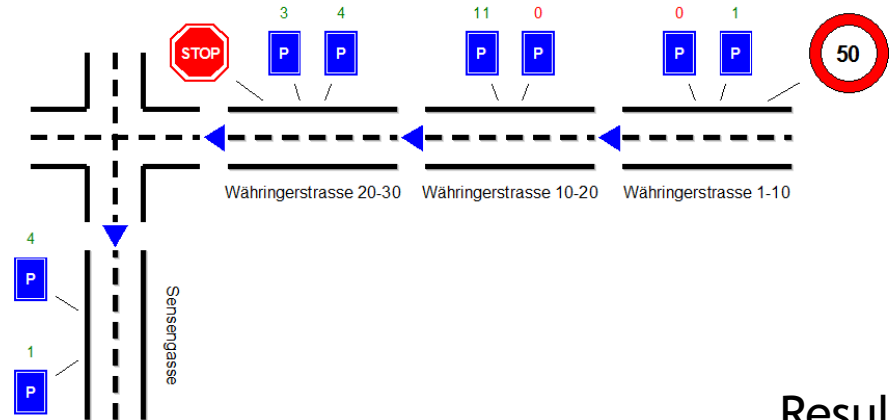
### User-defined Queries X

### Exercises



1. Find all Roads directly or indirectly preceding to a CrossRoad
2. Find all Roads directly preceding to a CrossRoad
3. Find all Roads directly subsequent to ,CrossRoad-12628‘
4. Find all objects directly or indirectly preceding to ,Sensengasse‘
5. Find all CrossRoads directly or indirectly preceding to ,Sensengasse‘
6. Find all objects directly preceding to the CrossRoads directly or indirectly preceding to ,Sensengasse‘

## 2. Model Queries in ADOxx User-defined Queries X



### Results

Find all Roads directly or indirectly preceding to a CrossRoad

`(<"CrossRoad"><<-"Subsequent">"Road">)`

Währingerstrasse 20-30  
Währingerstrasse 10-20  
Währingerstrasse 1-10

Find all Roads directly preceding to a CrossRoad

`(<"CrossRoad"><-"Subsequent">"Road">)`

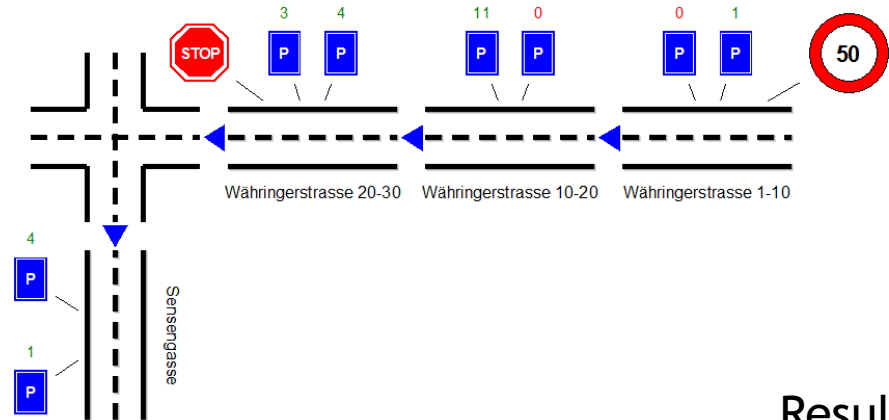
Währingerstrasse 20-30

Find all Roads directly subsequent to ,CrossRoad-12628'

`{"CrossRoad-12628"}->>"Subsequent"`

Sensengasse

## 2. Model Queries in ADOxx User-defined Queries X



### Results

Find all objects directly or indirectly preceding to ,Sensengasse'  
`{ "Sensengasse" } <<- "Subsequent"`

Währingerstrasse 20-30  
 Währingerstrasse 10-20  
 Währingerstrasse 1-10  
 CrossRoad-12628

Find all CrossRoads directly or indirectly preceding to ,Sensengasse'  
`( { "Sensengasse" } <<- "Subsequent" > "CrossRoad" < )`

CrossRoad-12628

Find all objects directly preceding to the CrossRoads directly or indirectly preceding to ,Sensengasse'

