

HELSINKI UNIVERSITY OF TECHNOLOGY  
Department of Computer Science and Engineering  
Telecommunications Software and Multimedia Laboratory

**Marko Luukkainen**

# **Use of 3D graphics for configuration and visualization of large scale process simulation: ontology-based approach.**

Master's Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Technology.

Espoo, May 10, 2007

## **Supervisor**

Professor Lauri Savioja

## **Instructor**

Timo Tossavainen, Ph.D.

HELSINKI UNIVERSITY OF TECHNOLOGY      ABSTRACT OF MASTER'S THESIS Department of Computer Science and Engineering	
Author Marko Luukkainen	Date May 10, 2007 <hr/> Pages 8+96
Title of thesis Use of 3D graphics for configuration and visualization of large scale process simulation: ontology based approach.	
Professorship Interactive Digital Media	Professorship Code T-111
Supervisor Professor Lauri Savioja	
Instructor Timo Tossavainen, Ph.D	
<p>Traditionally design- and simulation applications have been separated, but extensive use of simulation to support design process drives to find integration of those. This thesis studies integration of 3D industrial plant modelling, process simulation and visualization with ontologies.</p> <p>The aim was to develop a user interface for 3D plant modelling that supports process simulation and animations required by visualizations. The design focused on usability of modelling and visualization techniques. As a result, we developed a software that can be used for modelling equipment of a plant, and parts of the plant, which take account for the process. User may configure boundary values for the simulation and visualize behaviour of equipment and flows inside pipes.</p> <p>Ontology-based approach was used for linking simulators and plant data model: concepts of 3D graphics, 3D plant modelling, and example simulator's data structures were represented as ontologies. Using ontologies enables integrated use of different data models. Here the advantage is that the implementation is not bound to any particular simulator, and other simulators can be easily added to the system.</p>	
Keywords: 3D industrial plant design, simulation, visualization, ontology	

TEKNILLINEN KORKEAKOULU		DIPLOMITYÖN TIIVISTELMÄ	
Tietotekniikan osasto			
Tekijä Marko Luukkainen		Päiväys 10.5.2007	
		Sivumäärä 8+96	
Työn nimi Kolmiulotteisen grafiikan käyttö laajaskaalaisen prosessisimulaation mallintamiseen ja visualisointiin: ontologiapohjainen ratkaisu			
Professori Interaktiivinen digitaalinen media		Koodi T-111	
Työn valvoja Professori Lauri Savioja			
Työn ohjaaja Timo Tossavainen, FT			
<p>Perinteisesti suunnittelu- ja simulointiohjelmistot ovat olleet erillisiä, mutta simuloinnin laaja käyttö suunnittelun tukena vaatii simulointiominaisuuksien integrointia suunnitteluohjelmistoon. Tämä diplomityö tutkii kuinka 3D-laitosmallinnus, prosessisimulointi ja visualisointi voidaan integroida ontologioita käyttäen.</p> <p>Työn tavoitteena oli kehittää 3D-laitossuunnitteluun soveltuva käyttöliittymä, joka tukee prosessisimulointia ja visualisoinnissa tarvittavia animointeja. Suunnittelun painopisteenä oli sekä käytettävyys, että visualisointitekniikat. Työn tuloksena syntyi ohjelmisto, jolla pystyy mallintamaan prosessiteollisuuslaitoksen laitteet ja itse laitoksen prosessiin osallistuvat osat, säätää simuloinnin raja-arvot, sekä visualisoimaan laitteiden ja putkissa kulkevien virtausten käyttäytymistä.</p> <p>Simulaattorien kiinnittämiseen laitosmallin käytettiin ontologia-pohjaista ratkaisua: sekä kolmiulotteisen grafiikan, kolmiulotteisen laitosmallinnuksen ja esimerkkisimulaattorin käsitteet kuvattiin ontologioina. Ontologioiden käyttö mahdollistaa eri tietosisältöjen yhteiskäytön, ja tässä tapauksessa toteutus ei ole sidottu yksittäiseen simulaattoriin, vaan uusien simulaattorien käyttöönotto järjestelmässä on helppoa.</p>			
Avainsanat: 3D-laitossuunnittelu, simulaatio, visualisaatio, ontologia			

# Acknowledgements

I want to thank my supervisor professor Lauri Savioja for his input, and instructor Timo Tos-savainen for valuable guidance with the thesis.

I would like to thank entire Simantics platform development team, Toni Kalajainen, Kalle Kon-delin, Tuukka Lehtonen, Antti Villberg, and our team leader Tommi Karhela. Without you this would not have been possible.

I would also like to thank Petteri Kangas for his valuable help with example bleaching line simulation.

Finally, I would like to thank my parents, my sister, and my friends to whom I never tried to explain what I was writing about.

Espoo, May 10, 2007

Marko Luukkainen

# Table of Contents

1.	Introduction .....	1
1.1.	Goals and Scope .....	2
1.2.	Structure .....	2
2.	Background .....	3
2.1.	Process Simulation .....	3
2.2.	Plant modelling .....	4
2.2.1.	Plant Design .....	4
2.2.2.	Pipe Classes .....	5
2.2.3.	Benefits of 3D modelling .....	6
2.2.4.	An example of plant design software .....	6
2.3.	3D Interaction .....	8
2.3.1.	Object Manipulation .....	9
2.3.2.	Viewpoint Handling .....	13
2.4.	Solid modelling .....	14
2.4.1.	Data representation .....	15
2.4.2.	Parametric Modelling .....	16
2.4.3.	Initial Graphics Exchange Specification .....	18
2.4.4.	Standard for the Exchange of Product Model Data (ISO 10303) .....	19
2.5.	Data Visualization .....	20
2.5.1.	Animation techniques .....	20
2.5.2.	Flow visualization .....	21
2.5.3.	Visualization of large, complex 3D models .....	21
2.5.4.	Visualizing manufacturing simulation with animation .....	24
2.6.	Ontology based programming and modelling .....	24
2.6.1.	Communication .....	26
2.6.2.	Inter-Operability .....	27
2.6.3.	Systems Engineering .....	28
2.6.4.	Information represented by ontologies .....	28
2.6.5.	Design process, criteria and evaluation. ....	29
2.6.6.	Resource Description Framework .....	31
2.6.7.	Web Ontology Language .....	32
2.7.	Using ontologies with graphics and simulation .....	32
2.7.1.	Linking graphics to domain ontologies .....	32
2.7.2.	Linking graphics to simulation concepts .....	34

2.7.3.	Semantics based geometric simplification .....	34
3.	Requirement analysis .....	37
3.1.	Functional requirements .....	37
3.2.	Technical requirements.....	39
4.	Implementation platform.....	41
4.1.	Layer0.....	42
4.2.	User Application .....	45
4.3.	Ontology Development.....	46
4.4.	Basic Ontologies .....	47
4.5.	Data Visualization .....	48
5.	Design .....	49
5.1.	Shape Editor.....	49
5.1.1.	Modelling .....	49
5.1.2.	Animation.....	51
5.1.3.	Parameterisation.....	51
5.2.	Process Editor .....	51
5.2.1.	Process modelling .....	51
5.2.2.	Mapping the plant model to a simulation model.....	53
5.2.3.	Configuring and visualizing simulation .....	53
6.	Implementation.....	55
6.1.	Used software components .....	55
6.1.1.	jME .....	55
6.1.2.	OpenCASCADE .....	56
6.2.	Scene-graph .....	56
6.3.	Shape Editor.....	58
6.3.1.	Geometry .....	59
6.3.2.	Animation.....	61
6.3.3.	Parameterization of geometry .....	63
6.4.	Process Editor .....	65
6.4.1.	Plant modelling ontology .....	65
6.4.2.	Pipeline modelling .....	66
6.4.3.	Plant modelling user interface .....	76
6.4.4.	Simulation configuration and visualization .....	77
7.	Analysis and Discussion .....	79
7.1.	An example of simulation configuration and visualization: bleaching line.....	79
7.2.	Evaluation against requirements .....	81
7.2.1.	Equipment modelling and animation .....	82

7.2.2.	Plant modelling .....	82
7.2.3.	Simulation and visualization.....	83
7.3.	General analysis .....	84
7.3.1.	Usability .....	84
7.3.2.	Scalability .....	85
7.4.	Future Work.....	85
7.4.1.	Other use cases for 3D modelling and 3D Plant Model .....	87
8.	Conclusions .....	89

## Abbreviations

B-rep	Boundary Representation
CAD	Computer-aided Design
CAM	Computer-aided Manufacturing
CSG	Constructive Solid Geometry
DAML	The DARPA Agent Markup Language
IGES	Initial Graphics Exchange Specification
ISO	International Organization for Standardization
LOD	Level of Detail
OIL	Ontology Inference Language
OPC	OLE for Process Control
OWL	Web Ontology Language
RDF	Resource Description Framework
STEP	Standard for Exchange of Product Model Data
VRML	Virtual Reality Markup Language
W3C	World Wide Web Consortium
X3D	Extensible 3D
XML	Extensible Markup Language

## 1. Introduction

Use of simulation has increased in process industry. Built plants have been getting more complicated, and requirements for simulation have grown. Today it is common that when new plant is designed, designs are tested and verified using simulators. Typical simulators contain their own user interfaces and their own data structures, and transforming modelling data to simulator compatible form is seldom automatic. Usually a plant model has to be redone for the simulator, increasing effort to use simulator and possible errors in simulator's model. In a common scenario, various systems in plant have been modelled with different software, which further increases risk of error.

Currently, the most common way to create a model for process simulation is to use two dimensional diagrams. At the same time, nearly all new built plants and changes to old plants are designed with some 3D modelling software. Combining 3D modelling with simulation combines advantages from both: Two-dimensional diagrams cannot show physical properties of the modelled process, but 3D models are similar to real world objects. 3D-modelling comes especially handy when sizes of objects are needed by the simulator. In 2D-diagrams numbers must be given explicitly, whereas in 3D-modelling all dimensional values are already in the modelled process.

Because simulators are used for testing and verifying the functionality of the modelled process, it is more practical to include simulation within the modelling software. This would allow an engineer to test his design earlier and improve his understanding of the modelled process and its behaviour. The improved understanding leads to better design practices and results in savings in time and money. In understanding the process and its behaviour, visualization is important. Animations and other visualization techniques can improve perception of simulation results over text-based numeric information or separate 2D graphs.

Ontology-based knowledge representation has gained popularity because of its flexibility and expressive power. Here using the ontology-based approach has critical advantages: 3D modelling user interface does not have to be bound to any particular simulator nor any particular simulator type. Instead, simulators can be linked to 3D model's data structures, and the same visualization package can be used with many different simulators and the same 3D model with multiple simulators. This is required, because different simulators are used in different stages of the modelling process: First mass and energy balances are simulated using steady state simulation; later dynamic simulation can be used for various purposes, including testing plant's behaviour in changing conditions, and in operator training.

## **1.1. Goals and Scope**

Our goal is to represent how 3D plant modelling interface can be used for configuring process simulation. This includes creating a simulation model from 3D plant model, configuring simulation input values (boundary conditions), and visualizing the simulation. Visualization includes equipment states and flows in pipes.

The aim of this thesis is to make tools that can be used for modelling equipment, components and pipes. The aim is not to make full scale 3D plant modelling software: we omit electrical, structural and other modelling features that are required when actual plant is built according to 3D model, because those do not affect the industrial process and therefore they are not useful for process simulation. Also piping functionality will be reduced: flanges and other pipeline components that are part of pipeline, but do not affect the actual process will be left out.

## **1.2. Structure**

Background, related research, and common concepts of the area are represented in chapter 2. It includes wide range of subjects: process simulation, plant design, 3D interaction, solid modelling, visualization, and finally ontologies. Chapter 3 presents requirement analysis: what the system should be able to do and why. Simantics platform, which is used in implementation, is presented in chapter 4. Chapter 5 discusses the design, and chapter 6 presents the implementation. Analysis of the work and result are in chapter 7, and conclusions are in chapter 8.

## 2. Background

This chapter begins with a brief explanation of process simulation, and plant modelling, since both of them are the basis of this work. In section 2.3 we discuss about 3D interaction, because our purpose is to design and implement 3D user interface for modelling plants. In section 2.4 we discuss about solid modelling, because it is widely used in the industry, and provides means to create parameterized geometries. Since visualization of the simulation is another major goal, in section 2.5 we discuss about that. In section 2.6 we present ontologies, and discuss how they are beneficial in combining different data models together. Then, in section 2.7, we discuss ontologies and their usage in graphics representation and linking graphics to domain and simulation models.

### 2.1. Process Simulation

Before presenting process simulation, we start by defining process industry and simulation.

ISO's STEP (Standard for the exchange of product model data) application protocol 227 Functional data and their schematic representation for process plants (AP227) defines processes as activities that operate on process materials. In the discrete manufacturing industry, these process materials are often solids, but in the process industry they are often fluids. We will use definition of AP227 for process and process industry. The process industry includes power plants, pulp and paper mills and for example oil refineries.

Braunschweig and Gani (2002, section 3.3) cited two sources for definition of simulation: "The study of a system or its parts by manipulation of its mathematical presentation or its physical model", and "The imitation of a physical process or object by a program that causes a computer to respond mathematically to data and changing conditions as though it were the process or object itself". From these sentences, we can conclude that simulation is the solution of a model of a system with a computer, and that modelling means construction of a mathematical representation of a system.

Process simulators are usually either steady state or dynamic. Steady state simulators calculate time independent values for a process, thus for one set of input values simulator calculates always the same output. Steady-state can be described as a condition, where the properties of a system at each point are constant in time (Braunschweig and Gani 2002, section 3.3.1.2). This means that the process is first configured for the simulation, then the simulator is run once, and finally the interpretation of calculated values can begin. Typically they are used in the early

design phases to calculate mass and energy balances, so that designer can verify that his model works as expected. One example of steady-state simulators is Balas (Balas, 2007), that focuses on pulp and paper industry.

Dynamic simulators calculate time dependent values for a process, and typically the values vary over time. Use cases for dynamic simulation are different from steady state simulations. While steady state simulation can provide basic operation conditions for a plant, dynamic simulation can provide further information on how the plant behaves under operational mode change, or how to keep it under certain operation mode (Braunschweig and Gani 2002, sections 2.1.5 and 3.3.1.2). Dynamic simulation can be used for instance for training of operators (training simulator), and for testing automation systems. One example of dynamic simulators is Apros (Apros, 2007), which has been used in multiple different cases, including optimizing pulp mills and training simulator of nuclear power plant operators.

## **2.2. Plant modelling**

This section represents traditional design flow of an industrial plant, position of 3D modelling in it, benefits of 3D plant and pipe modelling versus traditional 2D CAD drawings, and last one example application used widely in industry.

### **2.2.1. Plant Design**

Typical plant design process consists of three steps: conceptual design, process design, and detailed design. These steps are sequential, and information gained in the previous step is used in the next step.

The design process starts with *conceptual design*. In that phase major process units required to achieve the end result are selected. Basic thermodynamic information is required to determine approximate operating conditions. Conceptual design uses steady-state simulation.

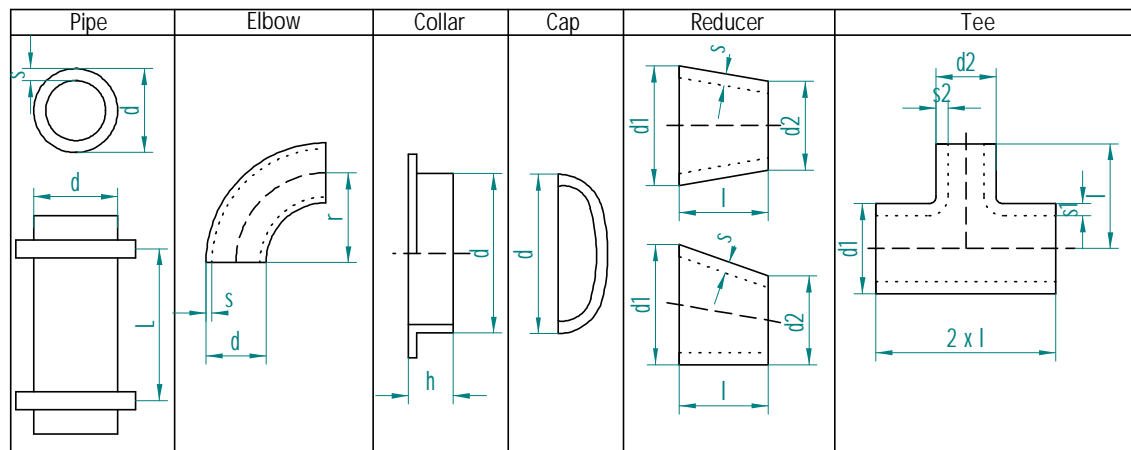
Next step in the design process is *process design*. The major process units are decomposed into smaller groups of units with more equipment and detail. Reasonably precise operation conditions are determined. Mostly steady-state simulation is used, but using dynamic simulation can provide advantages in designing the basic control scheme for the plant.

*Detailed design* is done after process design. It completes the equipment specifications and the 3D layout for the plant. Also complete PIDs (Piping and Instrumentation Diagram) and con-

struction diagrams are made. Simulation can be used for operator training, plant commissioning and online applications (Braunschweig and Gani 2002, sections 3.3.2 and 2.3).

### 2.2.2. Pipe Classes

Piping includes components like pipe, elbows, flanges, fittings, bolting, gaskets, valves, and pressure containing portions of other components. It also includes pipe hangers, supports and other items necessary to prevent overpressurization and overstressing of pressure containing components.



**Figure 1: Typical piping components.**

Currently used standards to describe size of piping are *diameter nominal* (DN) and *nominal pipe size* (NPS). DN is given in units of metric system, and it is developed by ISO. NPS uses inches and it is developed by ASME (American Society of Mechanical Engineers). DN or NPS does not directly translate to outer diameter of pipe. This comes from previous ways to describe pipe's size with inner diameter, but when wall thickness started to vary because of required pressure tolerances, acid- and corrosion resistances, and used materials, a measurement for describing both pipe diameter and wall thickness was needed. Real outer diameter and wall thickness for pipes are described in pipe class standards. ASME and ISO have their own standards; different countries may have their own standards, for example Finland has own pipe class specifications by Finnish Standard Association SFS. Typical components used in piping and some measurement that are specified for each pipe size and each pipe type are in Figure 1 (Nayaar 2000, SFS 123, 2000).

### 2.2.3. Benefits of 3D modelling

Piping Handbook (Nayyar 2000, section B.3) mentions several benefits of CAD applied to a piping system. One of the benefits is *interactive design*. When CAD-system allows interactive routing and modification of piping, it improves productivity of the designing. If the 3D-CAD system does not support interactive design, another step in the modelling process is necessary, because 3D model must be created according 2D drawings. Not just any interactivity will suffice: The CAD system must be designed for 3D pipe design to gain productivity. Another great benefit of CAD is *interference checking*. Using it on 3D-model is more practical than using 2D-drawings. It can be automated, and designer can fix the errors when the interference checking finds them. Checking hard interferences (metal-to-metal) is not enough; also personnel access equipment and for maintenance should be checked.

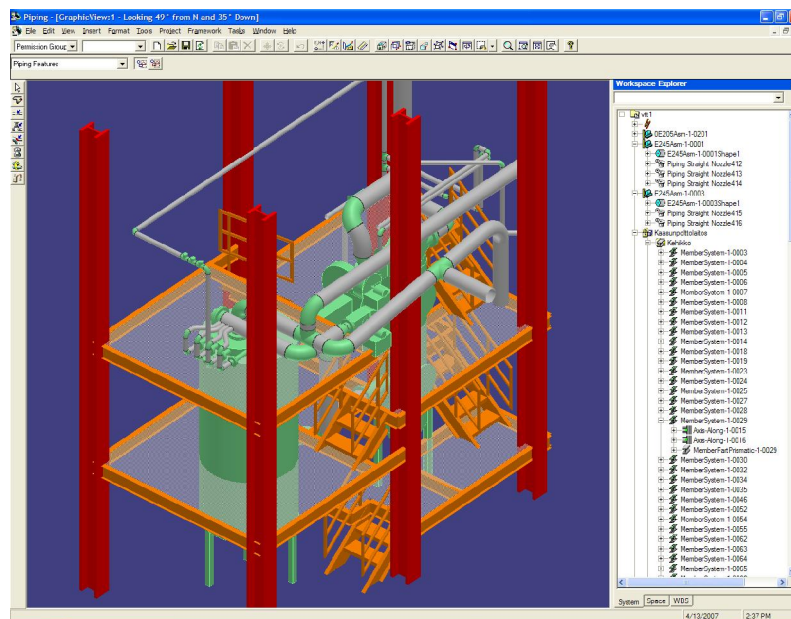
Benefits related to construction of a plant are *drawing generation* and *billing system*. Drawing generation is needed, because isometers and orthographic drawings are required in various design- and construction stages. The 3D CAD-system should be able to generate those drawings automatically or semi-automatically. An automatic billing system that keeps track of prices of used components is of great benefit, when the price of construction is to be kept as low as possible. When combined with interactive design, the modeller may test various designs and select the cheapest one.

The 3D CAD application that is used for pipe modelling has requirements. One critical part is training and implementation, since 3D-modeling is generally more complicated than 2D CAD processing. Therefore the needed expertise should be available to use the full potential of 3D piping software, and to train others to use the software. Using 3D-modelling also changes the design process. 2D drawings are not available until the 3D-model has been created and the 3D-model must be used for design review, including walkthrough views of the model. Software interface to other system is needed, because other stages of design- and construction process need the 3D-model and one must be able to export it to other systems.

### 2.2.4. An example of plant design software

SmartPlant3D is Intergraph's latest 3-dimensional plant modelling software (Intergraph, 2007). The application is database driven: One database contains all the information of a plant. It has been designed to work in multi-user environment.

The software contains viewpoints (tasks) that are used for modelling certain aspect of a plant. For example the *Piping* task is used for routing and modifying piping and inserting, removing and modifying inline components, the *Systems and Specifications* is used for managing structure of plant and specifications for parts that are used in the model, and the *Drawings and Reports* task is used for creating isometers, snapshots and various reports, for example a part list. The user interface updates according to the selected task. For instance, in Figure 2, SmartPlant3D is set to piping task mode and it shows tools that are used for routing pipe, tree view of the structure of the plant, and 3D view, but in the systems and specifications task, UI does not have the 3D view at all.



**Figure 2: User Interface of SmartPlant3D. All equipment and components are listed in tree view on the right, and available tools for modelling for current task are on the left.**

The reason for the strict division between different tasks is access rights. Each designer has his own area of responsibility and is not allowed to perform changes in other parts. For example the designer who is responsible for routing pipes for a certain area of the plant may not change piping specifications, or add new pipeline systems or pipelines to the plant. The lead designer adds necessary pipeline systems and pipelines, and only pipe routing is done by other designers.

Since designed plants are large, the designer must have a way to select what he can see and what is relevant to his current task. Too many visible components just complicate the design process and slow down the program. For this SmartPlant3D uses filters. Filters can filter plant components by several criteria, like system, permission group, spatial (volume) and object type. The user can save and load filters and so he can quickly change what he sees.

All equipment and components in SmartPlant3D are stored in a catalog where all models are reusable. The user has the capability to model new equipment, which cannot be parameterised. Parameterized geometry is created with Visual Basic by coding.

To ease modelling SmartPlant3D employs context dependent aids called SmartSketch. SmartSketch is used in every modelling aspect to help the designer to select right objects. The idea in SmartSketch is simple: The mouse cursor changes, when something relevant to current modelling action is below it. This is used in tasks like routing pipes along axes, placing objects parallel or perpendicular to each other, finding divisors in pipes and structural components.

SmartPlant3D has two ways to check interferences between objects: server-side and client-side. The client application does client-side checking and provides instant feedback, if the designed objects interfere with other objects. Client-side interference checking has one major setback: It only checks new and modified objects against visible objects. This means that if the user does not have the whole plant visible, which is quite uncommon, then possible interferences will not be found. Server-side interference checking is a constant process that runs on the database server, or dedicated server. Its task is to find interferences on the whole plant level and to keep a list of interfering objects, so that designers can solve problems. Interference checking is divided into two parts because doing plant level interference checking on client application would be too computationally intensive.

### **2.3. 3D Interaction**

Traditional ways to interact with computer are WIMP (Windows, Icons, Menus & Pointing devices) interfaces. When interacting with 3D environment, these concepts do not work all the time, but in typical CAD and CAM programs, it still is the prominent way to create user interface.

3D Interaction can be divided into three distinct cases (Hand 1997): object manipulation, viewpoint manipulation and application control. Also, there are two distinct phases in the development of 3D interaction techniques: 2D mouse and interfaces based on true 3D input devices. Since 2D mouse is still the dominant interaction technique in current engineering applications, in the next three sections we focus on describing it and how recent research has handled mouse and keyboard interaction in 3D.

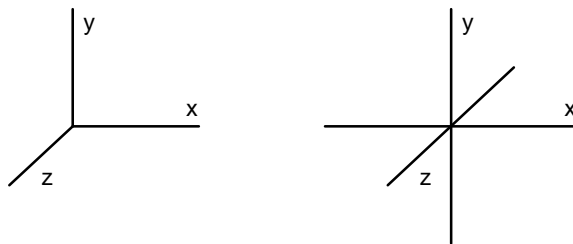
Application Control describes communication between user and system that is not part of the 3D environment. Hand (1997) finds this topic to be the least popular research topic of the three interaction types. Here 2D interfaces are usually sufficient. Therefore the problem is more related to full 3D interfaces, where user gives input using 3-DOF or 6-DOF devices like data gloves. Hence, the problem is not relevant for our topic and is not discussed here.

### 2.3.1. Object Manipulation

Object manipulation is defined as translating, rotating, creating, deleting, modifying, etc. of objects in a 3D environment. Some of these actions are comparable to real-world cases, and others, like deleting objects, are impossible to achieve.

Eric Bier (1986) used Skitters and Jacks to describe transformations. The Skitter is a cursor that shows positive unit vectors for each axis in 3D (Figure 3). The direction of axes defines an orientation and the origin of axes describes a point in 3D. The Jack is similar, but with axes extending to both negative and positive directions.

The basic principle of their use is that a user can place the skitter to surfaces of objects, centre points of objects, and in free space. Then he can add a jack to the position of the skitter. Moving of the skitter has four modes: In the first mode, the skitter is placed on the front face of an object, directly underneath mouse cursor. In the second mode, the skitter is placed on a back face of the object. In the third mode, the user can point centre positions of objects, and in fourth mode, the skitter moves along the three major axes relative to the selected jack, or in any of the three planes of the jack. Like with faces, the skitter is placed underneath the mouse cursor if it is possible, else to nearest point of it.

