DATA IN MOTION

Modeling for Smart Cities

What is Modeling?

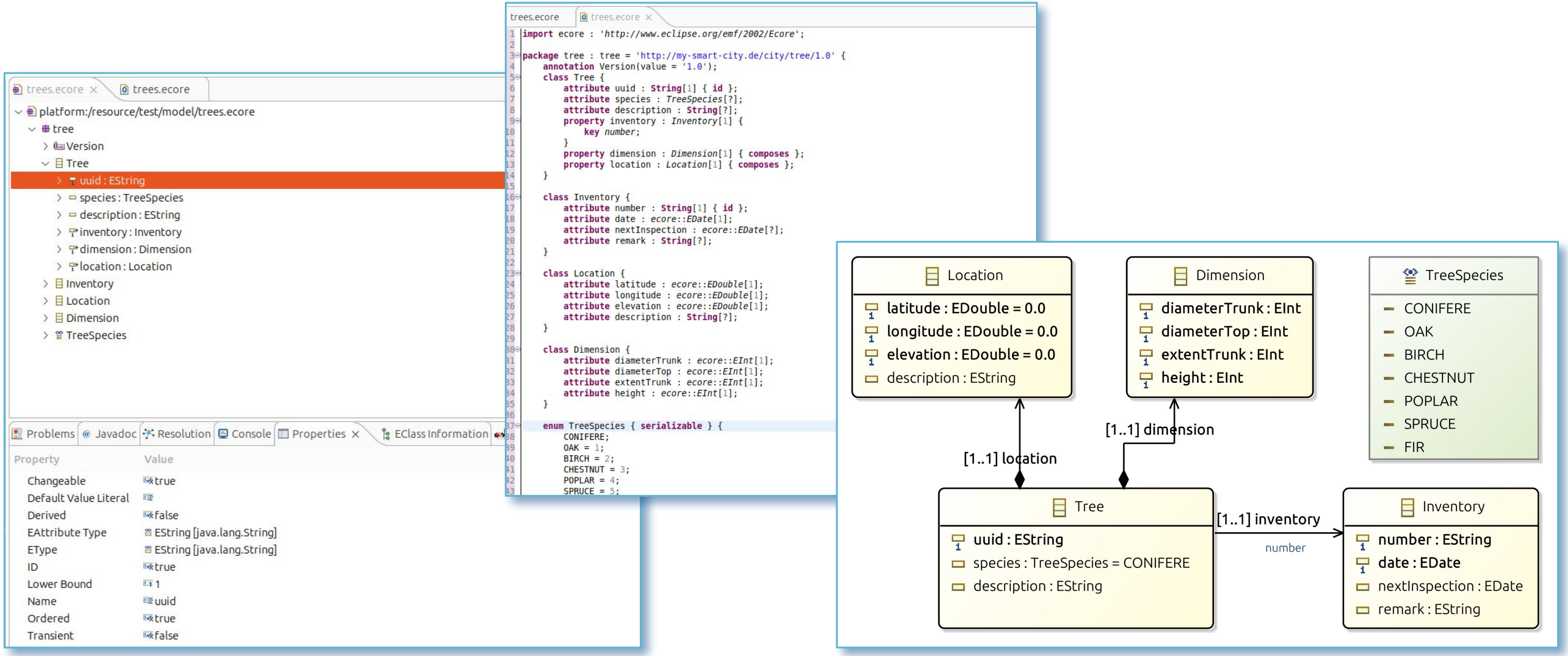
SMARTCITY
EXPO WORLD CONGRESS

OSGi®

What is modeling?

- Modeling is a methodology to create a common language
- It is a structured documentation how things are described
- Formalizing tacit knowledge in organizations
- Increase of interoperability between participating departments
- Non-technical activity, that can be technically used
- Modeling as a process helps to understand the own environment
- Different models can represent different perspectives on a context
- Much more than just a diagram …

How does a model look like?



- platform:/resource/test/model/trees.ecore
  - tree
    - Version
    - Tree
      - uuid : EString
      - species : TreeSpecies
      - description : EString
      - inventory : Inventory
      - dimension : Dimension
      - location : Location
    - Inventory
    - Location
    - Dimension
    - TreeSpecies

Property	Value
Changeable	true
Default Value Literal	
Derived	false
EAttribute Type	EString [java.lang.String]
EType	EString [java.lang.String]
ID	true
Lower Bound	1
Name	uuid
Ordered	true
Transient	false

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore';

package tree : tree = 'http://my-smart-city.de/city/tree/1.0' {
    annotation Version(value = '1.0');
    class Tree {
        attribute uuid : String[1] { id };
        attribute species : TreeSpecies[?];
        attribute description : String[?];
        property inventory : Inventory[1] {
            key number;
        }
        property dimension : Dimension[1] { composes };
        property location : Location[1] { composes };
    }

    class Inventory {
        attribute number : String[1] { id };
        attribute date : ecore::EDate[1];
        attribute nextInspection : ecore::EDate[?];
        attribute remark : String[?];
    }

    class Location {
        attribute latitude : ecore::EDouble[1];
        attribute longitude : ecore::EDouble[1];
        attribute elevation : ecore::EDouble[1];
        attribute description : String[?];
    }

    class Dimension {
        attribute diameterTrunk : ecore::EInt[1];
        attribute diameterTop : ecore::EInt[1];
        attribute extentTrunk : ecore::EInt[1];
        attribute height : ecore::EInt[1];
    }