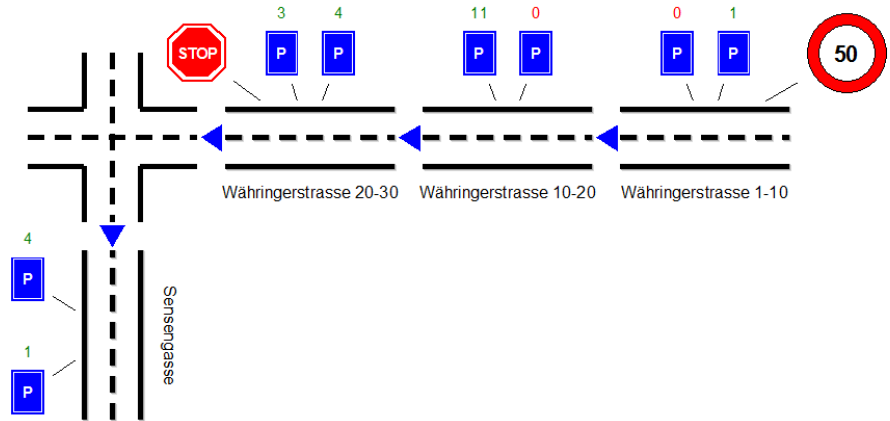
Währingerstrasse 20-30

`( { "Sensengasse" } <<- "Subsequent" > "CrossRoad" < ) <- "Subsequent"`

## 2. Model Queries in ADOxx

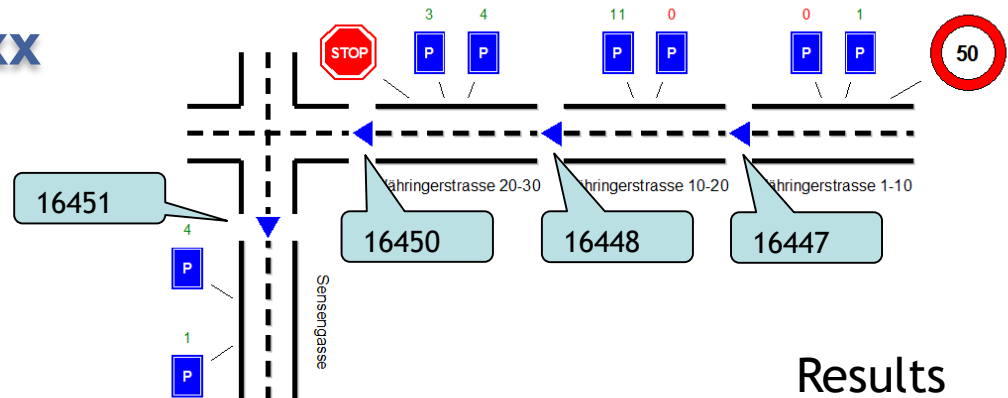
### User-defined Queries X

### Exercises



1. Find all relations of type subsequent
2. Find all relations of type subsequent incoming to a road directly preceding a CrossRoad
3. Find all relations of type subsequent outgoing from a road directly preceding a CrossRoad

## 2. Model Queries in ADOxx User-defined Queries X



Find all relations of type subsequent  
( "<\ "Subsequent\ ">" )

CrossRoad-12628 -> Sensengasse  
Währingerstrasse 1-10 -> Währingerstrasse 10-20  
Währingerstrasse 10-20 -> Währingerstrasse 20-30  
Währingerstrasse 20-30 -> CrossRoad-12628

Find all relations of type subsequent incoming to a  
road directly preceding a CrossRoad

Währingerstrasse 10,20 -> Währingerstrasse 20-30

( <"CrossRoad"> <- "Subsequent"> "Road" < ) <- <"Subsequent">












Find all relations of type subsequent outgoing from  
a road directly preceding a CrossRoad

Währingerstrasse 20-30 -> CrossRoad-12627

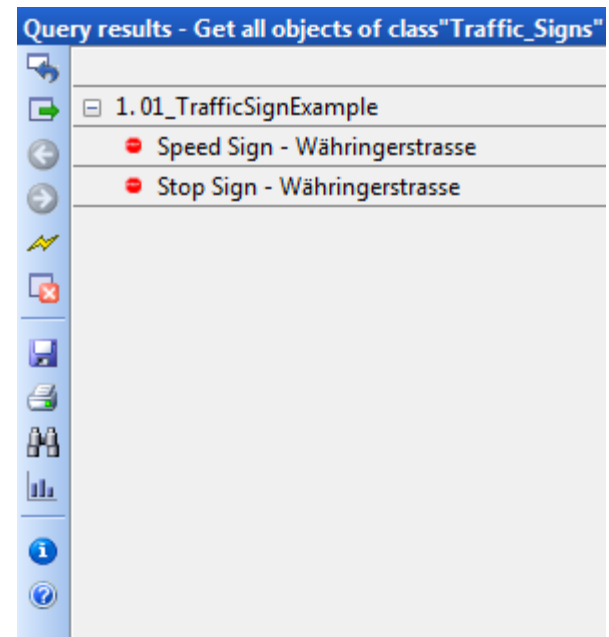
( <"CrossRoad"> <- "Subsequent"> "Road" < ) -> <"Subsequent">

## 2. ADOxx Query Results Window

### Query Results Options

- *Back to Query Definition* 
- *Continuous Query* 
- *Previous/Next Query Result* 
- *Close Query Result* 
- *Refresh* 
- *Save/Print/Search*   
- *Diagram* 
- *Query Info* 
- *Help* 

### Query Results





# AQL in ADOscript

- AQL can also be used in ADOscript

```
CC "Modeling" GET_ACT_MODEL  
#--> RESULT modelid: intValue .
```

Escaping necessary!

```
SET aqlGetPredecessorsForObj: ("(<\\"Road\\">)" )  
CC "AQL" EVAL_AQL_EXPRESSION expr: (aqlGetPredecessorsForObj)  
modelid: (modelid)  
# --> RESULT ecode: intValue objids: strValue  
  
CC "AdoScript" INFOBOX (objids)
```

## 2. Model Queries in ADOxx

### User-defined Queries X

Use the following code for the next examples!

```
CC "Modeling" GET_ACT_MODEL  
#--> RESULT modelid: intValue .
```

```
SET aqlGetPredecessorsForObj: ("({\"Sensengasse\"}<-\"Subsequent\">\"CrossRoad\"<-\"Subsequent\"")
```

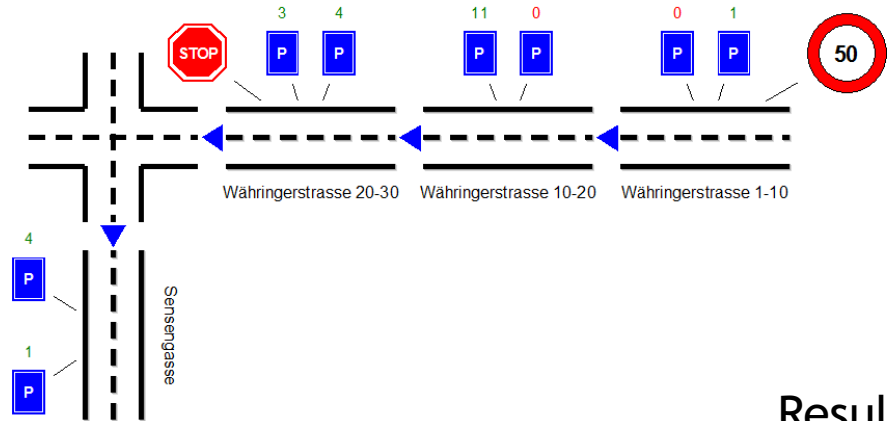
```
CC "AQL" EVAL_AQL_EXPRESSION expr: (aqlGetPredecessorsForObj) modelid: (modelid)
```

```
SET nameString:("")
```

```
FOR obj in:(objids) {  
    CC "Core" GET_OBJ_NAME objid:(VAL obj)  
    SET nameString:(tokcat(nameString,objname))  
}
```

```
CC "AdoScript" INFOBOX (nameString)
```