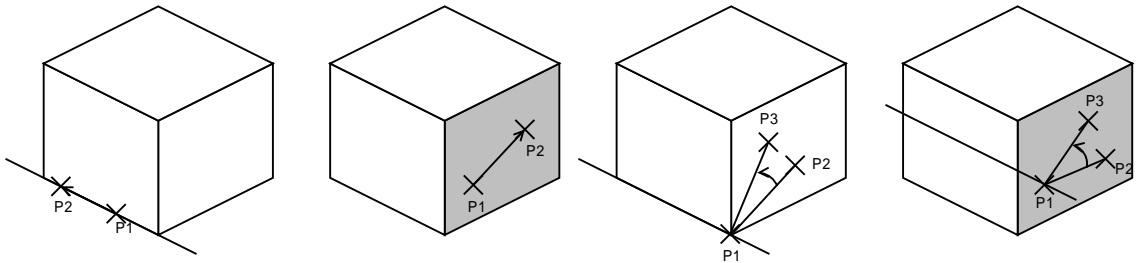


**Figure 3: Skitter (left) and Jack (right). The skitter is used as a cursor in 3D, while the jack is used for marking positions and orientations (Bier 1986).**

After the user has positioned the jack, it and skitters can be used for translating and rotating objects. Bier described several ways to do both. These actions were divided into discrete and smooth motion transformations: discrete actions move and rotate objects instantly to position

that the action describes, and smooth motion actions changes transformation interactively. Discrete actions included moving object along jack's axis until its coordinate value on that axis is equal to the other selected jack's, and moving an object to a position where a jack on its surface is coincident with an other jack. In practice each object's centre position could be used as a jack, and moving objects relative to each other without placing jacks first was possible. Bier's implementation included only smooth translation of objects, either on a plane parallel to the view plane or on a plane of a jack, but most discrete actions could be transformed into smooth actions.

Nielson and Olsen (1986) used similar cursors; the skitter was called the *triad cursor* and the jack was called the *full space cursor*. They also described how 2D mouse coordinates can be mapped to 3D coordinates relative to the triad cursor, giving ability to translate objects in 3D using the mouse. Therefore the user could control the plane along which he wants to translate objects by clicking on the proper position of the triad cursor instead of separately selecting the plane like in *Skitters and Jacks*. In their paper, rotations use axis and angle representation. When user rotates an object, he first selects an axis by selecting one of its edges or a normal of its face, and after that giving the angle by selecting two points (Figure 4). Likewise the user could input more precise translations, in comparison to triad cursor, by selecting two points on the same edge or on the same face.



**Figure 4: Translation and rotation mechanisms. Translation can be described by giving two points, either on the same edge or on the same face. Likewise rotation can be described by giving two points and edge or three points on the same face. (Nielson 1986)**

*Skitters and Jacks* developed further to *Snap-Dragging* (Bier 1990), which combined several previously introduced ideas of gravity (snapping), alignment objects and interactive transformations. The gravity function was created with a three-dimensional cursor, the snap-dragging skitter, which snapped to points, curves, and surfaces at a certain distance in screen space. The user could select the preferred order of snapping, e.g. try to snap points before edges. To control snapping there were three types of alignment objects: lines, planes, and spheres. The alignment was controlled using alignment values that were selected from menu. The selection of a value created one or more alignment objects. When alignment objects were added, intersection points

and intersection curves between objects were calculated and both alignment objects and their intersections could be used for snapping. To make the system more usable, in addition to manually added alignment values, the intersection could be calculated from objects in the 3D environment. For example vertices could be used as centre points of alignment spheres and distances between vertices as radii. Moving of objects was implemented so that if the cursor was not close enough to any of alignment objects, then the object moved along the default plane, which was parallel to the screen. The system allowed the user to move objects into contact with each other, to rotate objects so that their edges were coincident, etc. using the mouse as the only interaction device.

While the previously mentioned techniques can also handle the orientation of objects, there has been some research done on orienting objects in 3D. The most recent evaluation of orientation techniques using the mouse (Henriksen et al. 2004) describes three different ways: Chen et al. (1988) originally tested multiple ways to control orientation. One of them was a virtual trackball. Ken Shoemake (1992) created an arcball, and Bell created virtual trackball. Henriksen et al. show that Shoemake's and Bell's methods are variations of Chen's design. The only major change is mapping between 2D-mouse coordinates to object's orientation. In technical way, Shoemake's approach avoids hysteresis, where "closed loops of mouse motion may not produce the closed loops of rotation". When the mouse is moved back to its original position, the orientation is not the same as in the beginning. Hinckley et al. (1997) compared Chen's virtual trackball and Shoemake's arcball, finding no distinguishing features between them: time used in test cases and accuracy was nearly same and some of tested user did not even notice difference between them.

Phillips and Badler (1988) created a user interface, called Jack, for manipulating articulated figures. Translating of objects was done by mapping each of the three mouse buttons to one of the global axes. Holding down one button translated the object along an axis, and holding down two buttons translated along a plane. Holding down all three mouse buttons had no effect, because the system was not able to map 2D mouse coordinates to 3D position. The actual position was calculated using 2D mouse coordinate to cast a ray to 3D and calculate intersection between the ray and axis direction or plane when two buttons were hold down. When the user translated an object along axis, its position was set to the closest position along translation axis and mouse ray, which allowed moving mouse cursor everywhere on screen. When the user translated an object along a plane, the object was positioned exactly under the mouse cursor. Rotation was handled in the same way: Buttons were mapped to rotation axes and the rotation angle was calculated using intersection point between mouse ray and plane defined by reference point and origin of the rotated figure. In both cases the user was shown the axis on which he was

translating, or the direction about which he was rotating object using arrow symbols. The method using mouse ray is similar as in Bier's skitters and jacks, where the position of the skitter was placed underneath the mouse cursor when it was possible.

While these actions provided a flexible way to transform objects, other means were needed to input precise transformation. Jack implemented two different ways to achieve this. The first method was to input transformation values using keyboard. The second method was snapping translation to selected face, edge or vertex and rotation to selected face or edge.

Stephanie Houde (1992) performed usability tests on hand-style cursors and bounding boxes with handles. She found that there were many different preferred ways to move objects from place to place (sliding vs. lifting) and rotating objects. Users had expectations on how objects should move, based on identity of the object. In Houde's case users were interacting with furniture, and expected that the behaviour of chair is different from pictures since chairs lie on floor and pictures are hanging on wall. According to her Bier's Skitters and Jacks and virtual sphere by Chen are not sufficient for space planning systems where smooth switching between sliding, lifting, and turning actions is desired.

In order to achieve rapid changes in actions, Houde tested several different concepts. In the first case furniture had active areas where certain actions could be activated. For example, clicking on a standard lamp's neck would activate lifting so that the lamp could be moved, but if the user clicked the top of the lamp rotation would be activated. While the system allowed rapid changes in actions, users had difficulties to select the action desired. Active areas were not self explanatory, and different users expected them to be in different positions.

Next, she changed the interface to show active areas when piece of furniture was selected by showing hand icons as narrative handles for each action over active areas. The user could select the desired action by clicking the appropriate handle. This interface had problems with intuitive location of handles: For some of the furniture it was too difficult to decide where handles should be located, and still users had different expectations where they should be.

The next method that Houde tested was attaching handles to bounding boxes of furniture. When a piece of furniture was selected, a bounding box with handles appeared. Previously, the user could select desired actions using handles. The handle for lifting was on top of the bounding box and handles for rotation were in the box's bottom corners. Sliding operation did not have handles, since users preferred to click and drag the object itself, most likely because 2D-interfaces behave similarly. For better usability, only handles that are possible actions to se-

lected object appeared. For example, a picture on the wall cannot be rotated, so rotation handles are not visible. The system also restricted rotation capabilities of other furniture, since in space planning system it is unacceptable to tip them over.

### 2.3.2. Viewpoint Handling

Viewpoint manipulation includes movement in 3D, zooming, and changing field of view. It is especially critical in modelling applications, because the user has to see what he is modelling (Phillips et al. 1992).

Mackinlay, Card & Robertson (1990) classified four different viewpoint movement types:

- General Movement: Exploratory Movement, such as walking through a simulation of an architectural design
- Targeted Movement: Movement with respect to a specified target, such as moving in to examine detail of an engineering model.
- Specific coordinate movement: Movement to a precise position and orientation, such as to a specific viewing position relative to a molecule or a CAD solid model.
- Specific trajectory movement: Movement along a position and orientation trajectory, such as a cinematographic camera movement.

The Jack system (Phillips and Badler 1988) has simple set of routines for viewpoint manipulation: *sweeping*, *panning* and *zooming*. Sweeping is an action, where the user can rotate the viewpoint around a selected reference point while keeping viewpoint focused on it. Panning is also rotation, but when the view is rotated, its position is kept the same and the view direction changes. The zoom action translates viewpoint along its line of sight towards the focus point, and it could be used simultaneously with sweep.

Mackinlay et al. (1990) developed a viewpoint manipulation technique that was capable of moving large distances quickly and short distances accurately. The key idea was that user has selected a point of interest (POI) and viewpoint is moved relative to it. Movement speed depends logarithmically on the distance to target, being slower when closer to target and vice versa.

Controlling of the viewpoint required two different actions from the user. Using the mouse to select and update the POI, and using the keyboard to indicate movement direction, forward or backward. Viewpoint is moved towards or away from the POI. Another way to move the viewpoint was called orienting POI movement, where the viewpoint moves towards the object's sur-

face normal at the POI and simultaneously rotates the viewpoint to face the POI. When the viewpoint is manipulated this way, distance to POI does not change.

They added more features to improve usability by locking the POI to selected object when viewpoint was moved, which prevented the user from accidentally to select wrong POI. Another improvement was hovering over the POI by pressing both forward and backward keys down at the same time. This allowed inspection of object's surfaces, similarly to scrolling in 2D. While this method was considered a good way to manipulate the viewpoint for targeted viewpoint movements, usability degenerates for movements where there is no appropriate object to anchor the movement.

Tan, Robertson, and Czerwinski (2001) combined traditional flying with a method called orbiting. Orbiting is similar to POI movement, but instead of using the intersection point of mouse ray and the objects surface, the selected object's centre was used as the POI. The user could rotate around object and zoom towards it by dragging. He could select between flying and orbiting: if dragging started over an object, then orbiting movement was chosen. Likewise if nothing was under mouse cursor, flying mode was chosen.

To increase usability, automatic viewpoint handling is possible. Phillips, Badler, and Granieri (1992) used it to aid modelling. Automatic viewpoint handling was used with object manipulation and when it was used, the viewpoint was turned to a better position, either because translation axis was too close to view's direction or other object was occluding the modified object. They also noted that some viewpoint properties are better modified by the user only. One example is zooming factor, because it is related to user's preferences. Other cases where automatic viewpoint handling must be turned off is when the user has positioned viewpoint parallel to coordinate axis for 2D view of the model.

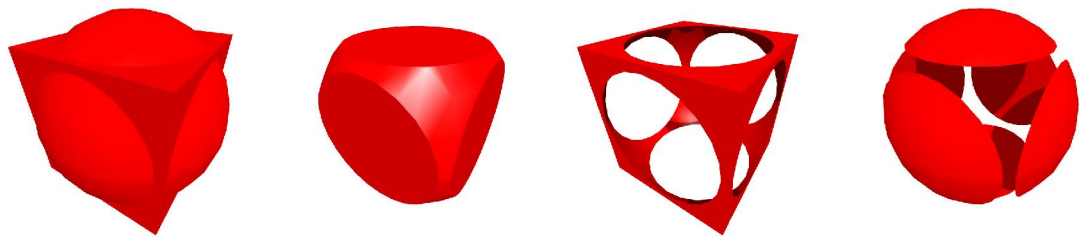
## **2.4. Solid modelling**

Solid modelling is widely used in the industry, because it provides precise representation of geometry and it is basis for more advanced modelling techniques: feature-based and parametric modelling. Our interest to the topic is twofold; parametric modelling provides better reusability of geometric models of equipment, and existing geometric models are commonly in some solid modelling format.

#### 2.4.1. Data representation

Currently there are two major representation schemes in solid modelling. Constructive solid geometry (CSG) defines solid objects as set-theoretic Boolean expressions of primitive solid objects. Both the surface and the interior of an object are defined, albeit implicitly. A boundary representation (B-rep) describes only the oriented surface of a solid as a composition of vertices, edges and faces. The orientation can be used for deciding which side of surface is towards the solid's interior, if the object is a bounded volume (Hoffman 1994). CSG and B-rep have different strengths and weaknesses. A solid constructed using CSG is always valid; its surface is always closed and orientable and encloses volume. B-rep model may not be closed and therefore not a solid, but B-rep enables a less restricted modelling scheme. Therefore both representations are often combined into single dual-representation modellers.

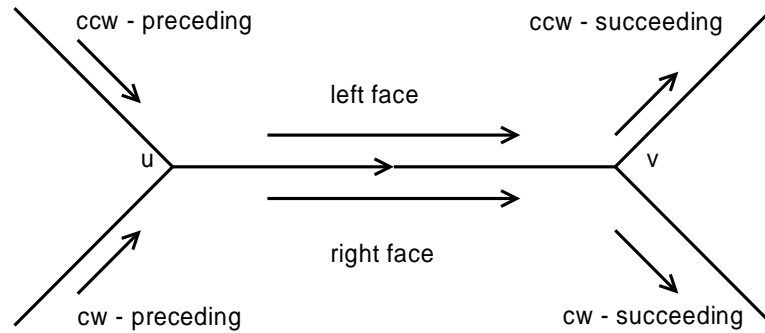
CSG is a method of representation, a design methodology and a certain standard set of primitives. A CSG object is built from the standard primitives by composing them using regularized Boolean operations: union, difference, and intersection. Regularized Boolean operations differ from set-theoretic ones in that the result is the closure of the operation the interior of the two solids, and unwanted lower-dimension structures are removed with them. Union operation is used for adding shapes together, intersection is used for subtracting shape from another shape, and difference takes the common part of the shapes (Figure 5). Geometric model's structure is then represented as a tree where leaves are primitives and interior nodes are Boolean operations.



**Figure 5: CSG Modelling: Union, intersection and difference of box and sphere. Difference of two objects can be done two ways: in third picture from left sphere is subtracted from box and in fourth picture box is subtracted from sphere.**

Boundary representation can describe a solid unambiguously. Surfaces must be oriented so that in each point of a surface it is possible to tell on which side is solid's interior. The description contains two parts: a topological description of the connectivity and orientation of vertices, edges and faces, and a geometric description for embedding these surface elements in space. Historically the representation evolved from a description of polyhedron.

The oldest formalized schema for representing the boundary of polyhedron and its topology appears to be the winged-edge representation. In it, each face is bounded by a set of disjoint edges, one cycle as faces outside boundary and other bounding holes in the face. Each vertex is adjacent to a circularly ordered set of edges. For each edge, incident vertices, left and right adjacent face, preceding and succeeding edge for clockwise- and counter clockwise order is stored (Figure 6). This enables traversing edges of a face in clockwise and counter clockwise order. Order of the edges is used for calculating which side of a face is outside and which side is inside of the solid.



**Figure 6: Winged-Edge representation of edge from vertex u to vertex v. The representation stores preceding and succeeding edge in clockwise and counter clockwise order.**

Geometric information in B-rep may contain coordinates of vertices and plane equations of faces: there are multiple representations that store information differently. Each face must be defined so that its normal points out from the solid. Geometric information may also contain equation of edges or those can be calculated from intersection of faces. Typical face defining representations are Bezier surfaces and rational B-splines.

#### 2.4.2. Parametric Modelling

Parametric modelling is used when several similar shapes are needed. It saves time when only one geometric shape with parameterization needs to be constructed instead of multiple geometric shapes. According to Anderl & Mendgen (1995), parametric design can be used in several different cases:

*Modelling variants of parts and assemblies* is a method for creating geometric and topological variants of a model. The shape of a part is modelled as a combination of features that are described by geometric parameters for its shape, position and orientation with respect to other features of the part. This way the parameters form a network and modifying a value of a parameter changes shapes of features of a part creating a new variation of the original part.

*Modelling the history of features* is a way to create topological variants of original geometry. Variants are formed by changing the number of features according to parameters. For example, a hole in a part may be removed if the part is small, but the same part may contain several holes when it is large.

*Modelling of catalog parts* is somewhat different from previous cases but can be achieved by those techniques. Catalog parts are modelled as geometric and/or topological variants where only predefined sets of dimensions are applicable. Usually catalog parts are used in engineering specifications, where for example a screw may have only certain set of sizes and its length depends on its size.

Geometric parameters of a feature with respect to other features are usually achieved by describing geometric relationships of shapes, e.g. parallelism. When a constraint is added to a geometric model, it is enforced by modelling application. Whenever user changes something in the model, the application tries to calculate properties of geometry according to all constraints in the model. These geometric relationships are called parametric constraints.

Parametric constraints can be applied in two ways: either the user explicitly defines constraints (constructive approach) or they are applied automatically (using a rule base). In the constructive approach, the user has to explicitly specify the usage of constraints, for instance “draw line parallel to”. In the automatic approach, rules of sketching system are detecting designer’s intentions. To ease the modelling effort, most systems can use both techniques, using first automatic detection and then letting the user to modify and add new constraints to the model.

Anderl & Mendgen (1995) presented several rules to detect constraints in sketching system:

1. Lines that are sketched approximately horizontal or vertical are considered to be intended as exactly horizontal or vertical
2. Lines that are sketched approximately parallel or perpendicular are considered to be intended as exactly parallel or perpendicular.
3. Elements that are sketched approximately tangent to arcs or circles are considered to be intended as exactly tangent to these arcs or circles.
4. Elements that are sketched approximately symmetrical about a centreline are considered to be intended as exactly symmetrical about this centreline.
5. Elements that are sketched approximately collinear are considered to be intended as exactly collinear.

6. Points that are sketched approximately lying on the other elements are considered to be intended as exactly lying on these elements.
7. Centre points that are sketched approximately lying on the same vertical or horizontal are considered to be intended as exactly lying on the same vertical or horizontal.
8. Elements of unknown length are assigned a length equal to that of a known element in the sketch of approximately the same length.
9. Arc and circles sketched approximately with the same diameter or radius are assigned exactly the same diameter or radius.
10. If a feature has the classification shaft and material strength  $R$  and torque  $T$  are assigned, then the dimensioning rule for the diameter  $D=f(R,T)$  must be applied.

These rules are just an example of a set of rules that can be used for interpreting the intentions of a designer. In practice if the sketch is not close to the intended shape, the rules will fail. Either the assumed constraints are not applied or constraints that designer does not want to apply are added. In these cases he is forced to reapply the constraints in a different way in order to achieve the intended result (Hoffmann 1994). The solution of Anderl & Mendgen (1995) to this was to allow designer to adjust sensitivity of sketcher. This can be problematic too, since if sensitivity is set to too strict, the sketch must be very close to intended shape or constraint detection will fail. On the other hand, setting sensitivity too relaxed may cause the system to detect too many unintended constraints. Other way to look at geometric rules (in the above example rules 1 – 7) is that all are related to the snapping feature used in interaction systems. The difference is that instead of applying constraint rule automatically, explicit user confirmation may be requested (Zelevnik et al. 1993).

Another thing that contributes to constraint detection is the number of objects in a sketch. If the sketch is large, it is practically inevitable that some of the rules apply to the currently sketched feature. Several solutions have been proposed. One solution is to use pixel distance: constraint rules are only checked against other objects that are  $n$  pixels way from current object. This of course means that constraint detection would depend on current display settings and zooming factor (Anderl & Mendgen 1995).

#### 2.4.3. Initial Graphics Exchange Specification

IGES is the first standard used for transferring both 2D- and 3D-CAD models between CAD software. It started in 1979, when mechanical CAD systems were less than 10 years old and only few products had significant market penetration. Even then users were disappointed of the fact that it was not possible to transfer CAD models from software to another (Kemmerer 2001).

IGES provides mechanisms to store geometry, graphical data and annotations. It establishes structures for digital representation and communication of product definition data focusing on essential engineering characters of physical objects such as manufactured products. IGES is supported by all major CAD systems even today, when STEP has been released and has gained popularity among CAD software providers.

First version of IGES was released 1980 and the last released version is 5.3 from 1996. According to the IGES 5.x Preservation Society (2007), the standard is now discontinued.

#### 2.4.4. Standard for the Exchange of Product Model Data (ISO 10303)

Similarly to IGES, the function of STEP is to provide a neutral data exchange format between applications. But instead of focusing just graphics, STEP is aimed to transfer all product related data (Pratt 2001). The STEP Application Handbook says: “The overall objective of STEP is to provide a mechanism that describes a complete and unambiguous product definition throughout the life cycle of a product, independent of any computer system.”

In the STEP standard, the application interface contains two layers: the application domain layer, which is called Application Resource Model (ARM), and the general layer, which is called the STEP Integrated Resources model. A STEP compliant intermediate file or database must be compliant with the Integrated Resources model so that data exchange is domain independent. Application Protocol (AP) specifications contain both domain specific part of the STEP standard and mapping of the domain specific data to the ARM model (Braunschweig and Gani 2002). The current list of Integrated Resources include, for example a representation for B-rep based geometries, materials, and mathematics. Application Protocols include standards for automotive industry, ship building, and the process industry. The list of these specifications is not final: New parts are under development and will be added in the future (Pratt 2001).

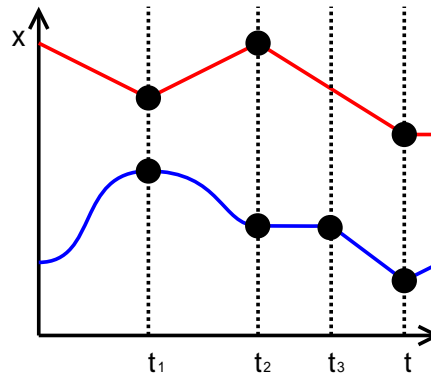
The most widely used part of STEP is the application protocol 203 (Configuration controlled 3D designs of mechanical parts and assemblies). It is used for exchanging of product shape models, assembly structure, and configuration control information, and it can be seen as replacement of IGES. AP203 contains a lot of features that IGES lacks (Pratt 2001).

## 2.5. Data Visualization

Visualization is a critical part of simulation. It helps us to understand a model's behaviour and decreases model building-, verification-, and validation time. The power of visualization comes from the capability of humans to process visual information quickly. It provides a natural way to transfer a lot of information between computer and its user. The simulation of manufacturing plants benefits from visualization techniques greatly, because the models tend to be complex and counter-intuitive (Rohrer 2000). We focus on parts of data visualization that are relevant to our topic: visualizing equipment's behaviour, dynamics of piping, and large CAD models.

### 2.5.1. Animation techniques

The impression of movement in animations is created with still pictures. Small changes between each picture and rapid change from picture to another create an illusion of moving objects. A basic technique in computer animation is keyframe animation. Its principle is that the user defines keyframes and computer calculates in-between frames by interpolating between them. A keyframe is defined by its particular moment in the animation timeline and by all the parameters associated with it. Since the keyframe technique is flexible, practically anything can be animated with it, ranging from position of geometric models to surface parameters (Kerlow 2004 chapter 11, Parent 2002 section 3.5).



**Figure 7: Interpolators are used with keyframe animations. Time is in horizontal axis and value in vertical axis tells interpolation value for particular time.**

Interpolators are simple but powerful way to express and control between keyframes and how attributes change. A common representation of interpolators is 2D graphs, where time is in horizontal axis and interpolator value in vertical axis (Figure 7). The slope represents the speed of change; path with steep angle changes interpolated value rapidly, while flat path keeps the value constant. Normal workflow with keyframe based animations is that a user modifies animated

model and the animation software automatically generates interpolators. After the animation is done, the user can fine tune it by editing interpolators in 2D graph representation.

Typical interpolation techniques are linear and curved interpolation. Linear interpolation is the simplest mathematically, but creating smooth, natural movements with it is hard, because speed of change changes instantly on keyframes. Curved interpolators are more sophisticated than linear interpolation. They use multiple keyframes for interpolating and depending on their implementation, give adjustable control parameters for each keyframe. They also avoid the discontinuity problem of the linear interpolation. Typically different interpolators are used with different cases; while 1D interpolators can be used for interpolating almost anything, but there are also cases where they will not suffice. For example, when interpolating 3D orientation, Slerp (Spherical Linear Interpolation) can produce better results (Kerlow 2004 chapter 11).

### 2.5.2. Flow visualization

Glyphs have proven useful in depicting spatially complex, multivariate data, because their graphical properties, such as colour and size, can be bound to visualized data. They can be seen in weather forecasts in TV, where arrows indicate wind direction, and their colour wind temperature. The power of glyphs is that they are intuitive and can present a large amount of spatial information in a compact form. One of problems with glyphs, like arrows, is that in 3D their direction is ambiguous and their proper positioning is difficult (Rosenblum et al. 1994, chapters 7.4, 23, 26).

Another technique to visualize flows is particle animation. While probably being most realistic visualization of flow, since flow velocity can directly be mapped to particle velocity, it requires constant updates of positions of particles, which can be computationally demanding (Rosenblum et al. 1994, chapter 23; Hansen & Johnson 2005, section 12.6).

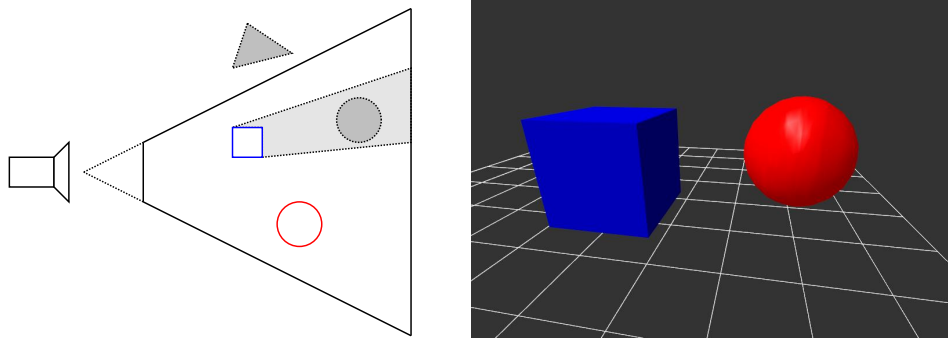
There are other methods, such as streamlines, streak lines, path lines, time lines, stream ribbons and stream surfaces, to visualize 2D and 3D flows (Hansen & Johnson 2005, section 12.6), but they are not usable in visualizing the flows in pipes that are essentially multiple 1D flows in 3D space.

### 2.5.3. Visualization of large, complex 3D models

Several variables affect rendering speed. One of them is amount of triangles that computer has to render. Generally, with fewer triangles to render the rendering gets faster. CAD models of

large industrial plants contain millions of triangles. In order to visualize them in real-time, special techniques must be applied to reduce amount of triangles rendered each frame. Visibility culling can prevent rendering of non-visible objects, and geometric simplification can reduce the amount of triangles used for rendering visible objects.

