User requirements document

PMoC

14/04/2003

Version 1.7

T-76.115 User requirements document PmoC

Version	Date	Author	Description
1.7	14.4.2003	Lönnberg	Added rotate/flip of symbols.
1.6	21.3.2003	Lönnberg	A17 and B2 no longer require scroll bars.
1.5	7.2.2003	Lönnberg	Fixed lots of minor omissions and mistakes in the use cases.
1.4	5.2.2003	Lönnberg	Revised functional requirements (use cases) and priorities to match the customer's UI design and changes of mind.
1.3	8.1.2003	Lönnberg	Added tree filtering operations.
1.2	7.11.2002	Lönnberg	Revised based on customer feedback
1.1	28.10.2002	Lönnberg	Revised based on customer feedback
1.0	25.10.2002	Lönnberg, Jakobsson and Backas	Sent to customer for checking
0.1	17.10.2002	Lönnberg, Jakobsson and Backas	Draft

Table of Contents

1. Introduction

1.1 Purpose of this document

The purpose of this document is to give a clear and precise definition of what the system should do from the user's point of view. User requirements and functional requirements are laid out without going into technical details. For user requirements user groups are identified and use cases are described. The functionality of the system is also defined with constraints and properties.

Group of the readers	Reasons for reading
Users and customers	To give feedback about the user requirements.
System developers	To understand what functions and properties the system must contain.
Testers	To test the system against the requirements.
Writers of user manuals	To get material for user manuals.
Project group	To follow-up the status of the project against the requirements.

Table 1: Readers of this document

1.2 References

Name of the document	Short description of the document contents
GML specification	The Gallery Markup Language XML specification
Architectural description	The architectural proposal of the Gallery system
Project plan	The plan for this project

Table 2: A list of all relevant documents

2. System overview

The terms used in this document when defining requirements, are:

- *Must*: this requirement is crucial and will be implemented.
- *Should*: this requirement is implemented if there is time and potential to do so.
- *Can* or *may*: this requirement is implemented only if there is time and interest.

This section describes the system to be implemented, which shall be known as *ProConf*. The system is intended to be a tool for creating and editing configurations for industrial process models for documentation and simulation purposes. The system can be divided into three subsystems: a *tree view* (or *browse view*) of the types and components, a graphical *symbol editor* and a graphical *net editor*.

The symbol editor must provide the user with tools to edit *symbols*, which are graphical representations in the SVG format of *components*, *terminals* and *connections*. It must also allow the user to define the *properties* of the different *component types* and edit the terminals of the component type. Terminals are the points on the instances of the component type between which connections (which are graphically represented as polylines) can be made. The properties of the component types are must be displayable and editable. The system should allow the user to display and edit these properties using the tree view and a separate form for each component type.

The net editor must allow the user to produce a *net* corresponding to a process flow diagram consisting of symbols of components and their terminals, symbols of connections and other SVG graphical objects. It must also provide a way for the user to define values for component properties. The net is a canvas on which graphics and symbols are drawn. Symbols and component types (and, similarly, symbol instances and components) must be able to have a one-to-one relationship. It should also be possible to create symbols and component types separately and join them later. Similarly, the user should also be able to create components and symbol instances separately and attach them to each other on demand.

The net editor provides a net view. The process configuration can be divided into a topological model and a graphical model. The topological model must be viewable completely through the tree view and partially through the net view (only those components that have corresponding graphical objects; in the first version all components are assumed to have a corresponding graphical object, although it may be empty). The graphical model can be viewed only through the net view and it must be possible to include graphical objects that do not correspond to anything in the topological model.

In the topological model, components form hierarchies. At first only a hierarchy of two component levels is allowed. Later on, support for hierarchies of with any amount of levels should be added.

The system should manage *categories* as described in the GML specification, and these categories should then be shown in the tree view. If this is not implemented, the system will rely on existing software to manage categories.

The intended user groups are described in section 4 and are summarised in Figure 1 together with their corresponding use cases. For clarity, similar use cases have been grouped together into larger tasks and only a few subtasks shown for each task group. Use cases are divided into *core functionality* (must be implemented; use of the program requires this), *desirable functionality* (should be implemented; adding this would make the program more useful) and *optional functionality* (can be implemented; may simplify some tasks). The use cases are:

- Edit symbol (type)
    - Core functionality
        - A1: New symbol
        - A2: Open symbol
        - A4: Save symbol
        - A5: Add graphical object to symbol
        - A6: Change graphical attribute
        - A7: Move graphical object
        - A8: Delete graphical object
        - A16: Add terminal
        - A17: Scroll symbol
        - A21: Import component type
        - A25: Attach symbol to type
        - A26: Make free symbol terminal symbol
        - A27: New connection symbol