## 2. Model Queries in ADOxx User-defined Queries X



### Results

Währingerstrasse 20-30  
Währingerstrasse 10-20  
Währingerstrasse 1-10

```
(" (<\ "CrossRoad\"><<-\"Subsequent\">\ "Road\"<) ")
```

Währingerstrasse 20-30

```
(" (<\ "CrossRoad\"><-\"Subsequent\">\ "Road\"<) ")
```

Sensengasse

```
(" {\ "CrossRoad-12628\">->>\ "Subsequent\" ")
```

Währingerstrasse 20-30  
Währingerstrasse 10-20  
Währingerstrasse 1-10  
CrossRoad

```
(" {\ "Sensengasse\"><<-\"Subsequent\" ")
```

CrossRoad

```
(" ({ {\ "Sensengasse\"><<-\"Subsequent\">\ "CrossRoad\"<) ")
```

Währingerstrasse 20-30

```
(" ({ {\ "Sensengasse\"><<-\"Subsequent\">\ "CrossRoad\"<)  
<-\"Subsequent\" ")
```

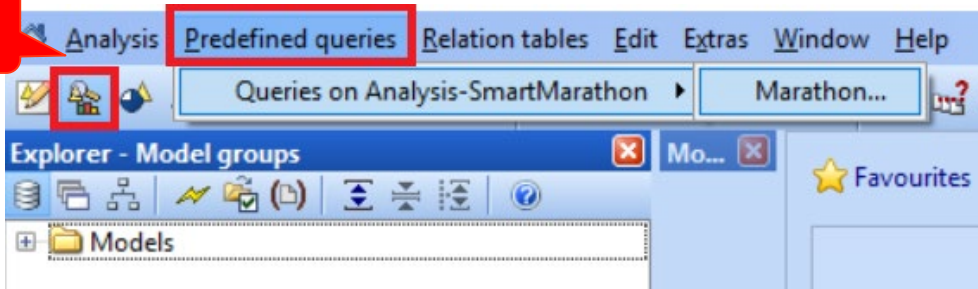
# 3. PREDEFINED QUERIES IN ADOXX



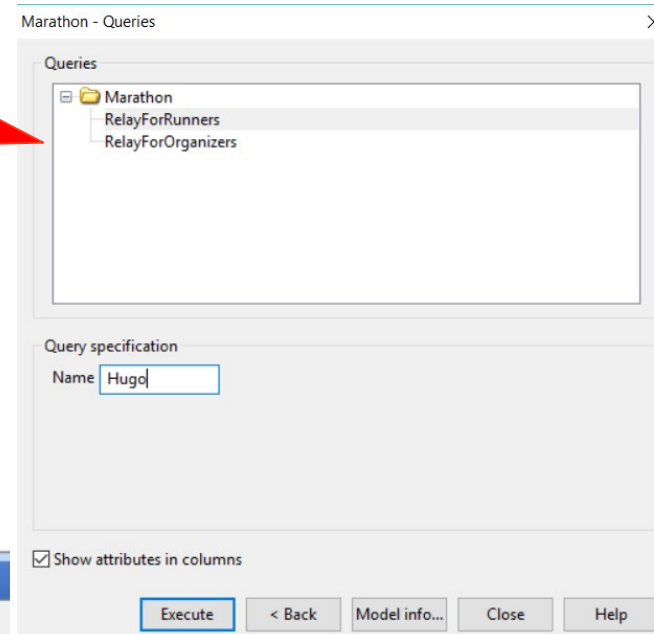
### 3. Predefined Queries I

Predefined queries can be user-specifically defined and be made available to the ADOxx users in the Analysis or Evaluation Component of the ADOxx Modelling Toolkit. They allow to create complex but easy to use queries.

1



2



3

The screenshot shows the 'Query results - RelayForRunners' table. The table has columns: Name, Number, Km, RelaySection, and AssignedRelaySection. The results are as follows:

	Name	Number	Km	RelaySection	AssignedRelaySection
1. Analysis-SmartMarathon - new					
Hugo	Hugo	0	0	not assigned	

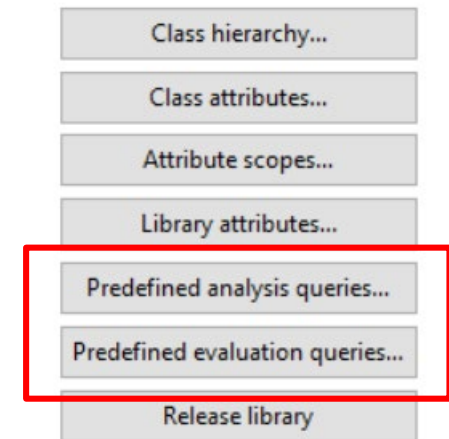




### 3. Predefined Queries II

The creation of a new query is defined through following steps:

- Create query
- Define input fields
- Define AQL-queries
- Define result attributes