In 3D graphics, scene is rendered from a certain viewpoint. Typically the rendering can be perspective projection or parallel projection of the 3D scene to a 2D display device. When perspective projection is used, a view forms a frustum in 3D space, which contains all visible geometry that is rendered to final image. In parallel projection the frustum is rectangular. An example of a view frustum is in Figure 8, where there are four objects in the scene. One of the objects is out of the view frustum. In the same figure grey ball is behind blue box and therefore it is not visible. Grey ball is then occluded by the blue box.



**Figure 8: View frustum: The grey triangle is not drawn because it is out of view frustum. The grey ball is not drawn because it is occluded by the blue box.**

View frustum culling rejects objects that are not inside the view frustum from rendering. Typical methods use bounding boxes, or spheres, and test them against view frustum. This technique is very simple, and it does not prevent rendering of objects that are behind other objects, and so do not contribute to final image. Algorithms which do that are called occlusion culling algorithms. Two types of algorithms are used: from-point and from-region algorithms. From-region algorithms typically calculate potentially visible set (PVS) as a pre-processing step, and use it in rendering time. For general scenes, PVSs are hard to compute and trade fast rendering time calculation for larger space requirements and more inefficient occlusion culling. From-point algorithms are run-time algorithms that compute potentially visible set at rendering time. Recent advancements in graphics hardware have made from-point based algorithms feasible. Bittner et al. describe an algorithm, coherent hierarchical culling, which uses hardware occlusion queries to perform visibility tests in a hardware friendly way by exploiting temporal coherence (Bittner et al. 2004).

Reducing complexity rendered geometry reduces the amount of time spend on rendering. There exists several different algorithms that may have special requirements for visualized data, and so can be used in limited cases. For instance many algorithms require pre-processing of the data, and can be used only with static data.

Polygon reduction algorithms can be divided by how they compose the whole scene into one hierarchy, and how they process it. The typical classification is Discrete Level of Detail (LOD), Continuous LOD, View dependent LOD, and Hierarchical LOD. Discrete LOD is the simplest hierarchy: each object contains several representations of its geometry, each with smaller geometric complexity. Continuous LOD is similar, but it contains so many different representations of the original object, that it can be said to be continuous. View dependent LOD is more complex, but allows detail adjustments for parts of the model individually depending on viewing distance and direction. Hierarchical LOD can combine whole scene into a tree structure, and therefore it can do level of detail selection for multiple objects at the same time. Plain discrete and continuous LOD techniques must test each object individually. (Hansen & Johnson 2005, section 8.4)

Geometric simplification of a model can be approached different ways. The simplest method that can be used with discrete LOD is that geometry is modelled with several different detail levels and it becomes a problem of content production. This gives absolute control of quality of all detail levels but burdens the modelling process. With continuous LOD and view dependent LOD, the representation must be calculated at runtime. Common geometric simplification of meshes are based on edge collapsing, vertex removal, or other similar techniques that change geometry of a mesh removing parts from it based on some error metrics, e.g. curvature, distance between vertices, and so on. The same approaches can be used with the discrete approach, but instead of calculating representation runtime, detail levels are calculated and stored as an pre-processing step. Another approach to simplification is to use image-based representations and impostors. They differ from previously described methods in that they do not create different resolution representations of the geometry, but capture the original representation to an image. Common image-based representations include point primitives, flat images (impostors), textured depth meshes and depth images.

All of these methods have something in common: selection of detail of the rendered geometry. Level of detail can be quality driven or performance driven. The principle of quality driven level of detail selection is that it guarantees quality of rendered result: The detail selection can be based on geometric pixel error estimates, preservation of silhouettes and preservation of texture maps. With performance driven selection, level of detail depends on frame rate estimates or pa-

rameters directly dependent on it, for instance, the amount of rendered triangles is kept constant. Therefore it tries to keep the frame rate of rendering constant, rendering with a quality that system is capable of handling. Performance driven management can be handled as reactive algorithm where selected rendering quality is based on time spent rendering the last frame. This is a setback in the approach since the information used is old, and the selected quality may not be correct for the next frame. The reactive approach is different, since it tries to predict the rendering quality which is needed to meet time requirements. The predictive model is more complex, because it needs knowledge of used hardware, but as a result gives more stable frame rates.

#### 2.5.4. Visualizing manufacturing simulation with animation

Quick, Zhu, Wang, Song and Müller-Wittig (2004) combined discrete simulation systems with animated scenes in virtual environments. Their system included a library, which contained reusable models and animation methods. The system supported automatic generation of virtual environments from process definition data. In most cases the data is not available, and the user has to manually compose the virtual model. The user is also responsible for binding animation properties to simulation properties.

Animations were driven by event based architecture: changes in simulation model launched events. Discrete events have their problems: when the animation system receives an event, the visualized model should be in a state that represents the simulation state. This does not give time for the animation system to react to state changes. For instance, animating the position of an object by interpolation is not possible. When the simulation was run before visualization, this was solved by look-ahead. In cases when simulation is running at the same time as visualization, look-ahead is not possible. The solution was to predict the next state of simulation, and to show the user that animation as a result of prediction, instead of assured information.

### 2.6. Ontology based programming and modelling

In modern world, in current information society, communication is a key issue. Organizations, people, and software systems must communicate with each other. Without proper communication they cannot co-operate, and bad communication leads to misunderstandings and information loss. Because of background and needs of those entities, there are many viewpoints and assumptions regarding what is essentially the same subject of matter. Each uses his own vocabulary and therefore same terms may have different meanings and different terms may have the same meaning. With software systems, this problem was initially solved case-by-case, creating communication and data transfer between two applications. Gradually it was noticed that

using ontologies, a shared taxonomy defining concepts and terminology used in communication, provides advantages over case-by-case conversions (Uschold and Gruninger 1996; Smith 2003).

Term "ontology" has several different meanings. The first confusion comes from the fact that the term "ontology" has different meaning in various contexts. In philosophy it is defined as "the science of being", started with Aristotle's work, and its original meaning of the word can derived from Greek: "ontos" means being, and "logos" meaning both language and reason. In the 19th century, at the beginning of modern philosophy, German philosophers defined ontology as systematic account of existence (Roche 2003).

When the word is used in the context of knowledge engineering, meaning is also ambiguous. Probably the most commonly referred definition for ontology was made by Thomas Gruber (1993): "An ontology is an explicit specification of a conceptualization", but as Guarino and Giaretta (1995) pointed out, there are many other definitions, and here exact meaning cannot be made without knowing what "specification" and "conceptualization" mean. After discussion, they decided that the Gruber's definition for ontology should be "a logical theory which gives an explicit, partial account of conceptualization", where "conceptualization" means "an intensional semantic structure that encodes the implicit rules constraining the structure of a piece of reality". This definition contains two keywords that make all the difference: partial and intensional. Conceptualization is partial, because then the degree and the detail of a specification may change, depending on where, how and why the ontology has been defined and used. The conceptualization itself is intensional semantic structure since the focus is in the meaning of things and extensional structure would make the ontology dependent of state of affairs. For example, with ontology of a table and boxes on the table, intensional definition does not take account order of the boxes, but with extensional definition different orderings of the boxes would be different conceptualizations. In practice, the meaning of the term "ontology" can be stated as shared understanding of some domain of interest (Ushchold and Gruninger 1996).

Ontologies are used in several contexts and for several purposes. Gruber (1995) used ontologies for information sharing. Uschold and Gruninger (1996) used them for communication, interoperability and systems engineering. Uschold and Callahan (2004) listed neutral authoring, ontologies as specification, common access to information, and ontology-based search. According to Guarino (1998), ontologies can be used as database components, user interface components, and application program components, either development time or run-time. Ontologies have become popular several scientific areas including Artificial Intelligence, Computational Linguistics, and Database Theory (Guarino 1998). The purpose of an ontology's existence also

drives how it should be designed, what it should contain, and how it should represent its information.

In the next three sections we look at how ontologies can be used. Section 2.6.4 shows how ontologies can represent information, and the ontologies can be classified by their use. Section 2.6.5 presents some design guides and criteria about ontology design. Sections 2.6.6 and 2.6.7 briefly introduce W3C's standards for semantic web and what kind of concepts they offer for creating ontologies.

#### 2.6.1. Communication

Ontologies can be used for communication in several ways. To create shared understanding, unified concepts and terms among those who must communicate with each other can be brought together and form an ontology for *normative model*. To form a normative model, one must identify all shared ideas and concepts, find exact matches and all important relationships. Using the normative model allows semantic transformation between contexts and provides possibility to use the same algorithms and problem solving techniques in all contexts.

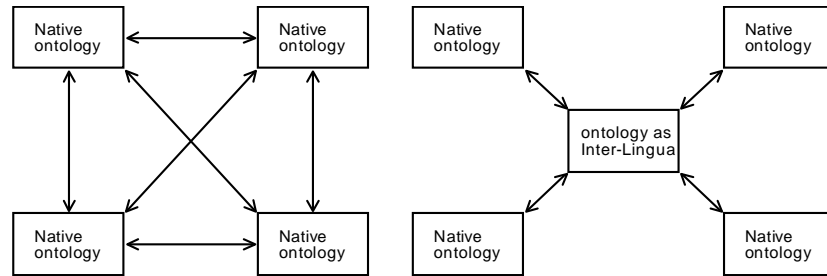
In the same way, ontologies can be used for forming networks of relationship. In cases where using a normative model would create too strict restrictions to different contexts, other mechanisms like agents and rules can be used for creating relationships between different contexts and allow communication between different contexts.

Another important goal of an ontology in respect of communication is to provide unambiguous definitions for terms used in a software system. All used software tools should be capable of supporting those terms by maintaining consistency of the system. User's and tool's ontologies are usually different and in such cases there must be components that can map those ontologies to each other.

Ontologies can also be used for integrating different user perspectives. This use is tightly related to the previous uses for communication; for each user perspective there exists one or more ontologies including concepts and terms for that particular user perspective. It allows each user can use vocabulary that he is accustomed to use for the same information. For example, a plant designer's perspective to a process plant is very different to a maintenance worker's perspective to the same plant, but it is critical that when concepts overlap, both perspectives use the same data.

### 2.6.2. Inter-Operability

In large software systems there is a need for exchanging data between users and tools. Ontologies can be used as a *Inter-Lingua* that supports translations between different languages and presentations requiring only creating translators between a native ontology and interchange ontology instead of creating translator for each native ontology pair (Figure 9) (Uschold and Gruninger 1996).



**Figure 9: Ontology as Inter-Lingua compared to translations done for each pair. Using ontology as Inter-Lingua reduces amount of translations.**

To ensure inter-operability, an interchange ontology must support all the features in native ontologies, and because of that it would be beneficial to use a global standard as an interchange ontology. Even then information loss may not be avoided, because native ontologies may not support the same features, and then all the information that the target ontology does not support will be lost (Uschold & Callahan 2004).

Inter-operability can exist in different dimensions. These dimensions according to Uschold & Gruninger are:

- *Internal inter-operability* is used with systems that are under direct control of the same organizational unit and it has old ontologies because of historical reasons or other legacy systems that cannot be changed.
- *External inter-operability* is used when an organizational unit needs to insulate itself from outside changes.
- *Integrated ontologies among domains* means integration of ontologies of different domains, usually for support some task. For example, an ontology for support simulation model creation will need to integrate ontologies for simulation and diagramming or an other user interface ontology.
- *Integrating ontologies among tools* is integrating different ontologies under the same domain, because of legacy systems that must be able to share information. This is a dif-

difficult case for ontologies since applications and their concepts already exist and cannot be changed.

### 2.6.3. Systems Engineering

Both communication and inter-operability sections dealt with ontologies that are used inside software systems. In systems engineering, ontologies are used for supporting development of software systems.

A shared understanding of the problem assists in specifications of a software system. When an informal approach is used for creating specifications, ontologies make identifying requirements, finding and understanding relationships among components of the system easier. In a formal approach, an ontology provides a declarative specification of a software system and allows to reason the purpose of the system.

Informal ontologies can improve reliability of software systems when they are used for manual checking the design against system's specifications. Formal ontologies can be used for automatic or semi-automatic consistency checking of a software system against its specifications. They can also be used for helping integration of software components because semantic constraints and relationships between different tools can be interpreted.

Ontologies can be used as shared libraries for modelling problems and domains. In order to do that, ontologies should be designed to be reusable and extendible. Problems will arise if an ontology is designed for a certain domain but is applied to another domain where assumptions are not the same. The result is that the application will not behave as expected. In these cases, order to increase reuse, ontologies must be analysed, so that reusable concepts can be found.

### 2.6.4. Information represented by ontologies

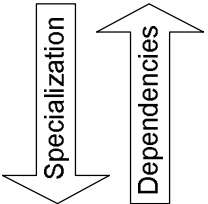
Ontologies can describe information on different levels. Roche (2003) listed four types of knowledge, and examples of them are in Figure 10.

- A *meta-ontology*, or representation ontology, specifies the knowledge representation principles used for defining concepts of domain and generic ontologies e.g. what is a class, a relation, and a function. Examples of meta-ontologies are OWL (Section 2.6.7) and Layer0 of ProConf (section 3.1).
- A *generic ontology*, also called top ontology, specifies general concepts, defined independently of a domain of application, which can be used in different application do-

mains. Time, space, mathematics are examples of general concepts. Because of independence of a domain, these ontologies can be reused (Guarino 1998)

- A *domain ontology* is dedicated to a particular domain and can be used and reused for particular tasks in the domain. Chemical, medicine, and enterprise modelling are domain ontologies. Guarino (1998) also listed *task ontology* to the same information level. It describes generic tasks and activities like diagnosing and selling.
- An *application ontology* gathers knowledge dedicated to a particular task including more specialized knowledge of the experts for the application. In general it is not reusable.

Type of ontology	Example Ontology	Example Concepts
Meta-Ontology	Layer0	Type, Relation
Generic Ontology	Mathematics	Vector, Equation
Domain Ontology	3D Graphics	Shape, Appearance
Application Ontology	3D Industrial plant modelling	Tank, Control Point



**Figure 10: Different types of ontologies, example ontologies and their concepts. Specialization increases from top to down and ontologies are dependent of their predecessors. In the example 3D Industrial plant modelling could be also domain ontology if it does not specify application specific concepts, like here the control point.**

Depending where and what for ontology is used, also its formality may be different. Uschold & Gruninger (1996) explained four categories: highly informal, semi-informal, semi-formal and rigorously formal, where first one is expressed loosely in natural language and last one is: meticulously defined terms with formal semantics, theorems and proofs of such properties as soundness and completeness.

#### 2.6.5. Design process, criteria and evaluation.

Ontology development starts by identifying purpose and scope of ontology. The purpose of the ontology can be derived from where and why it is used. These were listed in three previous sections. Also identifying possible users of the ontology is beneficial.

The next step is building the ontology. This includes two phases: capturing the ontology and coding the ontology. When the ontology is captured, key concepts and relationships in the domain of interest must be identified. Then unambiguous names and definitions for the concepts and the relationships must be found. After that, we have formed a conceptualization. Explicit

representation of the conceptualization is formed in the coding phase. The representation is designed and created on top of chosen meta-ontology, using its ontology modelling concepts and relationships. Integration of existing ontologies must be considered in the previous phases. There is no common rule when the existing ontologies should be integrated and when not. While reusing ontologies improves inter-operability, same effect can be achieved by mapping the designed ontology's concepts to the existing ontologies (Uschold and Gruninger 1996).

The third step in the ontology development is evaluation and the last step is documentation. In the evaluation step the developed ontology is reflected to its requirements and specifications, and so on. Documentation step may not be considered as a separate step at all, but should be practiced within all the previous steps. A good documentation is essential for re-using the ontology and knowledge sharing. Therefore all important assumptions, main concepts, and primitives used for expressing definitions should be written down.

T.R. Gruber (1995) proposed a preliminary set of design criteria:

- **Clarity:** An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective, independent of the context where they are used. Objective definitions can be specified in formal axioms. Therefore formal axioms should be used whenever possible. Documentation written in natural language, including examples of use will aid people to understand the ontology and reduce misconceptions.
- **Coherence:** An ontology should be coherent, it should sanction inferences that are consistent with the definitions. At least the defining axioms should be logically consistent. Coherence should also apply to the concepts that are defined informally, also those that are defined using natural language, including documentation and examples.
- **Extendibility:** An ontology should be designed to anticipate the uses of shared vocabulary. It should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that it does not require changing the existing definitions.
- **Minimal ontological commitment:** An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities. An ontology should make as few claims as possible about the world being modelled, and allow the parties committed to the ontology freedom to specialize and instantiate the ontology as needed. The effect of minimising ontological commitment is twofold: many ontological commitments may limit reusability and extensibility, but too few commitments may cause the ontology to be consistent in cases where it was not intended to be, for example allow creating incorrect models that are correct by the ontology.

- Minimal encoding bias: The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. This means that the conceptualization should be designed without taking account of the chosen representation, because it could influence the design, preventing use of the ontology in different representation systems.

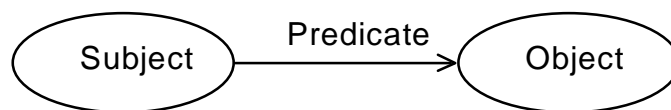
In order to evaluate designed ontologies, there is a need for objective criteria that are founded on the purpose of the design. But the problem is that within ontological engineering there is no classification of problem domains and no characterization of ontologies to evaluate and compare their adequacy or performance. On the other hand, theoretical analysis of ontologies can be done by comparing their axioms to the intended models of designer or user (Gruninger & Lee 2002).

#### 2.6.6. Resource Description Framework

Resource Description Framework (RDF) is W3C's standard for describing resources on the Web. RDF is built upon XML, but increases semantics of information by specifying standard modelling primitives like classes and properties. Therefore RDF provides semantic interoperability between documents, while plain XML provides only syntactic interoperability (Decker et al. 2000).

RDF has several purposes. One of them is to represent metadata about Web resources, such as title, author, and modification date of a Web page. In more general, its purpose is to provide a common framework for expressing information, and the information presented in RDF is intended to be processed by machines, not humans. RDF enables an application to exchange information without loss of meaning, and the information can be used in applications that were not designed to handle the information (Manola and Miller 2004).

The information model in RDF is based on facts. A fact is a statement of a relationship between two things. Facts are presented in RDF documents with a triple model; one triple encodes subject, predicate, and object. The predicate in a triple defines relationship between the subject and the object. The whole information in the document can be seen as a directed graph formed by the triples (Figure 11).



**Figure 11: Triple is formed from subject, predicate, and object.  
Whole information of RDF document is formed by these triples.**

For instance, in natural language sentence “Colour of the car is green.” the car is the subject, colour the predicate, and green the object. As a basic, a triple describes a property of a thing, thing being the subject of the triple, and the property being the predicate, and the value of the property is the object of the triple. From the predicate’s viewpoint the subject is domain of the relation and the object is its range.

RDF is designed to provide a basic object-attribute-value model for information, but it does not make any data-modelling commitments. Basic relationships used in RDF are: class, property, subclass and subproperty. Also domain and range restrictions of relations can be described with RDF.

#### 2.6.7. Web Ontology Language

Web Ontology Language (OWL) is an ontology language built top of RDF, and it is based on earlier ontology languages, most notable DAML+OIL (DARPA Agent Markup Language; Ontology Inference Language). Like RDF also OWL is W3C’s standard. The purpose of OWL is to “use by applications that need to process the content of information instead of just presenting information to humans”. It provides better machine interpretability than XML and RDF by providing richer means to express information, including formal semantics. OWL has three sublanguages, OWL Lite, OWL DL (Description Logics), and OWL Full, which increase expressiveness of the language but at the same time increase formal complexity making interpretation harder for reasoning systems. Therefore it is unlikely that there will be reasoning system for all OWL Full capabilities (McGuinness and van Harmelen 2004).

### 2.7. Using ontologies with graphics and simulation

In the previous section we discussed about uses for ontologies. This section represents few examples how ontologies have been used with graphics and simulation, and what benefits they have provided. This list of examples is small set of research related to the whole graphics and simulation area, but topics in the list are close to our subject.

#### 2.7.1. Linking graphics to domain ontologies

Kaloegeerakis, Christodoulakis and Moumoutzis (2006) presented an interoperable framework for integration of virtual reality scenes and semantic information. They used OWL-representation of a scene-graph, and it enabled creating mappings between graphical objects and domain specific knowledge. They defined fifteen primitive types of semantic mappings, which

could be used for binding classes or individuals of domain ontologies to classes or individuals of graphics ontology with a certain meaning. For instance, “equivalence”-relation can be used for describing that every NURBS curve represents a membrane or a polygon mesh represents a component in a specific shell. Using these relations for mapping graphics ontology to domain specific ontologies, which may be very different from graphics ontology, is possible because those relations are domain independent concepts. Hence mappings themselves are independent of domain ontologies and can be interpreted always in the same way.

Expressing arbitrary mappings using OWL object properties was considered to be impossible. Solution was to create fifteen OWL classes as an intermediate ontology to describe above mapping relations. And for last type of mappings, they added mathematical equations.

Using ontologies and inference provided several advantages. One of the advantages is creating new content based on their existing content or their incorporated domain knowledge and creating the scene based on the semantic instances of a specific domain. For instance a graphical model can be created from a representation of a chemical molecule by using rules and inference. Single atoms of the molecule can be mapped to spheres, and the position of the spheres can be calculated using rules. Similarly visualization of bonds between atoms can be created based on both structure of the chemical model and already generated geometric representation. Also the same approach can be used for visualization-aided decision making. Inference can be used for making automatic decisions, for instance, medical diagnosis based on the nature of the materials used with objects representing organs and tissue. Based on these decisions further visualization can be done, for instance, in form of animations.

Another advantage is querying the scene combining both their content and their domain knowledge. Using queries makes inference possible, for instance, the previous examples are depending inferences using queries. Critical information can be retrieved using ontological mappings. Queries like “what is function of that object?”, and “find all shapes and their materials” are easy to answer.

Ontologies also allowed personalizing the scenes by formalizing preferences of users about their content. This allows users to modify scenes so that other users will not see modifications. For example user may set material of objects that do not have material definition to a specified one, or scale all objects by two.

### 2.7.2. Linking graphics to simulation concepts

Park (2005) used ontologies to combine dynamic models used in simulation to geometric models of the phenomena being modelled, calling the methodology as *integrative multimodelling*. In multimodelling objects had three different descriptions: one for graphical model, one for dynamic model, and one for information model. Both the geometric model and the information model were used in visualization, while the dynamic model described objects' behaviour in simulation. Also the dynamic model had its own geometric representation and user could switch between visualization of the geometric and the dynamic model. Figure 12 shows a simulation of three aircrafts visualized in both ways. In the dynamic model visualization, user can also see connectivity of the models, which is visualized using arrows.



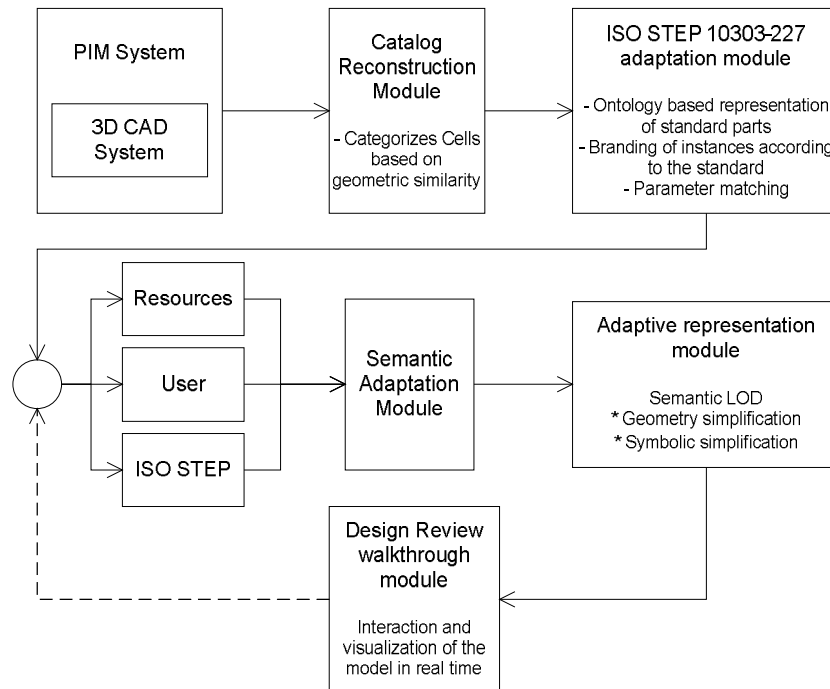
**Figure 12: Geometric (left) and Dynamic model of example scene. The geometric model visualizes the actual scene, while the dynamic model visualizes objects' behaviour and connectivity (Park 2005).**

The user interface of the system, built into Blender (2007), contained component called Ontology Explorer that could be used for creating and editing OWL ontologies. The purpose of ontologies was representing modelled phenomena; defining modelling components and formalizing mappings between them. The model components, their graphical representations and dynamic behaviours could be stored into component database, and reused in other simulations. Simulation itself was run in Blender using its game engine. The dynamic behaviour of objects was written in scripting language, either Python or JavaScript. Also user interface actions for models were created using concepts of VRML (Virtual Reality Markup Language) in ontological format.

### 2.7.3. Semantics based geometric simplification

Posada, Toro, Wundark and Stork (2005) used ontologies for semantics based simplification of large industrial plant CAD models. Their aim was to identify the objects in a CAD model and

use that information in geometric simplification to faster visualization. Their system consisted of several modules (Figure 13).

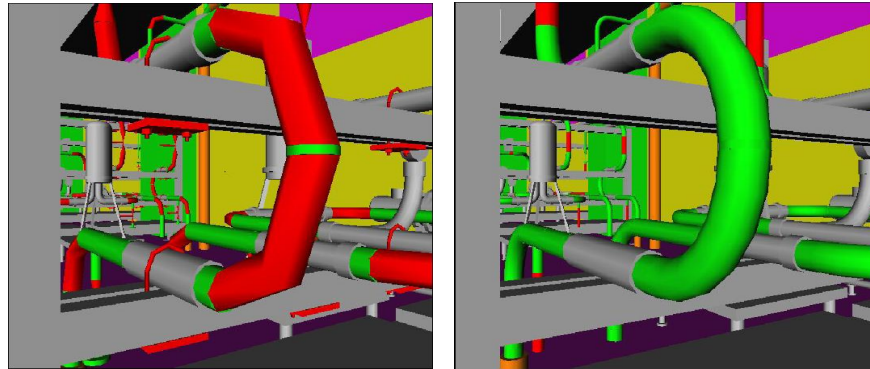


**Figure 13: Architecture of semantics based geometric simplification. Input of the algorithm is standard CAD model from Plant Information System. First geometries of the CAD are classified before rendering algorithm can use semantics.**

As input their system takes a plant model from a 3D plant modelling software. Usually those models do not contain semantic information, and Catalog Reconstruction Module searches for geometric similarity to classify all the objects in the CAD model. Then, with user's help, these classifications are mapped to concepts of STEP AP-227 (Plant spatial configuration), which is modelled as an ontology. This mapping is done in two phases: first the user matches the geometric objects to the concepts in the ontology and then the parameters specified in STEP standard to the geometric features of 3D objects.

