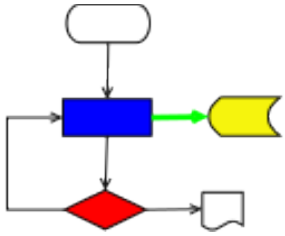
VICI



VISUAL CHART INTERPRETER

Architecture

Publication History

Date	Who	What Changes
25 April 2014	Brenton Ross	Initial version.
2 November 2014	Brenton Ross	Added Increment Plan references.

Table of Contents

1 Introduction

This document describes the architecture for the Visual Chart Interpreter (VICI) project.

1.1 Scope

The document will include the tactics, the conceptual architecture, and the implementation architecture for the VICI project. It will also include details of the interfaces between the components of the system, and also between the system and any externally connected systems.

In addition to the high level design the architecture will also include the policies and guidelines that are to be used during detailed design and implementation.

1.2 Overview

The requirements are examined and a set of tactics chosen to best satisfy them. The tactics imply a set of responsibilities which are then assigned to components of the conceptual model of the proposed system, or to aspects of the development process itself.

The conceptual model is then transcribed onto an implementation model consisting of third party products plus those components that will be developed for this project.

The interfaces between these components are then fully specified so that test harnesses can be constructed for the modules before any other work is undertaken.

Finally the policies and guidelines for the design and implementation are prepared so that future work is done in a consistent manner that preserves the architecture of the system.

1.3 Audience

This document is intended to be used by the developers and maintainers of the VICI project.

2 Tactics

The first step in developing the architecture for a system is to decide on the tactics to be used to satisfy each requirement.

In the following we list each tactic and the requirements it is meant to satisfy.

2.1 Functional Requirements

T1: Construct an administration tool for preparing commands.

A small separate GUI program is proposed as it will potentially be used by different users. It has a different pattern of usage to the main editor. It can be developed independently.

> R1: Provide a means of preparing a command for use by this system so that it can be more easily used by ordinary users.
>
> R3: The system will allow an administrator to specify the EBNF of the syntax for the commands that are supported by this program.
>
> R4: The EBNF used to define the syntax of a command's parameters and options will be that specified in Appendix B
>
> R5: The system will allow an administrator to set up aliases for common commands and some of their parameters.
>
> R90: Allow the user to export one or more prepared commands.
>
> R91: Allow the user to import one or more prepared commands.

T2: Use the administration tool to prepare the commands.

We will need to provide an initial library of prepared commands. We can use the VICI administration program to perform this task.

> R2: The commands listed in Appendix A will be prepared for use with this system.

T3: Construct an editor for the graphical layout of flowchart symbols.

> R6: Provide a means for users to create scripts using the graphical interface.
>
> R15: Allow the user to define the processing order within a script by laying out the commands in a flow chart.
>
> R16: Provide a palette of common flowchart symbols.
>
> R17: The visual scripting language (VSL) will use the symbols described in Appendix C.
>
> R18: Allow groups of symbols to be re-positioned while maintaining the links between symbols.
>
> RQC1: The system must record in the script the user name of anyone who edits it.

RQD3: The editor should automatically backup scripts and allow the user to easily revert to a previous version.

T8: Use a layout program to reorganise the layout of a diagram.

R20: Provide an option that attempts automatic layout of the flowchart diagram.

T9: Allow the user to enter text at any location on the diagram.

R21: Allow the user to insert comments into the flowchart.

T4: Allow user to attach commands to the symbols on the flowchart.

R7: The system will allow the user to select from the set of prepared commands.

R8: The system will allow the user to enter any command and its parameters, not just those that have been prepared for this system.

T5: Display help text for the selected command.

R9: Provide guidance for common commands, including guidance on the parameters and easy to understand documentation.

R30: The editor shall allow the user to view the man page for selected commands.

R31: The editor shall allow the user to view the info page for selected commands.

T6: Provide a user interface that guides the user through the options for a command.

R10: The system shall allow the user to enter short options, long options, parameters and name=value pairs as required by the commands.

R11: Guide the user in selecting the parameters for the commands used in their scripts.

R12: The system will allow the user to select the parameters from a palette of parameters and options applicable to the selected command.

R13: The system will hide or disable options and parameters which are inapplicable.

R14: The system will display a syntax chart for the selected command.

T7: Provide the ability to create script functions.

R19: Allow a group of symbols to be promoted into being a separate function.

R55: The VSL will include the ability to group commands and split them out into a separate function which can be reused.

R56: The VSL will allow the user to attach a function to a menu item so that it can be executed independently.

T10: Provide the means of managing the installation of scripts into the desktop.

R22: The editor will create desktop files according to the freedesktop.org specification. desktop-entry-spec-1.0.html

R23: The editor will create menu entries for the desktop according to the freedesktop.org specification. menu-spec-1.0.html

R24: The editor will place desktop files and menu entries according to the freedesktop.org specification. basedir-spec-0.6.html

R25: The editor will allow the user to remove a script and its associated desktop files and menu entries.

R26: Allow the user to install their script from the editor in which it being created.

T11: Provide a search capability over the help text.

R27: Provide a means of allowing a user to find commands that satisfy their needs.

R28: The editor will allow the user to use the apropos command in order to search for commands to use.

R29: The editor will allow the user to search the descriptions of the prepared commands.

T12: Provide a tag:command database where commands and tags can be classified using other tags, and which can be searched using set expressions.

R32: Allow the user to associate tag identifiers with the commands.

R33: Provide an initial set of tags for commands.

R34: Allow the user to select commands using tags.

R35: Allow the user to build set expressions for the tags, including union and intersection.

R36: The editor will allow the user to classify commands by associating one or more tags with each one.

R37: The search tags will be able to be classified into an hierarchy.

T13: Provide a runtime interpreter that can be controlled down to individual steps.

R38: Allow a user to test their script before installing it.

R39: Allow the user to step through the script one command at a time with a visual indication of what is happening.

R40: The VSL will allow the user to run a section of a script by defining start and break points.

R41: The VSL will allow the user to run all or part of a script in slow motion by inserting a configurable amount of sleep between each command.

R57: The VSL will allow a function to be run as a background task.

R58: The VSL will allow the user to determine which command to execute next based on the exit status of the previous command: either unconditionally, on success, on fail, or on a particular exit status value.

R59: The VSL will allow the user to connect commands using a pipeline. Connections may be from either stdout, stderr, or both.

R63: The VSL will allow commands to be run in the background. Such commands will be terminated when the script completes.

T14: Display the values for all variables during testing.

R39: Allow the user to step through the script one command at a time with a visual indication of what is happening.

R42: Allow a user to observe the internal working of a script while testing it.

R44: The VSL will allow the user to edit the values of variables while the system is paused.

R45: The system will allow the user to observe the values of variables when run from within the editor.

R46: The system will clearly indicate which commands are about to be processed, or are being processed, and which variables are being referenced during testing.

R47: The system will show the stack variables for the thread that is currently being displayed.

R50: The system will allow the user to select and display each thread separately.

T15: Provide snapshot and restore capability for testing.

R43: The VSL will allow the user to save and restore the values of variables and the state of the call stacks so that sections of a script can be re-run.

R51: The system will allow the user to take a snapshot of a file and restore it as necessary for their testing.

T16: Provide a file viewing capability.

R48: The system will allow the user to observe the content of selected files while being run from with the editor. An option to view the tail of the file shall be provided.

R49: The system will allow the user to monitor the size of files and other properties during testing.

T17: Provide a customisation option for the diagram editor.

R52: The VSL will allow the user to define the colours, patterns and textures used for the background of commands and lines to best suit their vision and cultural requirements.

T18: Provide a user configurable window management system.

R53: Provide the user with a configurable editor that allows sub-windows to be sized and positioned.

R54: Automatically configure sub-windows according to the size and shape of the display screen.

T19: Provide a set of built-in commands.

R60: The VSL will include a built in command for changing the current working directory of the script.

R61: The VSL will include a built in command that passes one line of input at a time to its output pipe. This will allow the user to build a loop structure that process each line of input.

R62: The VSL will include a built in command that sets one or more parameters to the values on its input pipe. This will allow the user to build a loop structure that processes variables separated by white space.

R64: The VSL will allow the user to connect the output of a command to a named pipe.

R65: The VSL will allow the user to send a signal to a background command.

T20: Provide the ability to pipe data between processes.

R64: The VSL will allow the user to connect the output of a command to a named pipe.

R66: The VSL will allow the user to connect the output of a command to a file, and may specify over-write or append modes.

R67: The VSL will include references to files.

T21: Provide an in-line file capability with variable expansion.

R68: The VSL will include the equivalent of the in-line file. The in-line file may contain references to variables that will be expanded when the file is referenced.

T22: Provide a clean up function that removes temporary objects.

R69:

T23: Provide a means of automatically signing scripts

R70: The VSL will allow the user to specify a file as being temporary, and such files will be automatically removed when the script terminates.

RQC2: The system must only open or run scripts which have been signed by the user or by one of an administrator controlled list.

T24: Support the use of variables in the scripts.

R71: The VSL will allow the user to store the output of a command into a variable.

R72: The VSL will allow the user to define variables and constant values which can be referenced as (parts of) parameters to commands by preceding their name with a $ sign.

R73: The VSL will allow the user to perform simple arithmetic on numeric variables. One of +, - , *, / and % can be specified as the operator.

R74: The VSL will include built in variables containing the process id of each command started in background mode.

R75: The VSL will include a built in variable containing the exit status of the last command to complete.

