Requirements document

Group Canvas

Helsinki, 13th June 2005

Software Engineering Project
UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cu)

Project Group

Duku-Kaakyire Michael
Karppinen Tony
Lamsal Pragya
Välimäki Niko

Instructor

Kauppinen Raine

Customer

Verkamo Inkeri

Supervisor

Taina Juha

Homepage

http://www.cs.helsinki.fi/group/canvas

Change Log

Version	Date	Modifications
0.1	19.05.2005	Document skeleton
0.2	30.05.2005	Publized for internal usage
0.3	01.06.2005	Deliver for customer review
0.4	27.07.2005	Converted from PDF to OpenOffice, fixed grammar
0.5	27.07.2005	Updated priorities, definitions and cross references
0.6	05.08.2005	Updated based on customer's comments

Contents

1 Introduction

Canvas is a project at department of computer science at University of Helsinki. The aim of this project is to design and develop an extendable generic drawing tool. This drwaing tool will allow the users to draw different kinds of diagram based on the built-in elements (circle, rectangle, etc). In addition, this tool will also allow the users to create new elements, save them and reuse them in later. The tool will be extendable, meaning that it will be possible for other developers to further develop it in future.

Section 2 provides definitions of terms used by this document. This section also outlines an overview of the system. User requirements are presented in section 3. System requirements in section 4 have been extracted from these user requirements[Pra05].

2 System overview

The drawing tool is mainly a workpiece problem. In addition, the system is also required to control the execution/behavior of elements and to export files to other formats. The problem frame is a combined workpiece, control and transformation frame[Bra02]. The figure 1 on the next page presents the data model of the drawing tool.

2.1 Definitions

Diagram: A unit that can be saved by the application.

Active diagram: The diagram that is open and currently selected.There can be multiple diagrams open at the same time but there can be only one active diagram, which has been selected by the user.

Element: A basic unit for drawing a diagram. For example, a circle, a line, etc. Elements can be composed of several elements. For example, the users can create new elements by combining two or more existing elements.

Complex element: An element that includes other elements or diagrams. Complex elements are not built-in with the application. The users can compose the complex elements.

Element repository: User defined elements that are stored with the program and shown on the tool bar.

Drawing area: An area where the user is able to place elements.There can be multiple drawing areas open at the same time and the user can switch to any of the drawing area to work on at any time.

Diagram properties: The attributes of a diagram that is saved with the diagram. For example, file name, author name, etc.

Element properties: The attributes of the elements. For example, name, color, etc.

Point: A position on a two-dimensional space, marked by the x and y coordinates.

Shape: A shape is an element. For example, line, rectangle, etc.

Line: An element marked by two (start and end) points, and a straight connection between them.

Rectangle: An element marked by four points, connected by four straight lines and creating an enclosed area.

Circle: An element marked by a point as a center and a circular line. The circular line is always at equal distance from the center.

Text: An element that is a collection of letters/strings.

View (viewport): An area of a diagram that is visible to the user.

Treeview: The open diagrams and their elements are presented as a tree and its branches. The root of the tree is 'Open Diagrams', which is specified by the application. The root contains the diagram names of the open diagrams as its children and the elements of each open diagram as its grandchildren, and so on.