    enum TreeSpecies { serializable } {
        CONIFERE;
        OAK = 1;
        BIRCH = 2;
        CHESTNUT = 3;
        POPLAR = 4;
        SPRUCE = 5;
```

Location
- latitude : EDouble = 0.0
- longitude : EDouble = 0.0
- elevation : EDouble = 0.0
- description : EString

Dimension
- diameterTrunk : EInt
- diameterTop : EInt
- extentTrunk : EInt
- height : EInt

TreeSpecies
- CONIFERE
- OAK
- BIRCH
- CHESTNUT
- POPLAR
- SPRUCE
- FIR

Tree
- uuid : EString
- species : TreeSpecies = CONIFERE
- description : EString

Inventory
- number : EString
- date : EDate
- nextInspection : EDate
- remark : EString

How do I do that?

- Load existing models
- Import Models out of existing schemas, like XSD’s
- With modeling tools (MagicDraw, Rational, Visual Paradigm, Eclipse)
- With own editors
- Customize existing editors depending on your needs
- Generate Model programmatically

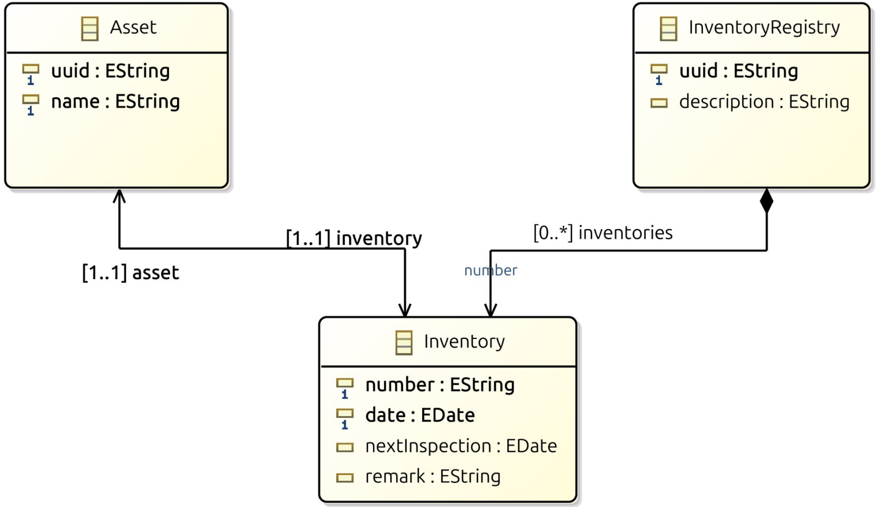
How is a model described?

- There are standards for defining models
- UML is the most known specification
- UML models are stored in a common format
- This is a XML-based format called XMI
- BPMN can be used for processes
- RDF is also a model for semantic web
- UML and RDF are both self-describing and therefore comapatible

So modeling is about …

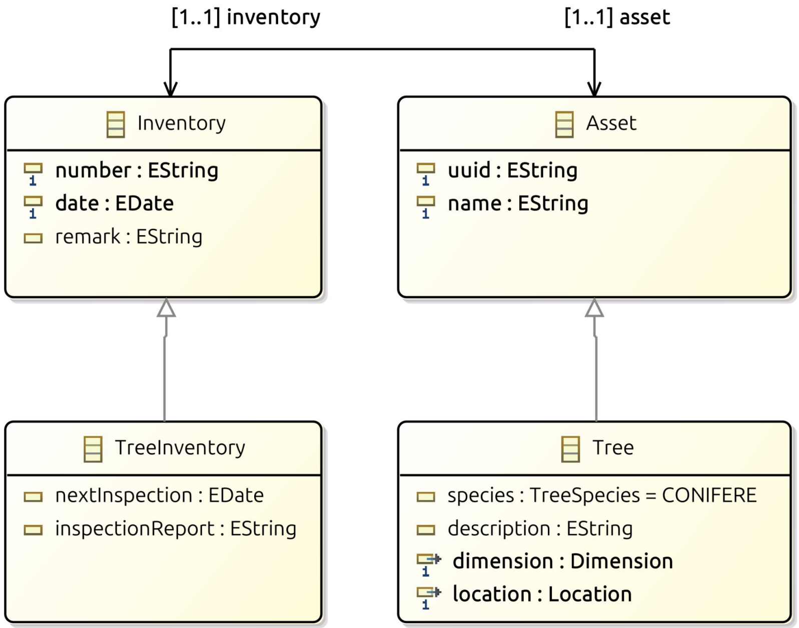
- Defining entities and their relations to other entities
- Inherit entities to make them more special
- Generalize and define a common base structure
- Transforming instances of one model into another
- Creating documentation / diagrams or code out of models or model-instance
- Load / Save model-instances in data formats like JSON, XML, binary, …

Defining Entities and Relations



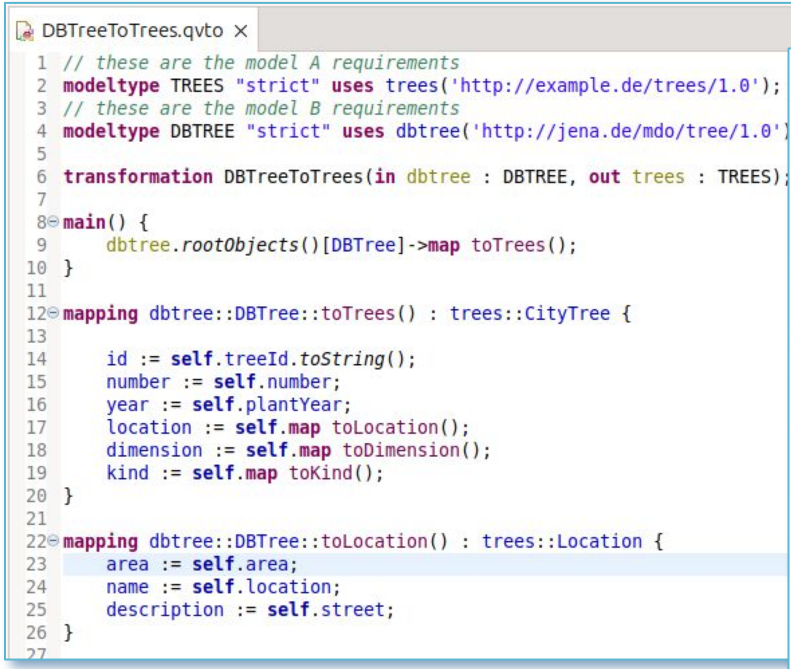
- An **Asset** is *linked* to an **Inventory**
- An **Inventory** is also *linked* to its **Asset** (bi-directionality)
- These links are mandatory (*1..1*)
- The **Inventory Registry** owns *many* **Inventories** (*0..n*)
- The **Inventory Registry** identifies the **Inventories** by the **number** attribute

Inherit from the General

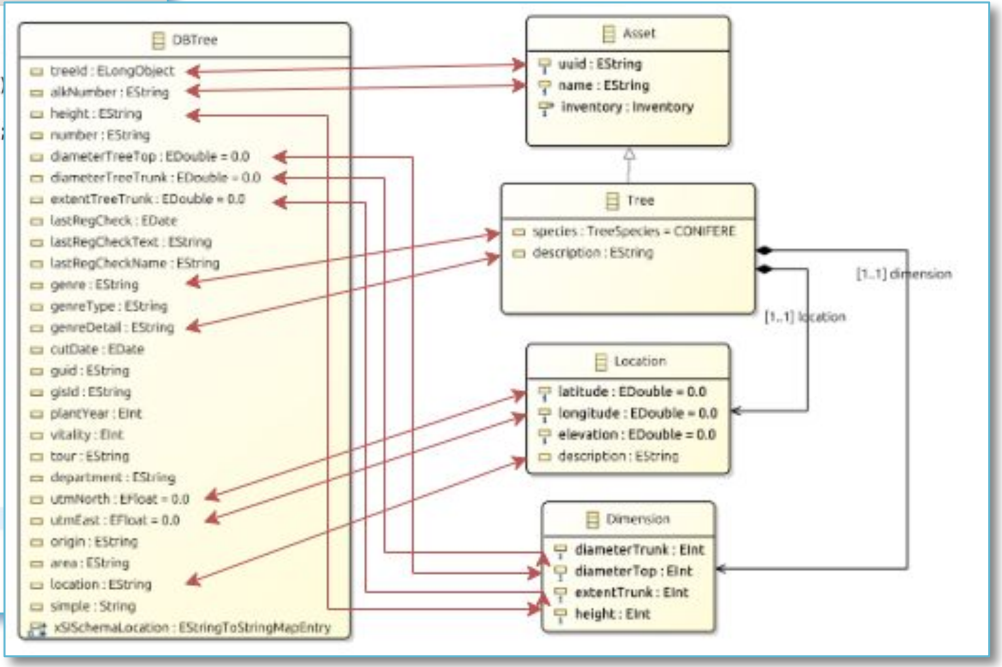


- A **Tree** is an **Asset**
- It owns *all* attributes from the **Asset**
- But the **Tree** also has *own* attributes.
- It is more *special* than the **Asset**
- The **Asset** is more *general*, than the **Tree**
- **Tree Inventory** *derives* from **Inventory**, like **Tree** from **Asset**
- This relationship is called **Inheritance**

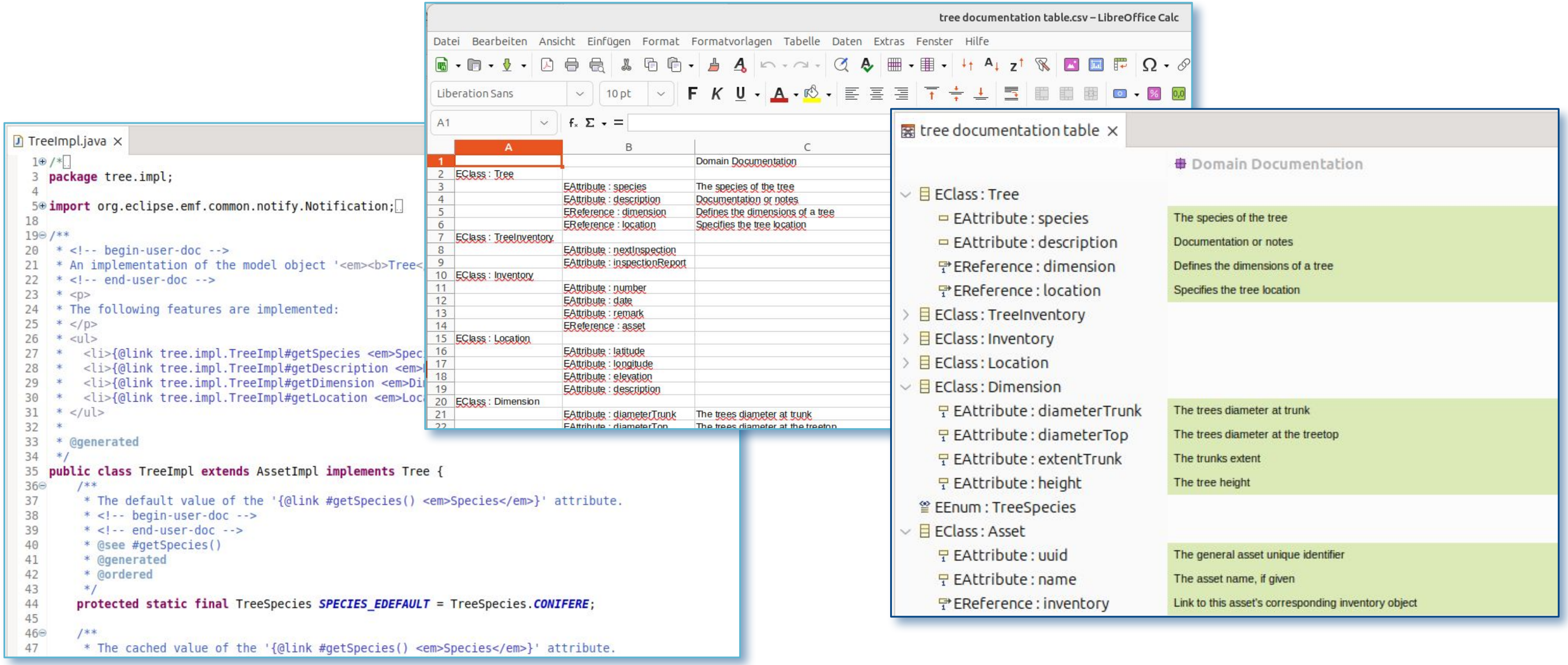
Mapping between Models