R76: The VSL will include built in variables that allow a function to have parameters. These will be numeric indexes into the parameters placed on the call stack.

R77: The VSL will enable the user to define variables that will take the value dropped onto a text entry field in the runtime. The runtime will display a text entry field for each of these variables.

R78: The VSL will enable the user to define mutex variables that can be used to synchronise the processing of two or more threads of control within the script.

R79: The VSL will enable the user to define a semaphore variable and a signalling command that can be used to increment the semaphore.

T25: Provide a runtime program for executing the scripts.

R80: Provide a means for users to run scripts from the graphical desktop.

R81: Allow the user to navigate the script to some folder.

R82: Allow scripts to be triggered from events such as program start, drag-n-drop, or menu action.

R83: Allow the user to terminate any running script from a menu option.

R84: The runtime shall include menu entries for any script function flagged as being attached to a menu.

R85: The runtime may provide text areas for accepting user input and/or displaying stdout and stderr if the script specifies them.

R86: Allow the user to launch their script by dragging and dropping a file name onto the script's desktop window.

R87: Allow the user to open a file browser using a menu in the scripts desktop window.

R88: The runtime shall include options to stop and to restart a script.

T26: Provide a user interface that allows the user to set a crontab entry for a script.

R89: Provide a means of automatically launching a script at a prescribed time.

2.2 Environmental Requirements

T27: Build using open source and cross-platform capable components.

RE1: The system must run on any Linux platform that supports the freedesktop.org specification.

RE2: The system must use open source, GPL components (or those which have no usage restrictions).

T28: Build using autotools.

RE1: The system must run on any Linux platform that supports the freedesktop.org specification.

T29: Verify freedesktop compatibility

RE1: The system must run on any Linux platform that supports the freedesktop.org specification.

2.3 Quality Requirements

T30: Use the Qt framework for GUI components.

RQC3: The system's UI components must present an attractive and non-intimidating interface.

T31: Hide UI components until they are needed.

RQC3: The system's UI components must present an attractive and non-intimidating interface.

T32: Use event driven design (rather than polling).

RQC4: The system should not place any significant extra load on the system beyond what is necessary for the commands being executed.

T33: Use clear and informative error messages.

RQC5: The system must provide clear feedback for all actions and error conditions. This must be provided in clear English and include some guidance on what action is required to resolve any problems.

T34: No privileged components will be allowed.

RQC6: It must not be possible to use the system to gain access that the user would not normally have.

T35: Provide comprehensive administration guides, user guides and tutorials.

RQC7: It should be easy to demonstrate the system. The system should be packaged with a set of tutorials and guides that make it easy to learn how to use.

RQD6: The system shall include an administrator's guide, a user's guide, tutorials and a video introduction that demonstrates a complete script being made, installed and run.

T36: Provide video guides illustrating the complete process of building, installing and running a script.

RQC7: It should be easy to demonstrate the system. The system should be packaged with a set of tutorials and guides that make it easy to learn how to use.

RQD6: The system shall include an administrator's guide, a user's guide, tutorials and a video introduction that demonstrates a complete script being made, installed and run.

T37: Use a software engineering approach to the construction.

RQC8: The runtime component must be dependable so that the user can trust that their scripts will be executed correctly and when needed.

RQT7: Support for testing the components must be included as part of the system.

RQP1: The system shall be developed using a well defined development methodology.

T38: Use a test driven development methodology.

RQC8: The runtime component must be dependable so that the user can trust that their scripts will be executed correctly and when needed.

RQT7: Support for testing the components must be included as part of the system.

RQP1: The system shall be developed using a well defined development methodology.

T48: Provide Linux standard packages.

RQD1: The system must be easy to administer. It should require little more than installing for it to be usable.

T39: Provide a script that promotes scripts to an administrator's site key.

RQD2: The system must allow an administrator to promote a user's script for site wide use.

T40: Use XML for persistent storage and configuration.

RQD4: Any configuration files must use a simple text format or an editor for the format must be included. (XML may be considered simple.)

RQD5: All data will be stored in open accessible formats such as XML.

RQT6: It must be easy to replace the system with an alternative product.

T41: Review design for unnecessary libraries.

RQD7: The number of libraries that this system depends on should be minimised, but not at the expense of best practice.

T42: Use an object oriented language.

RQT1: It must be easy to extend the system to support additional requirements.

T43: Provide clear guidelines and coding standards.

RQT2: It must be easy to understand the programs so that other developers can maintain and extend it.

RQT3: Variables and methods will have a consistent naming standard.

RQT4: It shall be easy to adapt the software for new locales.

RQT5: It shall be possible to adapt the software for any platform on which Linux runs a conforming desktop environment.

RQS4: The design shall use a consistent naming standard.

T44: The static design will include class diagrams showing the relationships between the classes.

RQS1: The design of the system must be easy to follow.

T45: The dynamic design will include automatically generated collaboration diagrams generated from trace information.

RQS1: The design of the system must be easy to follow.

T46: The system will include Makefiles and configure scripts.

RQS2: It shall be straightforward to build the system.

T47: The system will be created from separately constructed modules.

RQS5: The design shall allow the system to be extended to cater for additional functionality.

RQS6: The system shall be constructed in a modular manner with the separate components capable of independent testing.

RQS7: Components of the system will be as independent of each other as is practical so that they may be changed without adversely affecting other components.

3 Responsibilities

In this section we break each tactic down into a set of responsibilities.

A responsibility is something that can be attributed to either a part of the executable deliverable program, or to some step of the development process.

3.1 Functional Requirements

Req.	Tactic	Responsibilities
R1 R3 R4 R5 R90 R91	T1: Construct an administration tool for preparing commands. (Increment #3)	T1.1: Collect the name of the command.(#3)
		T1.2: Collect the syntax of the options. (#3)
		T1.3: Collect a short description of the command. (#3)
		T1.4: Collect the commands to show the help documentation. (#3)
		T1.5: Construct an XML document from the user input. (#3)
		T1.6: Save the XML document to disk. (#3)
		T1.7: Load the XML document from disk. (#3)
		T1.8: Restore user entered data from the XML document. (#3)
		T1.9: Generate a syntax diagram for the command. (#3)
		T1.10: Validate the EBNF of the option syntax. (#3)
		T1.11: Collect a name and some options to be used as an alias command. (#3)
		T1.12: Define the format of the XML file for commands. (#3)
		T1.13: Export selected commands. (#3)
		T1.14: Import commands. (#3)
R2	T2: Use the administration tool to prepare the commands. (Increment #3)	T2.1...T2.107: Prepare the command. (#3)
		T2.108: Arrange for the prepared commands to be included in the distributed package. (#3)
R6 R15 R16 R17 R18 RQC1	T3: Construct an editor for the graphical layout of flowchart symbols. (Increment #1)	T3.1: Display a set of flowchart symbols in a palette. See Appendix C of the Requirements document. (#1)
		T3.2: Indicate a default symbol on the palette. (#1)
		T3.3: Place default symbol on diagram. (#1)

Req.	Tactic	Responsibilities
RQD3		T3.4: Create a new empty diagram. (#1)
		T3.5: Indicate a symbol has been selected. (#1)
		T3.6: Move selected symbols to new positions on the diagram following the mouse. (#1)
		T3.7: Keep lines connected to other symbols. (#1)
		T3.8: Remove deleted symbols. (#1)
		T3.9: Save the flowchart as an XML file. (#1)
		T3.10: Reconstruct a chart from the contents of an XML file. (#1)
		T3.11: Define the format of the XML file. (#1)
		T3.12: Record the user name in the script.
		T3.13: Create backups automatically.
R20	T8: Use a layout program to reorganise the layout of a diagram. (Increment TBA)	T8.1: Create a description of the diagram in the language of the layout program.
		T8.2: Execute the layout program.
		T8.3: Update the layout of a diagram based on the results from a layout program.
		T8.4: Revert the layout to the previous arrangement when directed to by the user.
R21	T9: Allow the user to enter text at any location on the diagram. (Increment #1)	T9.1: Provide a palette of text attributes. (#1)
		T9.2: Collect the text from the user and place it on the diagram. (#1)
		T9.3: Save the text, its location and attributes, into the diagram XML file. (#1)
		T9.4: Restore the text to the diagram from an XML file. (#1)
		T9.5: Allow the user to edit the text, or its attributes. (#1)
		T9.6: Allow the removal of text from a diagram. (#1)
R7 R8	T4: Allow user to attach commands to the symbols on the flowchart. (Increment #1, #3)	T4.1: Display a list of prepared commands. (#3)
		T4.2: Attach the selected command to the selected symbol. (#3)
		T4.3: Verify the selected command is appropriate for the selected symbol.
		T4.5: Collect a command and its options and attach it to the selected symbol. (#1)