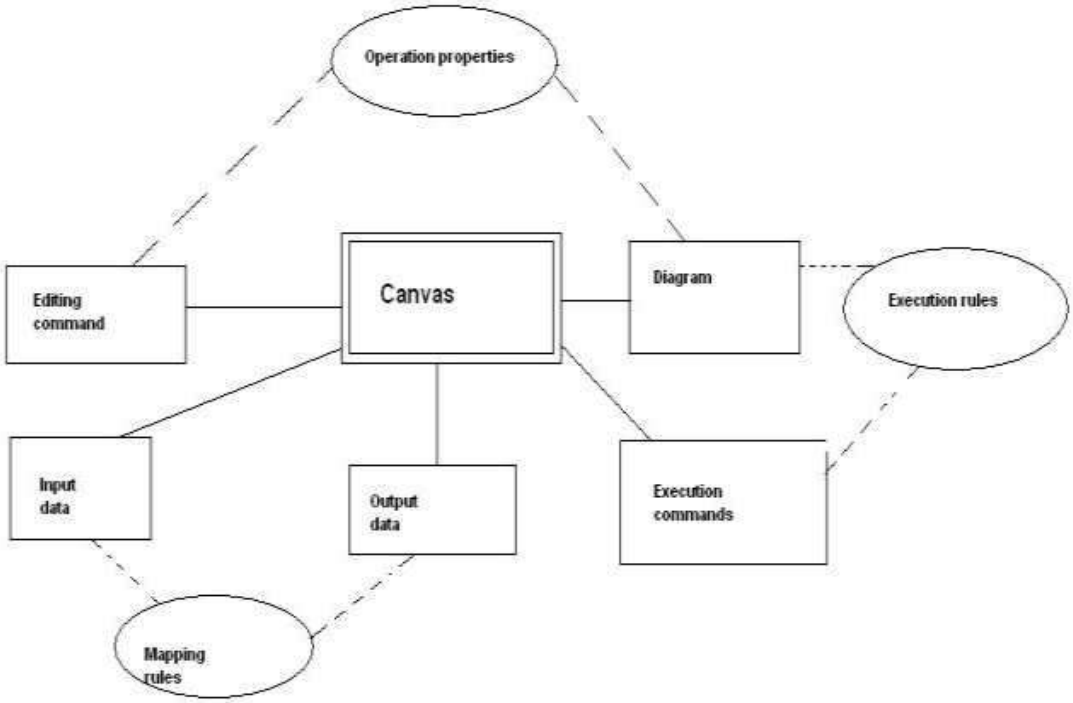

Figure 1: Data model

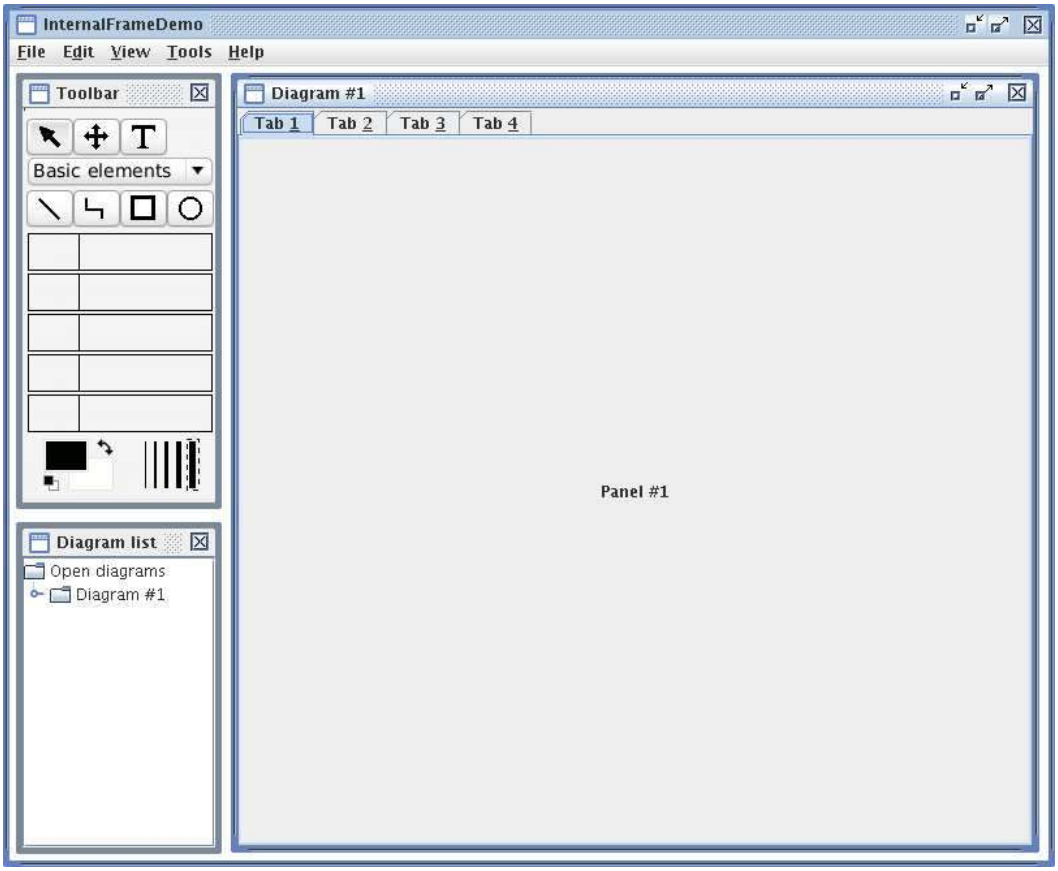

Figure 2: User interface

Figure 2 presents an example of the user interface of Canvas.

2.2 Use cases

2.2.1 Basic functions

Opening a diagram from a file: The user selects the correct diagram file from a file listing or types in the correct path and filename. The program checks if the file is in the valid diagram format and loads the diagram to the program memory for viewing and processing. The program also checks if there is any syntactic rules defined for the diagram. Example: A user opens a diagram from the file MyDiagram.cnv for editing.

Saving and exporting to a file: The user chooses to save a diagram and export it to another format. When saving a file, the user must give a filename for storing the diagram data. If the diagram is opened from an existing file, it can be stored over the old file. Otherwise the user need to provide tha application with a filename for saving the diagram. The user can export the diagram to encapsulated postscript format. When exporting a diagram, the user must give a filename. The user must be notified before writing over existing files if another file with the same name already exists. Example: The user saves the current open diagram to the file

MyDiagram.cnv.

Closing a diagram: The user chooses to close a diagram. A diagram is closed by closing all the views of the diagram. When closing the last view of the diagram, the user must be notified of any unsaved changes. All open diagrams are closed when the application is closed. Example: The user closes the active diagram.

2.2.2 Diagram drawing

Inserting a new element into a diagram: The user has to select the correct element to be inserted and then draw the element into the diagram. Drawing technique depends on the element. For example, a rectangle is drawn by clicking at the left upper corner of the rectangle and then at the right bottom corner. The program checks if the inserted element meets the syntactic requirements. If the element is not syntactically valid, it is highlighted and the user is notified. Example: The user selects a circle element and draws it by first selecting the center point and then specifying the radius.