  - Desirable functionality
    - A22: Create component type
    - A10: Change property name
    - A9: Add property
    - A11: Change property type
    - A12: Delete property type
    - A13: Change property attribute
    - A14: Edit default value of property (part)
    - A15: Add subtype to type
    - A19: Save component type
    - A23: Add constraint
    - A24: Remove constraint
  - Optional functionality
    - A3: Import graphic
    - A18: Zoom symbol
    - A20: Undo symbol change
- Examine net
  - Core functionality
    - B1: Open net
    - B2: Scroll net
  - Desirable functionality
    - B4: Expand/collapse tree representation
  - Optional functionality
    - B3: Zoom net
    - B5: Select root of tree
    - B6: Select visible element types in tree
- Edit net (instances)
  - Core functionality
    - C1: New component
    - C2: Save net
    - C3: Add graphical object to net
    - C4: Move graphical object
    - C5: Delete graphical object
    - C6: Add component to net
    - C7: Move component symbol
    - C8: Delete component
    - C9: Change component property or attribute value
    - C10: Add connection
    - C11: Delete connection
    - C17: Save component
  - Desirable functionality
    - C12: Change connection start/end terminal
    - C13: Change connection intermediate points
  - Optional functionality
    - C14: Add component as connection
    - C15: Undo net change
    - C16: Cut/copy/paste
- Edit tree structure
  - Core functionality
    - D1: Create category

    - D2: Change category attribute
    - D3: Delete category
  - Optional functionality
    - D4: Create department
    - D5: Change department attribute
    - D6: Delete department

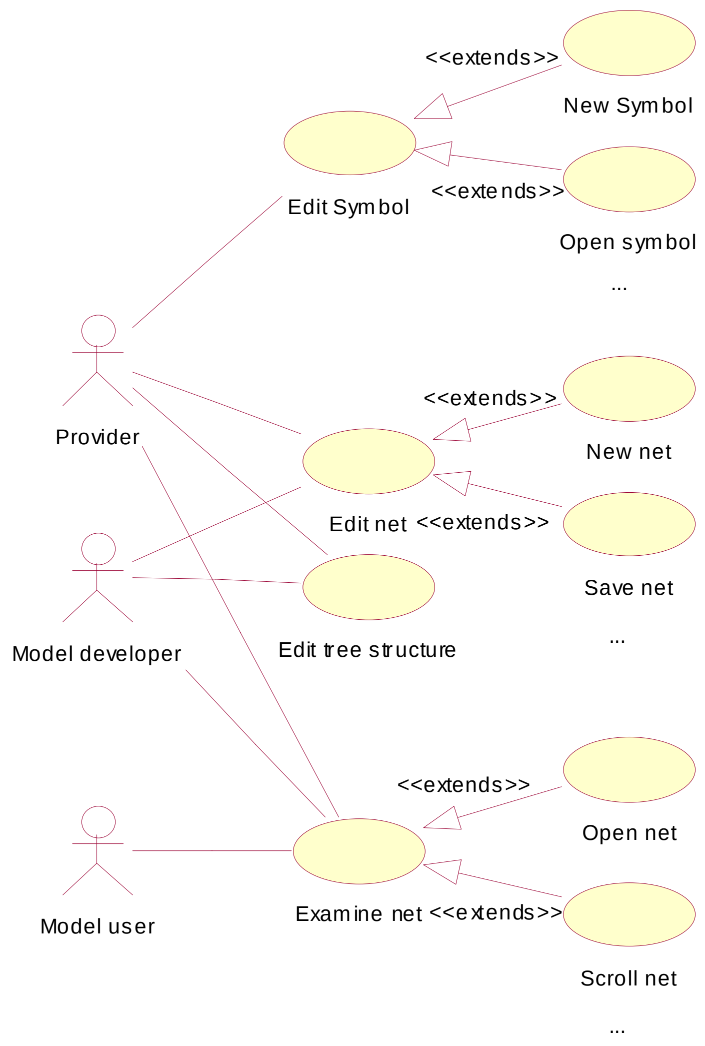

Figure 1: Use case diagram

At a later stage (probably after the completion of this project), the net editor will be connected to a simulator. The net editor will then provide the ability to start and stop simulation, and examine simulation results using *monitors* (that display the current value of a property) and *trend graphs* (that show how a property value has changed over time).

3. Business goals

The proposed SVG net and symbol editors are to be part of the larger Gallery System, an XML-based specification for process modelling. The current graphical solution for configuring process models utilises a badly documented proprietary graphics format and cannot be used within the GML solution. The use of SVG allows the graphics to be part of the XML system and allows a modular use of graphics. Any SVG editor can be used to produce graphics for the components. With SVG-based graphics use of the Gallery system would be very flexible and the system would be applicable to a wide variety of business areas.