Req.	Tactic	Responsibilities
R9 R30 R31	T5: Display help text for the selected command. (Increment #3)	T5.1: Provide a window for displaying help text. (#3)
		T5.2: Display the short description of prepared commands. (#3)
		T5.3: Display the results of running the help command for prepared commands. (#3)
		T5.4: Display the man page for non-prepared commands. (#3)
		T5.5: Display the info page for the selected command. (#3)
R10 R11 R12 R13 R14	T6: Provide a user interface that guides the user through the options for a command. (Increment #3)	T6.1: Display the syntax chart for the command. (#3)
		T6.2: Display the set of options and parameters available for the command. (#3)
		T6.3: Display the set of variables that are defined. (#3)
		T6.4: Indicate which options and parameters are legal at the current point in the command. (#3)
		T6.5: Build a command line for the command from the user's selection of options and parameters. (#3)
R19 R55 R56	T7: Provide the ability to create script functions. (Increment #1)	T7.1: Replace a group of symbols with a function symbol and place the group of symbols onto a new diagram.
		T7.2: Allow the user to define the name of a function. (#1)
		T7.3: Allow the user to attach a function to a menu item. (#1)
		T7.4: Allow user to arrange menu items.
R22 R23 R24 R25 R26	T10: Provide the means of managing the installation of scripts into the desktop. (Increment #5)	T10.1: Create desktop files for scripts while using the editor. (#5)
		T10.2: Create menu entries for scripts and categories of scripts while using the editor. (#5)
		T10.3: Install desktop files, menu entries and scripts while using the editor. (#5)
		T10.4: Remove desktop files, scripts and menu entries while using the editor. (#5)
R27 R28 R29	T11: Provide a search capability over the help text. (Increment #4)	T11.1: Key word search over the help text of prepared commands. (#4)
		T11.2: Use the apropos command to search for commands using a key word. (#4)
R32 R33	T12: Provide a tag:command database where commands	T12.1: Adding, editing and removing commands from the tag database. (#4)

Req.	Tactic	Responsibilities
R34 R35 R36 R37	and tags can be classified using other tags, and which can be searched using set expressions. (Increment #4)	T12.2: Adding, editing and removing tags from the tag database. (#4)
		T12.3: Classifying commands and tags as members of other tags. (#4)
		T12.4: Saving and loading the tag database. (#4)
		T12.5: Defining the format of the tag database. (#4)
		T12.6: Construct a query based on union and intersection of tags. (#4)
		T12.7: Display list of commands that satisfy a query. (#4)
R38 R39 R40 R41 R57 R58 R59 R63	T13: Provide a runtime interpreter that can be controlled down to individual steps. (Increment #1, #2)	T13.1: Construct an execution model for the script based on the diagram. (#1)
		T13.2: Create thread objects for each thread of control in the script. (#1)
		T13.3: Thread objects may wait for user input before executing next command. (#2)
		T13.4: Thread objects may wait for a specified interval before executing next command. (#2)
		T13.5: Thread objects can be repositioned to other locations in the execution model. (#2)
		T13.6: The execution model can be tagged with break points that stop the thread objects from proceeding until they have user input. (#2)
		T13.7: Create new thread objects when starting a background function or command. (#1)
		T13.8: Determine which command to execute next according to the execution model and the exit status of the previous command. (#1)
		T13.9: The thread is to start one or more process objects according to the execution model. (#1)
		T13.10: Terminate all threads when processing completes. (#1)
R39 R42 R44 R45 R46 R47 R50	T14: Display the values for all variables during testing. (Increment #2)	T14.1: Display a representation of the thread object on the diagram at its correct point in the execution model. (#2)
		T14.2: Allow user to specify the processing speed. (#2)
		T14.3: Display the value of each global variable. (#2)

Req.	Tactic	Responsibilities
		T14.4: Allow user to select which threads to display. (#2)
		T14.5: Display the value of each thread specific variable. (#2)
		T14.6: Allow user to edit the value of variables. (#2)
R43 R51	T15: Provide snapshot and restore capability for testing. (Increment TBA)	T15.1: Save and load for the variables.
		T15.2: Define the format of the snapshot file.
		T15.3: Allow user to define names for snapshot files.
		T15.4: Allow user to select files for inclusion in the snapshot.
		T15.5: Save and restore of snapshot files.
R48 R49	T16: Provide a file viewing capability. (Increment TBA)	T16.1: Display a list of the files referenced by the script, including name, permissions, and size.
		T16.2: Provide windows that can be used to display text files, or the tail thereof.
R52	T17: Provide a customisation option for the diagram editor. (Increment #7)	T17.1: Provide a logical to actual mapping for the symbol attributes. (#7)
		T17.2: Provide a palette of colours for the symbol attributes. (#7)
		T17.3: Provide a palette of patterns for the arrows and lines. (#7)
		T17.4: Provide a palette of textures for the area symbols. (#7)
R53 R54	T18: Provide a user configurable window management system. (Increment #7)	T18.1: Allow windows to be docked or floating. (#7)
		T18.2: Allow windows to be repositioned and resized. (#7)
		T18.3: Preserve user's settings. (#7)
		T18.4: Provide an ability to reset windows to a default state. (#7)
R60 R61 R62 R64 R65	T19: Provide a set of built-in commands. (Increment #1)	T19.1: Provide a cd command that changes the working directory of a thread. (#1)
		T19.2: Provide a “for each line” command. (#1)
		T19.3: Provide a “for each” command. (#1)
		T19.4: Construct and remove named pipes. (#1)
		T19.5: Provide a signal command that sends a signal to another process. (#1)

Req.	Tactic	Responsibilities
R64 R66 R67	T20: Provide the ability to pipe data between processes. (Increment #1)	T20.1: Allow a pipe to be connected to a named pipe. (#1)
		T20.2: Allow a pipe to be connected to a file in either append or over-write mode. (#1)
R68	T21: Provide an in-line file capability with variable expansion. (Increment #1)	T21.1: Expand variables embedded in in-line file. (#1)
R69	T22: Provide a clean up function that removes temporary objects. (Increment #1)	T22.1: Remove temporary files. (#1)
R70 RQC2	T23: Provide a means of automatically signing scripts (Increment TBA)	T23.1: Sign a script when saving.
		T23.2: Only allow a script to be opened if it has been signed by an approved key, or the user's key.
R71 R72 R73 R74 R75 R76 R77 R78 R79	T24: Support the use of variables in the scripts. (Increment #1)	T24.1: Assign the output of commands to variables. (#1)
		T24.2: Expand command lines with the current value of variables. (#1)
		T24.3: Perform arithmetic operations on variables. (#1)
		T24.4: Provide built in variables containing process ids of background threads. (#1)
		T24.5: Provide built in variables containing the exit status of the previous command. (#1)
		T24.6: Provide thread specific built in variables that are passed to functions as parameters. (#1)
		T24.7: Provide variables that represent the text field for drag-n-drop.
		T24.8: Provide mutex variables. (#1)
		T24.9: Provide semaphore variables that increments on receipt of a signal. (#1)
R80 R81 R82 R83 R84	T25: Provide a runtime program for executing the scripts. (Increment #1)	T25.1: Provide a GUI program to run scripts. (#1)
		T25.2: Provide a directory selection dialog to set the working directory. (#1)
		T25.3: Provide a file selection dialog to select the script to run. (#1)

Req.	Tactic	Responsibilities
R85 R86 R87 R88		T25.4: Dynamically create the menu according to the script. (#1)
		T25.5: Dynamically create the text fields for drag-n-drop.
		T25.6: Provide a menu option that terminates the running script. (#1)
		T25.7: Dynamically create the text fields for user input and script output and error messages. (#1)
		T25.8: Detect changes to drag-n-drop fields and cause the script to start the appropriate function.
		T25.9: Pause and restart a running script. (#1)
R89	T26: Provide a user interface that allows the user to set a crontab entry for a script. (Increment #6)	T26.1: Allow the user to select the script to be run. (#6)
		T26.2: Update crontab (#6)
		T26.3: Allow the user to define the schedule for a script. (#6)
		T26.4: Allow the user to remove a crontab entry for a script. (#6)

3.2 Environmental Requirements

Req.	Tactic	Responsibilities
RE1 RE2	T27: Build using open source and cross-platform capable components. (All Increments)	T27.1: Verify license of each component.
RE1	T28: Build using autotools. (All Increments)	T28.1: Use autotools.
RE1	T29: Verify freedesktop compatibility. (All increments from #5)	T29.1: Verify the system uses the freedesktop structure.

3.3 Quality Requirements

Req.	Tactic	Responsibilities
RQC3	T30: Use the Qt framework for GUI components. (All increments)	T30.1: Selection of Qt libraries.
	T31: Hide UI components until they are needed. (from Increment #7)	T31.1: Manage the display of windows and other GUI components according to the actions being performed.
RQC4	T32: Use event driven design. (rather than polling) (All increments)	T32.1: Rejection of any component that includes polling or other “busy-wait” mechanisms.
RQC5	T33: Use clear and informative error messages. (All increments)	T33.1: Easy to understand messages.
		T33.2: Guidance on resolution is included.
RQC6	T34: No privileged components will be allowed. (All increments)	T34.1: There are no components of the system that have anything other than the user's privileges.
RQC7 RQD6	T35: Provide comprehensive administration guides, user guides and tutorials. (Increment TBA)	T35.1: Write the guides and tutorials.
		T35.2: Verify that the guides are appropriate
	T36: Provide video guides illustrating the complete process of building, installing and running a script. (Increment TBA)	T36.1: Create a video tutorial.
RQC8 RQT7 RQP1	T37: Use a software engineering approach to the construction. (All increments)	T37.1: Define the tasks required to complete the project.
	T38: Use a test driven development methodology. (All increments)	T38.1: Build test harnesses for interfaces.
		T38.2: Build test harnesses for component modules.
RQD1	T48: Provide Linux standard packages. (Increment TBA)	T48.1: Provide an rpm package.
		T48.2: Provide a deb package.
RQD2	T39: Provide a script that promotes scripts to an administrator's site key. (Increment TBA)	T39.1: Provide a promotion script.
RQD4 RQD5 RQT6	T40: Use XML for persistent storage and configuration. (All increments)	T40.1: Verify persistent storage is XML.
RQD7	T41: Review design for unnecessary libraries. (All increments)	T41.1: Conduct design reviews.

