

# The Simantics System for Semantics-based Integrated Modeling

Hannu Niemistö  
VTT

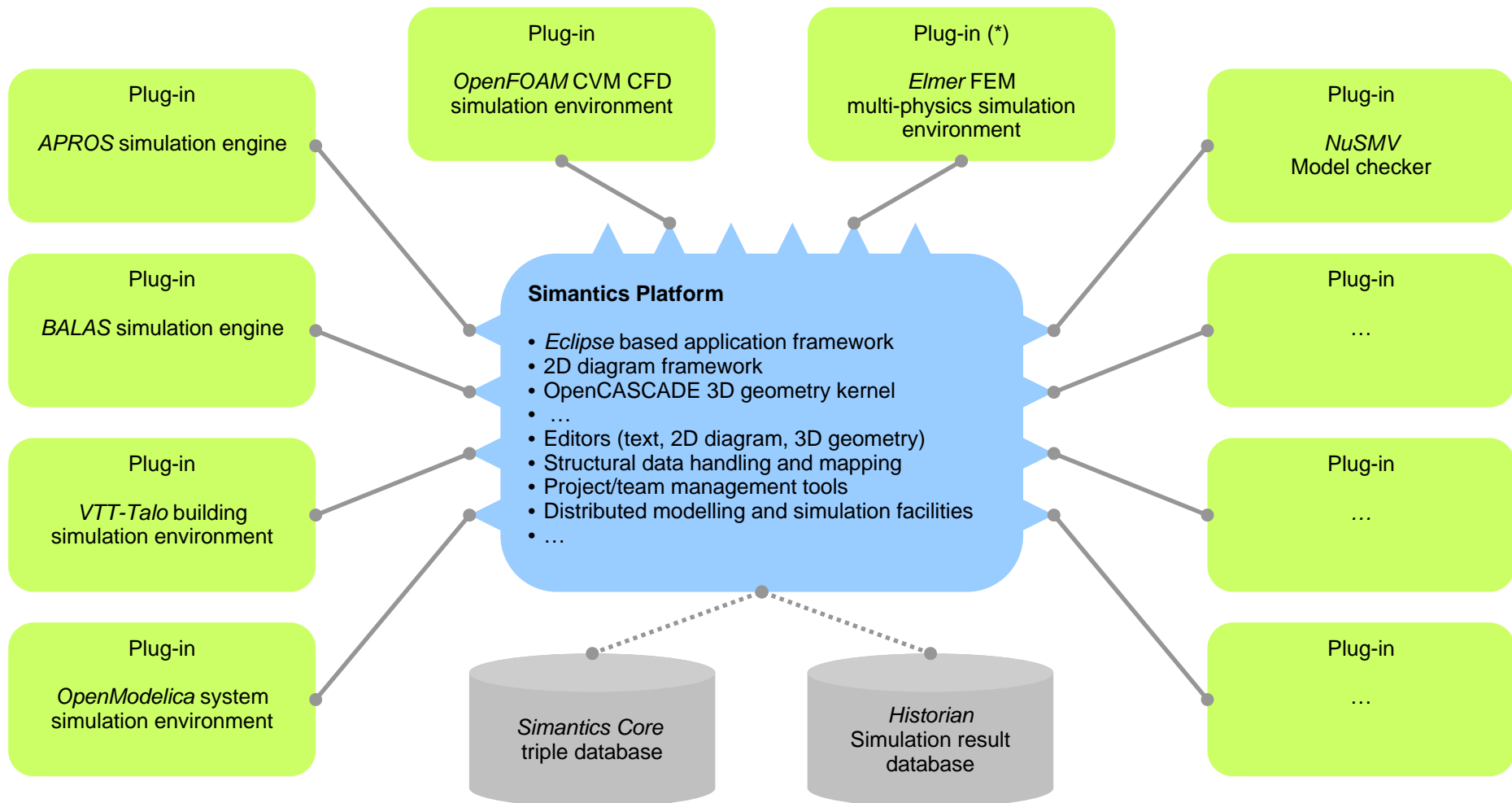
## Original goals

- There are lots of different kind of simulators built on research projects in VTT that never get used in industry because they are not productized.
- A simulation integration platform:
  - Support for multi-disciplinary multi-level simulation and modeling at VTT
  - Reusable components for modeling and simulation infrastructure (less focus on solver technology)
  - Open source
  - Mapping between different data models
  - Team work (version control)
  - Distributed simulation
  - Simulation integration