The full use of GML allows customers to tailor the process modelling tool to suit their own use. A completely XML-based process modelling solution is highly scalable, reusable and compatible with most environments. Gallery resources can be distributed over the Internet, thus improving usability and efficiency. The same Gallery system could be used for modelling anything from a nuclear power station to a paper mill.

With these long-term business goals in mind the customer is eager to validate the GML specification. By developing the SVG editors in this project the customer is one step closer to a full-fledged Gallery System implementation that will have a positive impact on Finnish process simulation research.

4. User groups

The Gallery system will have many different users and most of them will use the SVG net editor while some users, such as kernel developers, will not use it at all most of the time. The users can be divided into: those who design components (library developers or component providers), those who design process model by using the components (model developers) and those who use the process models for simulations (model users). The primary users of the symbol editor are the library developers while model developers and model users use the net editor.

User Group	**Description**	**Estimated number of users**
Provider	Develops component symbols and component types with the symbol editor	10
Model developer	Develops process models with the net editor	10–100
Model user	Users process models for simulations	100–1000

Table 3 – Users of the system

5. Functional requirements

The symbol editor and net editor can be considered two separate programs, and their functional requirements can therefore be defined separately.

To keep the exception descriptions short, all exceptions caused by insufficient system resources (e.g. insufficient memory, disk space, file handles) are left out from the following use case descriptions, as they can occur in all use cases. If a resource exception occurs, the program aborts the current action, displays an error message and continues to run if possible.

Unless otherwise specified, objects that exist both in the topological and graphical models can be manipulated using both models, e.g. components can be selected from both.

5.1 Symbol editor

Name:	A1: New symbol
Summary:	The user creates a new, empty symbol.
Actors:	Provider
Preconditions:	A category exists and is visible in the tree.
Basic sequence:	The user requests a new symbol from the tree view, specifying which category to put it in.
Exceptions:	
Postconditions:	A new empty free symbol is created.

Name:	A2: Open symbol
Summary:	The user opens an existing symbol file.
Actors:	Provider
Preconditions:	A symbol exists and is visible in the tree.
Basic sequence:	The user requests from the tree that a symbol be opened. The symbol editor then opens a new window displaying the loaded symbol.
Exceptions:	Invalid file: an error message is displayed and the action aborted.
Postconditions:	A new symbol editor window has been opened containing the loaded symbol.

Name:	A3: Import graphic
Summary:	The user creates a symbol based on an existing graphic.
Actors:	Provider
Preconditions:	A category exists and is visible in the tree. A graphics file exists.
Basic sequence:	The user requests that a graphic be imported (specifying the category to place it in). The symbol editor asks the user to select a file, and then creates a new symbol that consists solely of the imported graphics file.
Exceptions:	Invalid file: an error message is displayed and the action aborted. User cancels operation at file selection stage: action is aborted.
Postconditions:	A new free symbol has been created containing the loaded graphic.

Name:	A4: Save symbol
Summary:	The user saves an open symbol.
Actors:	Provider
Preconditions:	A symbol is open and active in the symbol editor.
Basic sequence:	The user requests that the symbol be saved. The open symbol is then saved.
Exceptions:	File system error: an error message is displayed and the action aborted.
Postconditions:	The currently activated symbol has been saved.

Name:	A5: Add graphical object to symbol
Summary:	The user adds a graphical object to the symbol.
Actors:	Provider
Preconditions:	A symbol is open and active in the symbol editor.
Basic sequence:	The user selects the graphical object to be added, and then specifies the co-ordinates of the object (e.g. by clicking and dragging from one point to another). An instance of the graphical object with the specified co-ordinates is then added to the symbol with current graphical attribute values.

Name:	A5: Add graphical object to symbol
Exceptions:	User cancels operation at co-ordinate selection stage: action is aborted.
Postconditions:	The graphical object is added to the symbol.

Name:	A6: Change graphical attribute
Summary:	The user changes a graphical attribute.
Actors:	Provider
Preconditions:	A symbol is open and active in the symbol editor.
Basic sequence:	The user chooses a new value for a graphical attribute (e.g. line width, colour, rotation, flipping). If a graphical object is selected, the corresponding attribute change is made to it. If not, the attribute values for newly created objects are changed.
Exceptions:	
Postconditions:	The specified attribute of the current object or the settings for new objects is changed.

Name:	A7: Move graphical object
Summary:	The user moves a graphical object in the symbol.
Actors:	Provider
Preconditions:	A symbol containing at least one graphical object is open and active in the symbol editor.
Basic sequence:	The user selects the graphical object to be moved, and then drags it to another position.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The graphical object is moved to the new position.

Name:	A8: Delete graphical object
Summary:	The user deletes a graphical object in the symbol.
Actors:	Provider
Preconditions:	A symbol containing at least one graphical object is open and active in the symbol editor.
Basic sequence:	The user selects the graphical object to be deleted, and selects deletion. The object disappears.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The selected object is removed from the symbol.

