Domain-specific Software Architectures

1. Objectives

- Engineers often draw annotated diagrams of components and the relationships between them to describe the high-level design or architecture of a system. We seek to make this intuitive notion of an architecture more rigorous by defining precise meanings and notations for such specifications.
- Reuse of source modules alone cannot achieve dramatic reductions in cost and schedule, since coding and unit testing typically account for only 10-20% of the overall software life cycle effort.
- Architecture-based analysis coupled with rigorously defined module interface conventions can lead to significant reductions in **defects** and reductions in risk and in the overall life cycle **costs** can by a factor of 3 to 10.

- Where are we in the software life cycle?

Requirements	**Design**	**Implementation**	**….**
	Software Architectures		
		Components	
	Software Component Architecture		
DSSA: Domain-Specific Software Architectures			
	Frameworks		
	Design Patterns		
		Which Programming language?	

2. Definitions

- Domain-specific architectures are a software development technology that promises to greatly reduce the cost and time to develop a family of systems for a common application domain such as satellite command and control.
- DSSA is also an architecture style and process that supports the development of a solution for a group of related applications within a problem domain.
- A domain is defined by a set of common problems, functions, and attributes that applications in such a domain can solve.
- According to [Hay 1994]:

> DSSA can be defined as an assemblage of software components, specialized for a particular type of task (domain), generalized for effective use across that domain, composed in a standardized structure (topology) effective for building successful applications.

3. DSSA Methodology

- First, the domain is analyzed to produce a generic breakdown into functional elements, with defined interfaces and data flows between elements. This can be done by traditional structured analysis or object-oriented analysis.

- Given a generic framework or architecture, developing systems within the domain can be accomplished more efficiently because many components will be reusable and the interfaces will be well-defined when new components must be developed.

4. DSSA main components

- Domain model
- Reference requirements
    - Stable requirements
    - Variable Requirements
- Reference architecture

4.1. Domain model

- The domain analysis is the **initial step** in Domain-Specific Software Architecture, and the analysis of the domain ordinarily consists of input from customers and "domain experts.
- Domain experts are usually familiar with previous systems of this type or other aspects of the domain of interest. They may specialize in a specific area of the domain, or may have actually help develop a similar problem domain or related applications under such a domain.
- The customers add their own requirements for the domain and make modifications based on their needs and interests.
- The customer considers their **needs statement as requirements**, however, the functional requirements define the problem domain, and the design and implementation requirements constrain the architecture.
- The domain model is referenced by developers and other professionals that perform maintenance on specific applications in a domain, and the domain model should provide them with an understanding of the domain aspects.
- It uses the domain dictionary, which is the text reference source for words and phrases found in scenarios and the customer's needs statement, and the context (block) diagram graphically represents the high-level data flow between major components in the system.
- The domain analyst performs an implicit functional decomposition of the system.

4.2. Reference Requirements

- There are composed of:
    - ✓ Functional requirements,
    - ✓ Non-functional requirements,
    - ✓ Design requirements, and
    - ✓ Implementation requirements.

- The functional requirements are the essential operations/processes, their numbers, etc.

- The non-functional requirements represent all other functionality of the domain, i.e., security, extendibility, etc.

- The design requirements focus upon the architectural style and the user interface style:
    - ✓ The architectural style will affect the performance, development cost, and the interfacing style of the corresponding components of the system.
    - ✓ The user interface style can be divided into three broad categories: command-line, menu driven, pull-down menu driven user, and web-based interfaces.

- The implementation requirements deal with the determination of the programming language, operating system, hardware platform, and networking capabilities when applicable.

4.3. The reference architecture

- Inheritance hierarchy is a form of reference architecture dependency diagrams.
- It consists of the followings:
    - ✓ Reference architecture models,
    - ✓ Configuration decision diagram,
    - ✓ Architecture schema/design record,
    - ✓ Dependency diagram,
    - ✓ Component interface descriptions, and
    - ✓ Constraints and rationale.
- The **reference architecture** models are simple abstractions based upon an existing architecture style.
- The **configuration decision diagram** includes, but is not limited to, decision trees and reference requirements.
- The purpose of an **architecture schema or design record** is to provide an understanding about components, and more specifically to cover knowledge about design alternatives and alternate implementations.
- The **component interface descriptions** section of the DSSA describes the interfaces to components in the reference architecture.
- **Constraints and rationale** are considered the expert system rules or can be considered an informal text that is included as a supplemental part of the design record or architecture schema:

    - ✓ Constraints are the ranges of parameter values, relationships, and component attributes.
    - ✓ The rationale is the retrospective "lessons learned" based upon using the reference architecture in development of applications. Constraints are the system guidelines.