## Simulators we have integrated or are integrating

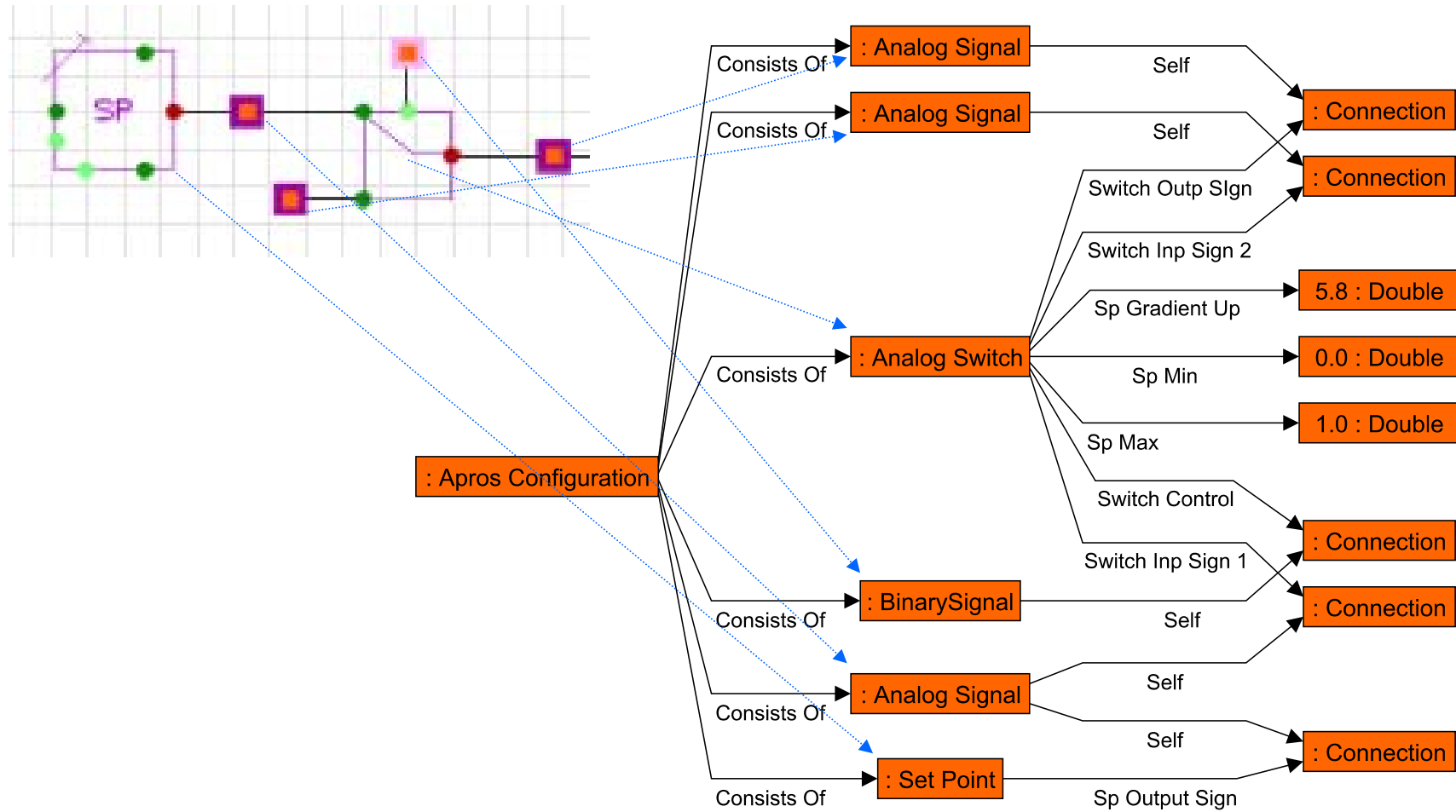
- Apros (to be released in 9/2010)
  - Dynamic process simulator in particular for conventional and nuclear power plants and paper production processes
  - Most used simulator developed in VTT
- Balas (to be released in 9/2010)
  - Steady-state process simulator
- OpenModelica
  - Multi-domain equation-based dynamic system simulator
- NuSMV
  - Model checking
- OpenFOAM
  - CFD toolbox
- An academic phase field simulator
- Nano cellulose solvers
- TempSimu
  - Continuous casting simulator
- A toy discrete event system simulator for testing purposes

# Plug-in Architecture for Modelling and Simulation



## How to unify the data model of different simulators?

- Started the experiments with XML in 2000
- Played with custom data models in 2003
- In 2005, embraced "semantic" technologies (Simantics project starts at 2006)
- In some respects, we deviated also slightly from RDF:
  - URIs are implicit (they encode ConsistsOf-paths of the resources).
  - Literals are not objects of statements but they are attached to a resource. The data type of the literal is indicated by a statement in the resource.
- Everything is put into one large semantic graph.