Name:	A9: Add property
Summary:	The user adds a property to the component type.
Actors:	Provider
Preconditions:	A component type is open and active in the tree view.
Basic sequence:	The user requests that a property be added to the component type and specifies its name. The user is then presented with a default type definition (e.g. a floating-point value).
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A new property is added to the component type with a default type and a new name.

Name:	A10: Change property name
Summary:	The user changes the name of a property.
Actors:	Provider

Name:	A10: Change property name
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects the property name and edits it using normal text editing features.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The name of the property is changed.

Name:	A11: Change property type
Summary:	The user changes the type of a property.
Actors:	Provider
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects the property type and selects a new property type from a list of predefined types.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The type of the property is changed.

Name:	A12: Delete property type
Summary:	The user deletes a property type definition.
Actors:	Provider
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects the type (or subtype) definition and selects deletion.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The type definition is deleted.

Name:	A13: Change property attribute
Summary:	The user changes an attribute of the property.
Actors:	Provider
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects the property (or subtype, if the type contains such) attribute he wishes to change (allowed vector size, unit name, minimum/maximum value or label), and types in the desired attribute value.
Exceptions:	User cancels operation: action is aborted. Invalid value: action is aborted.
Postconditions:	The value of a property of the attribute is changed.

Name:	A14: Edit default value of property (part)
Summary:	The user changes the default value of a (part of a) property.
Actors:	Provider
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects the type (or subtype, if the attribute contains such) default value, and edits it using normal text editing features (or, if it is an enumeration, selects it from the list of predefined values).
Exceptions:	User cancels operation: action is aborted. Invalid value: action is aborted.
Postconditions:	The default value of a (part of a) property is changed.

Name:	A15: Add subtype to type
Summary:	The user adds a type as part of a property type.
Actors:	Provider
Preconditions:	A component type with at least one property is open and active in the tree view.
Basic sequence:	The user selects a property type definition (or subtype, if the property contains

Name:	A15: Add subtype to type such), and specifies that he wants to create a new subtype. A new attribute with default type is then created as a subtype of the selected type.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A new attribute of default type is creates as a subtype of a specified attribute.

Name:	A16: Add terminal
Summary:	The user adds a terminal to a component type.
Actors:	Provider
Preconditions:	A component type symbol is open and active in the symbol editor.
Basic sequence:	The user requests that a terminal be added to the symbol and selects the symbol and the property to attach it to. The terminal symbol is added to the component type symbol.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A new terminal symbol is added to the component type.

Name:	A17: Scroll symbol
Summary:	The user scrolls the graphical representation of the symbol.
Actors:	Provider
Preconditions:	A symbol is open and active in the symbol editor.
Basic sequence:	The user can scroll a symbol somehow. Implementation is not specified.
Exceptions:	
Postconditions:	A different part of the symbol is shown.

Name:	A18: Zoom symbol
Summary:	The user zooms the graphical representation of the symbol.
Actors:	Provider
Preconditions:	A symbol is open and active in the symbol editor.
Basic sequence:	The user specifies a new zoom level. The symbol is then shown at this level.
Exceptions:	
Postconditions:	The symbol is shown in a different size.

Name:	A19: Save component type
Summary:	The user saves an open symbol.
Actors:	Provider
Preconditions:	A component type is open and active in the tree view
Basic sequence:	The user requests that the component type be saved. The open type is then saved.
Exceptions:	File error: an error message is displayed and the action aborted.
Postconditions:	The currently activated component type has been saved.

Name:	A20: Undo symbol change
Summary:	The user undoes a change to a symbol.
Actors:	Provider
Preconditions:	A change has been made to a symbol.
Basic sequence:	The user specifies that he wishes to undo the last change to a symbol.
Exceptions:	
Postconditions:	The last change to a symbol is undone.

Name:	A21: Import component type
Summary:	The user imports an externally defined component type.

Name:	A21: Import component type
Actors:	Provider
Preconditions:	A category exists and is visible in the tree.
Basic sequence:	The user requests that a new component type be imported into a specified category and selects the file to import.
Exceptions:	Invalid file: an error is reported to the user. User aborts action: action is cancelled.
Postconditions:	The component type is added to the specified category.

Name:	A22: Create component type
Summary:	The user creates a new component type.
Actors:	Provider
Preconditions:	A category exists and is visible in the tree.
Basic sequence:	The user requests that a new component type be added to a specified category.
Exceptions:	
Postconditions:	An empty component type is added to the specified category with default attributes.