```
// these are the model A requirements
modeltype TREES "strict" uses trees('http://example.de/trees/1.0');
// these are the model B requirements
modeltype DBTREE "strict" uses dbtree('http://jena.de/mdo/tree/1.0')

transformation DBTreeToTrees(in dbtree : DBTREE, out trees : TREES);

main() {
    dbtree.rootObjects()[DBTree]->map toTrees();
}

mapping dbtree::DBTree::toTrees() : trees::CityTree {

    id := self.treeId.toString();
    number := self.number;
    year := self.plantYear;
    location := self.map toLocation();
    dimension := self.map toDimension();
    kind := self.map toKind();
}

mapping dbtree::DBTree::toLocation() : trees::Location {
    area := self.area;
    name := self.location;
    description := self.street;
}
```



Generating Text / Code

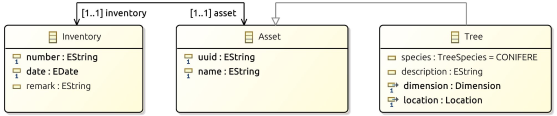

TreeImpl.java
tree documentation table.csv – LibreOffice Calc
Datei Bearbeiten Ansicht Einfügen Format Formatvorlagen Tabelle Daten Extras Fenster Hilfe
Liberation Sans
10 pt
A1
tree documentation table

```java
/*
package tree.impl;

import org.eclipse.emf.common.notify.Notification;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model object '<em><b>Tree<
 * <!-- end-user-doc -->
 * <p>
 * The following features are implemented:
 * </p>
 * <ul>
 *   <li>{@link tree.impl.TreeImpl#getSpecies <em>Spec
 *   <li>{@link tree.impl.TreeImpl#getDescription <em>
 *   <li>{@link tree.impl.TreeImpl#getDimension <em>Di
 *   <li>{@link tree.impl.TreeImpl#getLocation <em>Loc
 * </ul>
 *
 * @generated
 */
public class TreeImpl extends AssetImpl implements Tree {
	/**
	 * The default value of the '{@link #getSpecies() <em>Species</em>}' attribute.
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @see #getSpecies()
	 * @generated
	 * @ordered
	 */
	protected static final TreeSpecies SPECIES_EDEFAULT = TreeSpecies.CONIFERE;

	/**
	 * The cached value of the '{@link #getSpecies() <em>Species</em>}' attribute.
```

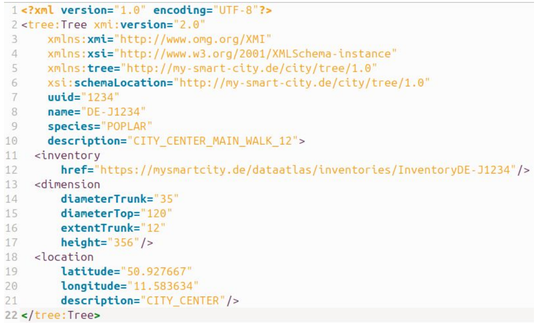
	A	B	C
1			Domain Documentation
2	EClass : Tree		
3		EAttribute : species	The species of the tree
4		EAttribute : description	Documentation or notes
5		EReference : dimension	Defines the dimensions of a tree
6		EReference : location	Specifies the tree location
7	EClass : TreeInventory		
8		EAttribute : nextInspection	
9		EAttribute : inspectionReport	
10	EClass : Inventory		
11		EAttribute : number	
12		EAttribute : date	
13		EAttribute : remark	
14		EReference : asset	
15	EClass : Location		
16		EAttribute : latitude	
17		EAttribute : longitude	
18		EAttribute : elevation	
19		EAttribute : description	
20	EClass : Dimension		
21		EAttribute : diameterTrunk	The trees diameter at trunk
22		EAttribute : diameterTop	The trees diameter at the treetop

	Domain Documentation
EClass : Tree	
EAttribute : species	The species of the tree
EAttribute : description	Documentation or notes
EReference : dimension	Defines the dimensions of a tree
EReference : location	Specifies the tree location
EClass : TreeInventory	
EClass : Inventory	
EClass : Location	
EClass : Dimension	
EAttribute : diameterTrunk	The trees diameter at trunk
EAttribute : diameterTop	The trees diameter at the treetop
EAttribute : extentTrunk	The trunks extent
EAttribute : height	The tree height
EEnum : TreeSpecies	
EClass : Asset	
EAttribute : uuid	The general asset unique identifier
EAttribute : name	The asset name, if given
EReference : inventory	Link to this asset's corresponding inventory object

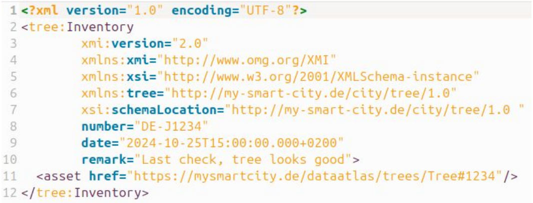
De- / Serialize Models



XML

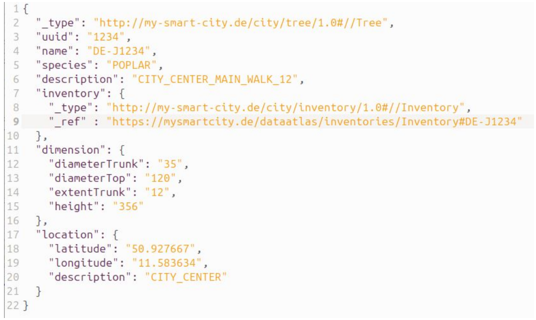


```xml
<?xml version="1.0" encoding="UTF-8"?>
<tree:Tree xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tree="http://my-smart-city.de/city/tree/1.0"
    xsi:schemaLocation="http://my-smart-city.de/city/tree/1.0"
    uuid="1234"
    name="DE-J1234"
    species="POPLAR"
    description="CITY_CENTER_MAIN_WALK_12">
  <inventory
      href="https://mysmartcity.de/dataatlas/inventories/InventoryDE-J1234"/>
  <dimension
      diameterTrunk="35"
      diameterTop="120"
      extentTrunk="12"
      height="356"/>
  <location
      latitude="50.927667"
      longitude="11.583634"
      description="CITY_CENTER"/>
</tree:Tree>
```

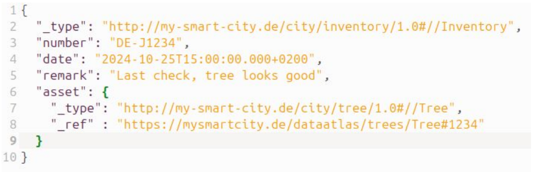


```xml
<?xml version="1.0" encoding="UTF-8"?>
<tree:Inventory
        xmi:version="2.0"
        xmlns:xmi="http://www.omg.org/XMI"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:tree="http://my-smart-city.de/city/tree/1.0"
        xsi:schemaLocation="http://my-smart-city.de/city/tree/1.0 "
        number="DE-J1234"
        date="2024-10-25T15:00:00.000+0200"
        remark="Last check, tree looks good">
  <asset href="https://mysmartcity.de/dataatlas/trees/Tree#1234"/>
</tree:Inventory>
```

JSON

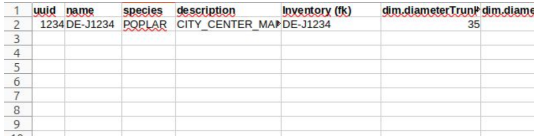