Connecting elements together: The user selects a connection line element and makes the connection between two desired elements by selecting them. The program makes syntactic checks for the connection line and the connected elements. Example: The user selects a connection line element and then selects a rectangle and a circle. Then the rectangle and the circle are connected by a line.

Editing an element: The user selects an element from the diagram to be edited. The selected element can be modified by resizing, moving or changing properties of the element. The program retains connections between elements. Example: The user selects a rectangle element and writes a few lines of text into it.

Copying, cutting and pasting elements: The user can select one or more elements, and copy or cut them to the clipboard. Elements in clipboard can be pasted to another location on the diagram. The program checks if the operation is valid from syntactic properties of the affected elements. Example: The user copies a rectangle to the clipboard and pastes several copies of it to the diagram.

Undoing an operation: The user can undo one or more previous operations that were committed. The program keeps track of the operations. The number of the previous operations that are saved for undo operation is limited to some number (for example, 10). Example: The user removes an element by accident and undoes the last command and the program restores the removed element and its connections.

Including elements/diagrams inside other elements: The user picks an element to be inserted. Then he draws the new element inside an existing element. The inclusion can be either physical or logical. The physical inclusion is appropriate when the included element is a simple element like a rectangle. But if the included element is a complex element and if it makes the diagram

look messy when physically included in the diagram, then the logical inclusion is appropriate. A reference to the included element/diagram is saved with the host element. A host element is the element inside which another element/diagram is included. The program checks if the operation is valid from syntactic properties of the affected elements. Example: The user draws a diagram of two circles inside a rectangle element.