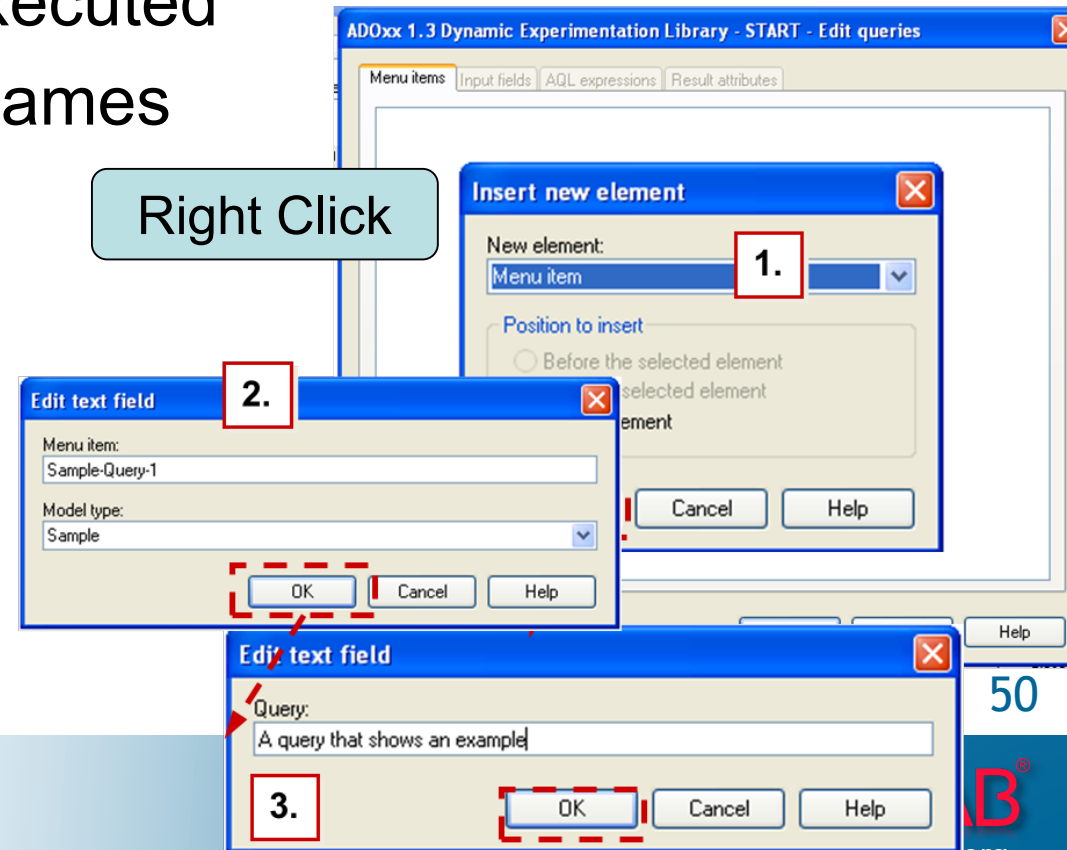


Queries can be created in the Development Toolkit via the Library Management by choosing a (dynamic or static) library and clicking ,Predefied analysis queries...‘ or ,Predefied evaluation queries...‘.

# 3. Predefined Analysis Query I

## Define Query Appearance

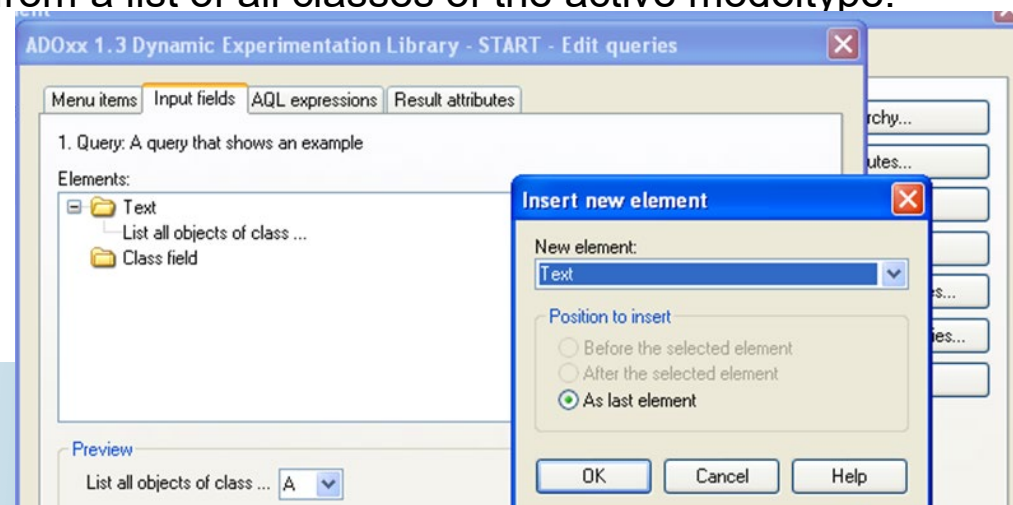
- Create a new menu entry
- Choose a model type for which the query should be executed
- Customize the menu names



# 3. Predefined Analysis Query II

## Define input fields

- The following types of fields are available:
  - **Text**
  - **Input fields:** For Attributes of type Text, time, date, date and time, integer, and double.
  - **Enumeration value field:** For attributes of Type enumeration.
  - **Attribute value field:** For the takeover of attribute values from attributes of different classes.
  - **Enumerated attribute field:** For the takeover of attribute values from enumerated attributes of different classes.
  - **Attribute field:** For choosing of attributes from a list of all attributes of all classes.
  - **Class field:** For choosing of a class from a list of all classes of the active modeltype.
- Add trough right click



# 3. Predefined Analysis Query III

## Define AQL-Queries

- Switch to the chapter "AQL expressions"
- Choose option "AQL Part" (right click)
- Copy manually designed AQL statement
- Add "References" to link input fields with query
- Follow proposal to place input fields into query statement