```json
{
  "_type": "http://my-smart-city.de/city/tree/1.0#//Tree",
  "uuid": "1234",
  "name": "DE-J1234",
  "species": "POPLAR",
  "description": "CITY_CENTER_MAIN_WALK_12",
  "inventory": {
    "_type": "http://my-smart-city.de/city/inventory/1.0#//Inventory",
    "_ref" : "https://mysmartcity.de/dataatlas/inventories/Inventory#DE-J1234"
  },
  "dimension": {
    "diameterTrunk": "35",
    "diameterTop": "120",
    "extentTrunk": "12",
    "height": "356"
  },
  "location": {
    "latitude": "50.927667",
    "longitude": "11.583634",
    "description": "CITY_CENTER"
  }
}
```

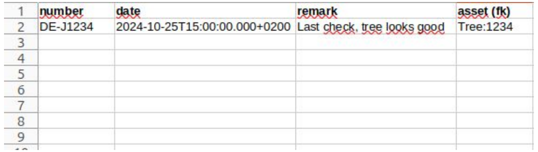


```json
{
  "_type": "http://my-smart-city.de/city/inventory/1.0#//Inventory",
  "number": "DE-J1234",
  "date": "2024-10-25T15:00:00.000+0200",
  "remark": "Last check, tree looks good",
  "asset": {
    "_type": "http://my-smart-city.de/city/tree/1.0#//Tree",
    "_ref" : "https://mysmartcity.de/dataatlas/trees/Tree#1234"
  }
}
```

Database



1	uuid	name	species	description	Inventory (fk)	dim.diameterTrunk	dim.diame
2	1234	DE-J1234	POPLAR	CITY_CENTER_MAI	DE-J1234	35	
3							
4							
5							
6							
7							
8							
9							



1	number	date	remark	asset (fk)
2	DE-J1234	2024-10-25T15:00:00.000+0200	Last check, tree looks good	Tree:1234
3				
4				
5				
6				
7				
8				
9				

De- / Serialize Models

- Serialization and Deserialization are decoupled from model
- Many frameworks have plug-able modules
- Allows end-to-end usage of models
- De-couple transport from serialization
- Implementations for XML, JSON, RDF, CSV, R-Data, MongoDB, JPA, …
- Model Exports XMI, Ecore, XML, XSD, Json-Schema
- Instance Exports for PlantUML, Mermaid, XLSX, ODS

Everything can be model

- Models can be linked to each other, even the meta-models
- Self-describing ability: You can describe UML using UML
- Models can be:
  - **Entity definitions** - What is a building?)
  - **Process definitions** - What do we do when watering trees? Which entities are linked?
  - **Mapping definitions** - We have to map two different entities and want to define how to do that
  - **Text/Code generation descriptions** - Describe the additional information for the generation process itself, like providing path or file-name information in another model
  - **UI / Dashboard descriptions** - Define UI and layout in a model and bind entity attributes to UI widgets.
  - **Configurations** - Provide configuration for software systems the also include processes or entities

Modeling for Smart Cities

Model based Data Platform

SMARTCITY
EXPO WORLD CONGRESS

OSGi®

Status Quo

- Many existing technical inventory systems
- Different products solve one process problem
- Different systems are not per-se inter-operable
- Lots of regulations influence processes
- Lots of different standards in different areas of activities
- Different departments “islands” with own perspectives to the same context
- Existing systems can not easily replaced

What does it really make smart?

- Understanding the existing assets and their values
- Increasing interoperability between existing systems