Req.	Tactic	Responsibilities
RQT1	T42: Use an object oriented language. (All increments)	T42.1: Select appropriate language.
RQT2 RQT3 RQT4 RQT5 RQS4	T43: Provide clear guidelines and coding standards. (All increments)	T43.1: Specify the coding standards.
		T43.2: Specify the guidelines for code level documentation.
		T43.3: Specify the guidelines for making the software locale independent.
		T43.4: Conduct reviews of the code.
RQS1	T44: The static design will include class diagrams showing the relationships between the classes. (All increments)	T44.1: Construct class diagrams.
		T44.2: Conduct design reviews.
	T45: The dynamic design will include automatically generated collaboration diagrams generated from trace information. (All increments)	T45.1: Include trace code.
		T45.2: Generate trace logs.
		T45.3: Generate collaboration diagrams.
RQS2	T46: The system will include Makefiles and configure scripts. (All increments)	T46.1: Create configure and make scripts.
RQS5 RQS6 RQS7	T47: The system will be created from separately constructed modules. (All increments)	T47.1: Create a modular design.
		T47.2: Review design for unnecessary interactions between modules.

4 Conceptual Model

This section allocates the responsibilities from the previous section to conceptual modules. This provides the basic structure to the design of the system.

4.1 Process Tasks

Some responsibilities are handled by the development process rather than the constructed system. This section lists the tasks necessary to satisfy the process responsibilities.

4.1.1 Architecture

This task involves defining the high level design of the system and setting the policies and guidelines for the remainder of the project. It also includes:

T37.1: Define the tasks required to complete the project.

T42.1: Select appropriate language.

T43.1: Specify the coding standards.

T43.2: Specify the guidelines for code level documentation.

T43.3: Specify the guidelines for making the software locale independent.

T47.1: Create a modular design.

4.1.2 Interface Stubs

This task involves creating stub versions of the interfaces between each component so that they may each be tested in isolation.

T38.1: Build test harnesses for interfaces.

4.1.3 Component Exploration

This task involves becoming familiar with the third party products that have been selected for the project.

4.1.4 Test Planning

Describe how the testing will be organised and what testing is proposed for the project. For this project there will be sections for system testing and unit testing.

4.1.5 Application Design

This task is responsible for creating the class diagrams that define the structure of the application programs. It includes:

T44.1: Construct class diagrams.

4.1.6 User Interface Design

This task is responsible for creating the layout of the visual components of the application programs.

4.1.7 Use Case Design

This task involves defining the sequence of events that the classes generate as each use case is processed.

4.1.8 Persistent Storage Design

We need to define the format, and rules, that apply to the storage of the application's data. This usually involves experts in the storage component, and can to some extent be started prior to the application specific work. It will usually involve creating some additional classes to interface the application specific classes to the storage mechanisms.

The following responsibilities can be assigned to this task:

T1.12: Define the format of the XML file for commands.

T3.11: Define the format of the XML file for the diagram.

T12.5: Defining the format of the tag database.

T15.2: Define the format of the snapshot file.

4.1.9 Test Case Design and Implementation

This task involves creating the unit test harnesses so that we have something to test against when we do the implementation.

4.1.10 Class Design

This task involves creating the interfaces for each class and defining the actions for each method.

4.1.11 Design Review

This task is responsible for ensuring that the design conforms to the policies and guidelines established for the project. It includes the following responsibilities:

T27.1: Verify license of each component.

T30.1: Selection of Qt libraries.

T40.1: Verify persistent storage is XML.

T41.1: Conduct design reviews focussing on finding unnecessary

components..

T44.2: Conduct design reviews focussing on ease of understanding.

T47.2: Review design for unnecessary interactions between modules.

4.1.12 Implementation

This is the task responsible for building the system. It includes the following responsibilities:

T28.1: Use autotools.

T38.2: Build test harnesses for component modules.

T39.1: Provide a promotion script.

T45.1: Include trace code.

T46.1: Create configure and make scripts.

4.1.13 Code Review

This Task is responsible for ensuring that the code conforms to the standards, policies and guidelines established for the project. It includes the following specific responsibilities:

T32.1: Rejection of any component that includes polling or other “busy-wait” mechanisms.

T33.1: Easy to understand messages.

T33.2: Guidance on resolution is included.

T34.1: There are no components of the system that have anything other than the user's privileges.

T43.4: Conduct reviews of the code.

4.1.14 Configuration Preparation

This task is responsible for preparing the configuration data that is required for the correct operation of the system.

T2.1...T2.107: Prepare the commands.

4.1.15 Documentation

This task is responsible for preparing the deliverable documentation for the project. It includes:

T35.1: Write the guides and tutorials.

T36.1: Create a video tutorial.

T45.2: Generate trace logs.

T45.3: Generate collaboration diagrams.

4.1.16 Documentation Review

This task involves verifying that the documentation is up to standard. It includes:

T35.2: Verify that the guides are appropriate

4.1.17 Testing

This task is responsible for verifying that each use case can be successfully completed.

4.1.18 Packaging

This task is responsible for creating a package that can be easily installed on a user's computer.

T2.108: Arrange for the prepared commands to be included in the distributed package.

T29.1: Verify the system uses the freedesktop structure.

T48.1: Provide an rpm package.

T48.2: Provide a deb package.

4.1.19 Integration Testing

This task is responsible for ensuring that the product can be successfully installed and run on a user's machine.

4.2 Components

These are the conceptual modules that will be the basis for the implementation modules that will make up the delivered product.

4.2.1 Administration GUI

This component will allow an administrative user to configure the application, prepare commands and to promote scripts for site wide use. It has the following responsibilities:

T1.1: Collect the name of the command.

T1.2: Collect the syntax of the options.

T1.3: Collect a short description of the command.

T1.4: Collect the commands to show the help documentation.

T1.5: Construct an XML document from the user input.

T1.6: Save the XML document to disk.

T1.7: Load the XML document from disk.

T1.8: Restore user entered data from the XML document.

T1.9: Generate a syntax diagram for the command.

T1.10: Validate the EBNF of the option syntax.

T1.11: Collect a name and some options to be used as an alias command.

T1.13: Export selected commands to an XML file.

T1.14: Import commands from an XML file.

4.2.2 Graphical Script Editor

This component allows the user to create and edit a script using a graphical flowchart like interface. It has the following responsibilities:

T3.1: Display a set of flowchart symbols in a palette. See Appendix C of the Requirements document.

T3.2: Indicate a default symbol on the palette.

T3.3: Place default symbol on diagram.

T3.4: Create a new empty diagram.

T3.5: Indicate a symbol has been selected.

T3.6: Move selected symbols to new positions on the diagram following the mouse.

T3.7: Keep lines connected to other symbols.

T3.8: Remove deleted symbols.

T3.9: Save the flowchart as an XML file.

T3.10: Reconstruct a chart from the contents of an XML file.

T3.12: Record the user name in the script.

T3.13: Create backups automatically.

T8.1: Create a description of the diagram in the language of the layout program.

T8.2: Execute the layout program.

T8.3: Update the layout of a diagram based on the results from a layout program.

T8.4: Revert the layout to the previous arrangement when directed to by the user.

T9.1: Provide a palette of text attributes.

T9.2: Collect the text from the user and place it on the diagram.

T9.3: Save the text, its location and attributes, into the diagram XML file.

T9.4: Restore the text to the diagram from an XML file.

T9.5: Allow the user to edit the text, or its attributes.

T9.6: Allow the removal of text from a diagram.

T4.1: Display a list of prepared commands.

T4.2: Attach the selected command to the selected symbol.

T4.3: Verify the selected command is appropriate for the selected symbol.

T4.5: Collect a command and its options and attach it to the selected symbol.

T5.1: Provide a window for displaying help text.

T5.2: Display the short description of prepared commands.

T5.3: Display the results of running the help command for prepared commands.

T5.4: Display the man page for non-prepared commands.

T5.5: Display the info page for the selected command.

T6.1: Display the syntax chart for the command.

T6.2: Display the set of options and parameters available for the command.

T6.3: Display the set of variables that are defined.

T6.4: Indicate which options and parameters are legal at the current point in the command.

T6.5: Build a command line for the command from the user's selection of options and parameters.

T7.1: Replace a group of symbols with a function symbol and place the group of symbols onto a new diagram.

T7.2: Allow the user to define the name of a function.

T7.3: Allow the user to attach a function to a menu item.

T7.4: Allow user to arrange menu items.

T14.1: Display a representation of the thread object on the diagram at its correct point in the execution model.

T14.2: Allow user to specify the processing speed.

T14.3: Display the value of each global variable.

T14.4: Allow user to select which threads to display.

T14.5: Display the value of each thread specific variable.

T14.6: Allow user to edit the value of variables.

T15.1: Save and load for the variables.

T15.3: Allow user to define names for snapshot files.

T15.4: Allow user to select files for inclusion in the snapshot.

T15.5: Save and restore of snapshot files.

T16.1: Display a list of the files referenced by the script, including name, permissions, and size.

T16.2: Provide windows that can be used to display text files, or the tail thereof.

T17.1: Provide a logical to actual mapping for the symbol attributes.

T17.2: Provide a palette of colours for the symbol attributes.

T17.3: Provide a palette of patterns for the arrows and lines.

T17.4: Provide a palette of textures for the area symbols.

T18.1: Allow windows to be docked or floating.