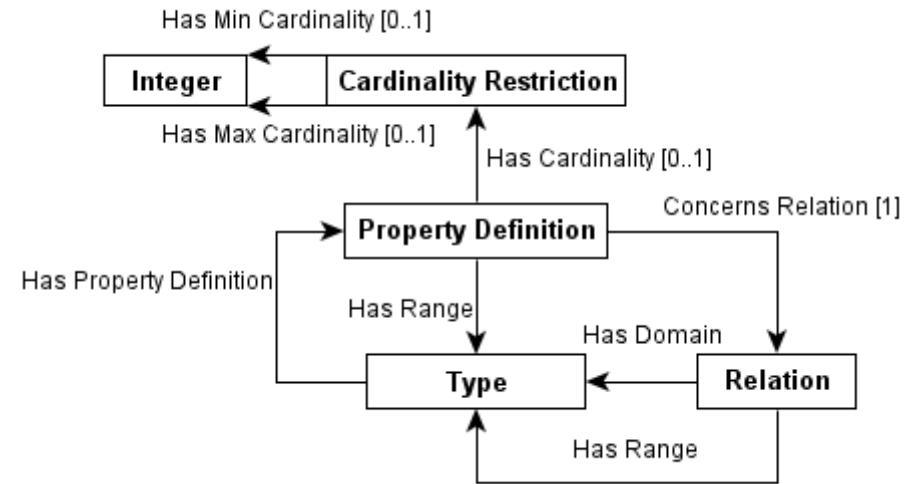
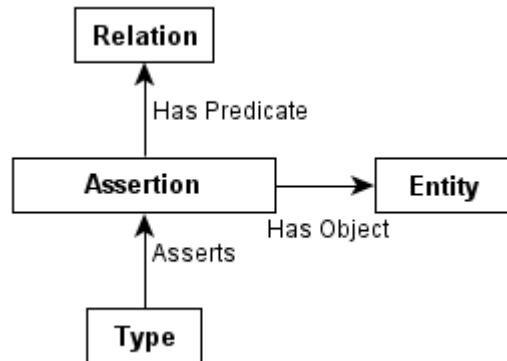
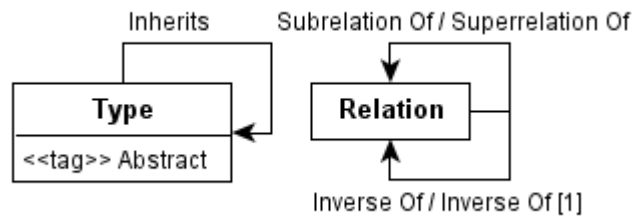
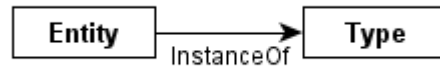
## Comparison of semantic graph and object oriented representation of data