- Combining information from different data sources to new information
- Create an foundation, that can handle this challenge
- Realizing that there is no one-size-fits-all solution
- Digitalization is a community act, dealing with your own organization
- Reveal tacit knowledge and solutions in organizations
- Technology can only support and assist but is no end in itself
- Sustainable solutions

System Architecture Level

- **Modularity / Reuse** - Many components need to interact with each other using a well defined interfaces including expectations and requirements of communicating parties / modules.
- **Service Orientation** - Services are a common way of communication between participants in a component and programming language agnostic way.
- **Distributed Components** - System components are distributed over heterogeneous infrastructures. This sets preconditions for distributed computing for the development of components.
- **Dynamics** - Every component and service in a distributed environment can come, change and go at anytime. Changes in components that belong to others needs to be reflected into the infrastructure.
- **Resilience** - The service dependencies must be well defined. Service can be mandatory, optional and conditional. Service replacement during runtime must be possible. Service availability means a functionality exists and is working. No service means a functionality is not available.

Professional Level

- **Process Re- / Engineering** - Review the existing and use technology support for an economic way
- **Formalizing** - Formalize information structures, processes in a way that is readable by machines.
- **Interoperability** - Sometimes processes involve more than one departments. Interfaces between departments and organizations have to take into account.
- **Tacit Knowledge** - Reveal tacit knowledge and respect it. It often shows a lived process and more efficient, practical and accepted way for a certain process.
- **Data Protection** - Data protection is a preset in all organizations. When IT is involved there is a need to always take care about data protection, in particular when designing processes.
- **Open Data** - Beside data protection open data is important for governmental organizations. There are a lot of specifications and regulations for it. Open Data and Data Protection are no competing topics!
- **Regulations** - Regulations demand certain aspects (e.g. documentation) of processes or specify a way a process has to work. Formalization can help here.

Modeling and the right technological architecture can create a common foundation for Smart Data Platforms!

- Infrastructure first approach, with use-case requirements in mind
- Apply use-cases to the infrastructure, instead vice versa
- Modify infrastructure depending on the requirements in a more generic way
- Focus on re-use of components
- Using models addresses exact the same aspects, like those for the system architectures
- Modeling can bridge the gap between professional and technical level

City Model Examples