5. Case Study [Gary A. Curry et al.]: A Graphical Editor domain.

- The graphical editor provides the user an interface to manipulate shapes, lines, and text in order to graphically represent objects, data, and relationships between them.
- The graphical editor domain includes the ability to create Data Flow Diagrams (DFD), structure charts, Entity/Relationship (ER) diagrams, state transition diagrams, as well as, numerous other types of design and architecture graphical modeling documents.
- The user needs to be able to show relationships and behavior by drawing multiple line types and arrows to connect these objects, and the editor must be able to support the insertion of text onto objects and corresponding lines to describe their functionality.
- The user should be able to create, delete, and move the lines in the document, and the ability to save or open a file is vital.
- Also, it is necessary to provide the option to output to a printer.
- In a more advanced version of a graphical editor, the support of color coding objects and lines should be available; further, the ability to create custom objects and relationships to better describe the user's information modeling.
- Some versions may also support the option of animation to show flow, and thus, improve the understanding of the objects.
- The editor should be able to handle image types of various formats such as PICT, RTF, SYLK, MIF, JPEG, GIF, BMP, TIF, and ICN to allow distinct products to interact by data interchange.

5.1. Reference Requirements for the Domain

- The goal is to specify the scope of the domain. It contains a list and description of functional requirements for the domain. The following is a refined preliminary report.
- There are two types of requirements:
  - Stable functional requirements
  - Variable functional requirements

5.1.1. Stable Functional Requirements:

- Stable functional requirements are the essential operations/processes of the domain.
- They are required for the entire domain and its subsequent applications.
- Stable functional requirements do not change from application to application.
- These act as the core requirements for the domain.
- The requirements are:

  - ✓ *file subsystem* - handles all user requests of file menu options
  - ✓ *file open* - opens an existing file
  - ✓ *file close* - closes the currently opened file
  - ✓ *file new* - creates a new file; templates
  - ✓ *file save* - saves the specified file
  - ✓ *file save as* - saves the specified file in a selected format
  - ✓ *file print* - spools the active file to a printer device
  - ✓ *file exit* - exit the program
  - ✓ *edit subsystem* - handles all user requests of edit menu options
  - ✓ *edit undo* - undo the last action; possibly several levels deep
  - ✓ *edit cut* - cuts the selected object(s) to the clipboard
  - ✓ *edit copy* - copies the selected object(s) to the clipboard
  - ✓ *edit paste* - pastes the selected object(s) from the clipboard
  - ✓ *edit text* - insert, delete, and modify text
  - ✓ *edit image subsystem* – handles all user requests of edit image options
  - ✓ *edit image* - insert, delete, and modify image
  - ✓ *edit arrow subsystem* - handles all user requests of edit arrow options
  - ✓ *interface subsystem* – handles user requests to change the current interface
  - ✓ *selector arrow* - select a graphical object on screen; targets object for application action
  - ✓ *ruler* - displays a ruler on the screen xy borders; vertical and horizontal measurements
  - ✓ *inches* - displays the dimensions on ruler in inches
  - ✓ *centimeters* - displays the dimensions on the ruler in centimeter
  - ✓ *background grid* - background of user screen is of grid type such as xy plane; allows for more precise depiction of images
  - ✓ *grid spacing* - provides a method of changing the background grid spacing
  - ✓ *snap to grid* - locks image onto background grid system as a graphical object is dragged across the screen
  - ✓ *grid lines* – indicates the vertical and horizontal background grid with dotted lines
  - ✓ *grid minor dots* – indicates incremental positions between grid lines

5.1.2. Variable Functional Requirements

- Unlike stable requirements, variable functional requirements may change from application to application.

- They represent the additional and supplemental functionality of the system domain. Such functionality adds to the user-friendliness and overall applicability of the system.