At runtime, three modules control the visualization quality. Three aspects are taken account: available system resources, the user's intentions and background, and the model characteristics. System's resources part is self explanatory; model can be visualized with higher quality on PC cluster versus single laptop. Semantic information of the plant model is used with user profiles: depending on who is using the visualization and for what it is used. Semantic Adaptation module controls the visualization quality by increasing or decreasing objects' quality depending of those two aspects. For example, a piping engineer doing piping fixation requires different visualization than a manager who is representing the model to customers. In the engineer's view an

elbow is rendered with lower quality than in the manager's view, but the manager's view hides the clamps (Figure 14).



**Figure 14: Visualization of a plant, where level of detail selection depends on who is using it. Engineer's view (left) show the elbow in lesser quality and shows the clamps, while manager's view (right) renders the elbow with better quality and hides the clamps. (Posada et al. 2005).**

Adaptive Representation Module performs the actual geometric simplification. It can use pure geometric simplification algorithms on the original CAD geometry, replace the original geometry with a standard parametric part used in STEP AP227, or use a symbolic replacement with low triangle count. Selection of simplification type depends on context; the symbolic models can be used only if they are familiar to the user.

As a case study, they used a CAD model of a chemical plant. While the algorithm was not able to recognize all objects in the model, semantic reduction reduced amount of triangles about 50% compared to pure geometric reduction. Most of unclassified geometry of the model was caused by complex objects like boilers and tanks, and other structures like columns, windows and square pipes, but these contributed just over one tenth of the whole geometry.

### 3. Requirement analysis

This chapter represents requirements to plant modelling. Requirements are divided into two sections: functional requirements section analyses the purpose of application and derives demands for the application. Technical requirements section sets constraints for the implementation.

#### 3.1. Functional requirements

The purpose was to design and implement 3D user interface for 3D process plant modelling, link that model to a simulation model, and last visualize result of the simulation. By analyzing these purposes, we can derive functional requirements.

Modelling of a plant starts by selecting, inserting, and placing equipment. In current plant design software, the user may select equipment from catalog, or create new equipment and model their geometries. Both of them are critical features, because equipment of a plant can consist of multiple similar piece of equipment, for instance, pumps, tanks, and valves that are used for creating desired process. Similarity here does not mean that equipment would be exactly the same: their size may change. Sometimes the plant may contain custom equipment that are designed and built for just that particular plant. Therefore also creating own equipment is necessary.

Another step in plant modelling is pipe routing. It includes routing pipe from a piece of equipment to a piece of equipment, and inserting necessary components, which are used for controlling flows inside pipes, to pipeline. Since it is awkward for the user to exactly position each component, automatic updating of pipelines and their components is necessary. Pipelines also contain components that do not affect the flows, but are required in real world when pipeline is built. One example is a flange that connects components to each other. These components restrict how components can be added or inserted into a pipeline, since they have a certain order how they can be connected to each other. For instance, a valve is connected to other components with flanges. A pipeline may also contain other components, like hangers that support weight of the pipeline. These components are not necessary to a user who is creating a model for process simulation.

Another aspect that contributes requirements of the pipe routing is layouting. Normally pipe layouting emphasises right angles that help to achieve symmetric design. They contribute to easier and cheaper construction and maintenance. Therefore pipe routing should support this type of design.

Pipes are connected to equipment with nozzles. Two types of equipment exist: equipment that have nozzles in predefined places, and equipment that allow custom positioning of nozzles. A good example of piece of equipment that has nozzles in predefined positions is pump. Input and output nozzles are always connected into the same position. A tank is an example of piece of equipment that does not necessarily have nozzles in the same position: depending of the process and layout, their positions may change. Both of these features are useful, but the free insertion of nozzles is necessary. With free insertion the user is able to insert nozzles to all equipment, but free insertion allows doing incorrect design. Supporting both cases would reduce errors and make plant designing faster, because the user would not have to insert nozzles manually to each piece of equipment.

Since the main purpose 3D plant modelling is to act as an user interface to process simulation, it sets certain requirements. The most necessary feature is generating simulation model according to plant model. Our purpose is not to create our own simulator, but instead use existing ones. Anticipating what information is required by a simulator is not possible: steady-state and dynamic simulators have different needs, and many steady-state simulators cannot use dimensional information at all. Sometimes real world values will not produce a correct simulation and some adjustment may be necessary to fine tune the used simulator, and the simulation case. Therefore it is required that simulators can provide their own user interface components, which can be used for configuring simulation values.

The most intuitive way to visualize behaviour of equipment is either mimicking their natural behaviour or to using colour to represent some information about them. For instance, visualizing tank fill with an animation that changes liquid's height inside the tank is natural to every user and requires no further explanation. Sometimes this is not enough, for example, while status of a valve can be visualized by animating the position of its handle, the handle will most likely be too small to be seen from long distances. Because understanding of the behaviour of the simulated process requires that some part of the process can be seen at the same time, it may be more practical to visualize state of the valve by changing its colour. Since the colour of equipment can visualize multiple different properties, the user must be responsible for binding the animation to a simulation value. Animations bound to natural features are different, for instance, the fluid height changing animation can be used only to visualize fluid height.

In the end, what means are used for visualizing simulation depends on the user and his preferences. One may prefer animations that mimic natural behaviour, and other may prefer changing colours. Therefore the user must have ability to select the animations that he wants to use. Since animations may only visualize limited set of simulation data, it is necessary that animations and

what they visualize can be changed rapidly. Sometimes it may be possible to animate multiple features at the same time, for instance, using the same tank example, one could visualize fluid height with an animation, and use colour changing animation to visualize pressure or temperature of the tank. Supporting this would allow more information about simulation to be presented to the user. While it is not necessary, interpretation of behaviour of simulated process would benefit of it.

Animating just equipment will not bring complete visualization of a process plant: also flows inside pipes are in critical part. Therefore we need means to visualize flows. Similarly as with equipment, flows may have multiple properties that the user may want to visualize. The most common of them is probably mass flow, but other properties, like temperature and pressure are also important.

While animations and other visualization techniques are great in representing large amount of information, the user must have ability to see exact numeric values of the simulation. This way the user can spot interesting places by watching visualizations, and then inspect a single piece of equipment or a pipeline and get more precise information to form a better picture of behaviour of the process.

Complete table of the analyzed requirements and their priorities depending of usability of the requirement is collected to Table 1. It is divided to six groups: modelling, animation, equipment layouting, pipe routing, simulation, and visualization.

### **3.2. Technical requirements**

The used software architecture for the implementation is Simantics Platform. ProConf, the user application of Simantics platform, is Eclipse based. It sets certain restrictions of what libraries can be used in implementation. They must be either pure Java libraries, or libraries with JNI-interfaces.

Simantics platform is released with open source licence, but it allows commercial application created with it, without releasing them with open source license. Since our aim is to design and implement generic user interface for process simulation, GPL type of licences must be avoided, because their use would restrict commercial usage.

**Table 1: Functional requirements**

ID	Requirement	Priority
M1	Geometric models of equipment must be reusable	High
M2	Geometric models should contain sizing parameters	High
M3	Geometric models of equipment must be animatable	High
M4	Geometric models of equipment should be easy to create	High
M5	One geometric model must be able to have multiple animations	High
A1	Transformation and size of equipment, or its feature must be animatable	High
A2	Colour of equipment must be animatable	High
E1	Inserting new equipment into a plant	High
E2	Changing rotation and translation of equipment	High
E3	Modifying sizing properties of equipment	High
E4	Adding and removing nozzles	High
E5	Placing equipment relative to each other	Medium
E6	Using custom coordinate systems to place equipment and nozzles	Low
P1	Routing pipe from a nozzle to another nozzle	High
P2	Inserting inline components to pipe	High
P3	Modifying routed pipe	High
P4	Modifying inline components	High
P5	Creating branches	High
P6	Structure of pipeline should be kept correct automatically	High
P7	Pipe routing should reflect real world situations	Medium
P8	Logical order of components	Low
S1	Simulation model must be created from graphical model	High
S2	Simulators must be able to provide UI components	High
V1	Selecting animations for equipment type and single equipment	High
V2	Visualizing dynamics of flows	High
V3	Natural linking of animation and simulation property	Medium
V4	Selecting more than one animation per object that is used at the same time	Medium
V5	User must have mean to see exact numeric information of simulation	High

## 4. Implementation platform

Simantics is ontology based modelling and simulation platform developed (and is still under development) by Semantics Models research team at VTT. One of the key concepts of Simantics platform is separation of modelling data, simulation algorithms, and real-time data access. With this separation, the platform is capable of supporting multiple different models and multiple different simulation algorithms, with their own data structures, at the same time. To support different modelling needs, Simantics platform uses semantic graph: all the information is stored with triples. The idea is similar to RDF and actually saving and loading of ontologies and models is done in RDF format.

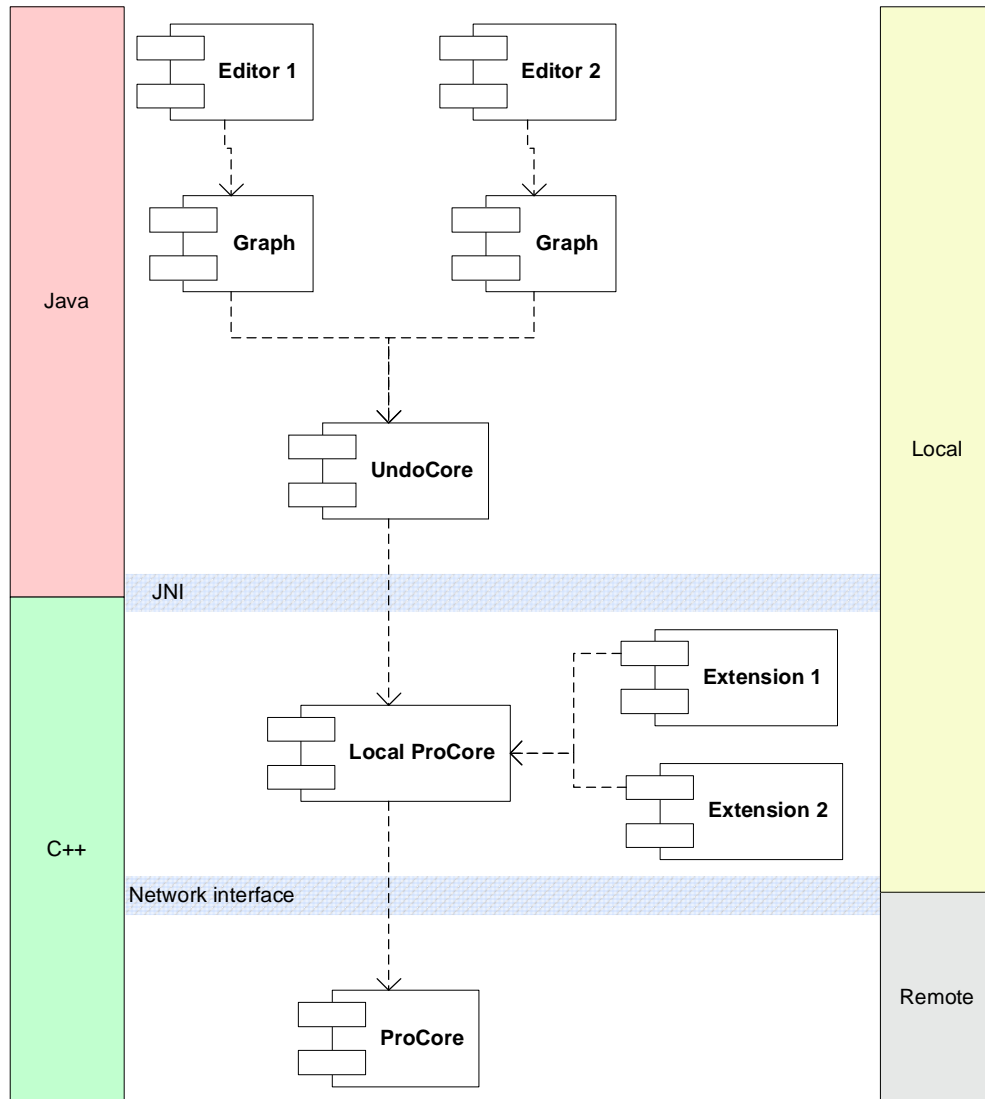
Simantics platform is multi-user modelling environment, hence it consist of two components: client application and server. Server, called ProCore, is decentralized, versioning triple storage, where all client applications are connected. Client application, called ProConf, is built on top of Eclipse RCP (Rich Client Platform).

For simulators, platform provides Extension mechanism. Extensions are client applications to ProCore, similarly as ProConf, but they are not tied to Eclipse and do not contain user interface. The most common use for the extension mechanism is simulators; extensions are registered to ProCore and it is able to launch them when needed. Simulation data transfer between extensions, ProCore, and ProConf is done with Value Sets, real-time data access interface of Simantics platform. Value Sets are separated from triple database, and therefore are capable of transferring much larger amount of data. The basic operation of Simantics platform goes as follows: first the user creates model with ProConf based editors, and starts the simulation. ProCore launches the extension of simulator, which initializes its calculation structures from information in triple database, and starts the simulation. Simulation values are transferred back to ProConf with Value Sets, and ProConf based editors can visualize the simulation.

Architecture wise, ProConf contains three components that are used for accessing triple storage on server: UndoCore, Graph and Local ProCore component (Figure 15). Local ProCore component is lowest level component of the client application, using SOAP (Simple Object Access Protocol) to connect the server and provides native interface for extensions. UndoCore acts as local triple storage for client application, caching all triples that client is using. It also provides undo features for user interface components. Graphs are components that are used for transforming triple-based data access to more user (programmer) friendly resource-based data access. All user interface components are accessing the database using graph components. These graphs

contain their own caches, and whenever something is changed in one of the graphs, changes are updated to other graphs via event based mechanism.

To further decrease programming effort, ProConf contains Java code generator, which is able to generate Java stubs for handling ontologies. Since ProConf supports multiple inheritance, multi instantiation (instance is instance of multiple classes), Java's instanceof checks cannot be used with stubs. Therefore ProConf provides its own methods to do the same thing.



**Figure 15: Architecture of Simantics platform.**

#### 4.1. Layer0

Layer0 is the base ontology of the system and all other ontologies that are added to the system are created using its concepts. It defines many basic concepts, like relations and properties as OWL, but it has several features that add support for separation of model configuration data and

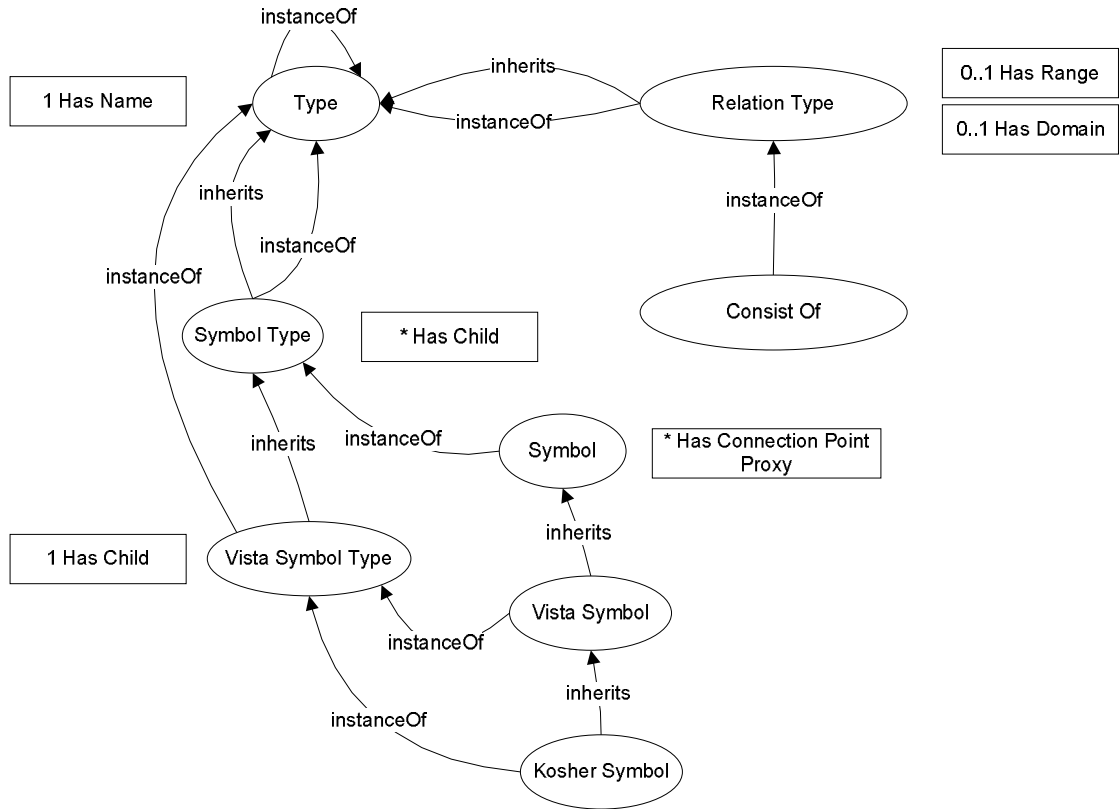
real-time (simulation) data. When ProConf was initially designed, using OWL was considered, but using OWL's DL-level that most of the available tools support, does not support all necessary features to describe all use-cases, like product-function-individual relationships and class-of-class concept. Class-of-class concept describes mechanism, how an instance of a class can be used as a class and further instantiated. This is used for expressing products where product is a class, and manufacturer's certain product is an instance of that class. That instance is then used as a class and instantiated into an individual piece of equipment. For example pump is product type, Sulzer Ahlstar A Series is a product, and a pump installed to a plant is an individual.

Another thing is that OWL relies on inference in various cases, for example when checking if an individual is an instance of a class. In the system, which purpose is to handle large amounts of data, using inference for too many things was considered to be too slow. Therefore decision was made to create own base-level ontology.

Basic concepts of Layer0 are *types*, *instances*, *relations* and *properties*. Types are similar as classes in object oriented programming languages: they define what relations and properties instances of a type may and must have. Relations are used for describing relationship of two resources. A relation can restrict its domain and range; where it starts and where it ends.

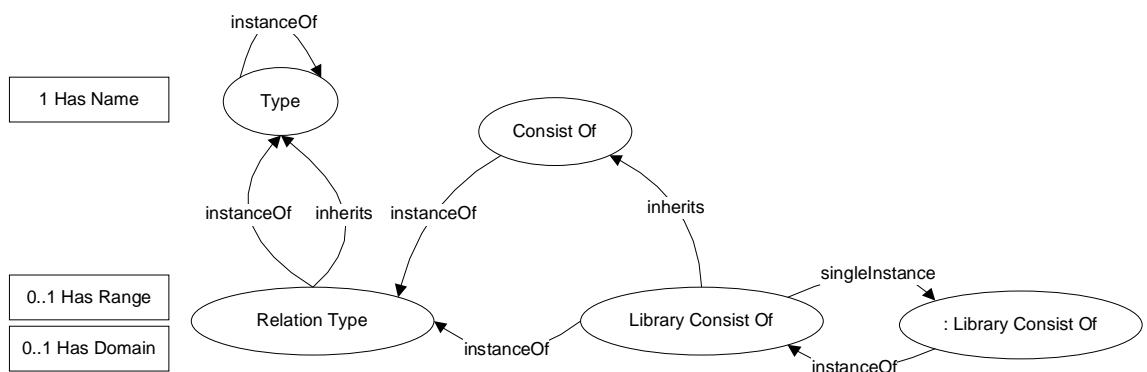
Layer0 has two basic relations that are used for modelling types and instances: *instanceOf* that describes instantiation of type, and *inherits* that describes inheritance of types. With inheritance, restrictions of a type are transferred to the inherited type, if the inherited type does not override them. For example, in Figure 16 Vista Symbol defines that its instances may have connection proxies because it is inherited from Symbol, but Vista Symbol Type overrides restriction of Symbol Type and its instances must have one, and only one child, while instances of Symbol Type may have zero or more children.

A resource must always be an instance of some type. This also applies to Type that is the lowest type in Layer0: it is an instance of itself. With instantiation type's restrictions are forced to instance. In Figure 16 Symbol may have children because Symbol type defines such restriction, but Vista Symbol must have one child. A resource is a type if it is instance of a resource that is inherited from Type or if it is an instance of Type. In figure Symbol is a type because it is an instance of Symbol Type that is inherited from Type, and Vista Symbol is a type because it is an instance of Vista Symbol type that is inherited from Type.



**Figure 16: Basic concepts of Layer0 and example of their usage.**

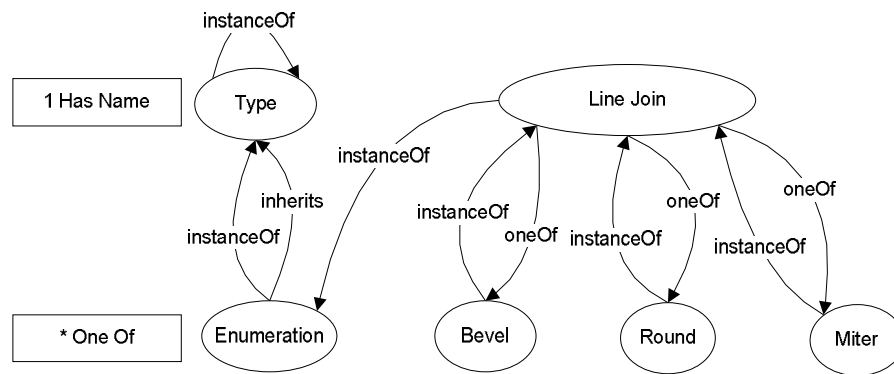
A resource is a relation type if it is an instance of Relation Type. Relations can be used in two ways: either as singleton relations or as relations instances. Relation instances may have their own properties. Singleton relations are similar to relations (properties) in OWL, and they only describe semantics of a relationship without giving any properties to the relationship itself. In Figure 17 Library Consist Of is a relation type and :Library Consist Of is its singleton instance.



**Figure 17: Relation concepts: Library Consist Of is a relation type and :Library Consist of is its singleton instance.**

Properties can be divided into two cases: structural properties and literal properties. Structural properties are similar to normal types, the difference is only conceptual: they are used for describing data related to the actual types. Literal properties are for storing primitive data: Strings, doubles, integers, and boolean values. Properties can be flagged as stateful or stateless. Stateful properties are used for real-time data access with simulators, and are not stored to graph. Stateless properties are stored into graph and therefore versioning of them is supported, which is not the case with the stateful properties.

Enumerations are used for describing restricted set of types that may exist. For instance, in Figure 18 Line Join may be only Bevel, Round, or Miter, nothing else. Enumerations are formed by instantiating Enumeration and adding all items to instantiated Enumeration with instanceOf- and onOf-relations.



**Figure 18: Enumerations: Line Join is an enumeration that is either Bevel, Round, or Miter.**

## 4.2. User Application

ProConf uses Eclipse development platform (Eclipse, 2007) as its basis for user interface. Eclipse itself is Java-based application, using its own toolkit for user interface (SWT, Standard Widget Toolkit). SWT is wrapper for native user interface components, which allows it to look similar to native applications.

Eclipse itself is plug-in based; core components contain only OSGi-based plug-in managers, which are capable of loading and unloading plug-ins when needed (McAffer and Lemieux, 2006, chapter 2). Whole Eclipse-based application is just a set of plug-ins, which use each other's services to provide functionality that user needs. Plug-ins can use services of other plug-ins two ways: either extending other plug-in and its classes, or adding new functionality to existing plug-in using Extension / Extension Point mechanism. Only way the first method differs from traditional object oriented programming is that plug-ins may restrict what classes are visi-

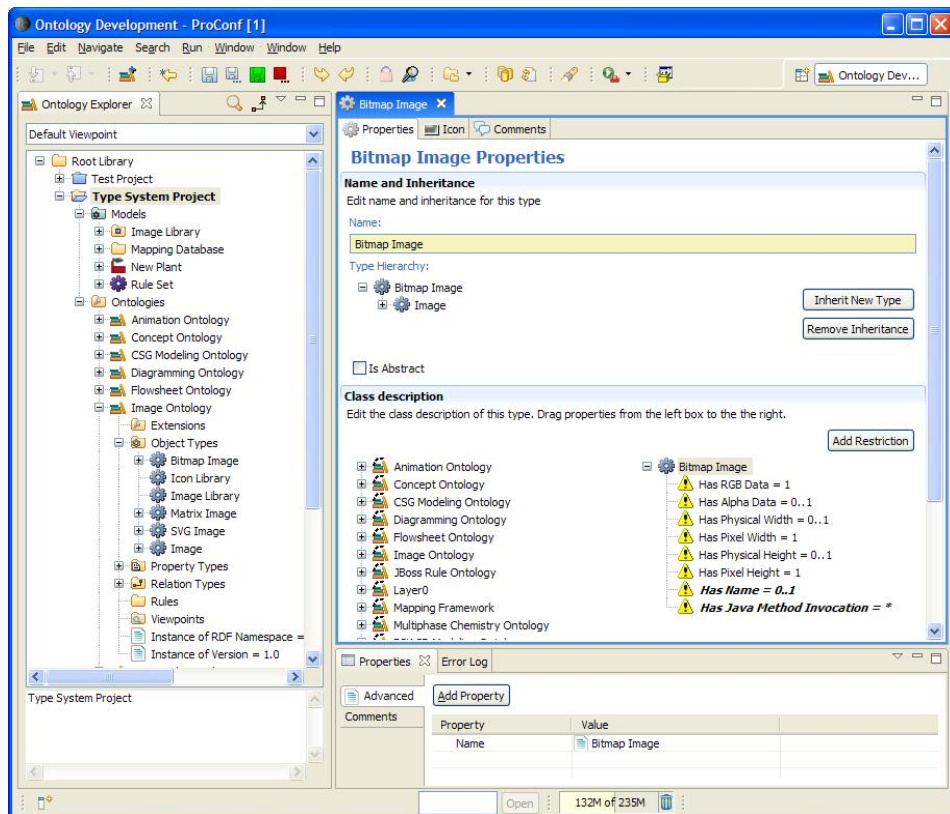
ble outside of the plug-in and if a class is not visible to other plug-ins, it cannot be extended. Latter way is somewhat more complicated process, but is heavily used throughout whole Eclipse, and it is beneficial to ProConf too. When plug-in defines an Extension Point, it defines an interface that other plug-ins can use for adding services by creating Extensions. The interface can be used for transferring data, like images, but the most common way is to describe a Java-interface, or multiple of them, which plug-in that creates Extension must implement. In Extension Point providing plug-in, all Extensions that are attached to the Extension point can be listed, but can be used only through the defined Java-interfaces, and so actual Java-classes that implement the Java-interfaces are not visible. For example, Eclipse defines an Extension Point to editors, and all editors must be Extensions to that Extension Point so that they can be used in the system.