Name:	A23: Add constraint
Summary:	The user creates a new constraint.
Actors:	Provider
Preconditions:	A constraint or component type exists and is visible in the tree.
Basic sequence:	The user requests that a new constraint be added to a specified component type or constraint, and chooses the type of constraint.
Exceptions:	
Postconditions:	A new constraint is added to the specified type or constraint.

Name:	A24: Remove constraint
Summary:	The user removes a constraint.
Actors:	Provider
Preconditions:	A constraint exists and is visible in the tree.
Basic sequence:	The user requests that a constraint be deleted.
Exceptions:	
Postconditions:	The constraint is deleted.

Name:	A25: Attach symbol to type
Summary:	The user attaches a free symbol to a component type.
Actors:	Provider
Preconditions:	A free symbol exists and is open, and a component type exists and is visible in the tree.
Basic sequence:	The user selects the symbol and drags it onto the component type.
Exceptions:	User aborts operation.
Postconditions:	The free symbol is changed into a component type symbol attached to the specified type.

Name:	A26: Make free symbol terminal symbol
Summary:	The user converts a free symbol to a terminal symbol.
Actors:	Provider
Preconditions:	A free symbol exists and is open.
Basic sequence:	The user selects the symbol, chooses to make it a terminal symbol and specifies

Name:	A26: Make free symbol terminal symbol
	the terminal type.
Exceptions:	User aborts operation.
Postconditions:	The free symbol is changed into a terminal symbol of the specified type.

Name:	A27: New connection symbol
Summary:	The user creates a new connection symbol
Actors:	Provider
Preconditions:	A category exists and is visible in the tree.
Basic sequence:	The user chooses a category and requests that a connection symbol be created in it.
Exceptions:	User aborts operation.
Postconditions:	A new connection symbol is created.

5.2 Net editor

Name:	C1: New component
Summary:	The user creates a new component with an empty net (without adding it to a net).
Actors:	Provider or model developer
Preconditions:	A category exists and is visible in the tree view.
Basic sequence:	The user requests a new component from the tree view and selects the component type to instantiate. A new component is added to the tree view.
Exceptions:	
Postconditions:	A new component is added to the topological model.

Name:	B1: Open net
Summary:	The user opens an existing net.
Actors:	Provider or model developer/user
Preconditions:	A component exists and is visible in the tree view.
Basic sequence:	The user selects a component from the topological model and specifies that he wishes to view its net.
Exceptions:	Invalid file: an error message is displayed and the action aborted.
Postconditions:	A new net window has been opened in the net editor containing the loaded net.

Name:	C2: Save net
Summary:	The user saves an open net.
Actors:	Provider or model developer
Preconditions:	A net is open and active in the net editor.
Basic sequence:	The user requests that the net be saved.
Exceptions:	Invalid file: an error message is displayed and the action aborted.
Postconditions:	The currently activated net has been saved.

Name:	C3: Add graphical object to net
Summary:	The user adds a graphical object to the net.
Actors:	Provider or model developer
Preconditions:	A net is open and active in the net editor.
Basic sequence:	The user selects the graphical object to be added, and then specifies the co-ordinates of the object (e.g. by clicking and dragging from one point to another). An instance of the graphical object with the specified co-ordinates is then added to the net with current graphical attribute values.

Name:	C3: Add graphical object to net
Exceptions:	User cancels operation at co-ordinate selection stage: action is aborted.
Postconditions:	The graphical object is added to the net.

Name:	C4: Move graphical object
Summary:	The user moves a graphical object in the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least one graphical object is open and active in the net editor.
Basic sequence:	The user selects the graphical object to be moved, and then drags it to another position.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The graphical object is moved to the new position.

Name:	C5: Delete graphical object
Summary:	The user deletes a graphical object in the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least one graphical object is open and active in the net editor.
Basic sequence:	The user selects the graphical object to be deleted, and selects deletion. The object disappears. The corresponding component or connection (if any) is also deleted from the topological model.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The selected object is removed from the net and from the topological model if a corresponding object exists there.

Name:	C6: Add component to net
Summary:	The user adds a component to the net.
Actors:	Provider or model developer
Preconditions:	A component/net is open and active in the net editor or tree view.
Basic sequence:	The user selects the type of the component to be added (from a library of component types), and then specifies the position of the corresponding symbol by dragging onto the canvas. An instance of the symbol with the specified position is then added to the canvas and a component of the specified type with default attribute values and no connections is added.
Exceptions:	User cancels operation at position selection stage: action is aborted.
Postconditions:	The component is added to the net and to the topological model.

Name:	C7: Move component symbol
Summary:	The user moves a component symbol in the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least one component is open and active in the net editor or tree view.
Basic sequence:	The user selects the symbol to be moved, and then drags it to another position.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The symbol is moved to the new position.

Name:	C8: Delete component
Summary:	The user deletes a component in the net.
Actors:	Provider or model developer
Preconditions:	A component/net containing at least one component is open and active in the net editor or tree view.