- The requirements are:

  - ✓ *square image* - draws a square on grid system; object representation
  - ✓ *circle image* - draws a circle on grid system; object representation
  - ✓ *triangle image* - draws a triangle on grid system; object representation
  - ✓ *line image* - draws a line on the grid system; object representation
  - ✓ *line arrow image* - direct function call; shows image referencing on screen
  - ✓ *dash arrow image* - asynchronous call; dashed arrow image
  - ✓ *parameter arrow image* - smaller and shorter arrow image used to show direction of parameter being passed between graphical objects
  - ✓ *sticky arrow* - arrows connecting graphical objects will reposition themselves in accordance with object location as objects are edited
  - ✓ *filled parameter arrow image* - used to represent data couple
  - ✓ *unfilled parameter arrow image* - used to represent control couple
  - ✓ *various arrow head(s)* - variations on the type of pointers on the arrow
  - ✓ *sharpen* - sharpen the quality of the image
  - ✓ *smooth* - smooth the image
  - ✓ *rotate* - rotate the image; clockwise or counterclockwise
  - ✓ *invert* - invert the image; flip 180 degrees
  - ✓ *RGB controls* - three slider bars for controlling the color scheme
  - ✓ *mirror* - mirror the image
  - ✓ *left-right justify* - justify the image according to margins
  - ✓ *various baseline symbols* - used to show such properties as aggregation and inheritance
  - ✓ *iterative function* - recursive type graphical object(square with arrow returning to itself)
  - ✓ *module* - graphical object with double borders to denote a module, possible complex
  - ✓ *data store* - graphical object used to represent a type of data store

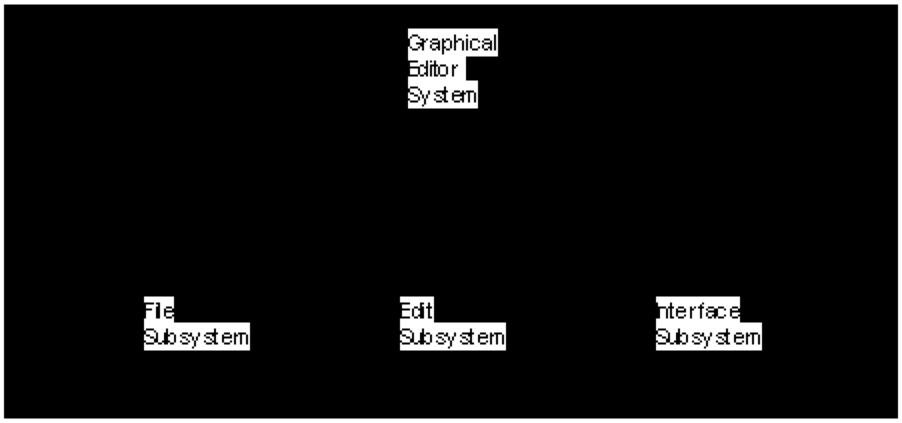
- ✓ *external entity* - graphical representation of an external person, place, or thing

5.2. Reference Architecture

- Reference architecture provides a basic structure of the domain and can be used as a foundation to develop specific application designs within the graphical editor domain.
- It helps a designer to quickly produce a useable design that can then be given to a programming team to produce the application software.

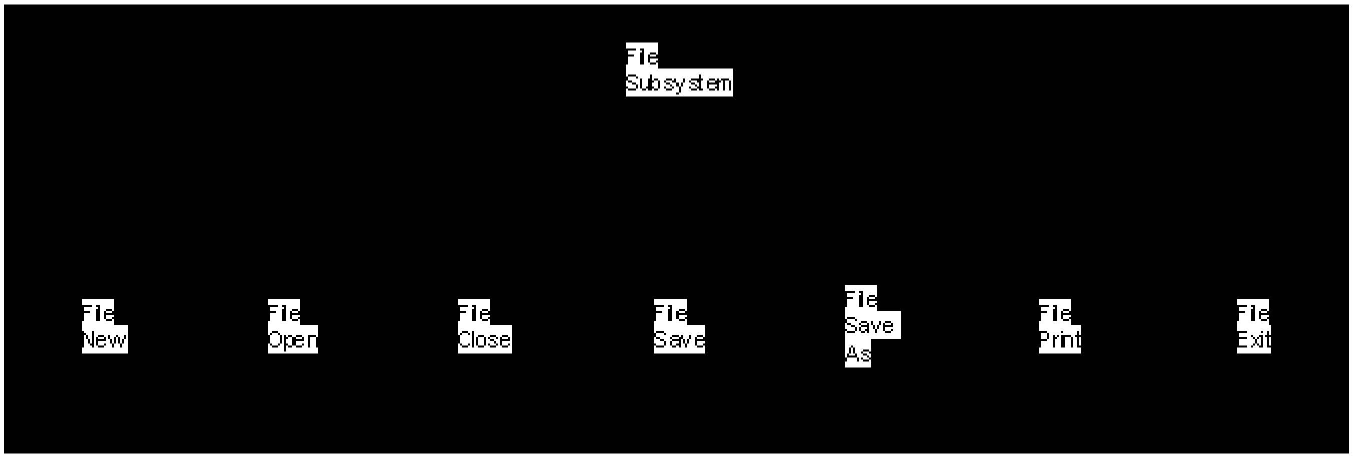
5.2.1. Reference Structure Chart

- It is a reference structure chart for the graphical editor domain.
- It is a "generic" description of the modules, which are available for all possible applications.