For ontology based programming Eclipse's plug-in architecture suits perfectly because the editors related to an ontology can be loaded only if the ontology is used, which decreases memory footprint of the program, when the user needs only few of the all available ontologies to do his tasks. An other way, how Eclipse's plug-in architecture is used, is that when an ontology extends concepts of another ontology, it can provide code that can handle those extended concepts by using the Extension Point mechanism.

### **4.3. Ontology Development**

ProConf contains its own ontology development user interface, capable of creating Layer0-based ontologies. Basic editors include tree viewer of semantic graph (Ontology Explorer) and editors for properties and types (Figure 19). To filter out unnecessary information, Ontology Explorer uses viewpoints that can be used for specifying which classes and instances are shown, which relations are traversed and shown, and which properties are shown. Viewpoints make the Ontology Explorer and its tree component reusable in various cases, where there is a need to show information to the user in a tree structure.

Viewpoints are also used in other parts of ProConf. When data is exported out of ProConf, for instance, when the user wants to export an ontology, viewpoints are used for selecting what is exported. Another case where viewpoints are used is when there is a need to copy data. In sense viewpoints are definitions of sub-graphs. The problem with viewpoints is that the ontology developer must create a viewpoint for exporting model if the ontology is used for creating models, and if the developed ontology contains certain custom structures, also a viewpoint for exporting the ontology itself must be defined. Hence, the ontology developer must be familiar with the concept of viewpoints.



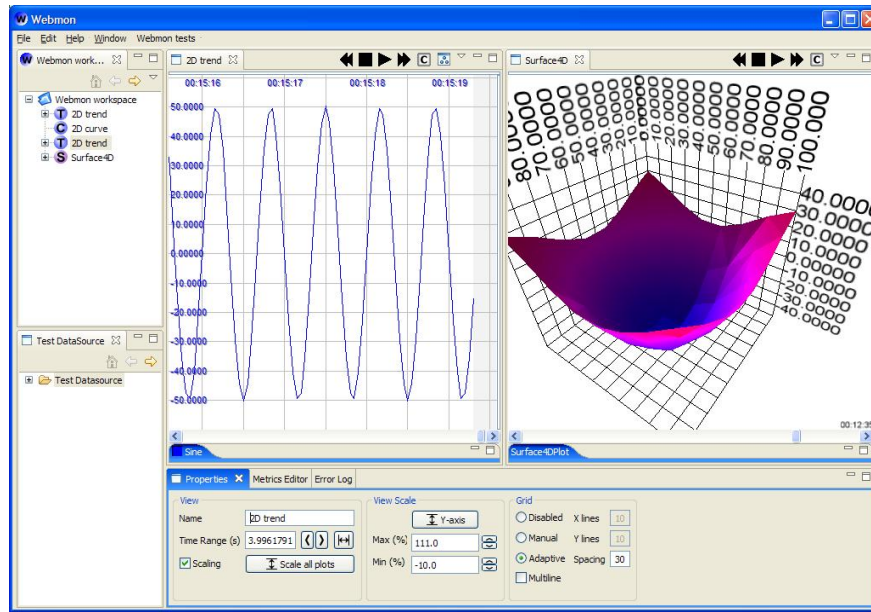
**Figure 19: Ontology Development in ProConf.** Ontology Explorer is on left, general properties view is in bottom. Ontology Explorer uses Default Viewpoint to represent information in graph. Type Editor has been opened for Bitmap Image. Its restrictions are in bottom right corner of the editor and type hierarchy on top.

#### 4.4. Basic Ontologies

In addition to Layer0, ProConf contains several basic ontologies that are used in the system.

- *Image Ontology* provides image support to ProConf. It is used for attaching SVG and bitmap images into ontologies. Image Ontology provides mechanism to set icons to concepts of ontologies, which are used in Ontology Explorer and other UI components.
- *Structural Modelling Ontology* defines concepts of structural model. It is a base ontology for several modelling cases, including flowsheets and diagrams.
- *JBoss Rule Ontology* is currently used mechanism to rules. It uses JBoss Rules rule-engine, which is based on pattern matching.
- *Mapping Framework* and its ontology can be used for creating generative mappings between ontologies. This means that if user is creating model using ontology a, and mappings from a to b are defined, mappings generate model based on ontology b.

## 4.5. Data Visualization



**Figure 20: User interface of Webmon. Plot configuration is in the tree component on the top left, a data source browser in the bottom left corner, a 2D trend visualization in the middle, and a 4D surface visualization on the right.**

Since ProConf is used for simulation, data visualization capabilities are one of required features of such platform. ProConf contains data visualization package called Webmon (Web technology based process data monitoring tool) (Kalajainen and Luukkainen 2005). Webmon is extensible visualization package, where new visualizations and data connectivity capabilities can be added as plug-ins. Currently implemented visualizations are 2D trend, binary signal, 2D curve, 3D trend, 3D surface, and 4D surface (Figure 20). Webmon can retrieve data from OPC XML-DA servers (OPC Foundation, 2007), import and export ASCII-based files, typical in automation systems, import data from and export data to historian server. Similarly capability to read data from Simantics' ValueSets has been added to Webmon, which allows it to visualize any simulation that is run in Simantics environment.

## 5. Design

This chapter presents design phase. The design is based on the requirements gathered in the previous chapter. When we look at the requirements, we can see four cases: modelling equipment, modelling plants, and creating mapping between plant model and simulation model, and last configuring and visualizing simulation results. Creating mappings between simulation model and plant model is different task to the others: it has to be done only once for each simulator, and requires understanding of both plant model and simulation model. Therefore creating mappings is not a modeller's issue; he just uses them when he models a plant. Other tasks are more relevant to the process modeller: he must be able to compose a plant model from provided equipment models, create new equipment models if needed, configure and run simulation, and last visualize simulation results. These tasks require user interface that can support such actions.

There are two choices: either combining plant modelling and equipment modelling into the same user interface or creating two interfaces, one for each case. Since those tasks are different from each other, most likely using latter approach will lead to better usability. This also helps with the implementation, since one user interface component does not have to support all the functionality. It also contributes to reusability of the components: the user interface created for equipment modelling can be used for other 3D modelling needs in Simantics platform. The user interface for modelling equipment is dubbed as Shape Editor, and the interface for process modelling is called Process Editor. Since visualization of simulation is partially based on animated equipment, the equipment modelling contains both creating geometry for the equipment and creating animations for them.

### 5.1. Shape Editor

Creating and editing geometric models for equipment can be divided into three tasks: creating geometry of equipment, creating animations, and parameterising the geometry. The last step is optional, since it is not necessary to create parameterised geometry, because there is also equipment that is used only once or may not be resized.

#### 5.1.1. Modelling

When we look at the possibilities for geometric modelling there are two different techniques that can be used: polygon-based modelling and solid modelling. Polygon-based modelling is most popular of the two and it is widely used especially in the entertainment business. But in the industry, where geometric models are used for production and simulation, more precise representation is needed. Therefore most of the 3D-CAD programs are based on solid modelling,

which is more precise than polygon based modelling. When we compare these two, both of them have their good sides and bad sides. For example, creating texture coordinates for solid models can be tricky, while with polygon based models that is quite straightforward. On the other hand, with solid models feature based modelling is feasible and it allows creating sizing parameters for geometry, because geometric representation of solid models is parameterized. With polygon based models, only resort could be describing position of each vertex as a function of all sizing parameters of the model.

Then there is the issue how the geometric model is modelled, and how to design and implement a user interface for it. The aim was to design an interface that is easy to use, but is able to create geometric models of typical industrial plant equipment, like tanks, pumps, valves, and so on. Probably the most simplest modelling paradigm is CSG-modelling, which requires creating primitives, adjusting their properties and combining primitives with Boolean operations. Using CSG-modelling does not bound the geometric representation; both polygon based modelling and solid modelling can be used.

Third aspect for choosing modelling paradigm is availability of open source components that can be used. Such components would speed up implementation process and probably add other useful features that would not be included in the implementation otherwise.

When all these aspects are considered, CSG-modelling paradigm was chosen because of simple concept that users can easily adapt to, and the implementation for it will not be complex. Review of available modelling kernels with suitable licenses revealed that OpenCASCADE is probably the best choice, because it supports B-rep based solid modelling, CSG-modelling, and it can import both IGES and STEP files, which enables importing existing models given in those formats. It also contains other useful features that can be used in future research projects. Better description of OpenCASCADE is given in Section 6.1.2.

When looking at user interface side, and possibilities for it (Section 2.3.1), most intuitive way to translate and rotate objects are gizmos. In addition to that also support for inputting values using keyboard must be available so that user can input precise information, which can be hard task to do using gizmos. Luckily it will not require any effort, since ProConf is able to show editable textual representation of objects' properties in Eclipse's properties view.

### 5.1.2. Animation

Useful techniques for implementing animation system were described in Section 2.5.1. Since keyframe-based animation is easy to use and implement, it was chosen as animation technique for animating equipment models. This means that when a user is creating animations for equipment, the user interface has to adapt to that, adding capability to create and modify keyframes.

In addition to user interface used in modelling, also user interface for editing interpolator curves is needed. ProConf contains implementation for 2D graphics, and while it is currently used for diagramming purposes, its components should be reused in interpolator editor. This would reduce implementation effort, and make use of already implemented features, like zooming and panning.

### 5.1.3. Parameterisation

The aim of parameterization in the implementation is to add reusability of geometric models of equipment. The intention is to add sizing parameters to geometries that have meaningful value, which can be used in process simulation. While using automatic detection of constraints could be applied, implementing such a system would be a tremendous task requiring complex analysing of modelled geometry. Since that is not the research problem, but a mean to make the application easier to use, the parameterization is done as a proof of concept without putting a lot of time to implement it. Therefore, parameterisation of geometry is done explicitly after modelling and animating of equipment, using editor that can be used for forming equations to define size and position of geometric features.

## 5.2. Process Editor

Process Editor has to handle previously mentioned tasks: process modelling, creating simulation model, configuring simulation, and visualizing simulation results.

### 5.2.1. Process modelling

Process modelling can be divided to equipment related and dipping related tasks. Equipment related tasks include inserting them into plant, translating and rotating them, and modifying their properties. Piping related tasks include routing pipe, inserting components into pipeline, and modifying properties of piping.

Moving and rotating equipment and nozzles can be handled using gizmos as in Shape Editor, but since it should be possible to position equipment relative to each other, snapping feature must be introduced. As explained in Section 2.4.2, in heavily populated environments snapping against every object will not be usable. Also snapping against objects that are at certain range from the moved object is too restrictive. These are the reasons why snapping is implemented in a way that user can manually specify the components that are used in snapping.

As explained in Section 2.2.3, pipe routing requires user interface components that are specially tailored for that purpose. Automatic pipeline updates contribute a lot to usability: the user does not have to modify each component separately when he is modifying a pipeline. This is crucial in simulation configuration, because simulation can be used for testing and evaluating different designs, which may have different layouts, which require easy modifiability of the pipelines.

Keeping structure of pipelines automatically correct introduces restrictions to positioning of pipeline components. For example, when we think about a valve, it can be rotated only around the pipe, and moved along the pipe without causing major changes to the whole pipeline. While automatic updates with free editability of all components can be achieved, for usability reasons restricting modifiability is in order. But since it is good to keep user interface consistent, custom gizmos are used for translating and rotating them.

Since a pipeline may contain only limited set of components (some examples were given in Section 2.2.2), we can derive some basic user interface actions that can be used for routing pipe and modify existing piping. The preliminary list of actions is:

1. Starting a new pipeline from an unconnected nozzle
2. Ending a pipeline to an unconnected nozzle.
3. Continuing a pipeline from a loose pipe end.
4. Moving an elbow or multiple elbows.
5. Adding new elbows (splitting a straight pipe into two straights)
6. Removing an elbow from pipeline. (Joining two straights into one)
7. Starting a new pipeline from a straight pipe (making a branch)
8. Ending a pipeline to other pipeline's straight pipe (making a branch)
9. Ending a pipeline to a loose end of other pipeline (joining pipelines)
10. Inserting inline parts into built pipeline.
11. Removing inline parts from pipeline.
12. Moving inline parts along a pipe (movement is restricted to the pipe's path)
13. Rotating inline parts (rotation axis is always the pipe's path)

With these actions only way to change the route of a pipeline would be moving elbows, inserting new elbows, or removing existing elbows. The principle here is similar to editing polygons: elbows can be moved similarly to corners of a polygon, and components between the elbows are translated and rotated to line between the elbows automatically. Here the structure is more complicated though, because a pipeline may start and end as a branch to other pipeline.

#### 5.2.2. Mapping the plant model to a simulation model

A mapping between the plant model and a simulation model has to be done once for each simulator. Simantics platform already contains mapping framework that is used for mappings ontologies to each other, and it is used here for simulator mappings. In optimal case, mapping system requires only activation by a process modeller, and after that it should work automatically.

Two ways to create mappings exists: either generating simulation model from an already modelled process or generating simulation model at the same time as the user models the process. Both of these should be supported, since just resorting to one of the methods is not sufficient. Latter approach can take account changes to process configuration, but is not able to generate simulation model to already modelled process. On the other hand, the first approach requires manual synchronisation of the simulation model.

#### 5.2.3. Configuring and visualizing simulation

Simulation configuration depends on simulator and its data model (ontology). Since there are many different simulators with different concepts, creating a user interface that takes account every possible case is complicated, if not impossible task. ProConf contains already properties view that can be used for modifying everything in the graph, but it is usable only in most trivial cases, where input is only an unconstrained number or text in free form. Other approach would be using Ontology Explorer, and simulators could provide proper viewpoints for it. Only problem here is that tree structures are inadequate way to represent and edit some information. Therefore simulators must be able to provide their own user interfaces to part of Process Editor. They can be added either using ontologies, since Simantics supports attaching Java code as part of an ontology, or using Eclipse's Extension Point mechanism.

Two different types of simulators were identified in Section 2.1: steady-state and dynamic. Their difference in handling simulation also affects visualization: Steady-state calculates results and then stops, while dynamic simulation keeps calculating new values. In visualization, their

difference is apparent: steady-state simulation updates animated equipment, but does not create changing animation out of simulation.

The user must be able to select what he wants to visualize, and how it is done. As stated in the requirement analysis, one could prefer visualizing tank fill level with liquid height changing animation, while other would use colour change. Therefore we need user interface components to select and control animations, and linking them to properties of simulation model. Same goes to exact numeric representation of simulation: a simulation model may contain multiple properties, but most likely the user is interested only few of those.

## 6. Implementation

This chapter describes the implementation. The first section describes software libraries that were used in the implementation. Next is description of the scene-graph that is designed as a common data structure for 3D graphics in ProConf. The scene-graph is common for both Shape Editor and Process Editor, which are described last.

### 6.1. Used software components

To ease up programming effort, existing software components were used. These are jME, which is a library that renders 3D scenes and contains interaction capabilities: Another library is Open-CASCADE, which is modelling kernel for CSG- and B-rep based modelling.

#### 6.1.1. jME

jME (jMonkeyEngine, 2006) is open source (BSD type of license) scene-graph library implemented in Java. It abstracts the rendering implementation, but currently only implemented interface is LWJGL (Light Weight Java Gaming Library). jME contains several useful features that can be used in implementation:

- Scene-graph hierarchy: allows easier scene compositing capabilities, including animations.
- Frustum culling: renders only parts of the scene that are potentially visible, improving rendering performance.
- LOD-support: jME supports discrete LOD and CLOD techniques, which can be used for achieving better rendering performance.
- Render-to-Texture support: can be used for rendering more complex visualization effects.
- Impostors: can be used with discrete LOD as a lowest quality model.
- Picking (triangle-ray intersections): this is required for building user interfaces for 3D applications.
- Particles: can be used with flow visualization.

jME is designed for Java-based games, and so it has several limitations like support for only one window at once, but because of the BSD license, that was changed. Another feature, that jME lacks, is occlusion culling. While most likely scenes generated with process editor will not contain much occlusion, Coherent Hierarchical Culling (Bittner et al. 2004) was implemented.

### 6.1.2. OpenCASCADE

OpenCASCADE is modelling kernel that constructs solids using boundary representation and it is able to generate triangulation to solids. It also supports CSG-style of modelling by providing union-, intersection- and difference-operations between solids. Development of OpenCASCADE started in 1993 with name CAS.CADE (Computer Aided Software for Computer Aided Design and Engineering), and it got its current name in 1999 when Matra Datavision published it as an open source library.

Import and export functionality of OpenCASCADE is good since it supports importing and exporting of IGES and STEP (AP203/214) files, exporting STL (stereolithography) and VRML files, and importing and exporting its own file format. This kind of functionality is valuable because it allows data exchange between other CAD/CAM applications that support those formats.

OpenCASCADE also includes its own application framework and graphical user interface framework but those cannot be used without compromising consistency of the user interface because of Eclipse environment. There is no technical restriction for its use: the application framework contains also Java implementation, but it contains AWT-based user interface implementation, while Eclipse is based on SWT. Therefore only geometry modelling features, and import/export functionality of OpenCASCADE is used in the implementation.

## 6.2. Scene-graph

Since the system is ontology based, also scene-graph needs its own ontology. Instead of inventing own scene-graph concepts, we use X3D (2007) as specification as a basis for our scene-graph. Using ontology gives better ways to describe relationships between objects and properties than X3D; therefore its concepts are not just copied to ontology format.

Scene-graph and its concepts are put in ontology called 3D modelling ontology. Basic type for scene-graph node is GraphicsNode, which may have position, rotation, and centre, and as in typical scene-graph, node has one parent (or none if node is root) and may have unrestricted amount of children. Here the presentation differs from X3D that has transformations as separate nodes in scene-graph. This is done because it is more natural way to describe position and orientation of an object instead of creating separate nodes for them. Each of those properties is optional; missing property means identity transformation. Because of that, it is possible to construct a scene-graph in X3D's style with separate transformation nodes, while that is not the intended way to use it.

Because of parent and child relationship, GraphicsNodes form a tree structure that supports hierarchical transformations. Since the purpose of the scene-graph is to support editing, collapsed form of transformations are also stored. This allows scene-graph handling algorithms to handle all 3D objects in the same coordinate system. Collapsing of transformation hierarchies is achieved by propagating transformation changes in the scene-graph tree. Collapsed transformations are called world transformations, and nodes' transformations relative to the parent node are called local transformations. Propagation is based on rules:

- If local transformation changes, world transformation of the node and every child must be updated.
- If world transformation changes, local transformation must be updated and children's world transformations must be updated.

Rule-based transformation propagation has its weaknesses and strong points. Rule-based propagation is slower than propagating transformation directly in the scene-graph implementing code, and most of the time it is unnecessary, because collapsed form of transformations is rarely needed, but it allows other components and editors to modify scene-graph without them knowing dependencies between objects. Therefore without rules transformation changes would not be propagated and the scene-graph could end up in inconsistent state.

ShapeNode, which is used for geometric objects that have visual representation, is inherited from GraphicsNode. It does not specify any other features, but in the actual visualization implementation level it has special handling; ontologies that define subclasses of ShapeNode must provide code as an Eclipse extension to handle provided classes so that they can be visualized. This is the way how higher level 3D-editors (Process Editor as an example) can be implemented, without coupling them to specific geometric modelling concept, and other modelling concepts can be added, requiring no changes in higher level editors.

For shape modelling support, the ontology has a separate relation for defining internal structure of a shape. While "Has subnodes"-relation is used for describing standard scene-graph structure, "Has subgeometry"-relation, inherited from "Has subnodes"-relation, is used for describing that a child node is actually contributing to its parent's geometry. This allows easier handling of both visualizing and editing features; editors can traverse "Has-subgeometry"-relations to show all needed visual features for geometric modelling, but components that use shapes only for visualization purposes can omit those.

Material definitions, including colours and textures are described in 3D modelling ontology very similarly as in X3D. One of the differences is that X3D uses separate data arrays and assumes that the reader can map items in multiple arrays to each other by their index, but in ontological format such arrays can be decomposed, and no assumptions are necessary. This is used for describing multi-textures and other properties in the scene-graph. Also data structures for images can be left out, because ProConf contains Image ontology that is capable of handling bitmap images and rasterising vector graphics. For instance, Image ontology contains support for SVG (Scalable Vector Graphics) images.

While X3D has support for user actions in the scene-graph, 3D modelling ontology does not support them. Attaching user interface related nodes to scene-graph is convenient only when the scene-graph is built only for one purpose, which is contrary to the requirements since the same model of a plant is used both in modelling and editing the whole plant, modelling and animating a single equipment, and visualizing (and configuring) of simulation of a plant, which all require different user actions. Therefore user interface is provided by software components that must interpret used ontology and context of use and build user interface according to that.

Other feature that X3D has, but scene-graph ontology does not have, is basic geometric primitives. This is because primitives, such as spheres and cones, are not useful in more complex applications, and ontologies extending 3D modelling ontology can provide primitive shapes by themselves, if they need them. Also audio-support was not considered useful, structures for animatable humans and geospatial data are more domain specific cases and are not supported in base ontology, but could be easily added by creating new ontologies derived from 3D modelling ontology.

Basic design principle in 3D modelling ontology is that it supports only those features that are necessary, and other features can be added by creating new ontologies and code to handle those ontologies. This goes according to the design principles described in section 2.6.5, where minimal ontological commitment was considered as good feature that enhances reusability of ontologies.

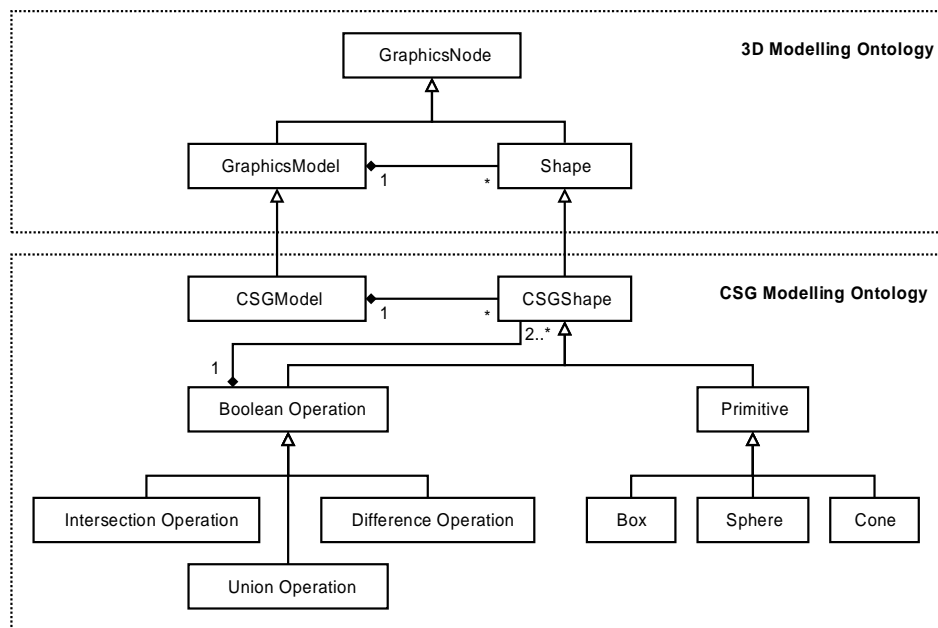
### **6.3. Shape Editor**

Shape Editor must be able to handle modelling of equipment, creating animations for them, and last parameterize the geometry. In the design chapter we decided to use CSG-modelling for modelling equipment. Therefore we will have to create an ontology for concepts of CSG-modelling. Another ontology that we need is Animation ontology. When concepts of animations

are separated to its own ontology, it can be reused in other modelling schemas. We will discuss about these two and the geometry parameterisation implementation in the next three sections.

### 6.3.1. Geometry

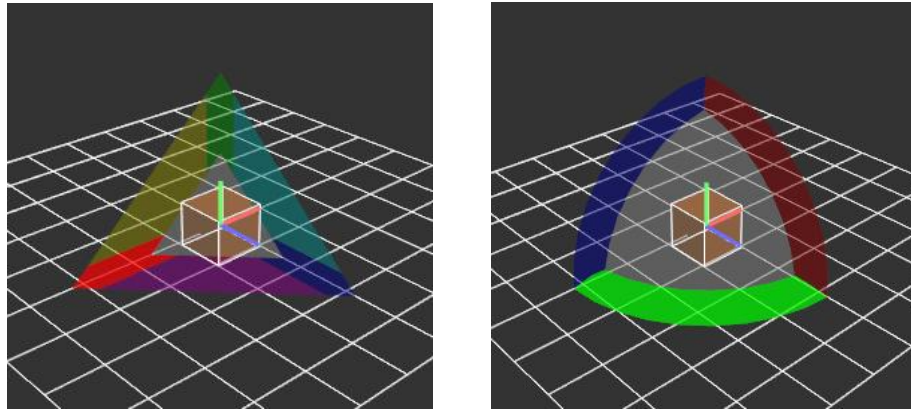
The implemented Shape Editor allows user to create new instances based on CSG-modelling ontology (Figure 21), which is derived from 3D modelling ontology. CSGShape-type is inherited from Shape, so that shapes used in CSG-modelling can be differentiated from other shapes. This is done because modelling kernel must be able to interpret all shapes to form visual representation, and it cannot be guaranteed that the used modelling kernel could support every shape in the system. CSGShape is divided to Primitive and Boolean Operation types. Boolean Operation is divided to Union, Difference, and Intersection operation, and it may contain multiple CSGShapes. Here the logic with Difference operation is that all shapes are intersected from the first shape attached to the operation. With Union and Intersection operations order of the shapes does not affect the result. Primitive type is further divided to actual primitives, like Box, Sphere, and Cone, which have unique properties for defining their sizes.