- Assets like buildings, intersections, trash can, driver license renewal processes
- Layering and linking different perspectives of an asset (e.g. building):
    - Electric plan of an building
    - Evacuation plan
    - Construction plan
    - Elevator maintenance plan
- Sensors, sensor values models - standardized and proprietary data formats
- ETL Processes - Mapping models for transformations
- Models for Open Data Schemas
- Analysis of data and their models for GDPR related information - Structured Reports
- Documentation creation / Auditing

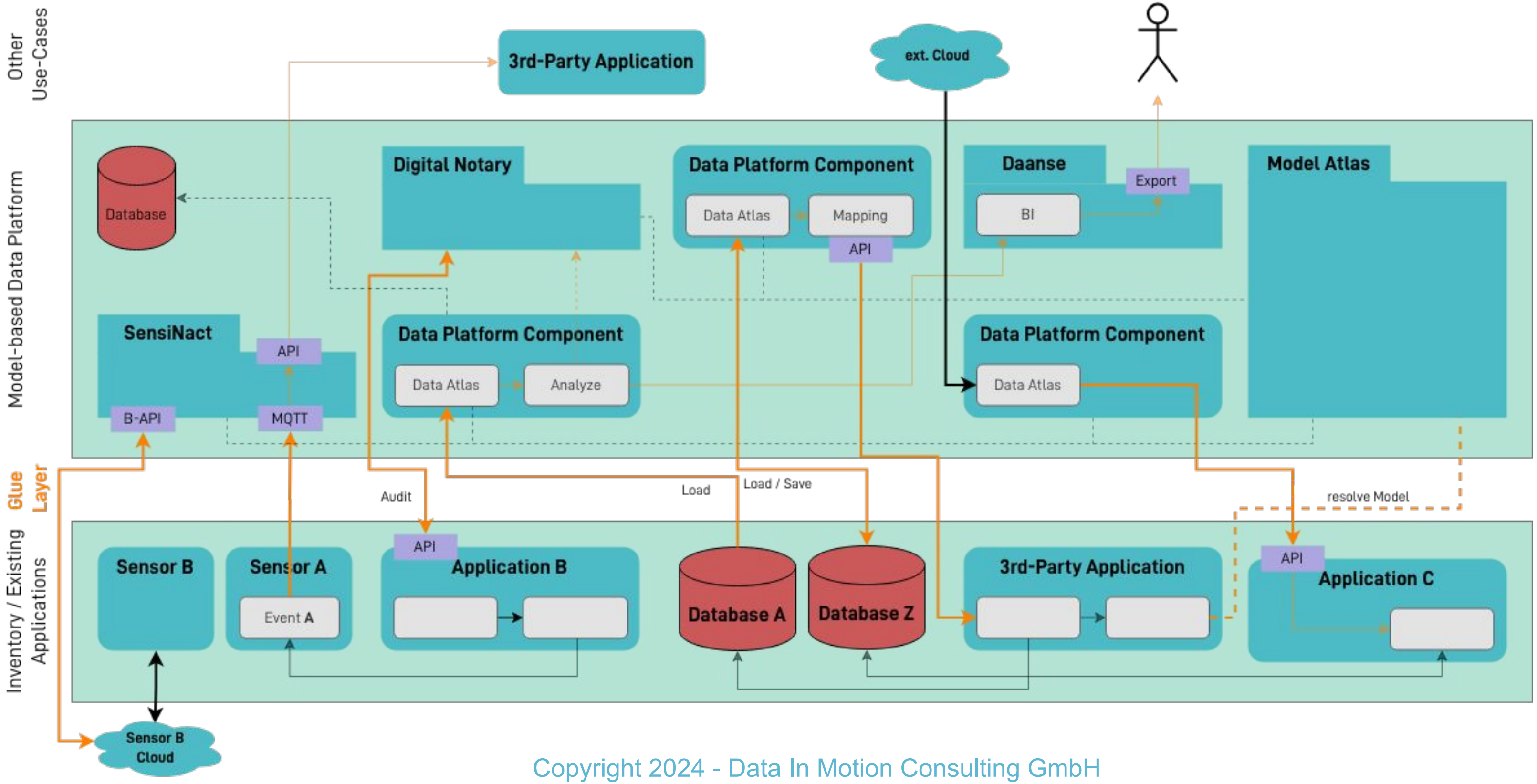
City Model Characteristics

- End-User models
- Many models may linked with each other
- Use of basic modelling features are sufficient
- Modeling process should not include deeper modeling knowledge
- So creating / modifying should happen with Low-Code tooling
- Less as possible, better non technician involved deploying models
- Place a “*Model Officer*” as review instance
- Necessity for release, audit workflows that involve automatically checks and human interaction (GDPR decontrol)

Smart Data Platform Requirements

- Handle data with low modification probability
- Handle data with high modification probability (“real-time data”)
- Request-Response and Event based data handling
- Data Analysis (Business Intelligence)
- GDPR, Open Data compliance as well as general data access rules
- Support publishing and consuming public standards
- Low Code - Usable for non-software developers
- Toolkit of components that can be combined or used standalone
- **Open Source**

Model-based Smart City Platform



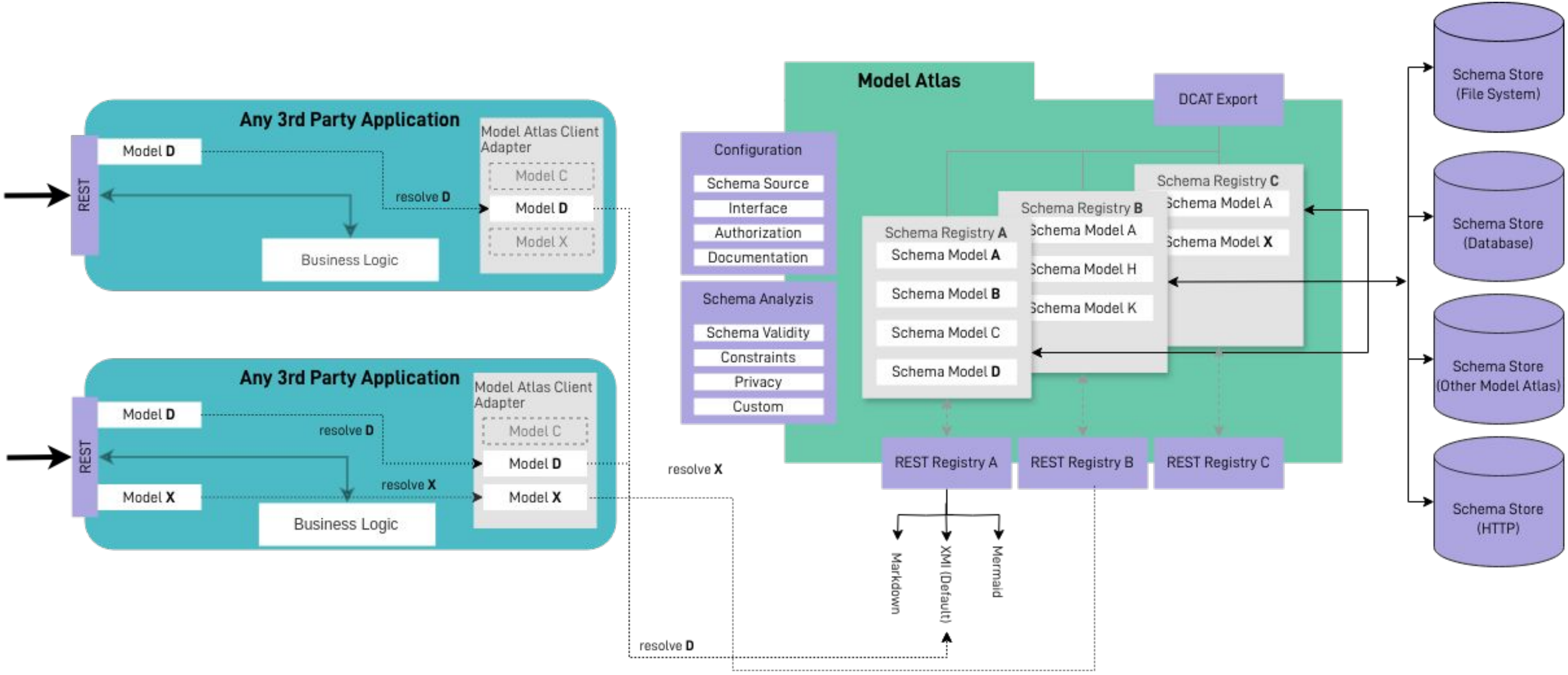
Model based Smart Data Platform

- Keep the existing inventory infrastructure of an organization
- Put an model-based application layer over it
- Connect the new application layer to existing data sources for inventory data and event data
- Hook into existing applications API’s whenever possible
- Adapt the new smart data platform to the existing authentication and authorization infrastructure
- Create bridges and API’s between the layers to enable bidirectional communication, when needed
- When procuring new systems, take the smart data platform integration into account

Used Technologies / Products

- **Gecko Model Atlas** - Distributed Model Registry
- **Gecko Data Atlas** - Model Connectors for Databases, Indexing, Transport Protocols
- Multi-Platform Client Support for Java, JavaScript, Python
- Model Mapping for Models from public standards
- User Interfaces for modeling and / or mapping
- **Eclipse SensiNact** - Event- / IoT Broker with model support
- **Eclipse Daanse** - Data Analysis for model based connectors
- **Gecko Model Analysis Tooling** - analyze models and model instances (Data quality, GDPR checks)
- **Gecko Notary** - Distributed Application Auditing (Transparency, GDPR Auditing)
- Service-based architecture based OSGi specification from Eclipse OSGi Working Group

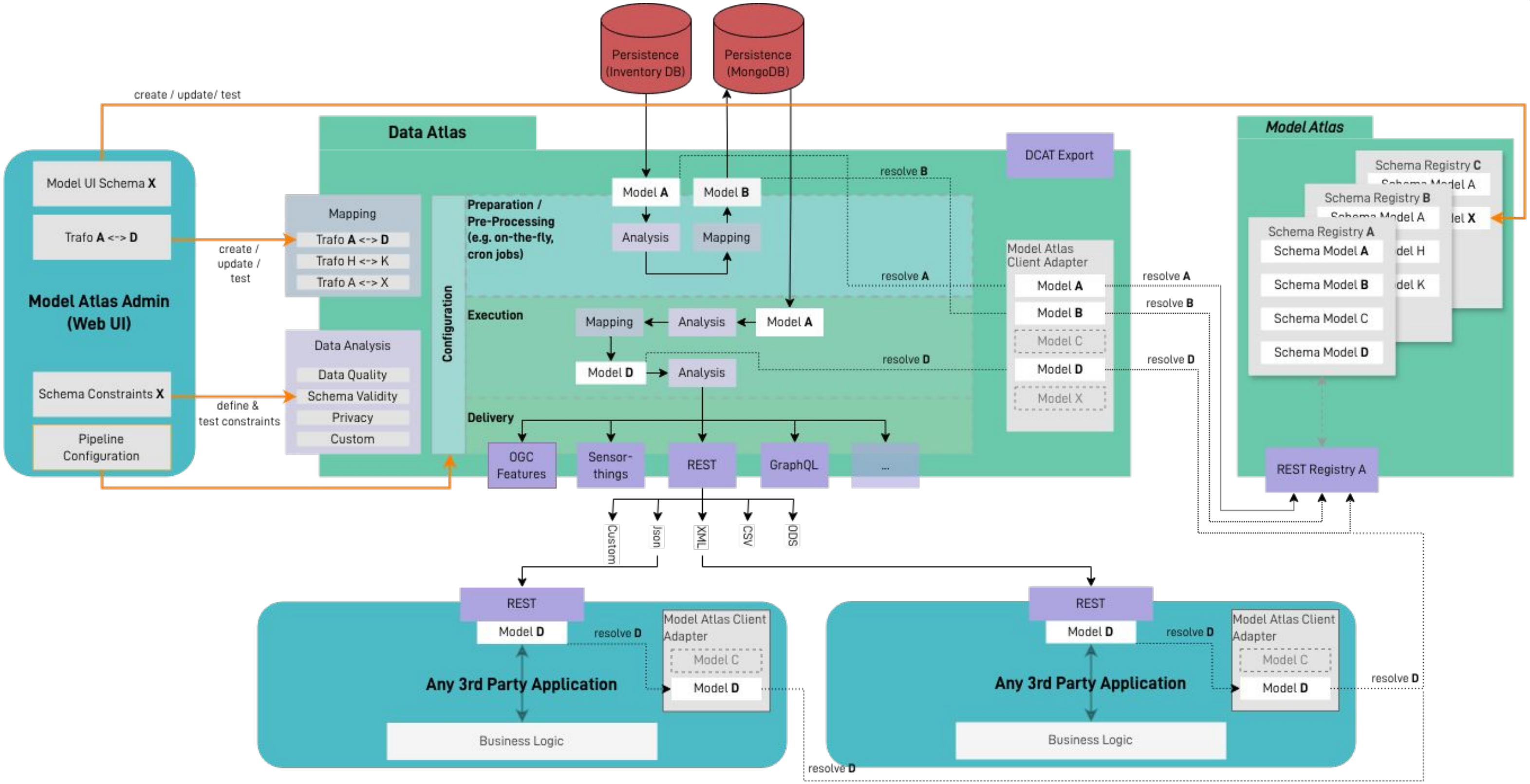
Gecko - Model Atlas



Gecko - Model Atlas

- Web-based EMF Model Registry
- Model-Isolation / Multi-Tenancy