Including a text into the diagram: The user can include text independent of any element into the diagram. Again, this inclusion can either be physical or logical. The user can either write the text on the diagram outside any element, or inside an element or add a reference to an external file. Text can be later modified or deleted by the user and the text will be saved with the diagram Example: The user writes the title of the diagram at the bottom of the diagram.

2.2.3 Element editing

Defining a new element by composition: A user defines a new element by combining existing elements. Syntactic rules can be given to determine the behavior of the new element, which can be implemented with Java. Example: A user creates a new element by combining a rectangle and a circle. The user can save this new element to the desired tool-type in the toolbar and reuse the element later.

Editing existing elements: The users can edit the existing elements of the program by modifying the element's shape or by changing element's properties. Syntactic rules can be modified with Java. The program updates the diagram according to the rules associated with affected elements. Example: The user modifies an old element by adding a new circle shape to it.

2.2.4 Extendibility of the application

Defining a new basic element: The user can extend the application by defining new elements using Java. The application provides an API for creating new elements Example: The user creates a bezierline element for the application.

Defining a new syntactic rule: The user can create new syntactic rules for the application using Java. The application provides an API for creating new rules Example: The user creates a must include rule for the application.

Creating new features: The user can extend the application by creating new features. The modular application structure makes it easy to create new features. Example: The user creates a semantic rule module for the application.

3 User requirements

3.1 Functional requirements

Functional requirements are described using the following structure[Som01]:

Identifier	Number of requirement
Name	Name of requirement
Description	Description of requirement
Priority	Priority from P1 to P3
Function(s)	Cross-reference to system requirements

P1, P2 and P3 are used to define the priorities of each requirement. Priority P1 means the requirement is implemented, P2 means the requirement is implemented if there is time after finishing P1 requirements, and P3 means the requirement that can be implemented by other developers in the future.

3.1.1 Drawings

Identifier	R1
Name	Create
Description	The tool is able to create a new diagram
Priority	P1
Function(s)	F1

Identifier	R2
Name	Edit
Description	The tool is able to edit diagram properties
Priority	P1
Function(s)	F9

Identifier	R3
Name	Save
Description	The tool is able to save a diagram
Priority	P1
Function(s)	F3

Identifier	R4
Name	Open
Description	The tool is able to open a diagram
Priority	P1
Function(s)	F2

Identifier	R5
Name	Export
Description	The tool is able to export a diagram as Encapsulated PostScript

Priority	P3
Function(s)	F6

Identifier	R6
Name	Scroll
Description	The tool is able to position a diagram at different viewports, i.e.scroll large diagrams
Priority	P1
Function(s)	F11

Identifier	R7
Name	Print
Description	The tool is able to print a diagram
Priority	P3
Function(s)	F5

Identifier	R8
Name	Multiple diagrams
Description	The tool can have multiple open diagrams at the same time
Priority	P1
Function(s)	

Identifier	R9
Name	Include
Description	The tool can include one or more element/diagram inside another element/diagram
Priority	P2
Function(s)	

Identifier	R10
Name	Save As
Description	The tool is able to save a diagram with a user specified name
Priority	P1
Function(s)	F4

3.1.2 Editing

Identifier	R11
Name	Draw
Description	Diagrams are created interactively on the screen using

	the menu of the tool, the mouse and the keyboard
Priority	P1
Function(s)	F17

Identifier	R12
Name	Line element
Description	The tool contains a line element
Priority	P1
Function(s)	F17

Identifier	R13
Name	Rectangle element
Description	The tool contains a rectangle element
Priority	P1
Function(s)	F17

Identifier	R14
Name	Circle element
Description	The tool contains a circle element
Priority	P1
Function(s)	F17

Identifier	R15
Name	Creating a new element
Description	The user is able to compose new elements using the existing elements and add the composed element to a element repository
Priority	P1
Function(s)	F18

Identifier	R16
Name	Creating a complex element
Description	The user can create complex elements by phsical inclusion or by reference.
Priority	P1
Function(s)	F19