**Figure 21: Type hierarchy of CSG-modelling ontology. Diagram does not contain all primitive shapes.**

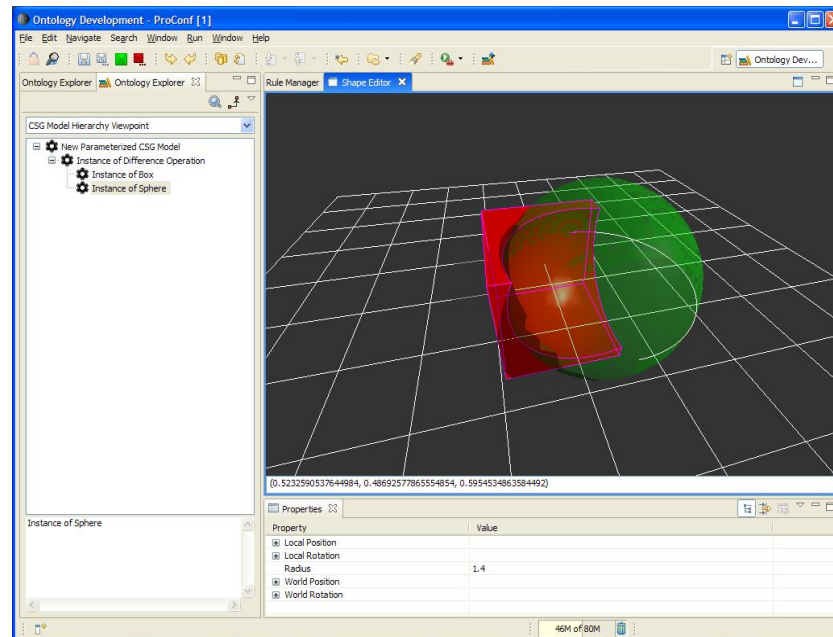
On the user interface side, use can move and rotate shapes using gizmos. Implemented translate gizmo is similar to triad cursor in (Nielson & Olsen 1986), but instead of using picking position to map mouse coordinate to 3D, user can interactively select one of the axes or one of the axis-aligned planes whom along he can move the selected object by dragging the mouse. In Figure 22, the user moves the cube along x-axis (red arrow is highlighted). Similar behaviour is used

with rotate gizmo: the user may interactively select rotation axis and dragging the mouse changes rotation angle. More exact translating and rotating can be done with Properties View, which is also used for changing sizing parameters of primitives. Mathematically the implementation of gizmos is similar to method described by Phillips et al. (1988), where translation of an object is calculated from intersection of the mouse ray and ray of an axis when the object is translated along the axis or the intersection of the mouse ray and a plane if the object is translated along the plane.



**Figure 22: Translate gizmo (left) is used for moving objects and Rotate gizmo (right) is used for rotating objects.**

When the user selects more than one object (primitive and / or shape of Boolean operation), the user can create a new Boolean operation (union, intersection or difference) using context menu. Shapes inside of a Boolean operation can be moved and translated, but they must be selected using Ontology Explorer before they can be modified. These interior shapes are only visible when user has selected them and to distinct them from the model, they are visualized with transparent colour. Selection of an interior shape can be seen in Figure 23, where the user has selected a sphere from an intersection of a box and the sphere. Selection of visible shapes is visualized by changing colour of the object and highlighting edges of the shapes in way that they can be seen through other objects. This evades the problem described in Section 2.3.2, where Phillips et al. (1992) changed the position of the viewpoint when modified object was occluded by other objects. Using see through edges avoids changing the position of the viewpoint, which can be problematic for the user. For performance reasons, the whole CSG-model is not updated real-time when the user moves or rotates a shape using gizmos. Instead the shape that the user moves is moved interactively on the screen, and when the user stops moving or rotating the shape, the whole CSG-model is updated. Another case, when the whole model is updated is when the user changes sizing parameter of a primitive, since that is done using Properties View, instead of gizmos and there is no interactivity in the action.

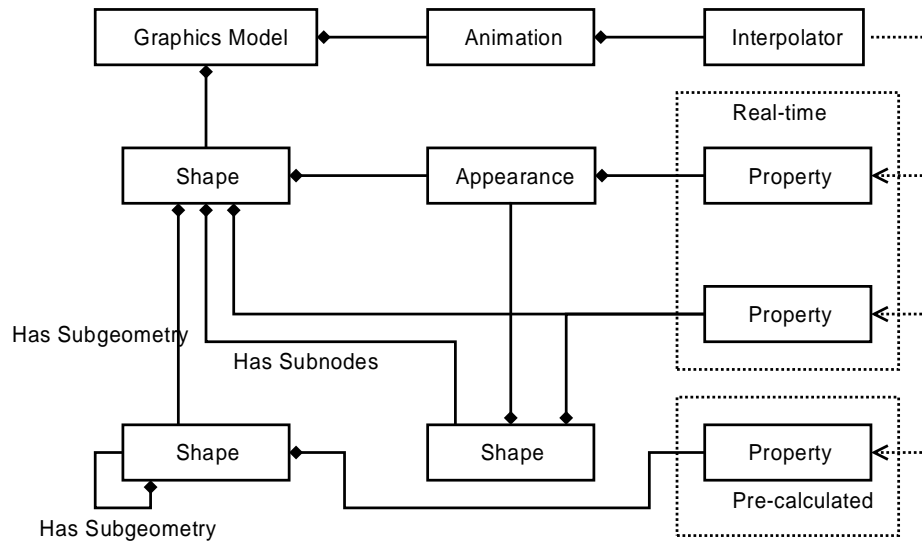


**Figure 23 : CSG-modeling interface. Editor shows intersection of a box and sphere. Structure of the model is shown in Ontology Explorer (right). The sphere is selected and its properties are shown in properties view (bottom).**

### 6.3.2. Animation

The animation implementation uses similar structure to X3D, where interpolators are used for changing geometrical and material properties. Interpolators are combined to a single animation, and one geometric model may contain multiple animations. Since there may be need for animation of geometry defining components of the model, like animating CSG- models by changing properties of primitives, animations are divided into two cases: real-time animations, where all animated properties can be changed on the fly, and pre-calculated animations, where properties of sub-geometry are modified and recalculation of geometry would take too much time. The limitation of pre-calculated animations is that the same animatable object can use only one of them at the same time, because using multiple geometry changing animations would require calculation of all possible combinations, which would result large amount of pre-calculated meshes.

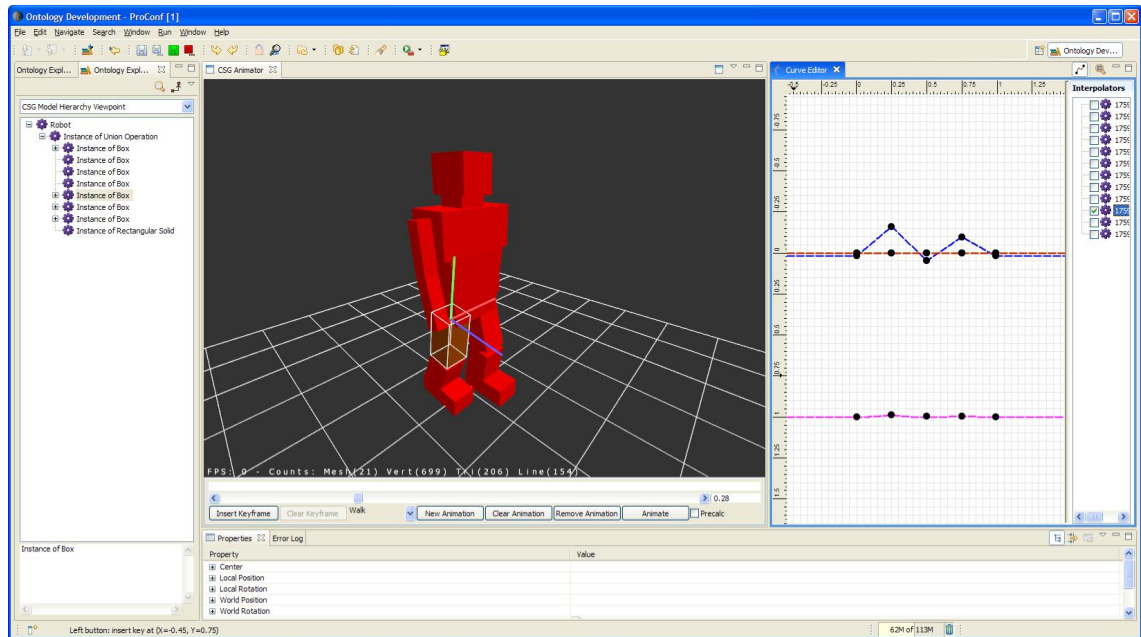
The structure of animations is in Figure 24, which shows difference of real-time and pre-calculated animations. Pre-calculated part of animations is interpolators that modify properties of shapes that are connected to their parent using Has Subnodes-relation. If a shape is connected to its parent with Has Subnodes-relation, all its children must be connected also with the same relation. Therefore if an animation needs pre-calculation, its interpretation is easy.



**Figure 24: Logical structure of animations. Animation can be either real-time or pre-calculated, which depends of if animation changes geometric features or not.**

Animation concepts are separated to their own ontology, but are considered as a basic feature of 3D modelling, and therefore it is dependency of 3D modelling ontology. By separating animation ontology, both 3D modelling and 2D graphics ontology, which is implemented in ProConf, may use the same ontology. Animation ontology itself does not contain any specific interpolators, and must be extended to give precise meaning for them. Similarly to handling of shapes, animation ontology contains an Eclipse extension point, and those ontologies that specify interpolators must also provide code for handling the interpolators. Two specific interpolator types were implemented: 1D TCB (Tension, Continuity, Bias) interpolator as generic interpolator and Slerp (Spherical Linear Interpolation) for interpolating orientations.

The user interface for creating animations is implemented into Shape Editor. Shape Editor can be set to animation-state, where it adds user interface components specific to editing animations. These include timeline controller, buttons to manage keyframes, and a separate view for editing and adjusting interpolator curves. The view for editing interpolator curves is implemented with the same principle as higher level editors: it is not bound to any specific interpolator type, and because ontologies that specify new interpolators have to export code to use them, interpolator editor is able to edit all interpolators that are provided by ontologies. An example of the user interface is in Figure 25, where the user is creating a walking animation for a model of a robot.



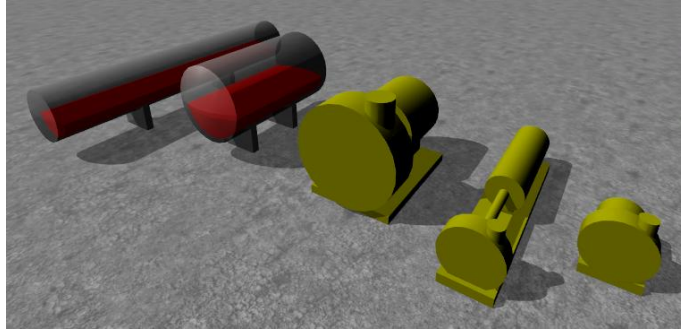
**Figure 25: The user interface for creating animations. The leg of the robot is selected and curve editor shows its animation curves. Buttons for managing animations and keyframes are beneath of the 3D view.**

### 6.3.3. Parameterization of geometry

Two types of parameterizations are supported: parameterization modelled as relations in the graph and code generated geometry. Parameterization made with relations contains two concepts: sizing parameter that is attached to a geometric model and relations that describe second order equations. These relations are used for linking sizing parameters and properties of geometry definitions, like position and size. When a sizing parameter of a model is changed, rules calculate new values of properties based on second order equations form of  $Ax^2 + Bx + C$ . In equation, constants A, B, and C can be customized for each property, sizing parameter is inputted as x, and solution of the equation is the property's new value. This kind of parameterization has a major restriction: each property of a shape can only depend on one sizing parameter and therefore only very simple geometric shapes can be parameterized.

Automatic parameterization of geometry is generated with an assumption that the size and the position of primitives are linearly dependent of sizing parameters. Therefore, it generates first order equations; in above equation constants A and C are set to zero. The user interface for geometry parameterization divides parameterisation process into tree steps: First the sizing parameter is selected, or new created and its value is set to a value that is used for calculating equations. Here the principle is that the user creates parameterization values such as they represent the model in its current state. In the second step the user selects shapes that are going to be parameterized, and in the third step he selects common properties of the selected shapes that

dependent to the selected sizing parameter. After that he presses a button that generates constraint equations. The second and the third step can be done and some times have to be done multiple times for the same sizing parameter, because in the third step common properties are selected, instead of properties of individual shapes. The user interface for parameterization is set to next of Shape Editor, and therefore the user can test how the generated parameterization works, and fix potential problems.



**Figure 26: Parameterized models of equipment. The tank contains fluid level changing animation, which is also parameterized.**

Figure 26 shows equipment, a tank and a pump, which are parameterized with the described procedure. The tank is composed from intersection of a cylinder and a sphere that rounds the ends of the tank, and two boxes acting as legs. Liquid inside the tank is also formed with intersection of a cylinder and a sphere, and in addition to that, the resulting shape is intersected with a box, which enables adjustment of the fill level. The pump is composed of multiple cylinders and a box. Parameterization of the tank could be done directly with the implemented user interface, but the pump was different case: sizes of motor, shaft, and volute chamber depend on each other, because linear mapping would make the shaft too long for larger pumps. Therefore manual adjustments to equation describing those sizes had to be made, so that the pump looks correct in different sizes.

ProConf supports attaching Java-code as part of an ontology and it is capable of compiling it runtime. Therefore a natural way to create “unrestricted” parameterized geometry is coding. When OpenCASCADE is already part of the system, the user has a powerful modelling kernel at his disposal, and so code based geometry generation is not restricted only to CSG-based modelling, but also sweeps, lathes and other more complex modelling methods can be used for composing geometry. The major restriction of code-based geometry is that there is no semantic information about the geometric structure and animations that modify the geometry cannot be used with them, at least with the current implementation. The problem is not attaching interpolators, but parameterizing interpolators themselves. Current parameterization implementation is

able to calculate parameterization of interpolator curves that change geometric properties, but only if geometric relationships are available.

## **6.4. Process Editor**

Process Editor contains features for modelling a plant, configuring simulation, and visualizing the simulation. We will introduce a new ontology for plant modelling, describe the system that handles pipe updates, user interface for plant modelling, and simulation related user interface concepts in the next four sections.

### **6.4.1. Plant modelling ontology**

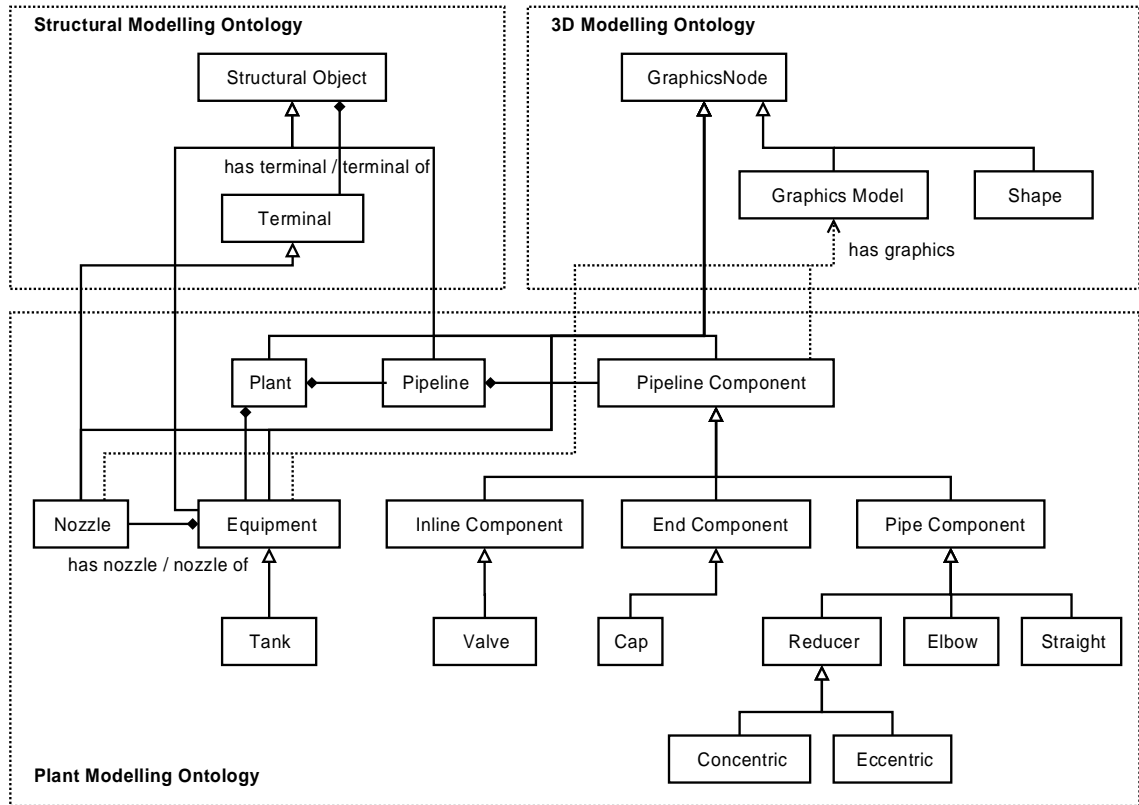
Plant modelling ontology is derived from 3D modelling ontology and Structural modelling ontology. 3D Modelling ontology is needed, because we are describing graphical 3D representation. Structural Modelling ontology brings various features, including easier mapping to other ontologies using it.

Similarly as with 3D Modelling ontology, we are not going to invent concepts of plant modelling ourselves, but use existing specifications and taxonomies as basis for it. In here we use combination of SmartPlant3D's and ISO 15926's names and concepts, but in very reduced form. For example, ISO 15926 defines thousands of concepts that are related to process industry, but using all of them would complicate our design without giving any real benefits.

Main concepts of Plant modelling ontology are Equipment, Nozzle, Pipeline, and Pipeline Component (Figure 27). Their basic usage is that equipment contains nozzles that are connected to each other with pipelines. Pipelines can be connected to nozzles and to each other with reducers and branch connections.

A pipeline is formed by one or more Pipeline Components. Pipeline Component is further divided into Pipe Component, Inline Component, and End Component. Pipeline Component, Equipment, and Nozzle are linked to their graphical representation using "Has Graphics" relation. This enables reuse of graphical components: for example, a plant may contain multiple tanks (instances), but each of them is linked to the same graphical representation of the tank. Equipment, Inline Component, and End Component are intended to be inherited for further specialization. For instance Tank and Pump are Equipment, Valve and Flange are Inline Components, and so on. Pipe Component describes basic components of pipeline: elbow, straight pipe, and reducers. These are already included in the ontology and no further classification is needed.

Difference between Inline Component and End Component is that End Component describes a component that is attached to the end of a pipe, connecting to only one other component, while Inline Component is attached to two other components allowing the pipeline to continue in a valid installation.



**Figure 27: Type diagram of Plant modelling ontology.**

#### 6.4.2. Pipeline modelling

Technically the most difficult part of the piping is modifying already routed pipe, because the structure of the pipes must be kept correct (Requirement P6). It means that all parts that are connected to each other before modifying the pipe must be connected to the same components after the modification, and their position and orientation must be updated so that all components are perfectly aligned to neighbour components and they do not overlap each other. These structural rules are listed into Table 2.

Just keeping components connected to each other will not suffice if editing is intended to be easy; designer's intentions must be taken account. Similar rules that were used for interpreting user's intentions in a sketching system Andler & Mendgen (1995) (Section 2.4.2), can be used in piping. These rules should reflect the requirements and typical scenarios in piping (Requirement P7). This includes that 90 degree elbows and 90 degree branches should be endorsed, be-

cause those are easier and cheaper to construct. Rules that are used for interpreting designer's intentions are described in Table 3. The basic principle with those rules is that the user places the equipment and nozzles where he wants them to be, and rules must not change their position. Rules IR3, IR4, and IR5 must be interpreted so that if the user modifies a pipe that was routed for instance along axis, that rule should not be forced. With opposite handling, the result could lead to modification of the whole pipeline because structure of the pipeline must be kept correct. Rule IR2 is also important because if new components are inserted into the pipeline without user conformation, it is most likely that the inserted component is something that the user does not want to appear in the design. Only case where components may be inserted automatically is a case where only way to fulfil the structural rules is to insert new components to a pipeline, or to fulfil logical rules and insert necessary components to a pipeline, for instance, insert flanges when a valve is inserted into the pipeline (Requirement P8).

**Table 2: Structural rules. These rules are for keeping structure of pipelines correct.**

<b>ID</b>	<b>Description of rule</b>
SR1	Connected components must be aligned to each other.
SR2	Inline components must be aligned to the path leg's direction.
SR3	Components may not overlap each other.
SR4	Components may not overlap branch connection.
SR5	Each inline component takes certain amount of space: straight part of the pipe must have enough length so that all components that are connected to it have enough space.
SR6	Inline components may not be inserted into elbows.
SR7	Unnecessary and invalid components should be removed automatically, including zero angle elbows and zero length straight pipes.

It must be noted that these restrictions cannot be met every time. Especially some of the rules that are used for catching user's intentions are hard to fulfil in many cases. When all the rules cannot be satisfied, it becomes very important to drop out right restrictions so that user's intentions may be interpreted correctly. Now the list of rules is ordered by importance level; rules that are dropped first are the last ones in the table. Only exception is the first rule (IR1), which must always be met, since it can be assumed that the user has placed equipment into positions where they should be. This adds another requirement for structural rules because the designer can place the nozzles in a way that there is no possible correct route for the pipe, and therefore rules should inform the user about bad design that they cannot fix.

These rules are also good basis for snapping, which can be used with translating pipeline components. But instead of using manual selection of components that are used in snapping, like

with translating equipment, components can also be selected automatically. For instance, if the user translates an elbow, the previous and the next elbow in the pipeline can be used for snapping the position of the translated elbow, and so the user can easily modify piping to go along axes or parallel to axis aligned planes.

**Table 3: Intention rules. These rules try to catch designer’s intentions with automatic structural updates.**

ID	Description of rule
IR1	Position of equipment and nozzles is never modified by the rules.
IR2	New components are inserted by rules only when there is no other way to satisfy structural rules.
IR3	If user models a pipe to go directly along certain coordinate axis, after modification the pipe should go along the same axis, if the pipe itself was not modified by the user (Requirement P7).
IR4	If user models a pipe to go parallel to a certain plane, after modification the pipe should go parallel to the same plane, if the pipe itself was not modified by the user (Requirement P7).
IR5	If branching pipes are perpendicular to each other (tee) after modification pipes should be perpendicular to each other, if the pipes were not modified by the user (Requirement P7).

When we think about geometry of pipeline, it can be divided into path legs that maintain the same run direction. When path leg is used for classifying components, it describes two types: path leg ends and inline components. Inline components are parts that always lie on a straight between path leg ends, which includes straight pipes, valves, and flanges. Path leg ends are components like elbows and caps. There are also components like tees that are both inline- and path leg end components, depending of the direction where the component is approached. This classification is different to previously described classification of equipment and pipeline components (Figure 27); the idea here is to describe geometric and structural classification, while previous one focused on the user’s point of view. Therefore it is easier to divide the piping structure to two parts: Structural rules that keep positions of pipeline components correct: valves and straights between path leg ends, and so on, and other set of rules that keep logical structure of pipelines correct (Requirement P8): valves are connected to flanges that are connected to other components, and so on. Those rules are fundamentally different because structural rules are required to update real-time, since the designer must be able to see changes in the pipeline in order to do interactive designing (Section 2.2.3). Logical rules are tested only once, either preventing the user doing wrong things or automating modelling by inserting necessary compo-

nents, for example, inserting flanges automatically when the designer inserts a valve into a pipeline.

In order to keep those rules differentiated, another structure in addition to equipment and component structure is needed. This structure is the control point structure, and it is based on the idea described in Section 5.2.1: piping is edited similarly as editing polygons. The control point structure is tightly linked to the component structure, describing positions and orientations of components. Ideally all position modifications are done on the control point structure level, and when it is changed, the component structure is updated. Those structures must be linked both directions, because when control points are positioned, sizing parameters of components are needed from the component structure.

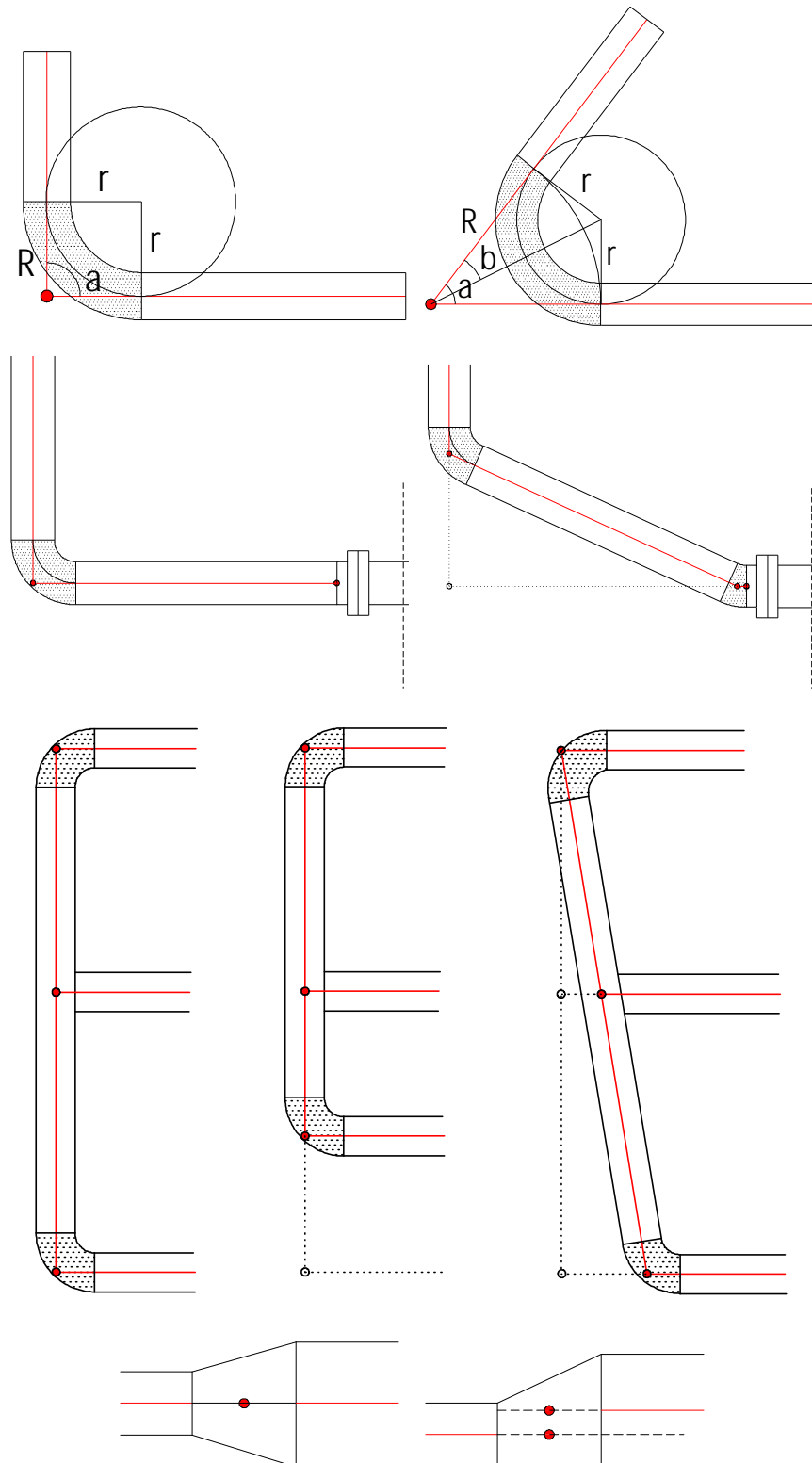
Using above classification, we divide control points to two types: *Path Leg End control points* and *Inline control points*. As described, Path Leg End control point is the base type for all control points that are in path leg ends: elbows, end components, branch ends, and pipe ends. Inline control point is the base type for control points that are used keep components and branches connected to a straight pipe. It is aligned directly between path leg's ends. There are several policies that are used when Path Leg End control points are modified and those depend on what type the control point is. We will list all specific control point types next, and type diagram of control points is in Figure 29. Examples of control point usage are in Figure 28.