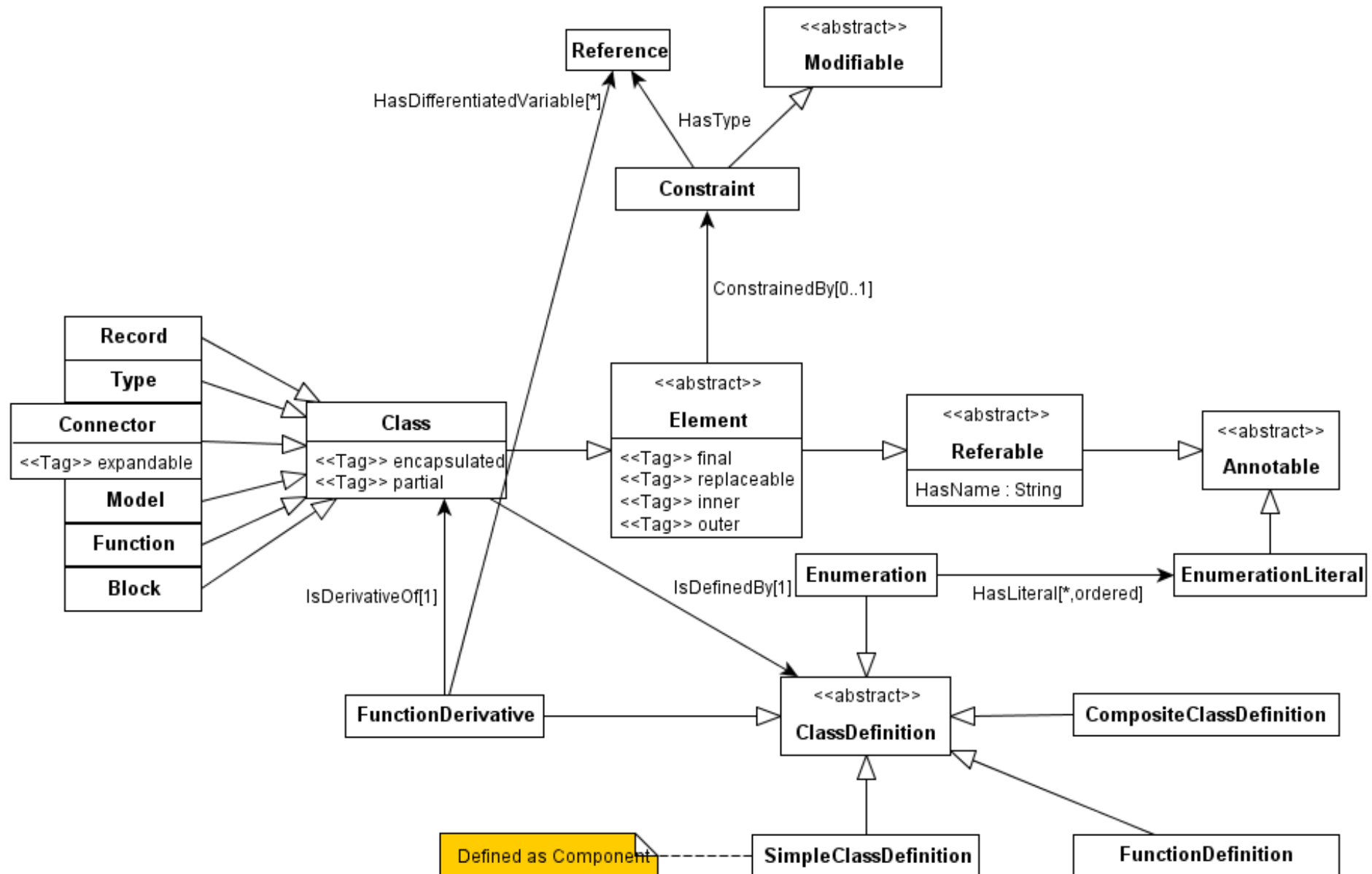
- Instances may contain arbitrary attributes (not only the attributes mentioned in the type).
  - More flexible
  - Easier to write invalid (not conforming to ontology) data
  - Does not allow as efficient serialization as OO-representation
  - But possible savings by storing only non-default values.
- No so clear separation between instance and type data:
  - Easy to add metadata to types
  - Model configurations may contain types
  - Of course, this can be modeled in OO-representation
- Relations are not bound to certain types (as attributes are). The relations may be subrelations of other relations.
  - Classification and grouping of attributes useful
  - Makes mapping to OO-representation harder

## How to describe ontologies?

- Tried existing OWL reasoners in 2006. Slow, not expressive enough and open world semantics not very useful in simulation and modeling context.
- We used OWL-like restriction language with closed world semantics in 2007-2008. Efficient for data validation, but not expressive enough. Also not optimal for code generation.
- Inspired by MOF, splitted the description language into two parts (in 2008):
  1. limited UML-like language
    - Useful for code generation, expressive enough for most restrictions
  2. more expressive constraint and transformation language (SCL) based on Datalog







## Literal data

- In RDF, literal types are usually described with XML schemas. XML documents are not very efficient representation of large arrays of numerical data.
- First we supported only fixed set of data types (primitives and arrays of primitives).
- In 2009, we designed a structural type system for literal data:
  - Primitives: Byte, Integer, Long, Float, Double, String
  - Array types
  - Record types
  - Union types
  - Recursive type definitions
  - Variants, optional types, maps
  - Numerical range restrictions, array&string length restrictions, regular pattern restrictions for strings

## Data storage

- Tried different triple stores in 2005 (for example Jena)
  - Not satisfactory performance for small writes and local reads.
- Started to implement our own triple store (Simantics Core) in 2006
- Features
  - Optimized with clustering for local browsing, not for large queries over the whole database
  - Transactional
  - Versioning
  - So far tested with up to 10Mtriples of data
- Numerical data is collected in a file based storage during the simulations and archived to triple store.