Identifier	R17
Name	Define syntactic rules for an element
Description	The user is able to define syntactic rules
Priority	P3

Function(s)	F19

Identifier	R18
Name	Define behaviour of an element
Description	The user is able to define semantic rules
Priority	P3
Function(s)	F19

Identifier	R19
Name	Undo and redo actions
Description	The user is able to undo previously made actions, and redo undone actions
Priority	P2
Function(s)	F12

Identifier	R20
Name	Cut, copy and paste elements
Description	The user is able to cut and copy selected elements to the clipboard, and paste elements from the clipboard.
Priority	P2
Function(s)	F13, F14, F15

3.1.3 Execution

Identifier	R21
Name	Run and analyze
Description	The tool is able to execute and analyze diagrams
Priority	P3
Function(s)	F23

3.2 Non-functional requirements

Identifier	R22
Name	Drawing utilities
Description	The file handling and drawing operations should, where reasonable, follow the conventions adopted by established drawing utilities.
Priority	P1
Function(s)	

Identifier	R23

Name	OO-design
Description	Implementation is done using object oriented design and programming techniques
Priority	P1
Function(s)	

Identifier	R24
Name	Used language
Description	The implementation is done using Java and by following the Java coding conventions [Mic99].
Priority	P1
Function(s)	

Identifier	R25
Name	Environment
Description	The primary operating environment is Linux.
Priority	P1
Function(s)	

4 System requirements

4.1 Diagrams

Following standard form is been used[Som01].

Identifier	Number of function
Name	Name of function
Description	What the function does
Pre-conditions	Condition to hold on entry to function
Inputs	Inputs for function
Outputs	Outputs from function
Post-conditions	Conditions to hold after function
Priority	Priorities from P1 to P3
User requirements	Cross-reference to user requirements

Identifier	F1
Name	New
Description	Create a blank diagram.
Pre-conditions	
Inputs	Diagram descriptor
Outputs	
Post-conditions	
Priority	P1

User requirements	R1

Identifier	F2
Name	Open
Description	Open an existing diagram specified by the user.
Pre-conditions	Diagram has to exist.
Inputs	File name
Outputs	Diagram descriptor
Post-conditions	Diagram is displayed to the user and diagram is active
Priority	P1
User requirements	R4

Identifier	F3
Name	Save
Description	Save the active diagram, which is already named by the user, to a file
Pre-conditions	Diagram has to be active
Inputs	Diagram descriptor
Outputs	
Post-conditions	The diagram is marked as unchanged
Priority	P1
User requirements	R3

Identifier	F4
Name	Save As
Description	Save the active diagram, which has not been named already, with the name given by the user.
Pre-conditions	
Inputs	Name to save as, diagram descriptor
Outputs	
Post-conditions	Diagram is saved with the nam given by the user.
Priority	P1
User requirements	R10

Identifier	F5
Name	Print
Description	Print the diagram
Pre-conditions	Diagram has to be active
Inputs	Diagram descriptor
Outputs	
Post-conditions	
Priority	P2
User requirements	R7

Identifier	F6
Name	Export
Description	Exports diagram to Encapsulated PostScript format
Pre-conditions	Diagram has to be active
Inputs	Diagram descriptor
Outputs	
Post-conditions	
Priority	P2
User requirements	R5

Identifier	F7
Name	Close
Description	Close the selected diagram
Pre-conditions	The diagram has to be active
Inputs	Diagram descriptor
Outputs	
Post-conditions	
Priority	P1
User requirements	R22

Identifier	F8
Name	Exit
Description	Terminate the software
Pre-conditions	
Inputs	
Outputs	
Post-conditions	
Priority	P1
User requirements	R22

Identifier	F9
Name	Edit properties
Description	Edit the properties of the diagram
Pre-conditions	Diagram has to be active
Inputs	Diagram descriptor
Outputs	
Post-conditions	Properties are changed
Priority	P1
User requirements	R2

Identifier	F10

Name	Help
Description	Show help, i.e. readme to user
Pre-conditions	
Inputs	
Outputs	
Post-conditions	Help is shown to the user
Priority	P1
User requirements	R22