T18.2: Allow windows to be repositioned and resized.

T18.3: Preserve user's settings.

T18.4: Provide an ability to reset windows to a default state.

T31.1: Manage the display of windows and other GUI components according to the actions being performed.

4.2.3 Layout Program

This is a component which takes a description of the nodes and edges of a graph and calculates the optimum layout.

4.2.4 Desktop Installer

This component is responsible for installing scripts onto the user's desktop, and removing them when they are no longer required. It has the following responsibilities:

T10.1: Create desktop files for scripts while using the editor.

T10.2: Create menu entries for scripts and categories of scripts while using the editor.

T10.3: Install desktop files, menu entries and scripts while using the editor.

T10.4: Remove desktop files, scripts and menu entries while using the editor.

4.2.5 Command Search Facility

This component allows the user to find commands for whatever task they have in mind. It has the following responsibilities:

T11.1: Key word search over the help text of prepared commands.

T11.2: Use the apropos command to search for commands using a key word.

T12.1: Adding, editing and removing commands from the tag database.

T12.2: Adding, editing and removing tags from the tag database.

T12.3: Classifying commands and tags as members of other tags.

T12.4: Saving and loading the tag database.

T12.5: Defining the format of the tag database.

T12.6: Construct a query based on union and intersection of tags.

T12.7: Display list of commands that satisfy a query.

4.2.6 Script Interpreter

This component executes the scripts by determining which commands to execute based on the exit status of the previous commands. It is used by both the Script Runtime component and the Graphical Script Editor. It has the following responsibilities:

T13.1: Construct an execution model for the script based on the diagram.

T13.2: Create thread objects for each thread of control in the script.

T13.3: Thread objects may wait for user input before executing next command.

T13.4: Thread objects may wait for a specified interval before executing next command.

T13.5: Thread objects can be repositioned to other locations in the execution model.

T13.6: The execution model can be tagged with break points that stop the thread objects from proceeding until they have user input.

T13.7: Create new thread objects when starting a background function or command.

T13.8: Determine which command to execute next according to the execution model and the exit status of the previous command.

T13.9: The thread is to start one or more process objects according to the execution model.

T13.10: Terminate all threads when processing completes.

T19.1: Provide a cd command that changes the working directory of a thread.

T19.2: Provide a “for each line” command.

T19.3: Provide a “for each” command.

T19.4: Construct and remove named pipes.

T19.5: Provide a signal command that sends a signal to another process.

T20.1: Allow a pipe to be connected to a named pipe.

T20.2: Allow a pipe to be connected to a file in either append or over-write mode.

T21.1: Expand variables embedded in in-line file.

T22.1: Remove temporary files.

T24.1: Assign the output of commands to variables.

T24.2: Expand command lines with the current value of variables.

T24.3: Perform arithmetic operations on variables.

T24.4: Provide built in variables containing process ids of background threads.

T24.5: Provide built in variables containing the exit status of the previous command.

T24.6: Provide thread specific built in variables that are passed to functions as parameters.

T24.7: Provide variables that represent the text field for drag-n-drop.

T24.8: Provide mutex variables.

T24.9: Provide semaphore variables that increments on receipt of a signal.

4.2.7 Script Runtime

This component provides the user interface around the script interpreter component. It has the following responsibilities:

T25.1: Provide a GUI program to run scripts.

T25.2: Provide a directory selection dialog to set the working directory.

T25.3: Provide a file selection dialog to select the script to run.

T25.4: Dynamically create the menu according to the script.

T25.5: Dynamically create the text fields for drag-n-drop.

T25.6: Provide a menu option that terminates the running script.

T25.7: Dynamically create the text fields for user input and script output and error messages.

T25.8: Detect changes to drag-n-drop fields and cause the script to start the appropriate function.

T25.9: Pause and restart a running script.

4.2.8 Secure Signing Facility

This component is responsible for creating a secure signature for a script, and confirming that a script has an approved key. It has the following responsibilities:

T23.1: Sign a script when saving.

T23.2: Only allow a script to be opened if it has been signed by an approved key, or the user's key.

4.2.9 Graphical Crontab Editor

This component is responsible for allowing the user to set up crontab entries for their scripts. It has the following responsibilities:

T26.1: Allow the user to select the script to be run.

T26.2: Update crontab

T26.3: Allow the user to define the schedule for a script.

T26.4: Allow the user to remove a crontab entry for a script.

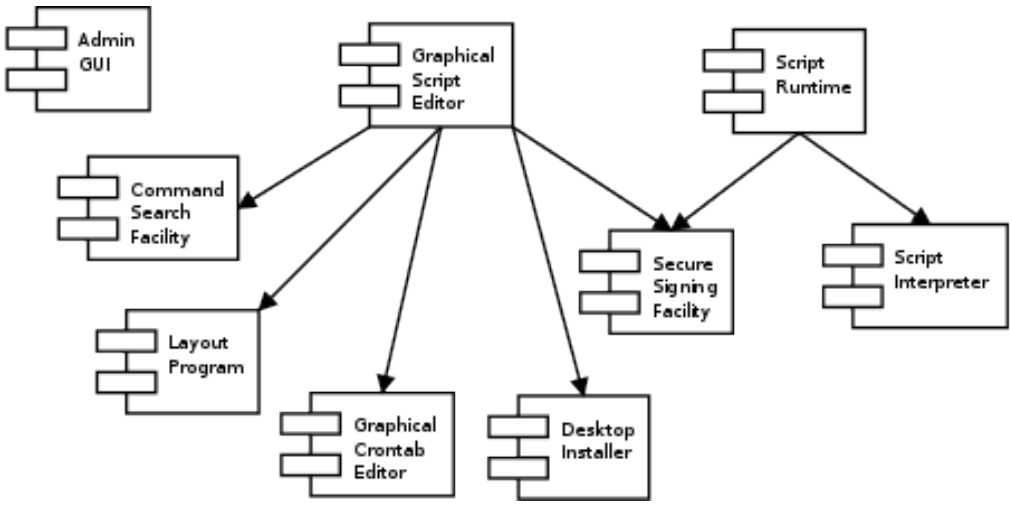


Figure 1: Conceptual Model

5 Logical Model

A logical architecture has a focus on design of component interactions, connection mechanisms and protocols, interface design and specification, and providing contextual information for component users.

5.1 Implementation Modules

This section describes the components as they are to be implemented. It breaks out the programs, libraries and persistent data stores into separate components.

5.1.1 The vici-adm Program

This program provides the user interface for the administration of the VICI system. It is part of the Administration GUI conceptual component that addresses the following responsibilities:

T1.1: Collect the name of the command.

T1.2: Collect the syntax of the options.

T1.3: Collect a short description of the command.

T1.4: Collect the commands to show the help documentation.

T1.5: Construct an XML document from the user input.

T1.8: Restore user entered data from the XML document.

T1.11: Collect a name and some options to be used as an alias command.

T1.13: Export selected commands to an XML file.

T1.14: Import commands from an XML file.

This component will use the libxml2 library to load and save the XML, an EBNF parser to validate the EBNF, and a syntax chart generator to create the syntax diagrams. It will have the following derived responsibilities:

1. Collect the EBNF for the command's options.
2. Display the syntax chart for the command's options.

The component will be developed during increment #3.

5.1.2 The vici-cmd Database

This will be an XML file containing the prepared commands. It will be written under the control of vici-adm and read by vici-ed.

The component will be developed during increment #3.

5.1.3 The libcfi Library

This is a library of common facilities infrastructure components. It includes a useful exception class, logging, configuration data access and tracing. It also includes a C++ wrapper around the general purpose C XML library.

It addresses the following responsibilities:

T1.6: Save the XML document to disk.

T1.7: Load the XML document from disk.

T1.13: Exporting commands.

T1.14: Importing commands.

T3.9: Save the flowchart as an XML file.

T3.10: Reconstruct a chart from the contents of an XML file.

T12.4: Saving and loading the tag database.

T45.1: Include trace code.

T45.2: Generate trace logs.

The component will be developed during increment #1.

5.1.4 The trc2dot Program

This program uses the trace logs to generate a file for the GraphViz dot program. This is used to generate the collaboration diagrams for the design documents. It addresses the following responsibilities:

T45.3: Generate collaboration diagrams.

The component will be developed during increment #1.

5.1.5 The libxml2 Library

This is a third party component that provides the ability to save and load XML files. It addresses the following responsibilities:

T1.6: Save the XML document to disk.

T1.7: Load the XML document from disk.

T1.13: Exporting commands.

T1.14: Importing commands.

T3.9: Save the flowchart as an XML file.

T3.10: Reconstruct a chart from the contents of an XML file.

T12.4: Saving and loading the tag database.

5.1.6 The libebnf Library

This library will be responsible for parsing an EBNF specification for the options of a command. It addresses the following responsibilities:

T1.10: Validate the EBNF of the option syntax.

In addition it will also address the following derived responsibilities:

1. Validate the options for a command against a provided EBNF.
2. Provide a list of valid next symbols for a partial option list for a specific command.

The component will be developed during increment #3.

5.1.7 The libsyntax Library

This library will be responsible for displaying a syntax chart for the EBNF of a given command. It addresses the following responsibilities:

T1.9: Generate a syntax diagram for the command.

The component will be developed during increment #3.

5.1.8 The libcc Library

This is a library of objects that are passed around between the major components. Such objects are sometimes known as currency classes, hence the libcc name. It provides a common implementation for objects which appear on the interfaces between modules, such as windows passed as parameters or parse trees passed to the syntax library.