Name:	C8: Delete component
Basic sequence:	The user selects the component to be deleted, and selects deletion. The component disappears.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The selected component is removed from the net.

Name:	C9: Change component property or attribute value
Summary:	The user changes the value of a property or attribute of a component in the net.
Actors:	Provider or model developer
Preconditions:	A component is open and active in the tree view or visible in a net.
Basic sequence:	The user selects the component, and selects the property value (or attribute) he wishes to change. He then specifies the new value using normal text editing facilities (or, for Boolean attributes, by checking/unchecking it). The user can also add and remove elements from the property value (as allowed by vector size limitations).
Exceptions:	User cancels operation: action is aborted. Invalid value: action is aborted.
Postconditions:	The selected component property or attribute value is changed.

Name:	C10: Add connection
Summary:	The user adds a connection to the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least two terminals is open and active in the net editor or tree view.
Basic sequence:	The user selects the type of the connection he wishes to add, and then selects the terminal the connection should start from. He then specifies any intermediate points the connection should pass through and finally specifies the terminal at which the connection ends. A new connection is then added to the net with the specified start and end terminals and intermediate points, and the properties corresponding to the connected terminals are updated to reflect this.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A new connection is added to the net and the terminal properties are changed to represent the new connection.

Name:	C14: Add component as connection
Summary:	The user adds a component as a connection to the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least two terminals is open and active in the net editor or tree view.
Basic sequence:	The user selects the type of the connection he wishes to add, and then selects the terminal the connection should start from. He then specifies any intermediate points the connection should pass through and specifies the terminal at which the connection ends. Finally, he specifies the type of the component that should be implemented in the connection (e.g. a pipe component). This component type must have exactly one start terminal and one end terminal. A new connection is then added to the graphical representation with the specified start and end terminals and intermediate points. A component of the specified type is added to the topological model with connections from the specified start terminal to the new component and from the new component to the specified end terminal.
Exceptions:	User cancels operation: action is aborted.

Name:	C14: Add component as connection
Postconditions:	A new connection is added to the graphical net, a new component is added to the topological representation and the terminal properties of the start and end components of the “connection” are changed to represent the new connection.

Name:	C11: Delete connection
Summary:	The user removes a connection from the net.
Actors:	Provider or model developer
Preconditions:	A net containing at least one connection is open and active in the net editor or tree view.
Basic sequence:	The user selects the connection he wishes to delete and selects deletion. The connection is then deleted both from the net and the topological model.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A connection is removed from the net and the terminal properties are changed to reflect the deletion of the connection.

Name:	C12: Change connection start/end terminal
Summary:	The user changes the start or end terminal of a connection.
Actors:	Provider or model developer
Preconditions:	A component/net containing at least one connection is open and active in the net editor or tree view.
Basic sequence:	The user selects the start/end terminal of a connection and moves the connection start/end to a different terminal. The connection is then changed both graphically in the canvas and in the component properties.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A connection start/end terminal is changed.

Name:	C13: Change connection intermediate points
Summary:	The user moves an intermediate point of a connection.
Actors:	Provider or model developer
Preconditions:	A net containing at least one connection is open and active in the net editor.
Basic sequence:	The user selects the intermediate point he wishes to change and moves it to a different point. The graphical representation of the connection is changed accordingly.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	A connection intermediate point is changed.

Name:	B2: Scroll net
Summary:	The user scrolls the canvas for a net.
Actors:	Provider or model developer/user
Preconditions:	A net is open and active in the net editor.
Basic sequence:	The user can scroll a canvas somehow. Implementation is not specified.
Exceptions:	
Postconditions:	A different part of the canvas is shown.

Name:	B3: Zoom net
Summary:	The user zooms the canvas for a net.
Actors:	Provider or model developer/user
Preconditions:	A net is open and active in the net editor.
Basic sequence:	The user specifies a new zoom level. The canvas is then shown at this level.

Name:	B3: Zoom net
Exceptions:	
Postconditions:	The canvas is shown in a different size.

Name:	B4: Expand/collapse tree representation
Summary:	The user shows or hides the child items of an item in the tree representation of the topological model.
Actors:	Provider or model developer/user
Preconditions:	
Basic sequence:	The user tells the net editor to expand or collapse a branch of the tree. If the branch is to be collapsed, it is collapsed. If it is to be expanded, the all child items of the selected item are added to the visible part of the tree representation. See the technical specification for details on what items are children of a specified item.
Exceptions:	
Postconditions:	The specified branch of the tree representation is shown or hidden.

Name:	C15: Undo net change
Summary:	The user undoes a change to a net.
Actors:	Provider
Preconditions:	A change has been made to a net.
Basic sequence:	The user specifies that he wishes to undo the last change to a net.
Exceptions:	
Postconditions:	The last change to a net is undone.