4.2 Drawings

Identifier	F11
Name	Scroll view
Description	Change the viewport of the diagram
Pre-conditions	Diagram has to be active
Inputs	Diagram descriptor, new viewport location
Outputs	
Post-conditions	The diagram's viewport is changed
Priority	P1
User requirements	R6

Identifier	F12
Name	Undo
Description	Undo the last change made to a diagram.
Pre-conditions	Changes must have been made on active diagram, or some changes must have been undone.
Inputs	Diagram descriptor
Outputs	
Post-conditions	The state before the previous change, or the state before undone change.
Priority	P2
User requirements	R19

Identifier	F13
Name	Cut
Description	Remove the selected elements from the diagram
Pre-conditions	Diagram is active
Inputs	Array of selected element descriptors
Outputs	
Post-conditions	The selected elements are removed from diagram
Priority	P2
User requirements	R20

Identifier	F14

Name	Copy
Description	Copy elements to clipboard
Pre-conditions	Diagram is active
Inputs	Array of element descriptors
Outputs	
Post-conditions	The clipboard contains the selected elements and connections between them
Priority	P2
User requirements	R20

Identifier	F15
Name	Paste
Description	Paste elements from the clipboard
Pre-conditions	Diagram is active
Inputs	Diagram descriptor
Outputs	
Post-conditions	The copied diagram to the clipboard is inserted into the active diagram from the clipboard at the user specified position in the diagram.
Priority	P2
User requirements	R20

Identifier	F16
Name	Delete
Description	Removes selected elements from the diagram
Pre-conditions	Diagram is active
Inputs	Array of descriptors
Outputs	
Post-conditions	The selected elements are removed from the diagram
Priority	P1
User requirements	R22

Identifier	F17
Name	Drawing an element
Description	Draw the element selected from the tool bar in the diagram
Pre-conditions	Diagram is active, element is selected from the tool bar
Inputs	Element descriptor, Element position, Diagram descriptor
Outputs	
Post-conditions	
Priority	P1
User requirements	R11

Identifier	F18
Name	Create a new element
Description	Create a new element by composing
Pre-conditions	Array of element descriptors
Inputs	Diagram descriptor
Outputs	Element descriptor
Post-conditions	A new element is created
Priority	P1
User requirements	R15

Identifier	F19
Name	Edit element's properties
Description	Change properties of the element
Pre-conditions	The element is selected
Inputs	Element descriptor
Outputs	
Post-conditions	The element is updated with given properties, diagram is now unsaved
Priority	P1
User requirements	R22

Identifier	F20
Name	Selecting elements
Description	Select element or elements for editing or moving
Pre-conditions	
Inputs	Element descriptor
Outputs	
Post-conditions	The selected elements are now marked as selected.
Priority	P1
User requirements	R22

Identifier	F21
Name	View toolbar
Description	Display the shape toolbar
Pre-conditions	Shape toolbar has to be closed
Inputs	
Outputs	
Post-conditions	Toolbar is displayed to user on screen
Priority	P1
User requirements	R22

Identifier	F22

Name	View treeview
Description	Display the treeview
Pre-conditions	Treeview has to be closed
Inputs	
Outputs	
Post-conditions	Treeview is displayed to user on screen
Priority	P1
User requirements	R22

Identifier	F23
Name	Execute/Analyze
Description	Support for execution of user defined rules to execute and analyze diagram
Pre-conditions	Rules for diagram has to exist
Inputs	Diagram descriptor
Outputs	Text
Post-conditions	
Priority	P3
User requirements	R21

References

Bra02 Bray, I. K., *An Introduction to Requirements Engineering*. Addison-Wesley, 2002.

Mic99 Microsystems, S., *Code Conventions for the Java Programming Language*, 1999. URL `http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html` .

Pra05 Pragya, L., Project plan., Univercity of Helsinki department of computer science, 2005.

Som01 Sommerville, I., *Software Engineering*. Addison-Wesley, 2001.