The component will be developed during increment #1.

5.1.9 The libQtGui Library

This third party library will provide the user interface components for the VICI project.

5.1.10 The libQtCore Library

This third party library will provide various components including one that wraps a running process which will form the basis of the interpreter.

5.1.11 The symbol Library

This component is responsible for the symbols used on the flowchart diagrams. It addresses the following responsibilities:

T3.1: Display a set of flowchart symbols in a palette. See Appendix C of the Requirements document.

T3.2: Indicate a default symbol on the palette.

T17.1: Provide a logical to actual mapping for the symbol attributes.

T17.2: Provide a palette of colours for the symbol attributes.

T17.3: Provide a palette of patterns for the arrows and lines.

T17.4: Provide a palette of textures for the area symbols

T9.1: Provide a palette of text attributes.

It will also have the derived responsibilities:

1. Drawing the symbols on the provided canvas.

The component will be developed during increment #1.

5.1.12 The canvas Library

This library will be responsible for allowing the user to draw their diagrams. It addresses the following responsibilities:

T3.3: Place default symbol on diagram.

T3.5: Indicate a symbol has been selected.

T3.6: Move selected symbols to new positions on the diagram following the mouse.

T3.7: Keep lines connected to other symbols.

T3.8: Remove deleted symbols.

T3.9: Save the flowchart as an XML file.

T3.10: Reconstruct a chart from the contents of an XML file.

T8.1: Create a description of the diagram in the language of the layout program.

T8.2: Execute the layout program.

T8.3: Update the layout of a diagram based on the results from a layout program.

T8.4: Revert the layout to the previous arrangement when directed to by the user.

T7.1: Replace a group of symbols with a function symbol and place the group of symbols onto a new diagram.

T7.2: Allow the user to define the name of a function.

T9.2: Collect the text from the user and place it on the diagram.

T9.3: Save the text, its location and attributes, into the diagram XML file.

T9.4: Restore the text to the diagram from an XML file.

T9.5: Allow the user to edit the text, or its attributes.

T9.6: Allow the removal of text from a diagram.

The component will be developed during increment #1. The automatic layout functionality will be added in a future increment TBA.

5.1.13 The GraphViz dot Program

This third party program can be used to calculate the layout for a diagram. It implements the Layout Program component of the conceptual design.

5.1.14 The command Library

This component provides the user interface for the command selection and parameter entry functions for the vici-ed program. It has the following responsibilities:

T4.1: Display a list of prepared commands.

T4.2: Attach the selected command to the selected symbol.

T4.3: Verify the selected command is appropriate for the selected symbol.

T4.5: Collect a command and its options and attach it to the selected symbol.

T5.1: Provide a window for displaying help text.

T5.2: Display the short description of prepared commands.

T5.3: Display the results of running the help command for prepared commands.

T5.4: Display the man page for non-prepared commands.

T5.5: Display the info page for the selected command.

T6.1: Display the syntax chart for the command.

T6.2: Display the set of options and parameters available for the command.

T6.4: Indicate which options and parameters are legal at the current point in the command.

T6.5: Build a command line for the command from the user's selection of options and parameters.

The component will be developed during increment #3.

5.1.15 The vici-ed Program

The program provides the user interface for the VICI script editor. It is part of the Graphical Script Editor conceptual component. It addresses the following responsibilities:

T3.4: Create a new empty diagram.

T3.12: Record the user name in the script.

T3.13: Create backups automatically.

T7.3: Allow the user to attach a function to a menu item.

T7.4: Allow user to arrange menu items.

T14.1: Display a representation of the thread object on the diagram at its correct point in the execution model.

T14.2: Allow user to specify the processing speed.

T14.4: Allow user to select which threads to display.

T18.1: Allow windows to be docked or floating.

T18.2: Allow windows to be repositioned and resized.

T18.3: Preserve user's settings.

T18.4: Provide an ability to reset windows to a default state.

T31.1: Manage the display of windows and other GUI components according to the actions being performed.

T6.3: Display the set of variables that are defined.

T14.3: Display the value of each global variable.

T14.5: Display the value of each thread specific variable.

T14.6: Allow user to edit the value of variables.

T15.1: Save and load for the variables.

T15.3: Allow user to define names for snapshot files.

T15.4: Allow user to select files for inclusion in the snapshot.

T15.5: Save and restore of snapshot files.

T16.1: Display a list of the files referenced by the script, including name, permissions, and size.

T16.2: Provide windows that can be used to display text files, or the tail thereof.

The component will begin development during increment #1. It will extended in subsequent increments.

5.1.16 The installer Library

This component is responsible for allowing the user of vici-ed to install (or remove) scripts from their desktop. It addresses the following responsibilities:

T10.1: Create desktop files for scripts while using the editor.

T10.2: Create menu entries for scripts and categories of scripts while using the editor.

T10.3: Install desktop files, menu entries and scripts while using the editor.

T10.4: Remove desktop files, scripts and menu entries while using the editor.

The component will be developed during increment #5.

5.1.17 The search Library

This component allows the user to find commands for whatever task they have in mind. It addresses the following responsibilities:

T11.1: Key word search over the help text of prepared commands.

T11.2: Use the apropos command to search for commands using a key word.

T12.1: Adding, editing and removing commands from the tag database.

T12.2: Adding, editing and removing tags from the tag database.

T12.3: Classifying commands and tags as members of other tags.

T12.4: Saving and loading the tag database.

T12.5: Defining the format of the tag database.

T12.6: Construct a query based on union and intersection of tags.

T12.7: Display list of commands that satisfy a query.

The component will be developed during increment #4.

5.1.18 The tag Database

This database component provides the persistent storage for the classification of commands.

The component will be developed during increment #4.

5.1.19 The libvici Library

This component executes the scripts by determining which commands to execute based on the exit status of the previous commands. It is used by both the vici program and the vici-ed program. It addresses the following responsibilities:

T13.1: Construct an execution model for the script based on the diagram.

T13.2: Create thread objects for each thread of control in the script.

T13.3: Thread objects may wait for user input before executing next command.

T13.4: Thread objects may wait for a specified interval before executing next command.

T13.5: Thread objects can be repositioned to other locations in the execution model.

T13.6: The execution model can be tagged with break points that stop the thread objects from proceeding until they have user input.

T13.7: Create new thread objects when starting a background function or command.

T13.8: Determine which command to execute next according to the execution model and the exit status of the previous command.

T13.9: The thread is to start one or more process objects according to the execution model.

T13.10: Terminate all threads when processing completes.

T19.1: Provide a cd command that changes the working directory of a thread.

T19.2: Provide a “for each line” command.

T19.3: Provide a “for each” command.

T19.4: Construct and remove named pipes.

T19.5: Provide a signal command that sends a signal to another process.

T20.1: Allow a pipe to be connected to a named pipe.

T20.2: Allow a pipe to be connected to a file in either append or over-write mode.

T21.1: Expand variables embedded in in-line file.

T22.1: Remove temporary files.

T24.1: Assign the output of commands to variables.

T24.2: Expand command lines with the current value of variables.

T24.3: Perform arithmetic operations on variables.

T24.4: Provide built in variables containing process ids of background threads.

T24.5: Provide built in variables containing the exit status of the previous command.

T24.6: Provide thread specific built in variables that are passed to functions as parameters.

T24.7: Provide variables that represent the text field for drag-n-drop.

T24.8: Provide mutex variables.

T24.9: Provide semaphore variables that increments on receipt of a signal.

The component will be developed during increment #1. Some responsibilities will be done in subsequent increments.

5.1.20 The vici Program

This component provides the user interface around the libvici component. It addresses the following responsibilities:

T25.1: Provide a GUI program to run scripts.

T25.2: Provide a directory selection dialog to set the working directory.

T25.3: Provide a file selection dialog to select the script to run.

T25.4: Dynamically create the menu according to the script.

T25.5: Dynamically create the text fields for drag-n-drop.

T25.6: Provide a menu option that terminates the running script.

T25.7: Dynamically create the text fields for user input and script output and error messages.

T25.8: Detect changes to drag-n-drop fields and cause the script to start the appropriate function.

T25.9: Pause and restart a running script.

The component will be developed during increment #1. Some responsibilities will be done in subsequent increments.

5.1.21 The libsecure Library

This component is responsible for creating a secure signature for a script, and confirming that a script has an approved key. It addresses the following responsibilities:

T23.1: Sign a script when saving.

T23.2: Only allow a script to be opened if it has been signed by an approved key, or the user's key.

The component will be developed during an increment TBA.

5.1.22 The libcron Library

This component is responsible for allowing the user to set up crontab entries for their scripts. It addresses the following responsibilities:

T26.1: Allow the user to select the script to be run.

T26.2: Update crontab

T26.3: Allow the user to define the schedule for a script.

T26.4: Allow the user to remove a crontab entry for a script.

The component will be developed during increment #6.

5.1.23 The libifstubs Library

This is a temporary library providing some scaffolding for the construction process. It allows us to build each module in isolation. Without it we would need to build the components in order of their dependencies, which is not always a possibility. It addresses the following responsibilities:

T38.1: Build test harnesses for interfaces.

T38.2: Build test harnesses for component modules.

The component will be developed during increment #1.