Name:	C16: Cut/copy/paste
Summary:	The user cuts or copies a component and pastes it.
Actors:	Provider
Preconditions:	A net with at least one object is open and active.
Basic sequence:	The users selects an object and then cuts/copies it. The object the selects the net in which he wishes to paste it (he may perform any operations except cut/copy in between) and selects paste. The cut/copied object is then inserted into the specified net.
Exceptions:	User cancels operation: action is aborted.
Postconditions:	The specified object is cut or copied and pasted.

Name:	B5: Select root of tree
Summary:	The user chooses which element of the topological model is the root of the visible tree.
Actors:	Provider or model developer/user
Preconditions:	
Basic sequence:	The user selects an element in the tree and tells the program to use this as the tree view root, or requests that the root element of the topological model ("The World" or whatever) is made root of the tree.
Exceptions:	The user cancels the operation.
Postconditions:	The tree view is now rooted at the selected element.

Name:	B6: Select visible element types in tree
Summary:	The user shows or hides elements of specified type in the tree view
Actors:	Provider or model developer/user

Name:	B6: Select visible element types in tree
Preconditions:	
Basic sequence:	The user tells the net editor that he wishes to change the visible element types in the tree. The program allows the user to select from a list of types (e.g. a set of checkboxes) the types he wishes to see. The tree view is changed to reflect this.
Exceptions:	The user cancels the operation.
Postconditions:	The specified element types of the tree representation are shown or hidden.

Name:	C17: Save component
Summary:	The user saves an open component.
Actors:	Provider or model developer
Preconditions:	A component is open and active in the tree.
Basic sequence:	The user requests that the component be saved.
Exceptions:	Invalid file: an error message is displayed and the action aborted.
Postconditions:	The selected component has been saved.

5.3 Edit tree structure

Name:	D1: Create category
Summary:	The user creates a new category.
Actors:	Provider or model developer
Preconditions:	A department or category is visible in the tree view.
Basic sequence:	The user requests a new category from the tree view (specifying where to create it).
Exceptions:	
Postconditions:	A new category is added to the topological model.

Name:	D2: Change category attribute
Summary:	The user changes an attribute of a category.
Actors:	Provider or model developer
Preconditions:	A category is visible in the tree view.
Basic sequence:	The user asks to change an attribute of a category from the tree view and specifies the new value.
Exceptions:	
Postconditions:	The specified attribute is changed.

Name:	D3: Delete category
Summary:	The user deletes a category.
Actors:	Provider or model developer
Preconditions:	A category is visible in the tree view.
Basic sequence:	The user requests that a category be removed.
Exceptions:	
Postconditions:	The category is removed from the topological model.

Name:	D4: Create department
Summary:	The user creates a new department.
Actors:	Provider or model developer
Preconditions:	A host is visible in the tree view.
Basic sequence:	The user requests a new department from the tree view (specifying where to create it).

Name:	D4: Create department
Exceptions:	
Postconditions:	A new department is added to the topological model.

Name:	D5: Change department attribute
Summary:	The user changes an attribute of a department.
Actors:	Provider or model developer
Preconditions:	A department is visible in the tree view.
Basic sequence:	The user asks to change an attribute of a department from the tree view and specifies the new value.
Exceptions:	
Postconditions:	The specified attribute is changed.

Name:	D6: Delete department
Summary:	The user deletes a department.
Actors:	Provider or model developer
Preconditions:	A department is visible in the tree view.
Basic sequence:	The user requests that a department be removed.
Exceptions:	
Postconditions:	The department is removed from the topological model.

6. Properties

6.1 Quality requirements

The following steps are to be taken to ensure the quality of the product. Most of these requirements are handled and explained separately in the project plan.

Documentation

The documenting follows the general principles of the course T-76.115 Software Project (Tietojenkäsittelyopin ohjelmatyö). Additional documents are produced upon agreement with the customer. The preferred format is Portable Document Format (PDF). The language of the documentation is English.

Customer satisfaction

The requirements specification must be agreed upon with the customer and met by the group.

Risk management

Risks are identified and dealt with. More information about risk management is found in section 8 of the project plan.

Testing

The product should be tested. Information about the testing practices is found in the section 8 of the project plan.

Version control

The version controlling is ensured through CVS. More information: section 8 of the project plan.

Coding conventions

Clear and consistent coding conventions are used in this project. See section 8 of the project plan.

Integration and regular builds
Integration and regular builds are managed separately to ensure the quality of the product. For more information please refer to section 8 of the project plan.

Communication and meeting practices
Communication and meeting practices are coordinated. See section 8 of the project plan.

Use case and design
Use cases and design are quality requirements that are taken into consideration during the life of this project. See section 8 of the project plan for more information.