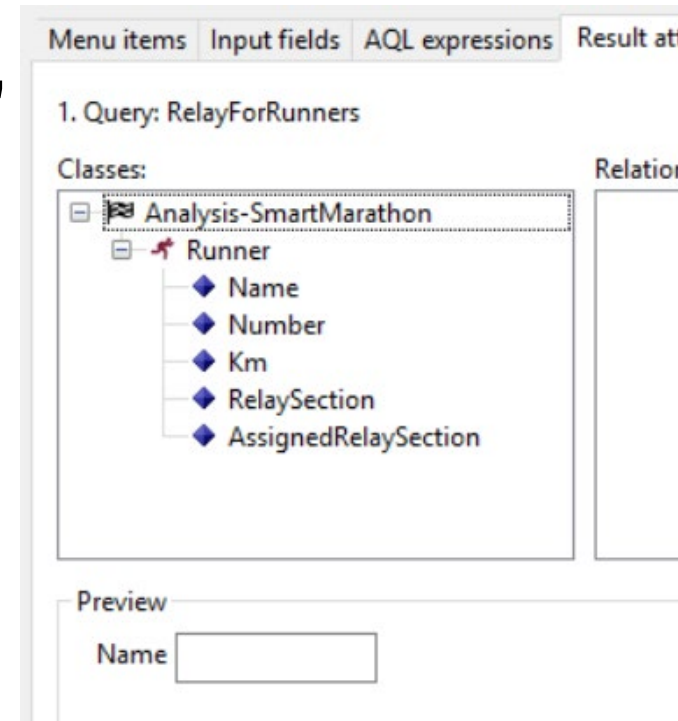
The screenshot displays a software interface for defining AQL queries. It features four tabs at the top: "Menu items", "Input fields", "AQL expressions", and "Result attributes". The "AQL expressions" tab is currently active. Below the tabs, the interface is divided into two main sections. The left section, titled "1. Query: RelayForRunners", contains a tree view labeled "AQL elements:". This tree view shows a hierarchical structure with three folders: "AQL part", "Reference", and another "AQL part". The first "AQL part" folder contains the text "<"Runner">[?"Name" like "\*\*", the "Reference" folder contains the number "2", and the second "AQL part" folder contains the text "\*)""]". The right section, titled "AQL expression:", contains a text area with the following XML-like query: `{<"Runner">[?"Name" like "**@R@@2@ *"]}`. At the bottom of the interface, there is a "Preview" section with a label "Name" followed by an empty rectangular input field.

# 3. Predefined Analysis Query IV

## Result Attributes

- In the chapter "Result attribute" it is specified which objects and attributes should be displayed in the result representation.
- Switch to chapter "Result attributes"
- Choose Option "Attribute"
- Determine Position
- Confirm

Hint: You can find an example in the ADOxx help. Search for 'Example of definition of a pre-defined query'.





# **4. EXERCISES**

## **SMART CITY MODEL QUERIES**

## 4. Exercises: Smart City Model Queries Overview

- I. The query dialog and the query result window
- II. Smart Parking with Standard Queries
- III. Construction of Combined Queries

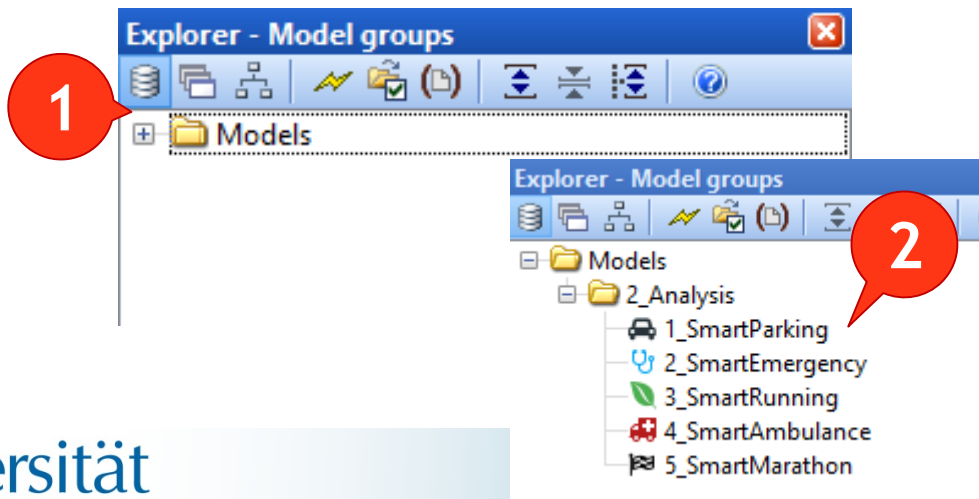
## 4. Exercises: Smart City Model Queries Overview

- I. The query dialog and the query result window
- II. Smart Parking with Standard Queries
- III. Construction of Combined Queries