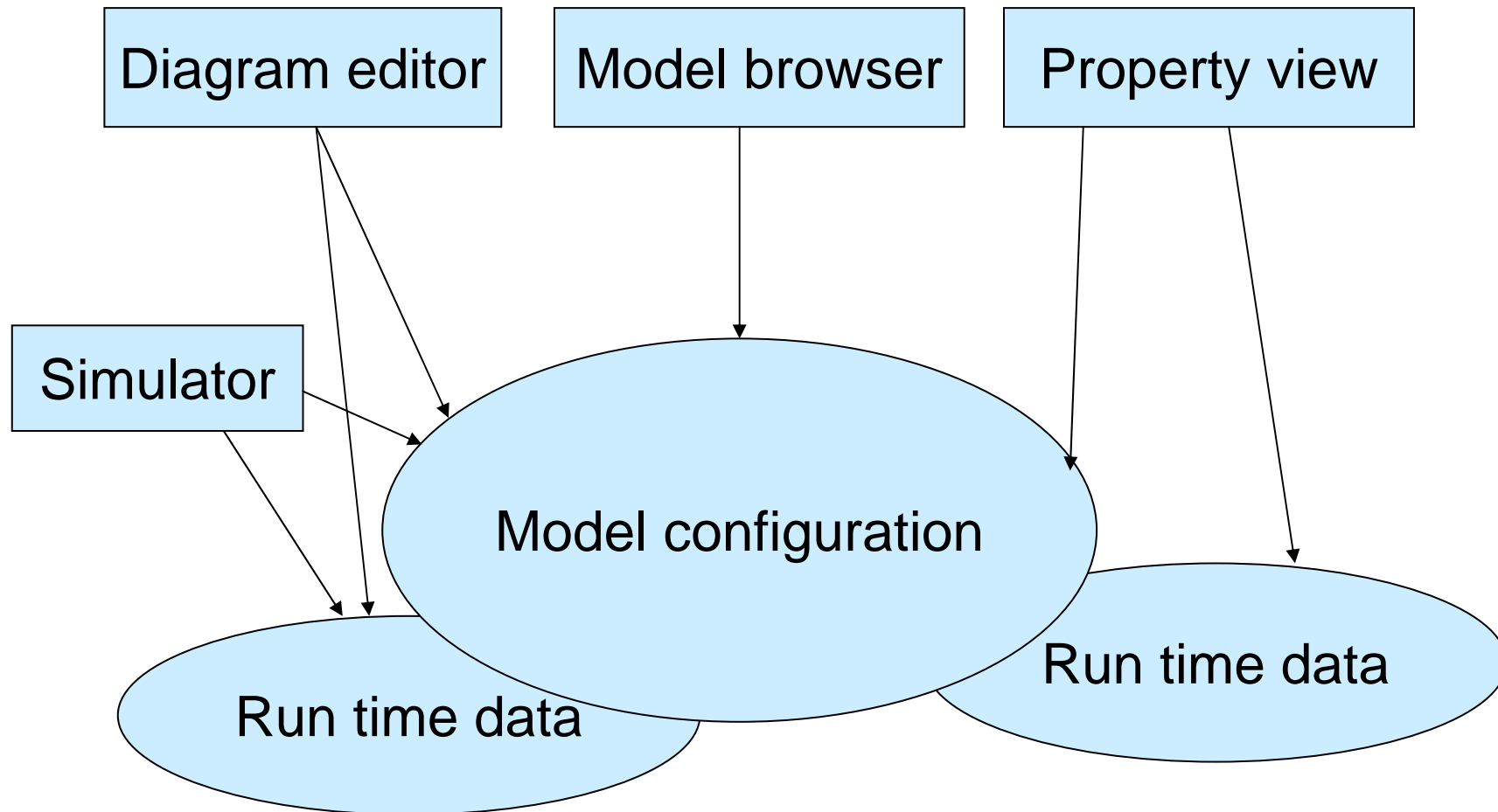
## Primitive DB API example

```
public static void printChildren(final Resource library)
                                throws GraphRequestException {
    SimanticsUI.getSession().syncRequest(
        new ReadRequest() {
            public void run(ReadGraph g) throws GraphRequestException {
                Builtins b = g.getBuiltins();
                for(Resource child : g.getObjects(library, b.ConsistsOf))
                    System.out.println(g.getRelatedValue(child, b.HasName));
            }
        });
}
```