- Plugable / extensible model analysis
- Pluggable model output formats (XMI, XSD, Json-Schema)
- Documentation generation (Diagram image, Plantuml, ODS, ...)
- DCAT / RDF Support for Open Data or Dataspace registries
- Client adapter for model discovery (EMF Java, TypeScript, Python)

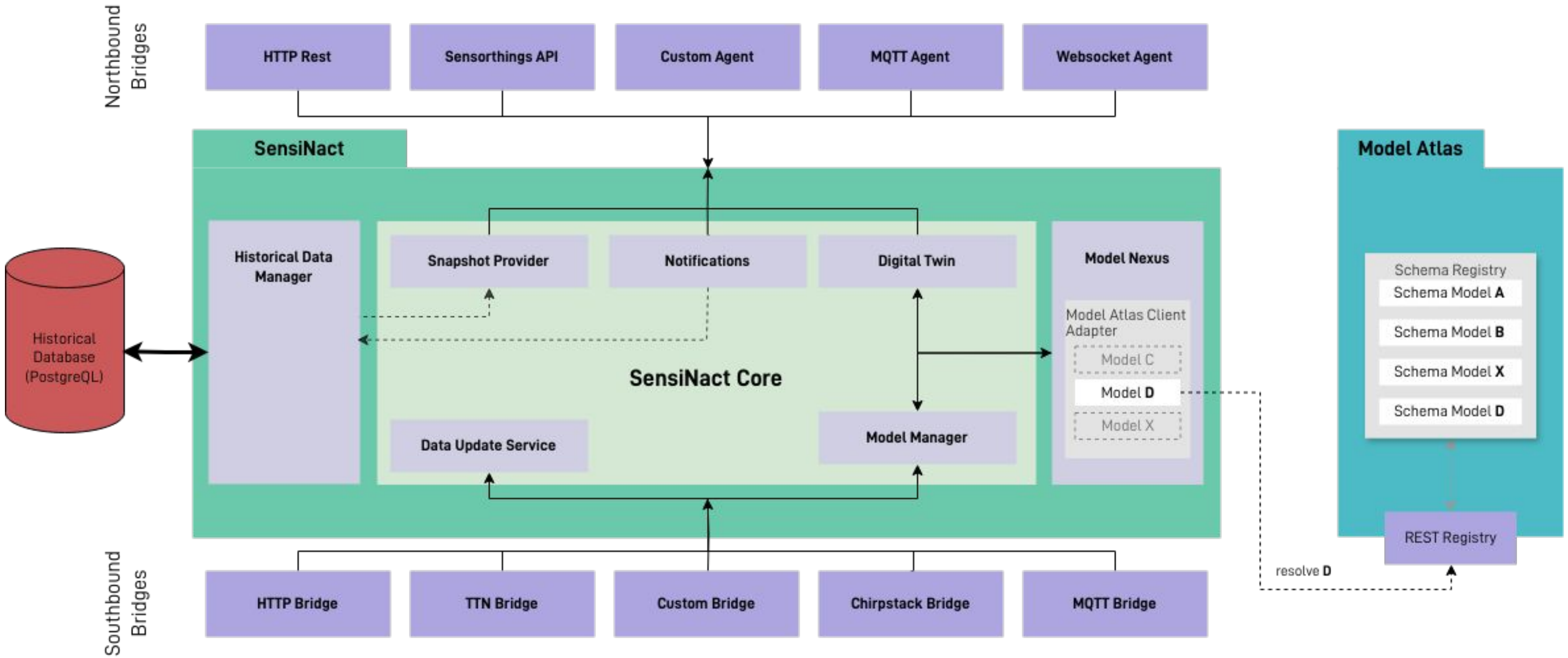
Gecko - Data Atlas



Gecko - Data Atlas

- Persistence Adapter for databases (Relational, Document)
- Configurable processing pipeline
- Model transformation support
- Validation support
- Model Atlas connection
- Scalable and modular (can be embedded or run standalone)
- Cron support for recurring tasks (data quality checks, GDPR checks)
- DCAT / Open Data registry connector

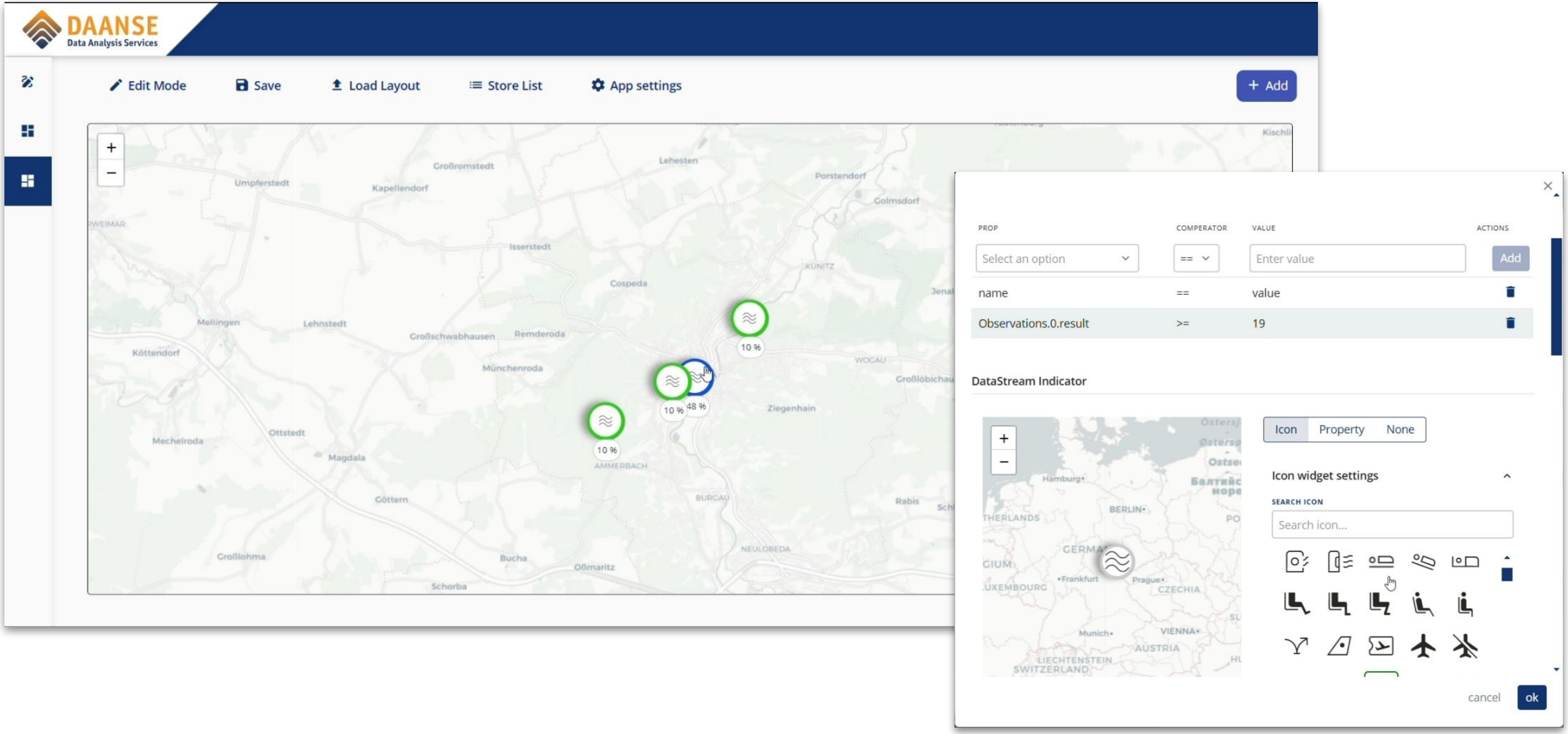
SensiNact - Data / Event Broker



SensiNact - Data / Event Broker

- Eclipse SensiNact Project
- Extensible Southbound Adapters for resources / sensors / actors
- Extensible Northbound Adapters for 3rd party interaction
- Adapters have built-in support for several protocols and plugable
- Core contains intermediate Digital Twin Data Model
- Model-based digital twin adapter for the Model Atlas
- Routing, Filtering, Combining, Mapping of event
- Storing of historical data
- Modular approach - Customize product with just what you need
- Runs embedded or on small IoT devices (e.g. Raspberry Zero)

Eclipse Daanse



PROP	COMPERATOR	VALUE	ACTIONS
Select an option	==	Enter value	Add
name	==	value	
Observations.0.result	>=	19	

DataStream Indicator

Icon | Property | None

Icon widget settings

SEARCH ICON

Search icon...

cancel ok

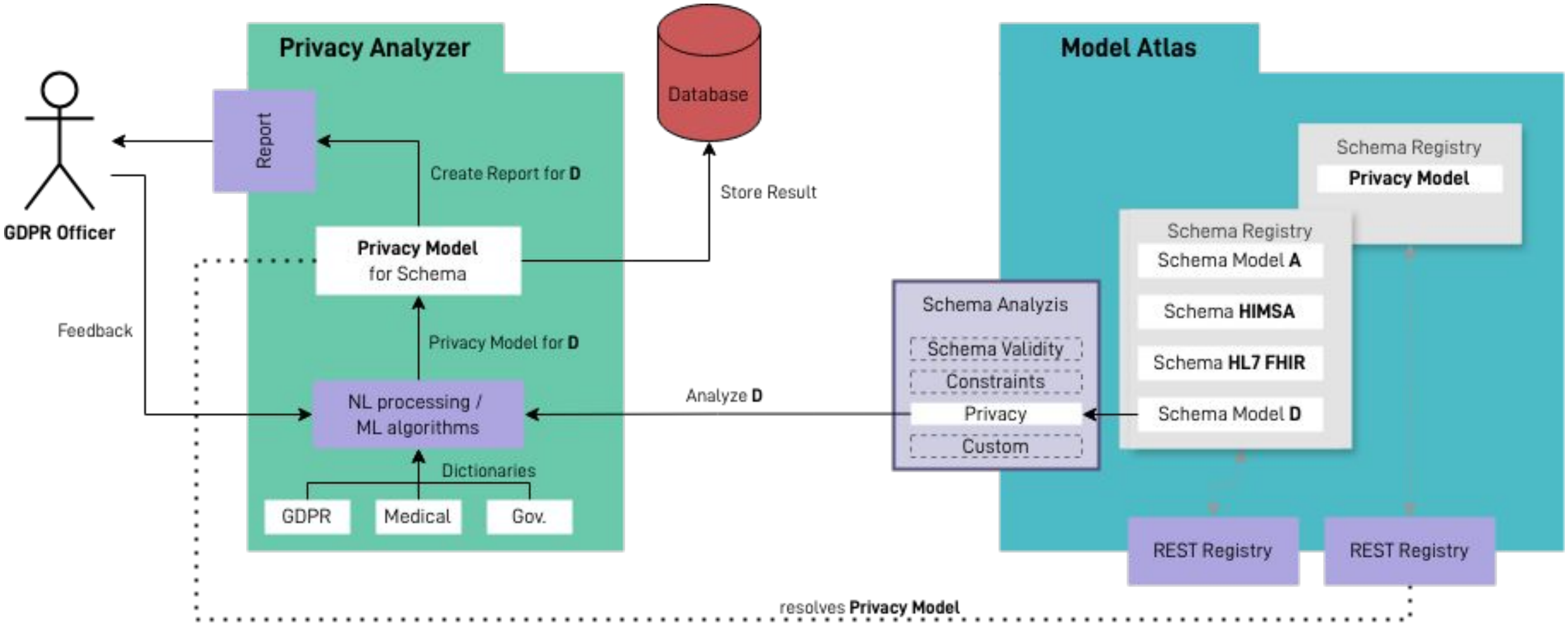
Eclipse Daanse

- Eclipse Data and Analysis Services
- Statistical Analysis, Big-Data Analysis
- Java API
- OLAP Datasources, Database connections
- XMLA Support
- Dashboard engine with variety of visualizations
- Dashboard datasources for Sensorthings, OGC (WMF, WFS), REST
- Queries / Data-Cubes based on models instead of tables

Privacy Tooling

- We developed a model-based approach for privacy tooling
- Models are inspected for possible field definitions, that may contain sensitive information
- Natural Language Processing helps in the analysis
- There is a 2 layer analysis:
  - Model / Schema Analysis
  - Model instance analysis
- Decision support system for the realization of the GDPR
- Analyse basics are inspired from discussions in the *Models 4 Privacy* Interest Group within the Eclipse Foundation
- Can also be used for non-privacy related purposes

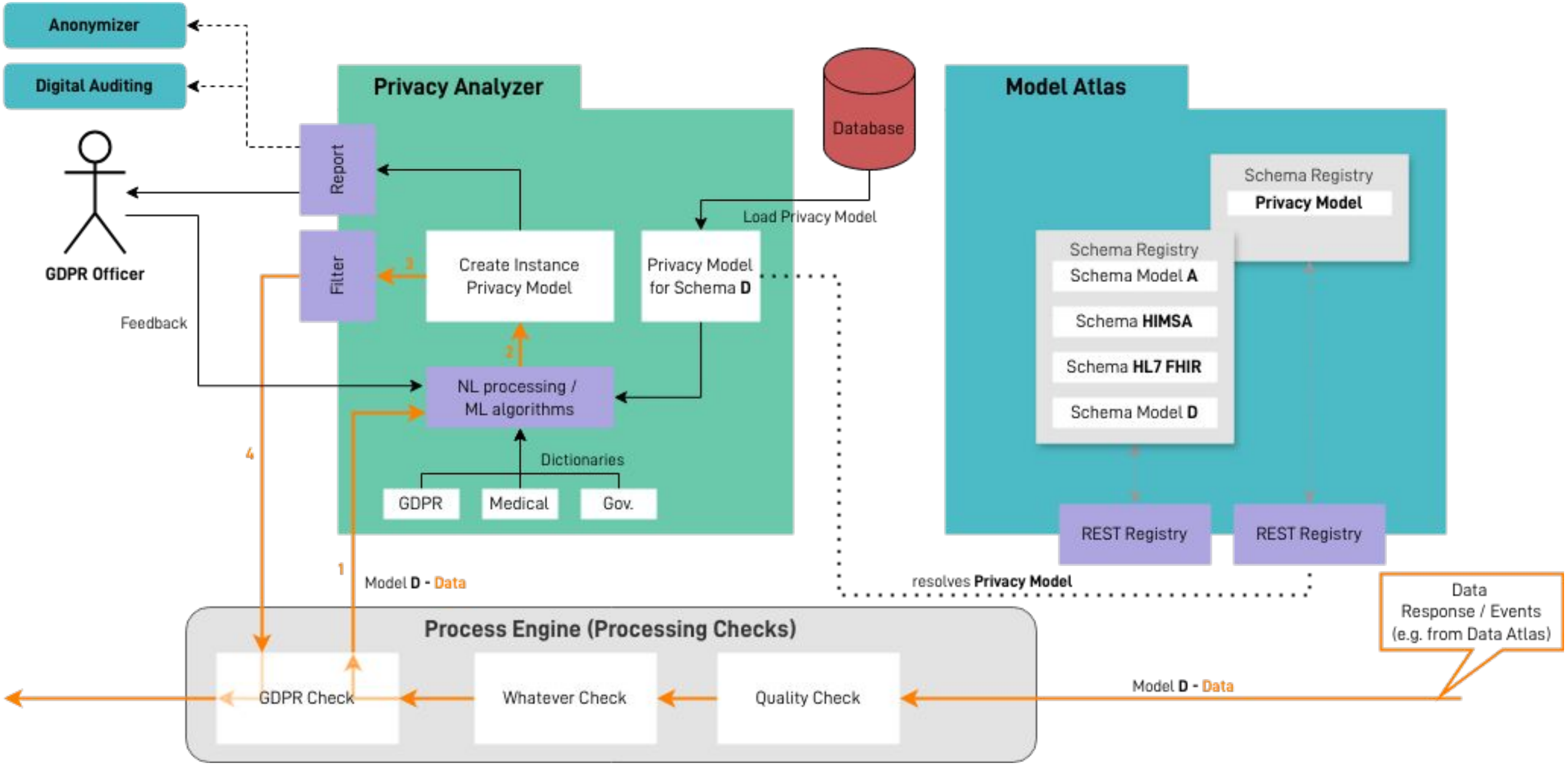
Model Privacy Analysis



Gecko - Model Analysis Tooling

- ML based model analysis
- Analysis report generation with feedback option
- Machine readable report model
- Supports GDPR, medical dictionaries
- Pluggable support new dictionaries
- Decision support system for e.g. GDPT officers
- Remote Service communication to Python ML component

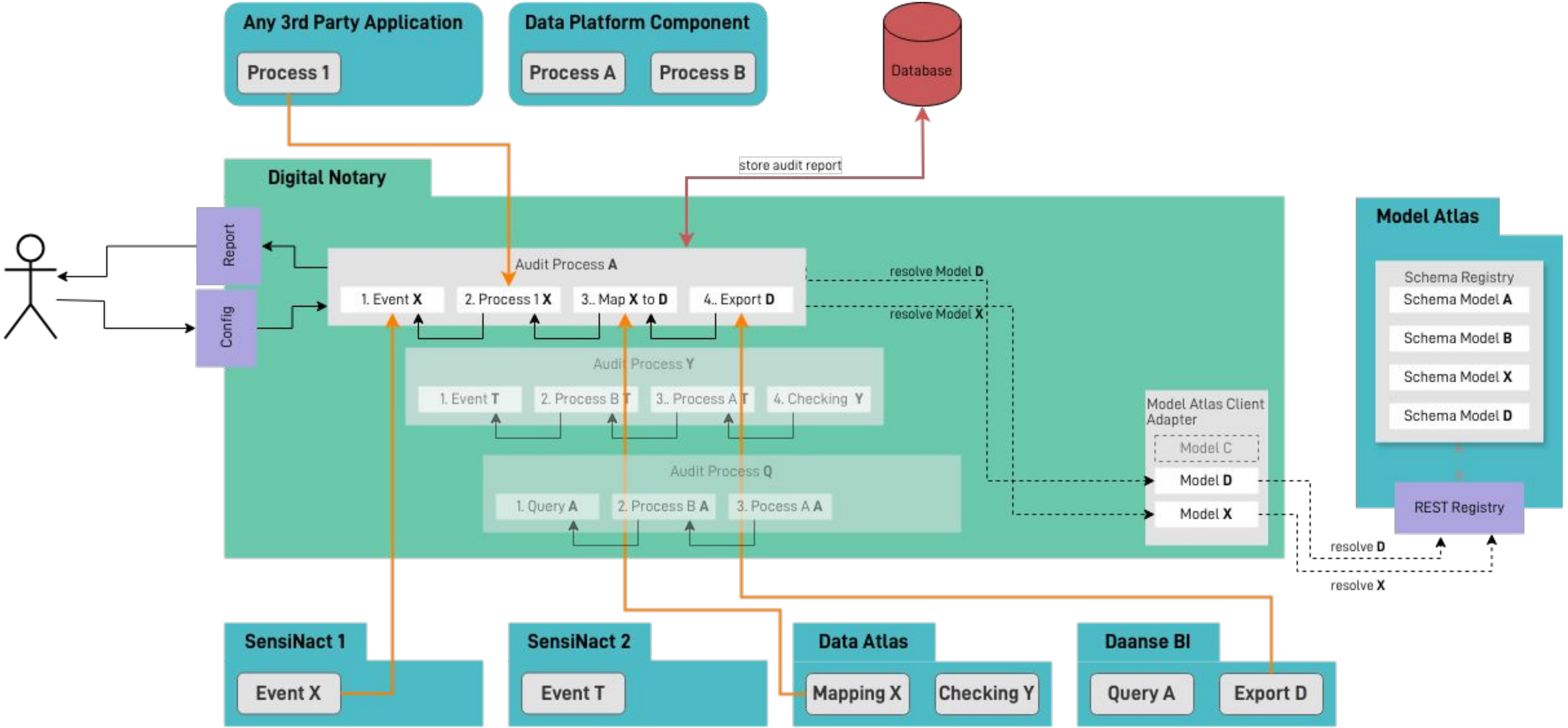
Model Instance Analysis



Gecko - Model Instance Analysis

- **Inconspicuous fields can contain sensitive information!**
- Observe data for elements that are not captured by the model analysis