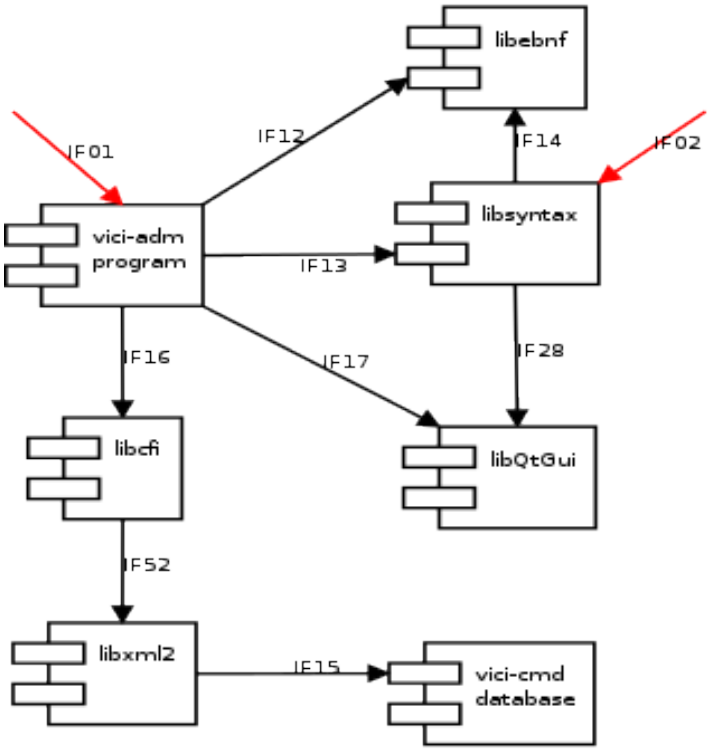


Figure 2: Administration - Logical Diagram

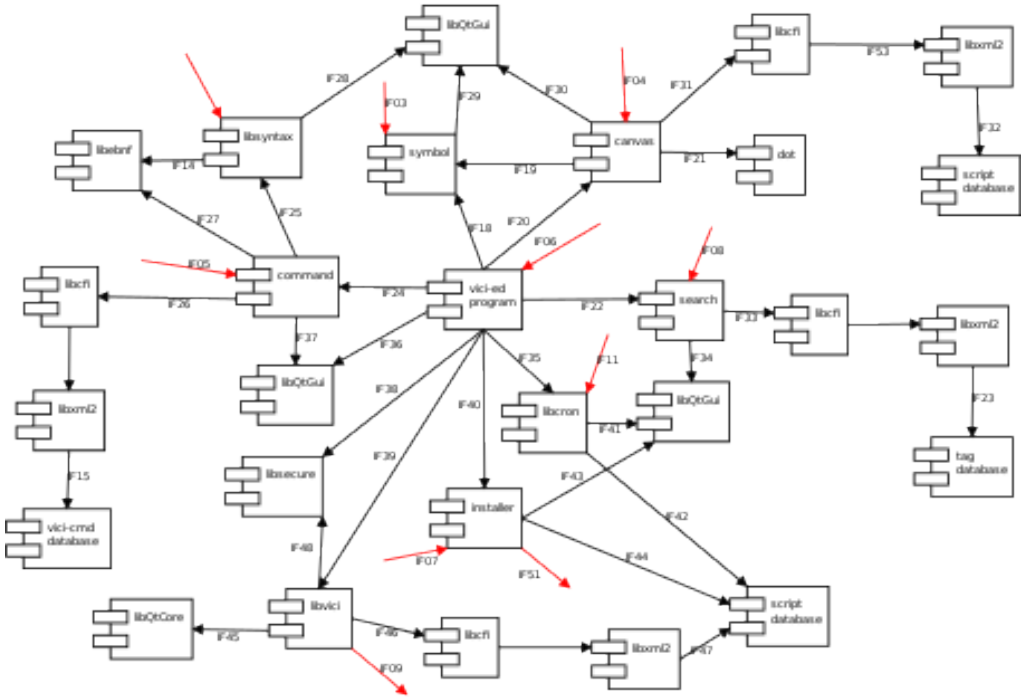


Figure 3: Chart Editor - Logical Diagram

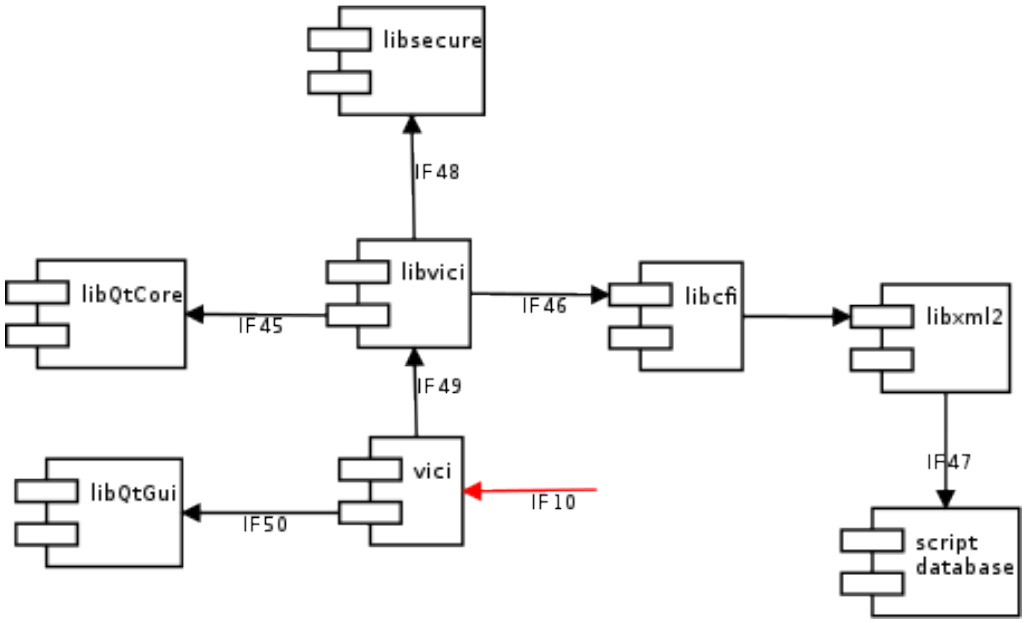


Figure 4: Run Time - Logical Diagram

6 Interfaces

This section describes the interfaces between the components of the system, and between the system and the outside world.

Defining the interfaces in detail allows us to develop the modules independently – changes to the internals of one module will not usually have any impact on the other modules. The alternative can lead to chaotic behaviour in the development process.

An important consideration when documenting the interfaces is to ensure that data is conserved – that all the data that a component requires, or transmits to other components, is provided to it.

6.1 External Interfaces

These are the interfaces between the system and external objects or the system's users.

6.1.1 IF01 Vici Admin UI

This is the graphical interface that allows an administrative user to interact with the vici-adm program.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Command name. (I)
2. Option syntax as EBNF text. (I)
3. Command short description. (I)
4. Help command. (I)
5. Alias name (optional) (I)
6. Error messages (O)
7. Status information (O)

6.1.2 IF02 Syntax Chart

This is an interface that displays a syntax chart in a window.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Syntax diagram (O)

6.1.3 IF03 Symbol Palette

This is the graphical interface that allows a user to select and configure the symbols used on the flowchart diagrams.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Palette of flowchart symbols. (O)
2. Indication of currently selected symbol. (O)
3. Palette of colours. (O)
4. Palette of line patterns. (O)
5. Palette of textures. (O)
6. Palette of text attributes. (O)
7. User selected colour. (I)
8. User selected line pattern. (I)
9. User selected texture. (I)
10. User selected text attributes. (I)

6.1.4 IF04 Canvas UI

This is the user interface for the canvas, the component responsible for allowing the user to create and edit the diagrams.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Flowchart diagram. (O)
2. Current selected symbols. (O)
3. User symbol selection. (I)
4. Desired position of symbol(s). (I)
5. User edit action (I)
6. Name of function. (I/O)
7. User text (I/O)
8. Location of execution points (O)

6.1.5 IF05 Command UI

This is the user interface that allows a user to select a command and set its options and parameters.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Prepared Command List (O)
2. User command selection (I)

3. User suggested command and options. (I)
4. Short description of a prepared command (O)
5. Use selection of type of help text. (I)
6. Help text. (O)
7. Option and parameter list (O)
8. User option or parameter selection. (I)
9. Command line (I/O)

6.1.6 IF06 Vici Editor UI

This is the user interface that allows a user to interact with the editor.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. File name (I)
2. Menu name (I)
3. Function selection (I)
4. Processing delays (I)
5. Function test action (I)
6. Thread list (O)
7. Thread selection (I)
8. Window arrangement (I/O)
9. Global variable list (I/O)
10. Thread variable list (I/O)
11. Snapshot file name (I)
12. File list (O)
13. File selection (I)
14. File attributes (O)
15. File contents (O)

6.1.7 IF07 Installer UI

This is the user interface that allows the user to manage the installation of their scripts onto their desktop.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Category list (O)
2. Category name (I)
3. Category selection (I)
4. Menu list (O)
5. Menu name (I)

6. Menu selection (I)
7. Edit action (I)

6.1.8 IF51 Installing Scripts onto Desktop

This is the interface between the Installer component and the user's desktop files and directories.

Transport Medium: File System

Protocol: C Function calls

Content:
1. Lists of categories (O)
2. Menu files (I)

6.1.9 IF08 Search UI

This is the user interface that allows a user to search for a command.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:
1. Search key (I)
2. Search method (I)
3. Search results (O)
4. Command selection (I)
5. Tag name (I)
6. Classification (I)
7. Tag selection (I)

6.1.10 IF09 Command Execution

This is the interface between the shell interpreter and the underlying operating system.