*Turn control point:* Turn control point (Figure 28, top) is used with elbows. This control point connects two path legs together. In order to fit elbow's geometry, next and previous component will have a certain offset from the control point. The offset is calculated using angle between path legs' directions.

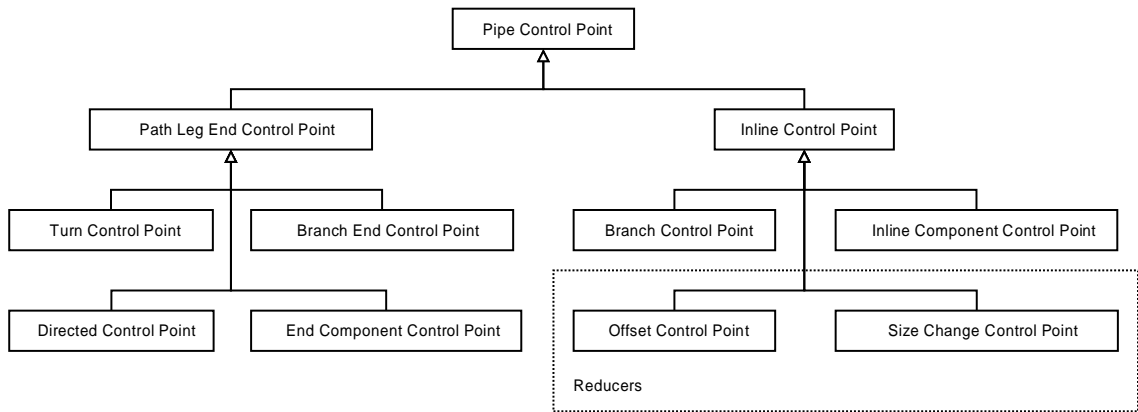
*Directed control point:* This type of control point is used when a path leg's direction cannot be changed, for example, when a pipe is connected to a nozzle. Using a directed control point requires that the path leg's other end must be in the position where directed control point points to. An example of this can be seen in Figure 28 (second from top), where a turn control point is moved upwards and a new turn control point is inserted to satisfy the restriction of directed control point.

The same figure can be used for visualizing another case: if the user moves the piece of equipment instead of the elbow, the pipeline can be updated two ways. On the right, the style is same as previous: generate new control point and add an elbow to the pipeline and leave the elbow on the left side intact (right). On the other hand the user may want to keep the elbow aligned to the

next straight pipe and avoid insertion of a new elbow. Therefore the second style is to move the elbow on the left side so that it is aligned with the nozzle and run direction does not change.



**Figure 28: Different control points and how they behave on modification. Turn Control Point (top), Directed control point, Inline Control Point and Size Change- and Offset Control Point (bottom).**



**Figure 29: Type diagram of Control Points**

Two types of control points are inherited from inline control point: *Inline Component control point* and *Branch control point*. Inline Component control point is used when an inline component, for example, a valve is placed into a straight pipe. Branch control point is used with branches. These cases require separate handling because one could add several branches into the same position, but components cannot overlap each other. Another reason is that when inline component is moved, updating rules need to take account only what is on the same path leg, but with branches also every branch's path leg and their components can affect the decision of the updated position. This is demonstrated in Figure 28 (second from bottom), where the branch's position in the modified path leg is calculated so that the branching path leg's direction stays vertical in both cases: when the lower elbow is moved up and down (middle figure) and when it is moved left and right (right figure).

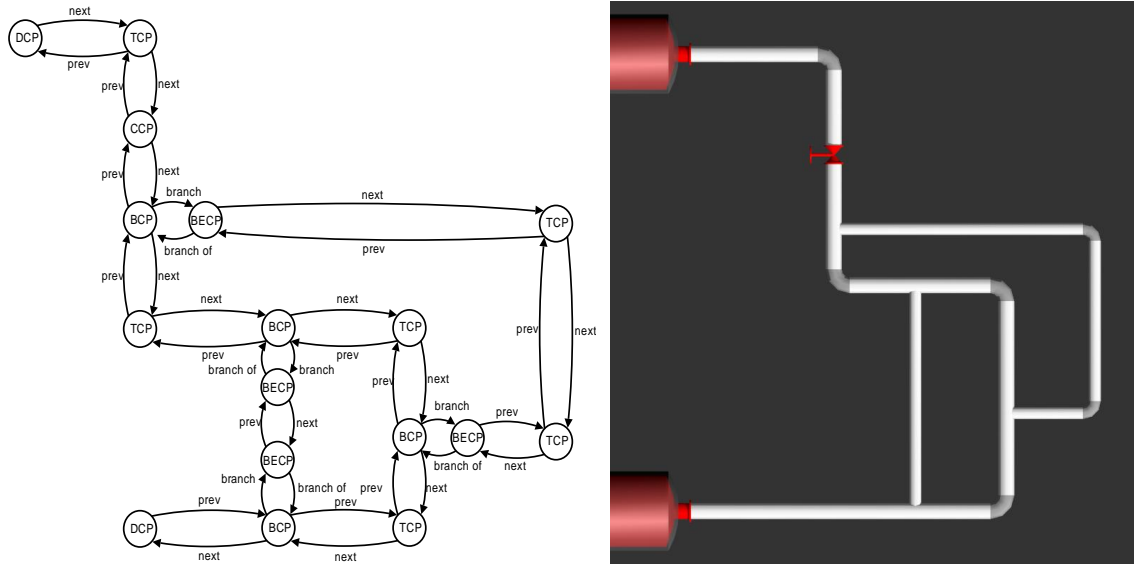
There were two different solutions that could be used with branches: either use separate "branch" relation with next and previous relations or duplicate branch control point for each branch. In the first case pipeline system with all branches would contain only one control point structure but traversing the structure and reasoning what control points belong to which pipeline would be very complicated process. On the other hand the second case requires special attention with the branches, because coordinates of all control points must be kept same. After some testing, the second approach was considered easier to handle. Therefore Branch control point also has a pair: *Branch End control point* that is used from branching pipelines side and it is inherited from Path Leg End control point. The idea is that for each Branch control point there is one Branch End control point for each branching pipeline, and all those control points are kept in the same position.

Like braches, also reducers need special attention. Two types of reducers exist: concentric and eccentric (Figure 28, bottom). Concentric reducer is simpler case of the two because it keeps the centreline of the pipe and the path leg intact. Eccentric reducer is more complex because the centreline of the pipe changes. To further complicate the situation, the reducer can be rotated and if the path leg's ends are kept in same positions, the path leg's direction changes. To solve this problem, instead of using one control point, reducers use two control points: *Size Change Control Point* and *Offset Control Point*. The position of Offset Control Point is always calculated relative to Size Change Control Point using required offset that depends on pipe diameters on both sides of the reducer, and the rotation angle of the reducer. When the control point structure is traversed, depending on the side the reducer is reached, traversing ends to Size Change Control Point or Offset Control Point. Using this mechanism avoids creating special case for straight parts, but also complicates changing control point structure, since when a control point is inserted (or removed) next to reducer's control points, both reducer's control points must be re-linked. A reducer acts as a boundary between two pipelines because their diameters are different. For easier handling, Size Change control point is kept as other pipeline's control point and Offset control point other pipeline's control point.

Last type of control points is *End Component control point*. Similarly as inline components, also end components must have their own control points. It is used with components like caps. It does not have any restrictions for its movement, and it calculates orientation from its angle property and position of the adjacent control point.

An example of a pipeline and its control points are in Figure 30. All control points are linked to each other with "next" and "previous" relations with an exception of branch control point, where all branches are linked with "branch" and "branch of" relations. Straight pipes are attached to two adjacent control points. The length of a straight pipe is calculated from the distance between control points and using offset value in control points, which prevents straight pipes from overlapping inline components and elbows.

When the user modifies piping parameters it is important to find correct set of control points that must be updated. Current implementation of the Piping Rules (Figure 31) assumes that only one control point is moved at the same time: modifying multiple control points at the same time can be handled with the same rules, but more work has to be done because the rules must be applied to each modified point separately.



**Figure 30: A sample pipeline and its control point structure. DCP: directed control point, TCP: turn control point, CCP: Inline Component control point, BCP: Branch Control Point and BECP: Branch End control point.**

Pipe updating rules are run when a change in the control point structure is detected. First is checked if the changed control point is inline or path leg end control point. If modified point was path leg end, all path legs connected to it must be updated. Path legs can be divided into three types; ones with freely modifiable ends, like elbows (free update), ones with one freely modifiable end (directed update), and ones whose both ends are locked into their positions (dual directed). In practice this means that depending of how many nozzles the path leg is connected to, updating rules are different.

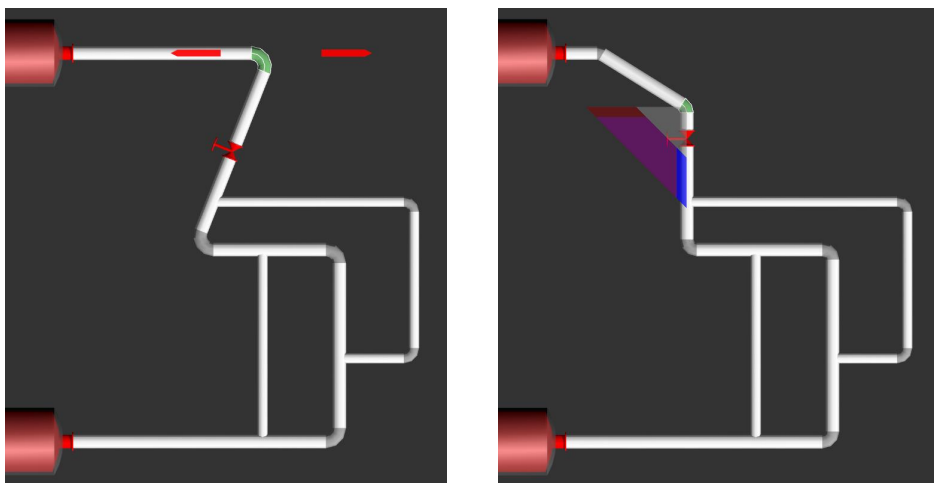
Since the rules must take account that elbows must be removed when their turning angle goes to zero, because zero angle elbows cannot exist (SR7), both free update and directed update must check if elbows can be removed. Removing is tested with recursion: algorithm tries to remove an elbow whose turning angle is near zero, and then checks if the resulting path leg (removing of an elbow joins two path legs together) is valid by using the same path leg updating procedures. Removing of an elbow may only fail when at least one of the new path leg ends is directed; if checking of alignment of directed ends fails, recursion takes one step back and then proceeds finalizing the path leg.



path leg is to insert two elbows; one elbow for each directed end. When a directed path leg is updated, the procedure depends on the path leg end that the user changed: if the user moved directed end, the other (free) end can be moved to proper position, but when the user moves the free end, a new elbow must be inserted. When a free path leg is updated, insertion of new elbows is not necessary. These update policies take account of rules IR1 and IR2, which state that nozzles cannot be moved and new components should be inserted only if necessary.

After creating new path legs or moving free ends, all modified path legs must be finalized by updating run ends (for example turning angle of elbows), and then updating all inline control points on the path leg, so that components are aligned to each other (SR1 and SR2). If a finalized path leg contains branches, each branching path leg is updated using above procedure. In this step, also directions of branches' path legs are taken account so that their run direction is kept unchanged if possible, which helps usability (IR3, IR4, IR5). At the same time, components' sizes can be taken account so that components will not overlap each other (SR5 & SR6), but when there is not enough space for components, only proper solution what could be thought of is to inform the user about the problem.

Two examples of pipe updates are in Figure 32. When the elbow on the top is moved from its initial position (Figure 30), path legs from both sides of the elbow has to be updated. Other of the path legs is directed path leg, because the pipe is connected to a nozzle. When the elbow is moved to the right (left figure), it is still aligned with the nozzle, but when it is moved down (right figure) a new nozzle has to be inserted. When the nozzle is moved back to its original position, the inserted elbow is removed. The designer's intentions are also take account; independent of how the elbow is moved, the branch keeps its run direction (horizontal pipe in the figure).

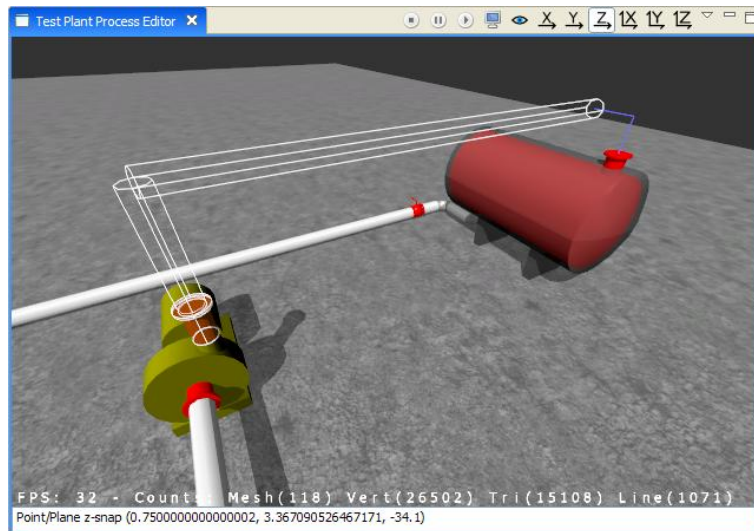


**Figure 32 : Automatic update of pipeline. When elbow is moved right, path legs are updated (left). When elbow is moved down, new elbow is created (right). Initial position of moved elbow is in Figure 30.**

#### 6.4.3. Plant modelling user interface

Process Editor shares part of the implementation with Shape Editor. Gizmos used for rotating and translating equipment are the same as were used in Shape Editor to move shapes. Similarly as with Shape Editor, proper actions that user can do, depend on selected objects. For example, if a tank is selected, UI shows actions to translate and rotate the object, remove the object, and focus on the object (rotate camera towards it), but when the user selects a nozzle, in addition to those actions the user has ability to route pipe, if the nozzle is not connected to a pipe. If the user translates or rotates an inline component, like a valve, gizmos are different than with standard equipment; the user may translate an inline component only along the pipe where it is connected and rotate it around the pipe. As in Shape Editor, user may select multiple pieces of equipment and move all of them at the same time. With inline components the case is different: Only those components that are on the same path leg can be moved or rotated at the same time.

To help pipe routing, the user is given option to lock one of the axes (X,Y,Z) and route pipe along that axis. Also routing parallel to axis-aligned plane (XY,XZ,YZ) is possible (requirement P7). To align a pipe to other pipe components and equipment, the user may click them and add them to snapping list. The snapping list contains all objects that are used for snapping. If the snapping list contains objects, pipe routing tries to snap to given positions or axis aligned planes defined by a point. For instance, the user may add a nozzle to snapping list, start routing a pipe from another nozzle towards the nozzle along a certain axis and tool will snap the position when coordinate value in selected axis of the snapped nozzle and the routed pipe is nearly the same. Example of this is in Figure 33, where the user is routing a pipe along Z-axis, and routing snaps to position of a nozzle. That way the user may easily route pipe that uses only 90 degree elbows. Another feature of snapping is that if a straight pipe is added to snapping list, pipe routing may be set to route pipe exactly the same direction as the straight pipe in the snapping list.



**Figure 33: Routing a pipe. Pipe routing snaps to position of the nozzle and it is highlighted with blue line. Routing is restricted to Z-axis, and Z-axis button in toolbar (top) is pressed down.**

Moving of elbows of already routed pipe is helped in the same way. When the user starts to move an elbow, the next and the previous path leg's end is automatically added to the snapping list and moving of the elbow can be snapped similarly to the same coordinate value of one of the main axes, if it is moved along an axis with the gizmo. When the elbow is next to a nozzle, movement can also be restricted into direction of the path leg. With restricted movement, the user can avoid a case, where piping rules must insert a new elbow to the pipeline (Figure 32, left).

Moving of branches is implemented similarly as movement of inline components: The user selects a straight pipe and gizmo appears over branch connection. Gizmo is the same as used with inline components, and behaves similarly, but adds the next elbow of the branching pipeline automatically to the snapping list so that 90 degree angles are easier to create.

#### 6.4.4. Simulation configuration and visualization

As explained in design chapter (5.2.3), creating an adequate user interface component for configuring simulators is difficult tasks. Therefore there is no generic implementation for that. In section 7.1 we will show an example simulator, and present how it uses Eclipse's Extension Point mechanisms to add its own user interface component to Process Editor.

Since visualization of equipment and flows in pipes require different amount of information, we have divided them to two different components. Configuration of equipment visualization goes as follows: First the user selects type of equipment and then the user interface shows all avail-

able animations for that type. When the user selects one of the animations, the user interface shows a list of simulation properties that can be used for driving the animation. Property selection is based on Ontology Explorer (Section 4.3) and each simulator must provide viewpoint for showing its data structures.

The pipe flow visualization is based on particle effects (2.5.2). Particle effects give ability to visualize both mass flow and some other property, mapped to colour of particles. Choosing particle effects was easy: jME (6.1.1) had already implementation of particles and using them required only little effort. Other choice would have been arrow glyphs, but implementing them would have taken more time. Configuration of flow visualization is similar to configuration of equipment. Difference is that the user does not select equipment type or animation, but he must select a simulation property used for particle velocity and another simulation property used for particle colour.

To represent precise numeric information in textual form (requirement V5), yet another user interface component is required. This component is called monitor. Since both equipment and flow visualization configuration rely on Ontology Explorer, it is natural to use the same method, but here the user may select multiple simulation properties that are shown in monitors. These configurations are stored per object type: For instance, tanks have different properties than pipes, and the user must be able to configure monitor for both of them.

## 7. Analysis and Discussion

This chapter presents analysis of the design and the implementation, but first we will represent a small example case of plant modelling and simulation, where we model a pulp bleaching line. This example is used as basis for our analysis. Last we will present some thoughts about future development ideas and challenges.

### 7.1. An example of simulation configuration and visualization: bleaching line

As practical use case, Process Editor was used for modelling a bleaching line. Bleaching line is part of a pulp mill, and its purpose is to improve brightness and cleanliness of pulp. In practice bleaching is removing residue lining from the pulp: Previous stages of pulp processing remove most of the lining, but some of it is still left there. In addition to that, previous stages dye the pulp to dark brown because of chemicals they use for lignin removal. Kappa value is used for describing amount of lignin, but in practice the number describes pulp's ability to consume permanganate, which correlates with the amount of lignin.

For this case, its own ontology had to be developed. Ontology had to contain equipment that was needed to model the process (bleaching tower, washer, and boundary component) and mappings, which generated simulation model from 3D plant model. Simulation model itself was divided to multiple ontologies, one describing basic flowsheet ontology, one multiphase chemistry ontology and one specific to this simulation case, dubbed as Vista ontology. Name Vista comes from the name of project where multiphase chemistry simulation algorithms were developed (Brink et al. 2007).

Flowsheet ontology describes basic concepts of flowsheets: flowsheet contains nodes that have input and output terminals, one input- and one output terminal can be connected with a stream, and so on. Similarly multiphase chemistry ontology describes concepts needed there: multiphase system has multiple phases, one phase has multiple species, one species has molar flow rate, and so on. Vista ontology uses both of those ontologies to describe simulation model by defining mixer, splitter, flow and bleacher. Mixers and splitters are used in describing internal simulation model of washer, and it is modelled using ProConf's diagramming tools. Bleacher has its own type since it is simulated using special simulation code, tailored to calculate pulp behaviour inside bleacher. Complete description of the used algorithm is in (Räsänen 2003).

Before mapping between 3D plant model and simulation model could be created, simulation models had to be linked to equipment models. Therefore simulation model of the washer had to

be modelled with diagramming tools before ontology for the test case could be created. Another problem with mappings between models is that the 3D plant modelling ontology does not have concepts for input and output terminals, and there is no information, how flowsheets' ports can be matched to the nozzles. Since it is not possible to solve this automatically, a dialog for selecting port type is shown to user when a nozzle is added to a piece of equipment. This approach is not an optimal solution, because user interface cannot directly show the made choice afterwards. The actual simulation model is generated by cloning flowsheets attached to equipment, and then linking ports in flowsheets to nozzles when the nozzles are connected to each other with pipelines. As a summary, one piece of equipment is always mapped to one flowsheet, and one pipeline to one flow.

Simulator was implemented as extension to Simantics platform, and it is able to initialize simulation from generated simulation model. The simulator is a steady state simulator and it calculates new values only if the user changes input values (input pulp and water). The input values can be configured through special inline components, which are named as boundary components. They are inserted to pipe ends. Simulation configuration can be edited through a dialog that contains an Ontology Explorer. The dialog was attached as part of the simulators ontology.

Visualization of simulation is in Figure 34. Pulp enters the process from the left, travels each bleaching tower upwards, and leaves the process from the washer that is the last one on the right. Water used in bleaching moves opposite direction, travelling from washer to washer from the right to the left. Snapping features that were designed and implemented worked as planned. All pipes that are connected to bleachers on the same side are the same length, because snapping could be used so that previously routed pipe was added to snapping list. Figure also shows the effect of routing pipe along the axes: All pipes are either horizontal or vertical, and all elbows are 90 degree elbows.

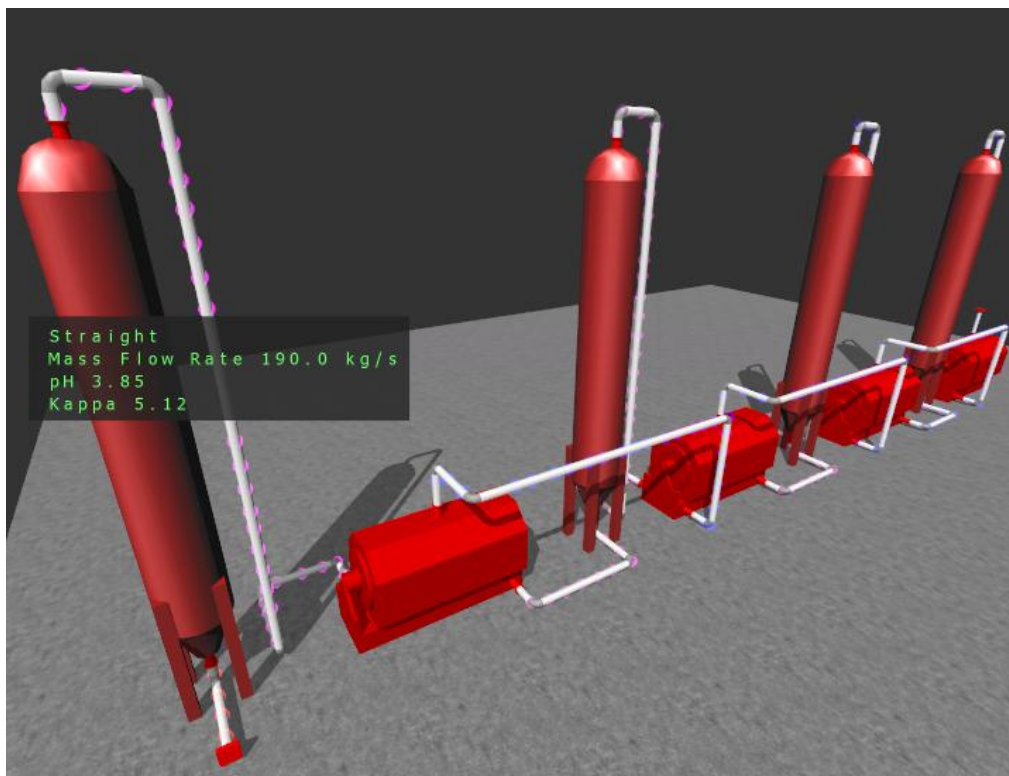
Mass flows of the process are visualized with particle effects, and kappa number is connected to colour of particles; red being high kappa value and blue low kappa value. Effect of the first reactor can be seen right from the image: particles are red when they enter the reactor and particles coming out of the reactor are violet. Difference of mass flow rates is much harder to see, even when pulp's mass flow is 190 kg/s and water's mass flow is 295 kg/s.

The problem of particle effects shows in the figure; particles are very hard to see at large distances. Another problem, which cannot be seen in the figure, is frame rate dependency of the particle effects that was mentioned in Section 2.5.2: When computer cannot render visualization fast enough, not only seeing velocity of mass flow is hard, but also seeing direction of mass

flow is impossible. This problem was hit when visualization was used in a laptop that used power saving mode, and its graphics processing capability was significantly reduced.

Functionality of the text based monitors appears in the figure. When the user enables monitors, simulation data from equipment that user is pointing is shown. Monitors can be configured, and here it is set to show mass flow rate, pH, and Kappa value of flows.

Test case showed that implemented animation features are not enough for all cases. The simulation model of reactors can provide Kappa in multiple points of the reactor. Simple colour change animation can change the colour of the whole object, and cannot create a colour gradient over the surface of model. Now Webomon's 2D curve and 3D tred plots were the only way to visualize kappa value's change inside reactors.



**Figure 34: Bleaching line, Reactors (Bleaching towers) are the tall objects; Washers are the smaller ones. Mass flow and cleanliness of pulp is visualized with particle effects. Exact information of the simulation can be retrieved with textual monitors.**

## **7.2. Evaluation against requirements**

In this section we will check what requirements we met and what requirements we did not met, and why that happened.

### 7.2.1. Equipment modelling and animation

When we look at the requirements for equipment modelling, we see that all high priority requirements are met. Geometric models are reusable, and when a piece of equipment is inserted into a plant, the system creates reference from the equipment to the geometric model. Whenever the geometric model is changed, equipment models in the plant are updated. Also geometric models can be animated and new animations can be added to models of equipment, and they are immediately usable in visualization.

There were also two medium priority requirements for Equipment modelling: parameterisation of geometric models and usability of modelling. The implementation contains two ways to create parameterized geometries, and both of them have good and bad sides. Semantic presentation of parameterization can be used with the Shape Editor, but in its current form it is difficult to use, and it has severe restrictions for the geometry. Code based geometry gives the user complete control of parameterisation, but the current implementation does not support animating geometric features. With some effort both of them could be made more useful, but in ideal case, the process modeller does not have to create parameterizations himself, because the application should contain models for all common equipment.