## 4. Exercises: Smart City Model Queries

### Execute a very first Query

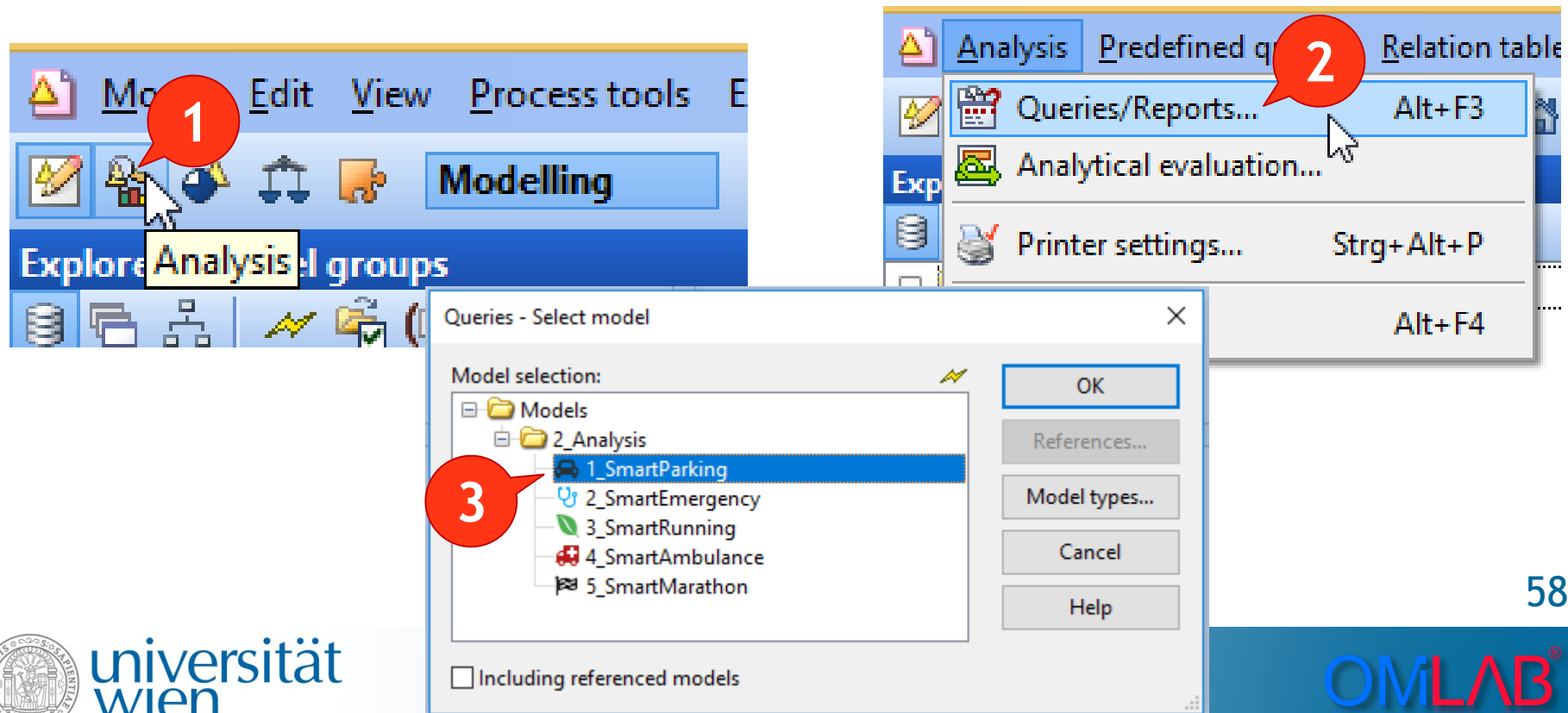
- The city of Vienna wants to change its smart traffic signs in the inner city for the European Song Contest. The first step is to identify all relevant signs via queries
  - Open the Modeling Toolkit with the [SmartCityLibraryII](#)
  - Open the [1\\_SmartParking](#) model (double click)



## 4. Exercises: Smart City Model Queries

### Execute a very first Query II

1. Select the *Analysis* component (in the toolbar)
2. Select Queries/Reports...
3. Select the **1\_SmartParking** model and click ,OK‘





## 4. Exercises: Smart City Model Queries

### Execute a very first Query III

Queries

Query scope

☐ Queries on models (model attributes)

☒ Queries on model contents

Standardised queries

Query:

Get all objects of class...

Input field

Get all objects of class Traffic\_Signs

Add Evaluate

User defined queries

AND OR DIFF Clear

☒ Show attributes in columns

< Back Execute Attributes... Model info... Cancel Help

Get all objects of class...

Get all objects of class ... with attribute ...

Get object ... of class ...

Get all objects connected with the object ... of class ... with the relation ...

Get all connectors of relation ...












Get all connectors of relation ... with attribute ...

4. Select a standardized query “Get all objects of class...”
5. Complete the query by filling in missing elements and input fields, i.e., “Traffic\_Signs”
6. Run the query by clicking on Evaluate

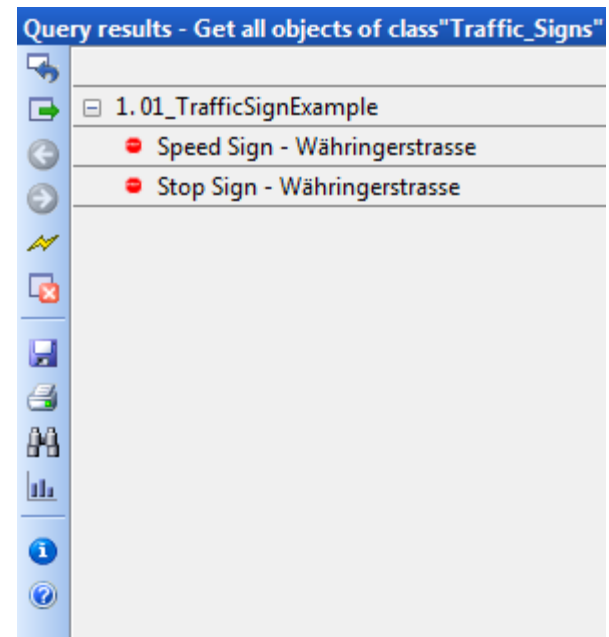
## 4 . Exercises: Smart City Model Queries

### ADOxx Query Results Window I

#### Query Results Options

- *Back to Query Definition* 
- *Continuous Query* 
- *Previous/Next Query Result* 
- *Close Query Result* 
- *Refresh* 
- *Save/Print/Search*   
- *Diagram* 
- *Query Info* 
- *Help* 

#### Query Results



## 4 . Exercises: Smart City Model Queries

### ADOxx Query Results Window II

- By clicking on the a query result, ADOxx highlights the corresponding model/object
- You can configure the *Query results* window by right-clicking in the grid (*Select Attributes*)
  - Hint: Press *Ctrl / Strg* to select multiple attributes
- You can edit model attributes directly in the *Query results* window

Query results - Get all objects of class "Traffic\_Signs"

	Name	Type	Max Speed	Number of Workers	Construction Site Cost	Traffic Control Cost	Hours/Day	Total Time	Bus_Numbers	Bus_Stop
1. 01_TrafficSignExample										
Speed Sign - Währingerstrasse	Speed Sign - Währingerstrasse	Speed_Limit	50	0	0,000000	0,000000	00:000:00:00:00	00:000:00:00:00		
Stop Sign - Währingerstrasse	Stop Sign - Währingerstrasse	STOP	50	0	0,000000	0,000000	00:000:00:00:00	00:000:00:00:00		