6.2 Non-functional requirements

Performance and scalability
This is an interactive system and it must be very scalable. It need not handle large models at a usable speed yet, but it should be ensured that it is capable of that within a period of five years, taken into consideration the development of modern computer systems during that period. The system should be as scalable as possible.

Portability and maintenance
Does not need to be portable, must work in modern Microsoft® Windows® based systems. The system must be open for further development. The source code must be available.

Usability
The product should be designed for the defined user groups.

Reliability
Reliability is a very important factor; it must not have any reliability issues since it is used to simulate critical systems for long periods of time.

Security
Security is not implemented at this point, but is made possible to integrate into the product at a later date.

Error recovery and other features
Errors must be clear for people, both IT professionals and non-professionals.

Copyright, distribution rights
Copyright and distribution rights are as in the written contract, signed by both parties.

7. Constraints

7.1 Standards

The system must meet the Java, SVG, XML and GML standards sufficiently to guarantee interoperability and to the extent to which they are relevant.

7.2 Software constraints

System software requirements
The system must work in modern Microsoft® Windows® operating systems.

Programming language / code conventions
The programming language is Java, written in Borland JBuilder.

7.3 Hardware constraints

Hardware requirements have not yet been properly specified; the customer merely wants it to run on some machines he has access to. To ensure that the product can be tested, we must require that the program can run with small test nets on Pentium machines with 64 MB of RAM. Larger nets may cause memory requirements to increase dramatically.

8. Main domain concepts

The main concepts of the problem domain are described in Table 4 and their relationships summarised in Figure 2.

Concept	***Description***
Net editor	An editor for defining topological and graphical model for process configuration.
Symbol editor	An editor for defining properties and symbol graphics for component types.
Process	A physical process or system that is modelled using the system (e.g. a power station).
Net	A part of a process model consisting of interconnected components.
Component	An element in a process model that is an instance of a component type. This is usually equivalent to a physical device, e.g. a pump.
Component type	A description of properties of a set of components. This is usually equivalent to a class of physical devices, e.g. pumps.
Symbol	A graphical representation of a component used in nets that include the component.
Connection	A link between terminals of components (e.g. a reference between a pipe output and pump input).
Terminal	A point on a component to which a connection can be made.
Property	An attribute of a component that reflects an aspect of the device that it represents, e.g. the water level of a tank.

Table 4: Main concepts of the users

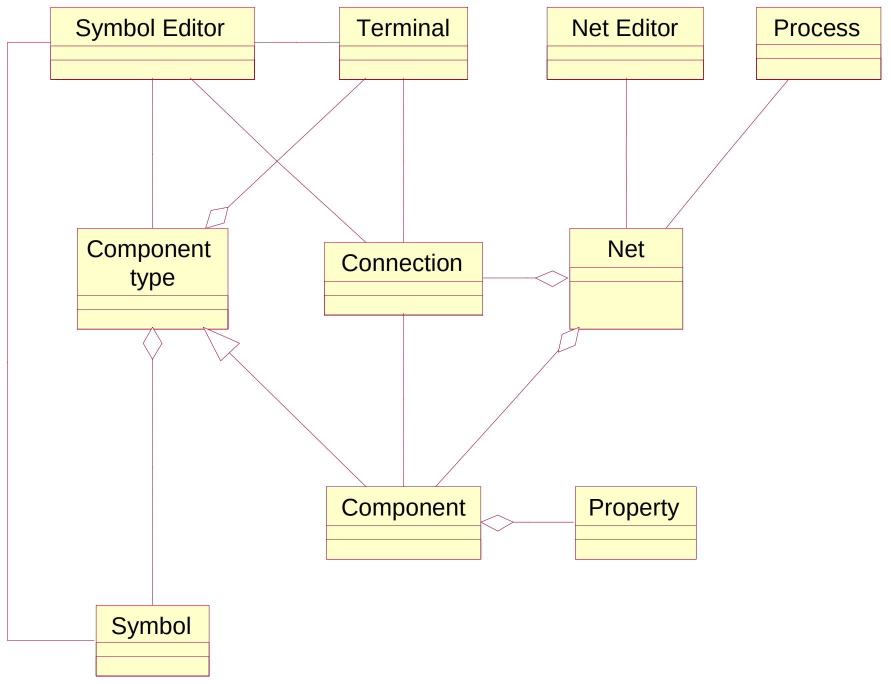

Figure 2: A graph describing the relationships of the main concepts

9. Acronyms and abbreviations

Acronym, abbreviation or special term	***Description***
SVG	Scalable Vector Graphics 1.0 is a language for describing two-dimensional vector and mixed vector/raster graphics in XML
XML	Extensible Markup Language is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.
GML	Gallery Markup Language, an XML based model configuration specification developed at VTT.