## Listening the data

- Much of the implementation effort of the user interfaces goes for listening and synchronizing the data models.
- We have two basic listening mechanisms for model configurations:
  - Making incrementally updated queries to the database. This mechanism is mostly used in user interfaces.
  - Listening and processing change sets (graph deltas) produced by write transactions. Used only in some cases where we cannot produce the query beforehand: for example validation.
- Also lots of utilities on top of these mechanisms

## Communication between components



```
@GraphType("http://www.simantics.org/Sysdyn#Dependency")
public class DependencyElement extends Element {
    @RelatedElement("HasTail")
    Connectable tail;
    @RelatedElement("HasHead")
    Connectable head;
    @RelatedValue("HasAngle")
    double angle = 0.1;
    ...
}
```

```
public class SysdynDiagramSchema extends SimpleSchema {
    ...
    addLinkType(MappingSchemas.fromAnnotations(g,
        DependencyElement.class));
    ...
}
```



```
public class SysdynDiagramEditor extends ResourceEditorPart {
    ...
    SysdynDiagramSchema schema = new SysdynDiagramSchema(g);
    mapping = Mappings.createWithListening(schema);
    diagram = (IDiagram)mapping.map(g, getInputResource());
    ...
    mapping.addMappingListener(new IMappingListener() {
        @Override
        public void domainModified() {
            session.asyncRequest(new ReadRequest() {
                @Override
                public void run(ReadGraph graph)
                    throws DatabaseException {
                    mapping.updateRange(graph);
                }
            });
        }
    });
    ...
}
```

## Real time data

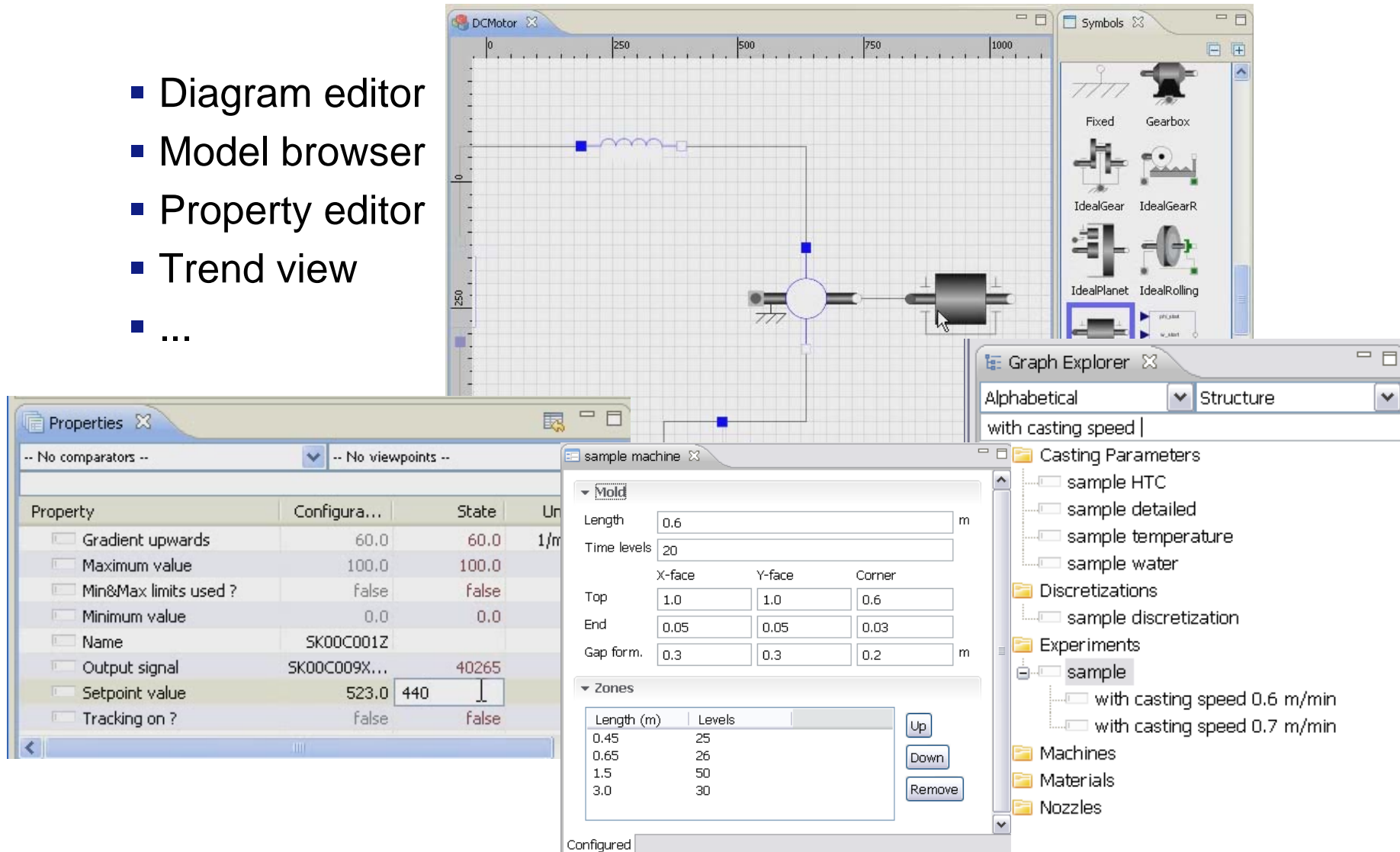
- Built on a simple RPC-protocol
- Communication described with the same data types that was used in literals.
- Protocols may be described using textual notation or by reflection with Java-interfaces
  