## 4. Exercises: Smart City Model Queries Overview

- I. The query dialog and the query result window
- II. Smart Parking with Standard Queries
- III. Construction of Combined Queries

## 4. Exercises: Smart City Model Queries

### Smart Parking with Standard Queries

*“Parking your car in big cities is becoming more and more difficult. Many drivers spend on average 20 minutes while looking for a parking spot. This increases CO2 emissions and most important of all; people waste their time.” [1]*

- Envision an app that searches for open parking spots, reserves a parking spot, and communicates with the smart parking meter to automatically process payment
- A Smart City model can provide real time data for a smart parking app



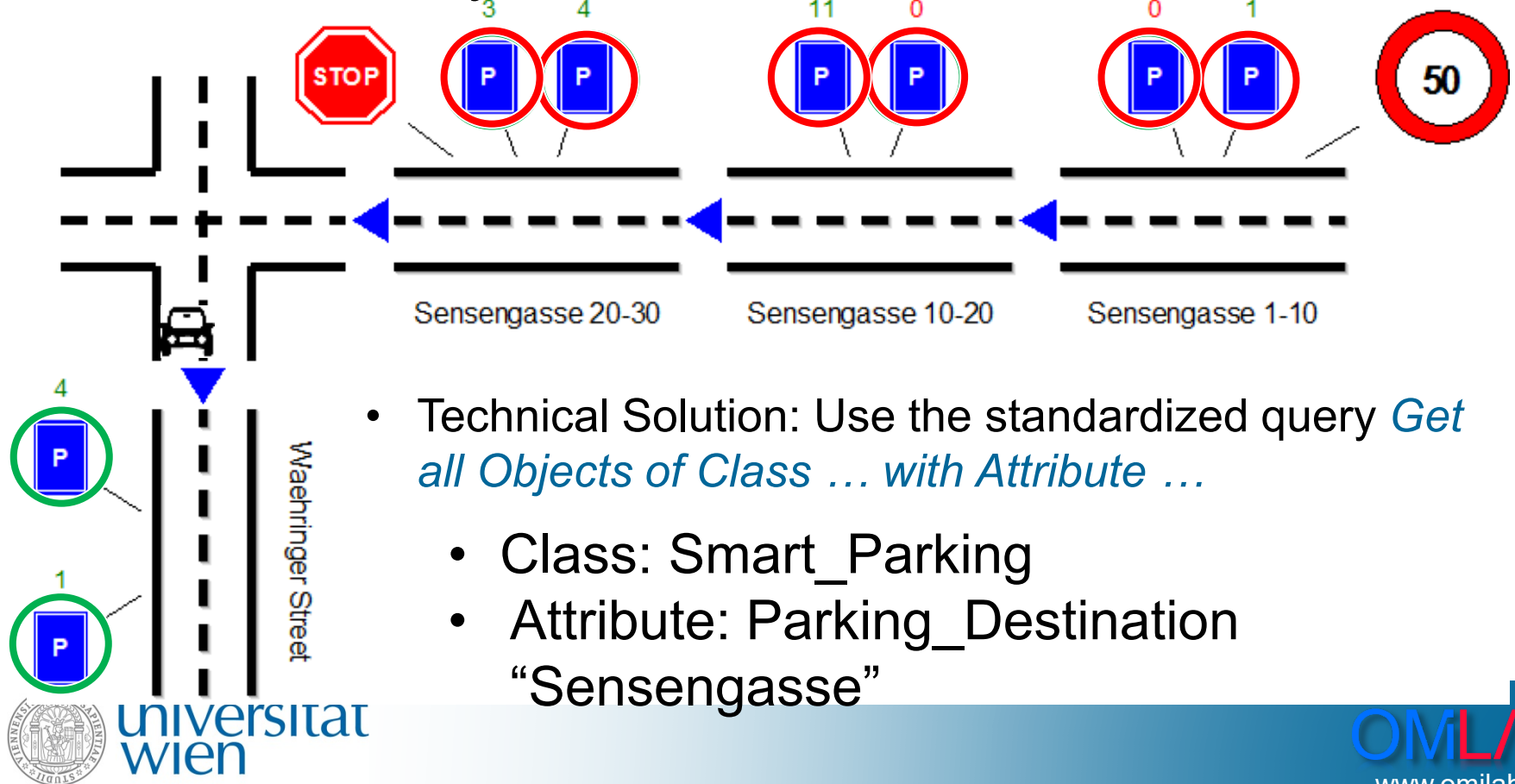
www.theparkerapp.com

[1] Amsterdam Smart City : <http://amsterdamsmartcity.com/projects/detail/id/64/slug/smart-parking?lang=en>

## 4. Exercises: Smart City Model Queries

### Smart Parking with Standard Queries II

- Envision that you drive home from work and want to search and reserve a smart parking spot at your home address near your front entrance



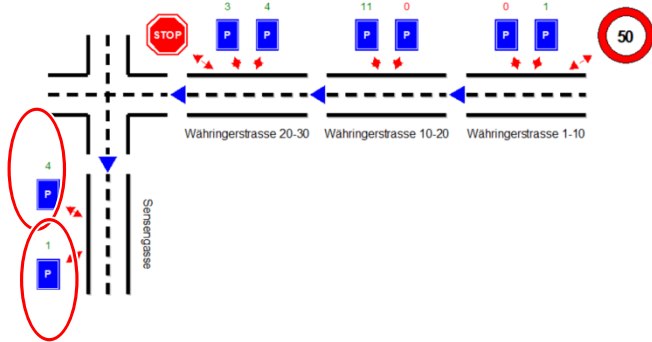
- Technical Solution: Use the standardized query *Get all Objects of Class ... with Attribute ...*
  - Class: Smart\_Parking
  - Attribute: Parking\_Destination "Sensengasse"



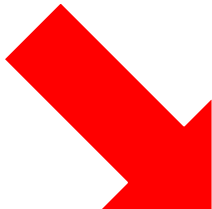
## 4. Exercises: Smart City Model Queries