Transport Medium: Memory.

Protocol: The exec system call.

Content:
1. Command (O)
2. Option string list (O)
3. Environment strings. (O)
4. Standard input pipe (O)
5. Standard output pipe (I)
6. Standard error pipe (I)
7. Signals (O)

8. Exit status (I)

6.1.11 IF10 Vici Runtime

This is the user interface that allows the user to interact with a running script.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:
1. Directory selection (I)
2. File selection (I)
3. Function menu (O)
4. Menu selection (I)
5. Text input (I)
6. Text output (O)
7. Error output (O)
8. Drag-n-drop input (I)

6.1.12 IF11 Cron Interface

This is a user interface that allows the user to schedule a script to start at a specific time.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:
1. Script list (O)
2. Script selection (I)
3. Schedule input (I)
4. Status (O)

6.2 Internal Interfaces

These are the interfaces between the components of the system.

6.2.1 IF12 Vici-adm to libebnf

This is the interface between the vici-adm program and the libebnf library.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Confirm option specification is a valid EBNF.
2. Advise the location and nature of any detected error in the EBNF.

6.2.2 IF13 Vici-adm to libsyntax

This is the interface the administration program uses to generate a syntax chart.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. The sub-window to display into (I)
2. The EBNF of the command (I)

6.2.3 IF14 libsyntax to libebnf

The libsyntax library uses libebnf to parse the EBNF and return a parse structure that it uses to construct the visual representation.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Request the parsing of an EBNF string returning a pointer to a parse tree data structure.

6.2.4 IF15 libxml2 to Command Database

This captures the reading and writing of the prepared commands by the vici-adm program.

Transport Medium: File System

Protocol: XML

Content:
1. An XML file containing the specifications for the prepared commands. (I/O)

6.2.5 IF16 Vici-adm using libcfi

This is the interface between our vici-adm program and the XML wrapper library used to read and write XML.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.6 IF52 libcfi using libxml2

This is the interface between our wrapper and the third party XML library.

Transport Medium: Memory

Protocol: C function calls.

Content:
1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.7 IF17 Vici-adm using libQtGui

This is the interface between our vici-adm program and the third party library responsible for the graphical interface.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Create and remove a main window.
2. Menu options.
3. Text entry and display widgets.
4. Sub-windows.

6.2.8 IF18 Vici-ed using Symbol

This is the interface between then vici-ed program and the component responsible for managing and displaying the symbols used on the flowchart.

Transport Medium: Memory

Protocol: C++ function calls, Qt signals and slots.

Content:
1. Sub-window in which it the symbol library uses to display its visual components. (I)
2. Signal that indicates a change of selected symbol. (I/O)
3. Attributes of text and symbols. (O)

6.2.9 IF19 Canvas using Symbol

This is the interface between the Canvas component and the Symbol component.

Transport Medium: Memory

Protocol: C++ function calls., Qt signals and slots

Content:
1. Drawing surface (I)
2. Position of required symbol (I)
3. Currently selected symbol (O)
4. Attributes of text and symbols. (O)

6.2.10 IF20 Vici-ed using Canvas

This is the interface between then vici-ed program and the component responsible for the layout and display of the diagram.

Transport Medium: Memory

Protocol: C++ function calls, Qt signals and slots.

Content:
1. Sub-window for the canvas to use. (I)
2. Menu actions. (I)
3. Command descriptions (I)
4. Currently selected symbol. (O)
5. Thread execution point. (I)

6.2.11 IF21 Canvas using GraphViz dot

This is the interface between the component responsible for the layout and display of a the diagrams and the program that can calculate the layout of objects on a diagram.

Transport Medium: UNIX Pipe

Protocol: Text files - ASCII.

Content:
1. The specification of the diagram in dot's syntax. (I)
2. The layout of the diagram in dot's plain output file format. (O)

6.2.12 IF22 Vici-ed using Search

This is the interface the vici-ed uses to display the search component.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Sub-window in which the search component operates. (I)
2. Selected command. (O)

6.2.13 IF33 Search using libcfi

This is the interface that Search uses to read and save the tag database.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.14 IF53 libcfi using libxml2

This is the interface between our wrapper and the third party XML library.

Transport Medium: Memory

Protocol: C function calls.

Content:

1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.15 IF34 Search using libQtGui

This interface allows the search component to interact with the user.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Lists of commands and tags. (I)
2. User selections (O)

6.2.16 IF23 libxml2 using the Tag Database

This is the interface that Search uses to read and save the tag database.

Transport Medium: File System

Protocol: XML File.

Content:
1. An XML file containing the tag database. (I/O)

6.2.17 IF24 Vici-ed using Command

This is the interface vici-ed uses to display the Command component that is responsible for constructing the command and its options and parameters.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Sub-window (I)
2. The command (I)
3. The current symbol (I)
4. The parameters and options (O)

6.2.18 IF25 Command using libsyntax

This interface allows the Command component to display the syntax diagram for the current command.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. The EBNF specification for the command (I)
2. The sub-window to display the syntax chart into. (I)

6.2.19 IF26 Command using libcfi

This interface allows the Command component to read the database of

commands.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.20 IF27 Command using libebnf

This interface allows the Command component to suggest valid alternatives for parameters and options.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Request the parsing of an EBNF string returning a pointer to a parse tree data structure.

6.2.21 IF37 Command using libQtGui

This interface allow the command component to provide the graphical elements that allow user interaction.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Command lists.
2. Text display fields.
3. Option and parameter lists.
4. User selections
5. User text input.

6.2.22 IF28 Syntax Library using libQtGui

This interface allows the syntax component to draw the syntax diagram, and the user to interact with the diagram, such as scrolling and zooming.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Drawing operations on a QGraphicsScene canvas.

6.2.23 IF29 Symbol Library using libQtGui

This interface allows the symbol component to display the symbol and text palettes.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Symbol palette
2. Text attributes

6.2.24 IF30 Canvas Library using libQtGui

This interface allows the canvas to display the diagram and interact with the user to edit it.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Drawing operations on a QGraphicsScene canvas.

6.2.25 IF31 Canvas Library using libcfi

This is the interface that the canvas component uses to load and save the scripts.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

6.2.26 IF32 libxml2 using the Script Database

This is the interface where the canvas uses the libxml2 library to load and save the script files.

Transport Medium: File System

Protocol: XML File.

Content:
1. An XML file containing a script. (I/O)

6.2.27 IF35 Vici-ed using libcron

This interface provides an environment in which libcron can interact with the user.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Sub-window (I)
2. Menu actions (I)

6.2.28 IF41 libcron using libQtGui

This interface allows the libcron component to create and use the graphical components necessary to interact with its users.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Lists of scripts. (I)
2. Text entry fields for schedules. (I)
3. User selections (O)

6.2.29 IF42 libcron using the Script Database

This interface allows the libcron to determine what scripts the user may wish to set a schedule for.

Transport Medium: File System

Protocol: C Function Calls.

Content:
1. The list of scripts. (O)

6.2.30 IF40 Vici-ed using Installer

This interface provides an environment in which the installer component can interact with the user.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Sub-window (I)
2. Menu actions (I)

6.2.31 IF43 Installer using libQtGui

This interface allows the installer component to create and use the graphical components necessary to interact with its users.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Lists of scripts. (I)
2. Lists of menu categories (I)
3. User selections (O)

6.2.32 IF44 Installer accessing the Script Database

This interface allows the installer component to determine the scripts available to be installed.

Transport Medium: File System

Protocol: C function calls.

Content:
1. Lists of scripts. (O)

6.2.33 IF38 Vici-ed using libsecure

This interface allows the vici-ed component to determine if its OK to open a script and to sign a script when saving it.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Script file name (I)
2. Verification of signature. (O)
3. Signature for the file (O)

6.2.34 IF36 Vici-ed using libQtGui

This is the interface between our vici-ed program and the third party library responsible for the graphical interface.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Create and remove a main window.
2. Menu options.
3. Text entry and display widgets.
4. Sub-windows.

6.2.35 IF39 Vici-ed using libvici

This is the interface that allows the editor user to test their scripts from the editor interface.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Script to execute. (O)
2. Variables (I/O)
3. Files (I)
4. Thread details (I/O)

6.2.36 IF45 libvici using libQtCore

The Qt library is used to provide a wrapper for system processes.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Command and options (I)
2. Pipes (I/O)
3. Signals (I)
4. Exit status (O)

6.2.37 IF46 libvici using libcfi

This interface allows the libvici component to read the script file.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Read an XML file

2. Get nodes from the document
3. Get properties from the document node.

6.2.38 IF47 libxml2 accessing Script Database

This is the interface where the vici interpreter reads the script files.

Transport Medium: File System

Protocol: XML File.

Content:
1. An XML file containing a script. (I)

6.2.39 IF48 libvici using libsecure

This is the interface that allows the vici interpreter to determine if it should open a script.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Script file name (I)
2. Verification of signature. (O)

6.2.40 IF49 Vici program using libvici

This is the interface that allows the user to interact with their scripts.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Script to execute. (I)
2. Menu actions. (I)
3. Text input (I)
4. Text output (O)
5. Error messages (O)
6. Status information (I)

6.2.41 IF50 Vici Program using libQtGui

This is the interface between our vici program and the third party library responsible for the graphical interface.

Transport Medium: Memory

Protocol: C++ function calls.

Content:

1. Create and remove a main window.
2. Menu options.
3. Text entry and display widgets.