- Currently developing
  - Graph based access to real time data. This would allow a uniform listening mechanisms.
  - Unified experiment control model. Current framework is restricted to dynamic system simulation and for example steady-state simulator (Balas) has a custom interface.

## User Interface

- Built on Eclipse platform
- We use
  - Eclipse RCP (SWT, JFace)
  - P2 (provision)
  - Mylyn
  - Plugins/Extensions/Extension Points
- We don't use
  - Eclipse file system (IResource etc.)
    - We store almost all data in one graph database
  - GEF
    - We have our own diagramming framework using AWT
  - EMF

## Generic Simantics UI components

- Diagram editor
- Model browser
- Property editor
- Trend view
- ...



The screenshot displays the Simantics software interface with several key components:

- Diagram Editor:** A central workspace showing a schematic diagram of a motor and gearbox assembly on a grid.
- Symbols Palette:** A panel on the right containing various mechanical symbols such as Fixed, Gearbox, IdealGear, IdealGearR, IdealPlanet, and IdealRolling.
- Properties Editor:** A table at the bottom left showing the configuration and state of a component.
- Graph Explorer:** A panel at the bottom right showing a hierarchical tree structure of the model, including Casting Parameters, Discretizations, Experiments, Machines, Materials, and Nozzles.
- Model Browser:** A panel at the bottom center showing the configuration of a 'Mold' component, including Length, Time levels, Top, End, Gap form, and Zones.

Property	Configura...	State	Un
<input type="checkbox"/> Gradient upwards	60.0	60.0	1/m
<input type="checkbox"/> Maximum value	100.0	100.0	
<input type="checkbox"/> Min&Max limits used ?	false	false	
<input type="checkbox"/> Minimum value	0.0	0.0	
<input type="checkbox"/> Name	SK00C0012		
<input type="checkbox"/> Output signal	SK00C009X...	40265	
<input type="checkbox"/> Setpoint value	523.0	440	
<input type="checkbox"/> Tracking on ?	false	false	

Length (m)	Levels
0.45	25
0.65	26
1.5	50
3.0	30

## Template of simulator integration process

- Write the data model (ontology) of the simulator.
- Customize existing user interface or implement new UI-components so that they listen and manipulate the configuration data in Simantics DB.
- Implement simulator integration (transforming data from graph to the native representation of the simulator)
  - Every simulator is different
  - Very big models require that the native representation of the simulator is updated incrementally. This is also necessary if the data model in Simantics does not contain all aspects of the native model (for example parts of the state).

**Thanks!**

More information in  
<http://www.simantics.org/>