### Smart Parking with Standard Queries III

- You need a parking spot at '*Sensengasse*'. Find all parking spots in *Sensengasse* and display their attributes in the Query results window
  - Use the *like* operator to compare strings and '?' for 1, or '\*' for multiple wildcards



- Reserve a parking spot in the query results window



	Parking_Status	Parking_Destination
1. 1_SmartParking2		
Smart Parking Spot - Sensengasse 14	Open	Sensengasse
Smart Parking Spot - Sensengasse 2	Open	Sensengasse

# 4. Exercises: Smart City Model Queries

## Smart Parking with Standard Queries IV

1. Click the **Analysis** menu.

2. Click **Queries/Reports...** (Alt+F3).

3. Select **1\_SmartParking** in the **Model selection** dialog.

4. Click **OK** in the **Model selection** dialog.

5. Select **Queries on model contents** in the **Query scope** section.

6. Select **Smart\_Parking** in the **Input field** section.

7. Click **Add**.

8. Right click on the query results.

9. Click **Copy to clipboard**.

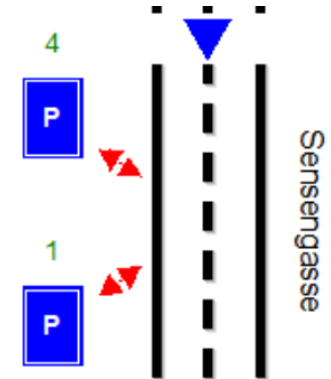
10. Select **Parking\_Status** in the **Queries - Attribute** dialog.

11. Click **OK** in the **Queries - Attribute** dialog.

# 4. Exercises: Smart City Model Queries

## Smart Parking with Standard Queries V

	Parking_Status	Parking_Destination	Parking_Capacity
1. 1_SmartParking2			
Smart Parking Spot - Sensengasse 14	Open	Sensengasse	4
Smart Parking Spot - Sensengasse 2	Open	Sensengasse	1



Smart Parking Spot - Währingerstrasse 1 - Pa...

Values:

- Occupied
- Open
- Reserved**
- Unknown

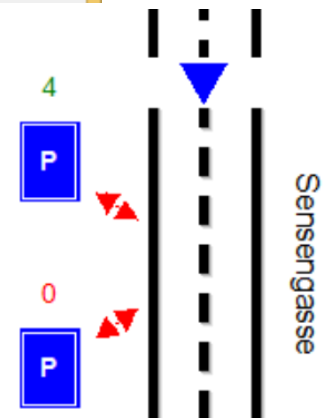
Apply

Cancel

Help

Query Results - (< Smart\_Parking >[ Parking\_Destination = Sensengasse ])

	Parking_Status	Parking_Destination	Parking_Capacity
1. 1_SmartParking2			
Smart Parking Spot - Sensengasse 14	Open	Sensengasse	4
Smart Parking Spot - Sensengasse 2	Reserved	Sensengasse	0



## 4 . Exercises: Smart City Model Queries Overview

- I. The query dialog and the query result window
- II. Smart Parking with Standard Queries
- III. Construction of Combined Queries

## 4. Exercises: Smart City Model Queries

### Construction of Combined Queries I

Queries

Query scope

☐ Queries on models (model attributes)

☒ Queries on model contents

Standardised queries

Query:

Get all objects of class ... with attribute ...

Input field

Get all objects of class **Smart\_Parking**

with attribute **Parking\_Status** **=** **Open**

Add Evaluate

User defined queries

`(((<"Smart_Parking">[?"Parking_Destination" like "Sensengasse*"]) AND (<"Smart_Parking">[?"Parking_Status" = "Open"]))) DIFF (<"Smart_Parking">[?"Parking_Capacity" <= 0])`

AND OR DIFF Clear

☒ Show attributes in columns

< Back Execute

	Parking_Status	Parking_Capacity	Parking_Destination	Name
1. 01_TrafficSignExample				
Smart Parking Spot - Sensengasse 1	Open	1	Sensengasse 2	Smart Parking Spot - Sensengasse 1
Smart Parking Spot - Sensengasse 5	Open	11	Sensengasse 16	Smart Parking Spot - Sensengasse 5
Smart Parking Spot - Sensengasse 6	Open	4	Sensengasse 20	Smart Parking Spot - Sensengasse 6
Smart Parking Spot - Sensengasse 7	Open	3	Sensengasse 22	Smart Parking Spot - Sensengasse 7

Using the logical operators **AND**, **OR**, **DIFF**, simple standard queries can be combined to answer complex questions.

- Select all **Smart\_Parking** objects whose **parking destination** is at ,Sensengasse‘
- whose **parking status** is ,Open‘
- without those objects whose **parking capacity** is  $\leq 0$ .

## 4. Exercises: Smart City Model Queries

### Construction of Combined Queries II

1. Select all Smart\_Parking objects whose parking destination is at ,Sensengasse‘

(<"Smart\_Parking">[?"Parking\_Destination" like "Sensengasse\*"])

2. whose parking status is ,Open‘

(<"Smart\_Parking">[?"Parking\_Status" = "Open"])

3. without those objects whose parking capacity is  $\leq 0$ .

(<"Smart\_Parking">[?"Parking\_Capacity" > 0])

Combined Query:

(<"Smart\_Parking">[?"Parking\_Destination" like "Sensengasse\*"])  
AND (<"Smart\_Parking">[?"Parking\_Status" = "Open"]) AND  
(<"Smart\_Parking">[?"Parking\_Capacity" > 0])



# 5. LESSONS LEARNED



## 5. Lessons Learned

- Domain-specific modeling methods come with specific requirements toward model queries
- Model queries can realize an added value by answering non-trivial questions based on the knowledge captured in models
- ADOxx provides a rich set of built-in functionality that can be utilized to realize model queries by configuration
- How to utilize ADOxx standard queries
- How to combine standard and pre-defined queries for more complex model queries
- How to create user-defined model queries
- How to apply model queries in a Smart City scenario