### 7.2.2. Plant modelling

All high priority requirements for equipment placing are met: new equipment can be inserted to a plant, equipment can be translated and rotated freely, and sizing parameters can be adjusted. Also nozzles can be inserted to equipment, and translated and rotated using either world coordinate system or equipment's coordinate system. Missing features are in equipment placing relative to each other, and custom coordinate systems are not implemented. Therefore requirement E6 is partially met.

Likewise all high priority requirements of pipe routing are at least partially met. Routing a new pipeline works as designed, and an existing pipeline can be edited by translating elbows, which causes rest of the pipeline to be updated, keeping the pipeline's structure correct. The speed of the algorithm is faster than expected: moving an elbow that causes updates of several dozens of path legs, because of large amount of branches, works interactively.

Pipeline routing and modification also supports snapping, which allows routing a pipe relative to equipment and other pipes and their components. While this feature's usability leaves things

to be desired, like highlighting objects that are selected for snapping, it still helps pipe routing process a lot.

Inserting and modifying inline components works as designed: precise placing of components is difficult, but since the purpose was to design a user interface for process simulation, precise positioning could have made the interface more complicated to use.

There are still quirks left in modification process: elbows can be translated too close to each other, and the end result is invalid piping. While this goes according to the design, rules are trying to calculate proper angles for elbows and lengths for straight pipe parts and the result is quite odd looking structure. Also automatic deleting of unneeded elbows is problematic, because rules that implement the functionality is not connected to the user interface in any way, and the user may not notice deletion of an elbow, or the elbow is deleted without such intention. Another problem caused by disconnection of rules and the user interface is that there is no way to inform the user about bad design: When there is not enough space for inline components, they are put top of each other. This distinction has also good side: Pipeline and its properties can be modified even when the editor is closed, and structure is still kept correct.

#### 7.2.3. Simulation and visualization

Requirement S1, which states that the simulation models must be generated according to 3D plant model, is harder to evaluate. This is, because mapping between plant model and simulation model must be implemented separately for each simulator. In the example case of bleaching line (7.1), mapping worked quite well. The only problem with it was that mapping code had to ask if a nozzle is input or output nozzle, and what material passed through it. This implied that there was no way to generate simulation model for already created plant model, at least automatically.

Another simulation requirement was about adding simulator specific user interface components (Requirement S2). As explained in the example case, this is possible. What has not been thought is a case when the same plant model is linked to multiple simulation models. Currently the user interface components of all simulators would be added to the user interface, which could confuse the user.

While the animation system does not fulfil all requirements that were set to it, it provides configurable environment, where the user can freely select properties that he wants to visualize and

how he wants to do that. Minor restriction in current implementation is colour changing animations of equipment, because they must be created separately for each equipment model.

Things that were left out are multiple active animations per object and configuring animations per object. Both of these are somewhat complicated in sense that if they were implemented, the user interface could get more complicated. Also automatic binding a certain animation to a certain simulation value is not possible because semantic information for that kind of binding has not been modelled into ontologies. This helps the person who creates equipment models, because he does not have to bind animation to specific behaviour, but makes visualizations harder to use, because the user is responsible to binding animations to simulation values.

Dynamics of flows inside pipes can be visualized (Requirement V4), but the problem is that the user cannot create new visualizations and is bounded to existing particle-based implementation. To cope with the performance problem that we hit in the example case, arrow style glyphs should be tested how good they are in visualizing flows.

### **7.3. General analysis**

#### **7.3.1. Usability**

Over traditional 2D diagramming user interfaces, which are used for creating simulation models, 3D user interface is more complicated to use. Among simulations model creators experience over 3D modelling varies, and those with no previous experience seem to have a lot of problems with the 3D interface. Just moving around a 3D-model and controlling rotation and zooming of camera was troublesome among those who used the 3D interface for the first time. Especially problematic was positioning of an object accurately because perceiving position of objects related to each other and related to world was hard. Selected objects with see through edges confused users and made them think that object was front of everything, even when it was behind other objects.

At the same time it is easy to see that creating a simulation model using 2D diagramming tool is faster than creating the model with 3D modelling tool. But power of 3D modelling is that sizing parameters like length of a pipe and height of a reducer, that are required by some simulators, can be calculated using 3D model, while in 2D interface they have to be inputted manually. This feature can be problematic at times, because simulators approximate actual physical and chemical phenomena in the real world, sometimes requiring unrealistic input values to calculate correct results. Then there are cases where 3D model is required for simulation. When CFD (Computa-

tional Fluid Dynamics) simulation is combined with large scale dynamic process simulation (Pättikangas et al. 2006), 3D geometric data is required by CFD simulation algorithms.

While plant design and layouting were not the aim, those are most common, if not the only case where 3D modelling is currently used. Implementation is far from usable in those cases, because it lacks functionality for modelling of structures, more precise tools for pipe routing, and so on, but as stated in requirement analysis, it was not our aim.

### 7.3.2. Scalability

Typical plants may contain thousands of objects, which creates burden to visualization engine. But at the moment, bottleneck of visualization is not the graphics side, but the Simantics environment itself. With very simple simulation model of the example case of bleaching line (Section 7.1), which has eight pieces of equipment and twelve pipes between them, database contained over 146.000 triples. The 3D plant model does not consume many triples, at least when compared to the simulation model. The used simulation ontology defines complex properties of multi-phase chemistry and the simulation model of a washer takes over 7000 triples. In the example, there are four washers and so just washers alone use almost 30.000 triples. When the system contains just required ontologies, size of the database is 87.000 triples. With ontologies and 3D plant model size of the database is 101.000 triples, and so the 3D model takes 14.000 triples and the simulation model with mappings about 45.000 triples. Hence in that example, generating the simulation model increases size of the whole model four times over plain 3D model.

While 146.000 triples is not much, it slowed down the implementation platform noticeably. The problem is that current ProConf implementation aggressively caches everything, but never releases anything. In top of that, in the current application architecture each component that accesses triple graph needs its own cache. When effects of these are added together, memory consumption increases rapidly. Especially problematic is current JBoss rule engine that the current piping system uses: it has to read the whole database in order to work.

## 7.4. Future Work

One of the key issues of plant modelling is at the same time one of the key issues of Simantics platform: scalability. As the example case of bleaching line shows, ontology based approach creates large amount of triples even for simple models. The future vision for Simantics platform is to use it as a user interface for Apros and Balas, and that sets great goal for scalability: cur-

rently Apros has been used for simulating pulp mills and nuclear power plants, which may contain hundreds, if not thousands of equipment, and many kilometres of piping. When all that is described in ontological format, database will contain several hundred million triples. Therefore example test case is just a tiny fraction of what the system should be capable of handling.

Work for this has already been started. Unnecessary relations have been removed from Layer0, and that reduces amount of triples needed by the models. Caching algorithms of ProConf have been re-implemented: Now it contains only one cache and it can also release cached triples. There has also been discussion of packing properties; instead of storing all relations, structured property is stored using arrays. This would reduce amount of triples especially with simulation models. Also clustering capability for ProCore has been implemented. This allows distribution of triples to multiple servers, reducing workload of one server.

If those means solve the scalability issue, the next limit will be graphics. Designed ProConf's caches will allow it to handle larger models that will fit to the computer's memory. When a whole industrial plant is modelled, that is also required from graphical editors. When user is modelling the plant, it is more practical to filter unnecessary objects from the view, similarly as SmartPlant3D does, but when whole plant level simulation is visualized, the case is bit different. It depends on the user and what he wants to do: he may want to simulate the whole plant but is interested one section of it. Then similar filtering can be used, but when the user wants to just browse the plant and see everything, dynamic loading and unloading of graphical models is required.

Another major issue of the platform is usability. While Simantics platform has users on multiple levels, including developers and ontology creators, plant modelling is aimed at users that use the platform and its tools to model something and then simulate it. For them the platform is currently too complex to use. For instance when the user wants to model a plant using Process Editor and he starts with empty installation of ProConf, first he must load ontologies to the system, and then load all rules and start the rule engine. After that, he can create a new plant and start modelling. These steps are currently too difficult for average users, and must be polished. In ideal case user never sees that the system uses ontologies and semantic graph.

All in all, one sentence describes well the future of Simantics platform:

*Simantics is ready when it says it is ready. – A. Villberg*

Usability of 3D modelling, creating equipment models and designing plants is a challenging task. When design is based on real world requirements, the user interface must implement several features that complicate the design process. On the other hand, current process simulator users have accustomed to 2D diagramming tools, and have hard time even with the basic concepts of 3D modelling. Therefore testing Houde's method (Section 2.3.1), which restricts orientation and movement only valid positions, could be beneficial, because it would let the user to do only valid designs. This would set other restrictions to the system, since usually plant equipment is at different levels, on top of structures, and therefore modelling of structures would be required. Also current user interfaces, while they contain some good usability enhancing features, require a lot of polishing before they can be used in productive work.

One of the key issues in plant modelling is piping rules. The current system does not support all components that can be found in piping systems. These include tees and other components that have more than two connection points and components that change path leg's direction. Current implementation took two iterations to produce: after first iteration the system did not support reducers and could not fulfil structural requirements all the time. In this process, at least one iteration is needed, so that piping rules can be brought to adequate level.

With the current practices (2.2.1), creating a 3D model of a plant is done in the later stages, where multiple diagrams and simulation of the plan has already been done. Therefore it is hard to say, how usable the simulation is in the design process. If usability of 3D modelling can be brought to higher level, it is possible that 3D modelling could gain ground among process modellers.

#### 7.4.1. Other use cases for 3D modelling and 3D Plant Model

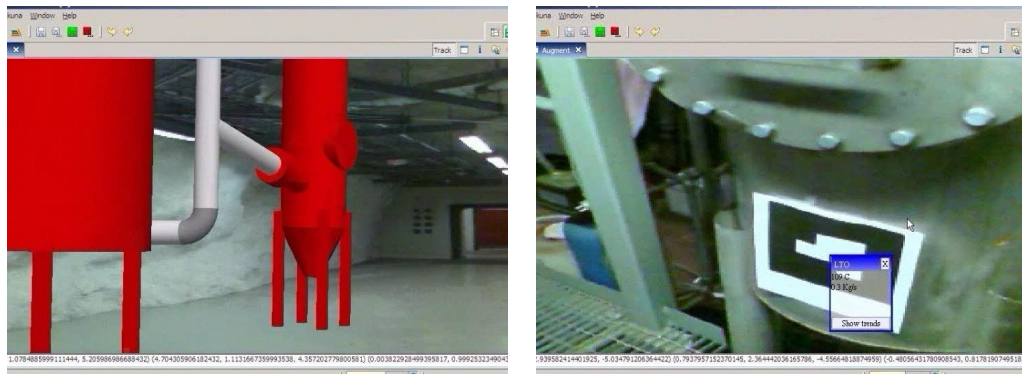
Within research project, where this thesis was made, the main topic was a portable device for maintenance workers. It was found that it would be useful to identify plant equipment from a video stream, and then allow the maintenance worker access information about the equipment in the picture. Few examples of information that was considered useful were maintenance history, manual, and current status of the equipment.

Ontologies allow combining different data models. This also applies to the 3D plant modelling ontology, and so any kind of information can be linked to a 3D plant model, including all the information that maintenance workers need. Because of the data model, there was no need to take account this use case when 3D plant modelling ontology was designed.

In the first stages of the project, the equipment was identified by markers (2D barcodes) (Figure 35. right side), and when a marker was detected on the camera's picture, information about equipment, linked to the marker was show to user. The major problem with this approach is that the user has to see the marker before he can get information about equipment.

Later markerless tracking was added. It detects changes in the camera's picture and calculates new orientation of the camera, while tracking of movement is difficult and unreliable. Attaching position and orientation information to each marker allows augmentation component to calculate the camera's position relative to 3D model from any marker. When this is combined with markerless tracking, the position of camera can be calculated from one marker, and after that, the user can rotate in place and see augmented information about equipment around him.

Two different ways to use 3D plant models were identified: either augmenting 3D model to empty hall, so that positions of equipment can be seen even when nothing has been built in the real world (Figure 34. right side). Other way is to hide the 3D model from visualization, but use it to detect what equipment user is looking at, and augment relevant information to the picture (left figure). More detailed description of how augmented reality is used in ProConf can be found in (Siltanen et al. 2007).



**Figure 35: Augmenting 3D model to video stream (left) and augmenting information about the process (right).**

Another case that tests reusability of the 3D modelling ontology is ongoing work of creating mechanical simulation into Simantics platform. It requires a user interface for modelling rigid bodies and attaching them together with joints. The scenario is different to both equipment- and plant modelling: mechanisms that were used with those, will not be sufficient for precise positioning. How well current ontologies and implementation fits to this case remains to be seen. Ideally the same features can be shared among all tools, which reduce effort of implementation.

## 8. Conclusions

The industry uses simulators to test and verify designs. Modelling and designing applications tend to be different than simulator applications, and that leads to manual input of design data to simulator. Combining the design application and simulators into a single package provides means to tackle this problem. The designer can use simulators without extra effort and without information loss in the translation process.

3D industrial plant modelling has been widely accepted by the industry, and most of the new plants are designed with some 3D modelling software. When compared to traditional 2D process simulation tools, 3D graphics provide more lifelike view of the plant that benefits process modeller.

We studied how 3D industrial plant modelling, process simulation and visualization could be combined with ontologies. Ontology-based approach was proven useful mechanism to break coupling of the user interface and the actual simulator. The visual model is separated from the simulator model, but since both models are in the same database, simulation can be configured through the 3D model. Therefore different simulators can be easily added without changing anything in 3D modelling and visualization.

Used visualization techniques provide means to better understanding of behaviour of a simulated process. When compared to traditional text based monitors and graphical trends, animations are more intuitive and faster to interpret. When more precise information is needed, older methods are better; animations should not be seen as replacement, but as additional way to help the designer.

Easy extendibility comes at its cost: semantic triple model takes large amount of space, and requires lot of caching to work fast. This creates huge memory requirements and scalability of the method is an issue that represents a future challenge for whole Simantics platform.

## References

Anderl, R. and Mendgen, R. 1995. Parametric design and its impact on solid modeling applications. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications* (Salt Lake City, Utah, United States, May 17 - 19, 1995). C. Hoffmann and J. Rossignac, Eds. SMA '95. ACM Press, New York, NY, pp. 1-12.

Apros – The Advanced Process Simulation Environment. [Cited 10 Apr 2007]. Available at: <http://apros.vtt.fi/>

Balas Process Simulator Software. [Cited 10 Apr 2007]. Available at: <http://balas.vtt.fi>

Bier, E.A., 1986. Skitters and Jacks: Interactive 3D Positioning Tools. *Proceedings of the 1986 workshop on Interactive 3D graphics*, 1987, ACM Press New York, NY, USA, pp.183-196

Bier, E.A. 1990. Snap-Dragging in Three Dimensions. Technical Report. UMI Order Number: CSD-88-416., University of California at Berkeley

Bittner, J., Wimmer, M., Piringer, H., Purgathofer, W., 2004. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum* (Grenoble, France, September, 2004)) EUROGRAPHICS 2004. Vol. 23, Num. 3. pp. 615-624.

Blender [Cited 10 Apr 2007]. Available at: <http://www.blender.org>

Braunschweig B., Gani, R., 2002. Software architectures and tools for computer aided process engineering. – (Computer aided chemical engineering, 11), Elsevier, Amsterdam, Netherlands, 700 pages. ISBN: 0-444-50827-9.

Brink, A. Lindberg, D., Hupa, M., Louhenkilpi, S., Wang, S., Fabritius, T., Riipi, J., Härkki, J., Kangas, P., Koukkari, P., Lilja, R., Pajarre, R., Penttilä K., Kankkunen A., Järvinen, M., Fogelholm, C-J., Bergström, F., Eriksson, K. 2007. Multi-phase Chemistry in Process Simulation MASIT04 (VISTA). *MASI Technology Programme 2005-2009, Yearbook 2007*. Technology Review. Tekes. To be published in 2007.

Chen, M., Mountford, S. J., and Sellen, A. 1988. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of the 15th Annual Conference on Computer Graphics and interactive Techniques* R. J. Beach, Ed. SIGGRAPH '88. ACM Press, New York, NY, pp. 121-129.

Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, D. and Horrocks, I. 2000. The Semantic Web: The Roles of XML and RDF. In *IEEE Internet Computing*. Vol. 4, Num. 5, pp. 63-74.

The DARPA Agent Markup Language (DAML). [Cited 15 Apr 2007]. Available at:  
<http://www.daml.org/>

Eclipse – an open development platform. [Cited 10 Mar 2007]. Available at:  
<http://www.eclipse.org/>

Guarino, N. and Giaretta, P., 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. pp. 25-32

Guarino, N. 1998. Formal Ontology and Information Systems. *Proceedings of the 1st International Conference* June 6-8, 1998, Trento, Italy. 1st. IOS Press. pp. 3-15.

Gruber, T. R. 1993. A translation approach to portable ontology specifications. In *Knowledge Acquisition*. Vol. 5, Number 2, (1993), pp. 199-220.

Gruber, T. R. 1995. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.* 43, 5-6 (Dec. 1995), pp. 907-928.

Gruninger, M. and Lee, J. 2002. Ontology: Applications and Design. In *Communications of the ACM*. Vol. 45, No. 2. February 2002. pp 39-41.

Hand, C., 1997. A Survey of 3D Interaction Techniques. *Computer Graphics Forum*, Vol. 16, Number 5, pp. 269-281, Blackwell Synergy, 1997.

Hansen, C.D., Johnson, C.R., 2005. The visualization handbook. Elsevier, Burlington, MA. 962 pages. ISBN: 0-12-387582-X

Henriksen, K., Sporring, J., Hornbaek, K, 2004. Virtual Trackballs Revisited. *IEEE Transactions on Visualization and Computer Graphics* ,vol. 10, no. 2, March/April, 2004 , pp. 206-216.

Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., and Kassell, N. 1997. Usability analysis of 3D rotation techniques. In *Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology* (Banff, Alberta, Canada, October 14 - 17, 1997). UIST '97. ACM Press, New York, NY, pp. 1-10.

Hoffmann, C.M, 1989, Geometric and Solid Modeling: An Introduction, Morgan Kaufmann, San Mateo, CA, 1989, 340 pages. [Cited 10 Mar 2007]. Available at:  
<http://www.cs.purdue.edu/homes/cmh/distribution/books/geo.html>

Hoffman, C. 1994. *Semantic Problems of Generative, Constraint-Based Design*. in "*Parametric and Variational Design*", Hoschek, J. and Dankwort, W. (eds.), Teubner Verlag, 1994

Houde, S. 1992. Iterative design of an interface for easy 3-D direct manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Monterey, California, United States, May 03 - 07, 1992). P. Bauersfeld, J. Bennett, and G. Lynch, Eds. CHI '92. ACM Press, New York, NY, pp. 135-142.

IGES, The Initial Graphics Exchange Specification. [Cited 12 Mar 2007]. Available at:  
<http://www.nist.gov/iges/>

IGES Preservation Society. [Cited 12 Mar 2006]. Available at: <http://www.iges5x.org/>

Intergraph's SmartPlant® 3D product web page. [Cited 22 Nov 2006]. Available at:  
<http://www.intergraph.com/smartplant/3d/default.asp>

ISO IS 15926-2:2003: Industrial automation systems and integration – Integration of life-cycle data for oil and gas production facilities – Part2: Data Model, 2003. International Organization for Standardization, Geneva, Switzerland.

ISO IS 15926-4:2004: Industrial automation systems and integration – Integration of life-cycle data for oil and gas production facilities – Part4: Initial Reference Data, 2004. International Organization for Standardization, Geneva, Switzerland.

jMonkeyEngine web page. [Cited 20 Nov 2006]. Available at: <http://www.jmonkeyengine.com>

- Kalajainen, T. and Luukkainen, M. 2005. Webmon's User's manual.
- Kalogerakis, E., Christodoulakis, S., Moutout, N., 2006. Coupling Ontologies with Graphics Content for Knowledge Driven Visualization. *Proceedings of the IEEE Virtual Reality Conference 2006 (IEEE VR '06)*, pp.43-50, Virginia, USA, 25-28 March 2006
- Kemmerer, S.J. 2001. Initial Graphics Exchange Specifications. In *A Century of Excellence in Measurements, Standards, and Technology - A Chronicle of Selected NBS/NIST Publications, 1901 - 2000*, David L. Lide, Editor; NIST Special Publication 958, January 2001
- Kerlow, I.W., 2004. The art of 3D computer animation and effects. 3<sup>rd</sup> edition. John Wiley & Sons Inc., Hoboken, New Jersey. 452 pages. ISBN: 0-471-43036-6
- KnowPulp 3.0 Learning Environment for Chemical Pulping and Automation. Prowledge OY & VTT Industrial Systems. [Cited 10 Mar 2007]. Available at: <http://www.prowledge.com/>
- Mackinlay, J. D., Card, S. K., and Robertson, G. G. 1990. Rapid controlled movement through a virtual 3D workspace. In *Proceedings of the 17th Annual Conference on Computer Graphics and interactive Techniques* (Dallas, TX, USA). SIGGRAPH '90. ACM Press, New York, NY, 171-176.
- Manola F. and Miller. E., 2004. RDF Primer. W3C Recommendation 10 February 2004. [Cited 15 Apr 2007]. Available at: <http://www.w3.org/TR/rdf-primer/>
- McAffer, J. and Lemieux, J-M., 2006. Eclipse Rich Client Platform: Designing, Coding and Packaging Java Applications. Addison-Wesley, 504 pages. ISBN: 0-321-33461-2
- McGuinness, D.L. and van Harmelen, F. 2004. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. [Cited 15 Apr 2007]. Available at: <http://www.w3.org/TR/owl-features/>
- Nayyar, M.L., 2000. *Piping Handbook 7th Edition*. McGraw-Hill ISBN 0-07-047106-1

Nielson, G.M., Olsen, D.R., 1986. *Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices* Proceedings of the 1986 workshop on Interactive 3D graphics, pp.175-182, 1987, ACM Press New York, NY, USA

OPC Foundation [Cited 10 Mar 2007]. Available at: <http://www.opcfoundation.org/>  
Open CASCADE Technology, 3D modeling & numerical simulation. [Cited 10 Mar 2007]. Available at: <http://www.opencascade.org/>

Parent, R., 2002. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann., San Francisco, CA. 527 pages. ISBN: 1-55860-579-7

Park, M., 2005. *Ontology-based Customizable 3D Modeling for Simulation*. PhD Thesis. University of Florida.

Phillips, C. B. and Badler, N. I. 1988. JACK: a toolkit for manipulating articulated figures. In *Proceedings of the 1st Annual ACM SIGGRAPH Symposium on User interface Software* (Alberta, Canada, October 17 - 19, 1988). UIST '88. ACM Press, New York, NY, 221-229.

Phillips, C. B., Badler, N. I., and Granieri, J. 1992. Automatic viewing control for 3D direct manipulation. In *Proceedings of the 1992 Symposium on interactive 3D Graphics* (Cambridge, Massachusetts, United States). SI3D '92. ACM Press, New York, NY, 71-74.

Posada, J., Toro, C., Wundark, S., Stork, A., 2005. Ontology supported semantic simplification of large data sets of industrial plant CAD models for design review visualization. *Lecture Notes in Computer Science* v 3683 LNAI, *Knowledge-Based Intelligent Information and Engineering Systems - 9th International Conference, KES 2005, Proceedings*, 2005, pp. 184-190

Roche, C., 2003. *Ontology: a survey*. *Proceedings of the 8th Symposium on Automated Systems Based on Human Skill and Knowledge*. IFAC, Göteborg, Sweden.

Pratt, M.J., 2001. Introduction to ISO 10303 – the STEP Standard for Product Data Exchange. *ASME Journal of Computing and Information Science in Engineering*. March 2001. pp. 102-103.

Pättikangas T., Manninen M., Ilvonen M., Huhtanen, R., Luukkainen M. 2006. Symbiosis between computational fluid dynamics and plant models. Research Report 2006, VTT. 32 pages.

Quick, J. M., Zhu, C., Wang, H., Song, M., and Müller-Wittig, W. 2004. Building a virtual factory. In *Proceedings of the 2nd international Conference on Computer Graphics and interactive Techniques in Australasia and South East Asia* (Singapore, June 15 - 18, 2004). S. N. Spencer, Ed. GRAPHITE '04. ACM Press, New York, NY, 199-203.

Rohrer, M. W. 2000. Seeing is believing: the importance of visualization in manufacturing simulation. In *Proceedings of the 32nd Conference on Winter Simulation* (Orlando, Florida, December 10 - 13, 2000). Winter Simulation Conference. Society for Computer Simulation International, San Diego, CA, 1211-1216.

Rosenblum, L., Earnshaw, R.A., Encarnacao, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., Thalmann, D., 1994. Scientific Visualization. Academic Press LTD, London. 532 pages. ISBN: 0-12-227742-2

Räsänen, E., 2003. Modelling ion exchange and flow in pulp suspensions. PhD (Tech.) Thesis. VTT Processes. Espoo. 62 p. + app. 111p. VTT Publications : 495.

SFS-Handbook 123 Pipe Classes. 2000. Finnish Standards Association. Helsinki, Finland.

Shoemake, K. 1992. ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics interface '92* (Vancouver, British Columbia, Canada). K. S. Booth and A. Fournier, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 151-156.

Siltanen, P., Karhela, T., Woodward, C., Savioja P. 2007. Augmented Reality for Plant Lifecycle Management. To be presented in *13<sup>th</sup> International Conference on Concurrent Enterprising. (ICE 2007)*. Sophia-Antipolis, France. 4-6 June, 2007.

Smith, B. 2003. Ontology. In *Blackwell guide to the Philosophy in Computing and Information*. Oxford: Blackwell. Pages 155-166.

Tan, D. S., Robertson, G. G., and Czerwinski, M. 2001. Exploring 3D navigation: combining speed-coupled flying with orbiting. In *Proceedings of the SIGCHI Conference on Human Fac-*

*tors in Computing Systems* (Seattle, Washington, United States). CHI '01. ACM Press, New York, NY, 418-425.

Uschold, M., Callahan, S., 2004. Semantics-Based Virtual Product Models: Unifying Product and Knowledge Data. *NASA Workshop on the Knowledge Integrating Virtual Iron Bird*. <http://ic.arc.nasa.gov/vib/>

Uschold, M. and Gruninger, M. 1996. Ontologies: principles, methods, and applications, In *Knowledge Engineering Review*, 11(2), 93--155, (1996).

West, M., 2004. Some Industrial Experiences in the Development and Use of Ontologies. *EKA04 Workshop on Core Ontologies*.

X3D Specifications. [Cited 10 Mar 2007]. Available at:  
<http://www.web3d.org/x3d/specifications/>

Zelevnik, R. C., Herndon, K. P., Robbins, D. C., Huang, N., Meyer, T., Parker, N., and Hughes, J. F. 1993. An interactive 3D toolkit for constructing 3D widgets. In *Proceedings of the 20th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '93*. ACM Press, New York, NY, 81-84.