- ML based model instance analysis
- Analysis report generation with feedback option
- Executed when data of a certain structure / model are retrieved
- Audit report data processing (Transparency, GDPR)
- Uses results from the previous model analysis
- Can run as recurrent job

Digital Notary



Digital Notary

- Usually component have an own logging, which is a technical log
- For regulatory, transparency or other reasons sometimes a non-technical audit is need
- Audit entries can have different information depending on the process
- Audit hook services can be consumed by an application.
- Alternatively an application can use a REST interface to publish audit entries
- Audit entries are chained and hashed to prevent modifications
- Entries and process can be individually configured
- There is an extensible model that can be extended

Conclusion

- *Model-Atlas* is single point of contact for models
- Model changes are reflected to DCAT Registries (Open Data Portals, Data Spaces)
- Data broker like *SensiNact* can handle model instances for event based data
- Data Atlas acts a connector existing systems like databases
- Data Analysis Tools like Eclipse *Daanse* can use Model-Data for e.g. Dashboard creation
- Model and instance analysis for privacy related data for GDPR conformance
- De-centralized auditing for processing steps as transparency documentation system
- TypeScript and Python support for EMF Ecore

Conclusion

A model-based platform like this is a toolkit consisting of dynamic, distributed, modular components.

- The principle of **Modularity** is crucial for an extensible architecture
- Modular, distributed components or models always have to deal with the same challenges like dynamic behavior and the tenets of distributed computing
- The model-driven approach enables low-code development for non-technical people
- End-to-end usage of model / instances within the whole system not only within a single component
- Service-orientation is a basic principle
- All components a **Open Source**

Outlook

- *Model Atlas, Data Atlas, Model Analysis Tooling* and *Digital Notary* are currently available under **https://github.com/geckoprojects-org**
- We are currently in the process of moving these components to the Eclipse Foundation
- The project proposal was accepted and the project name will probably be **Eclipse Fennec**
- As well a DIN specification for Urban Data Platforms as architecture models like from Civitas Connect e.V. containing model registries like the *Model Atlas*
- There are lots of existing models for open standards like OGC, KML or HL7 that can be reused
- A business process engine based in BPMN is planned for next year
- We also think about Git support for model creation and support for existing alternative model editors

SMARTCITY

EXPO WORLD CONGRESS

Visit us in Barcelona from 05 - 07.09.2024

Hall 2 booth D 111

Meet Jürgen Albert (CEO) and Mark Hoffmann (CEO/CTO)

We are looking forward talking with you.

All materials are available at: ***https://www.datainmotion.com/scewc24/***