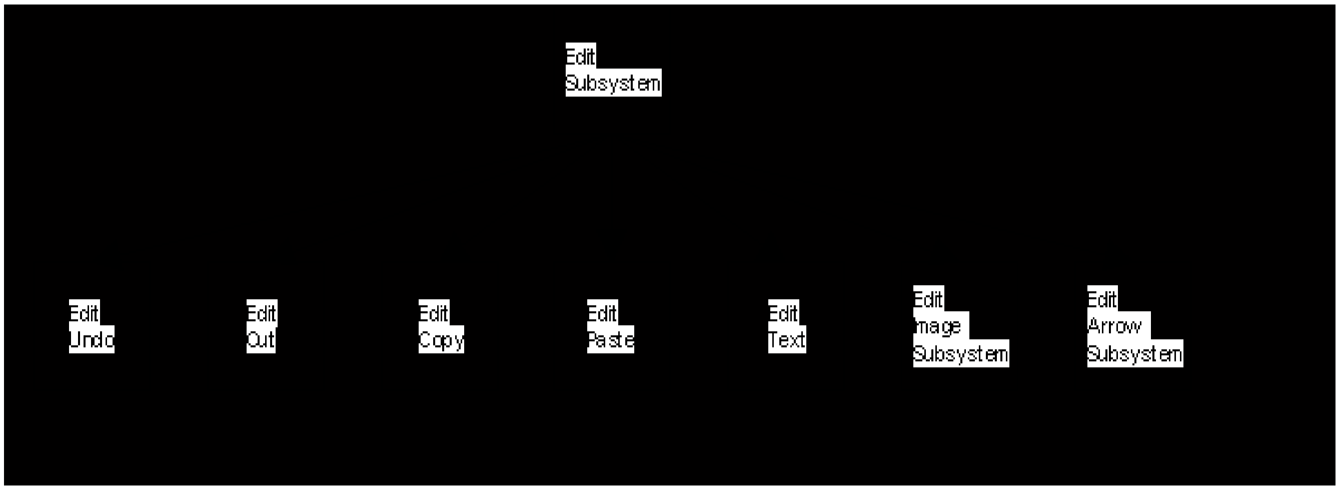


Structure Chart Image

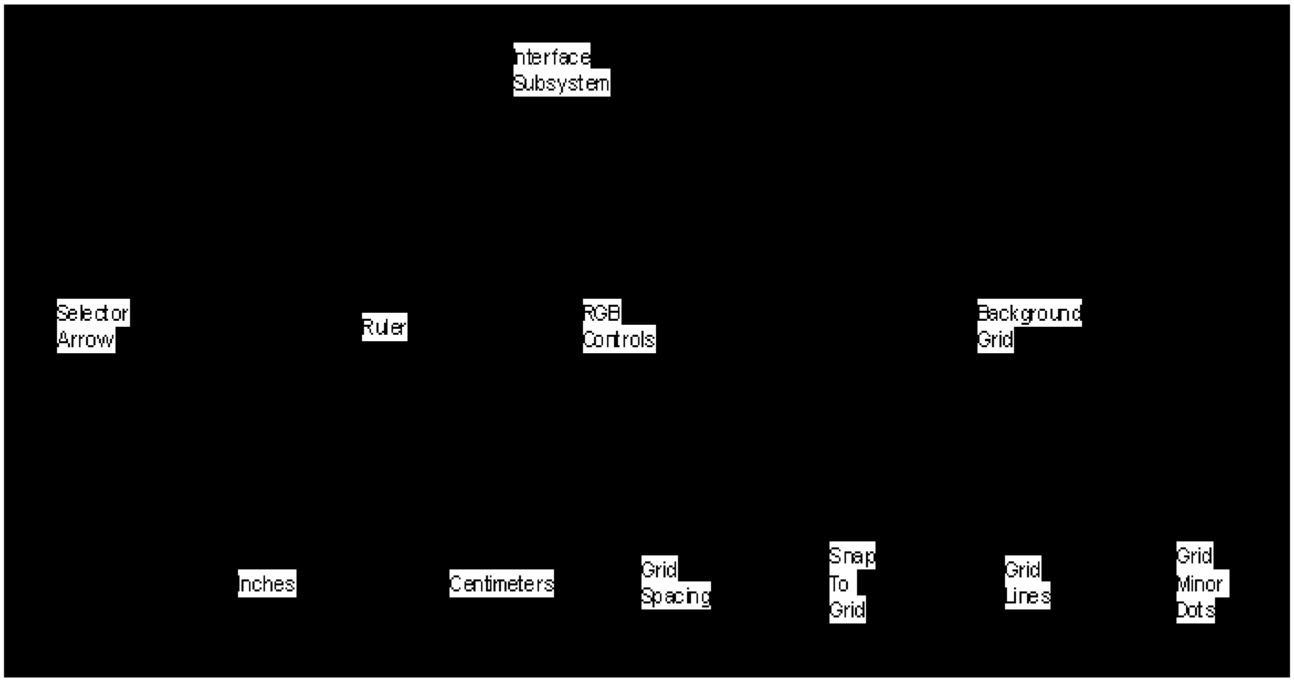
Subchart A:



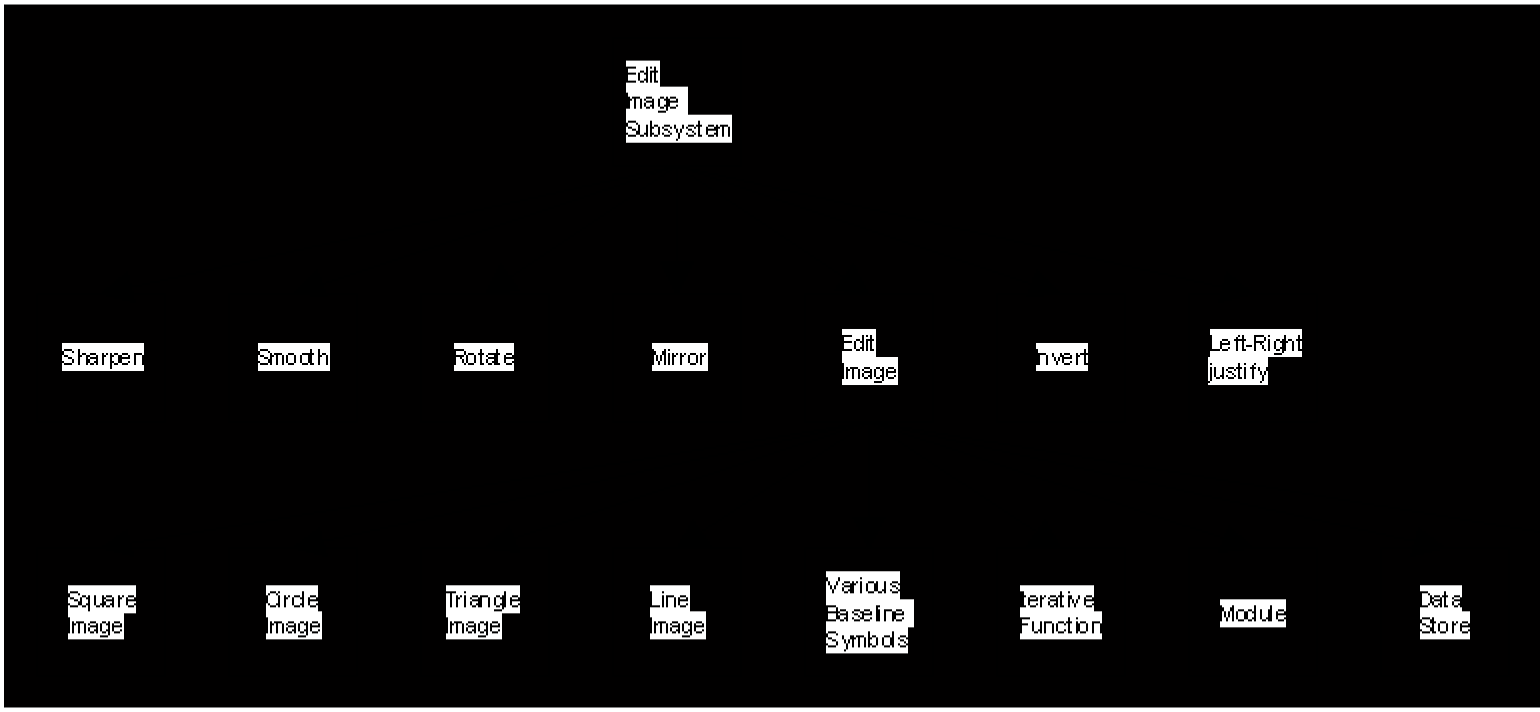
Subchart B:



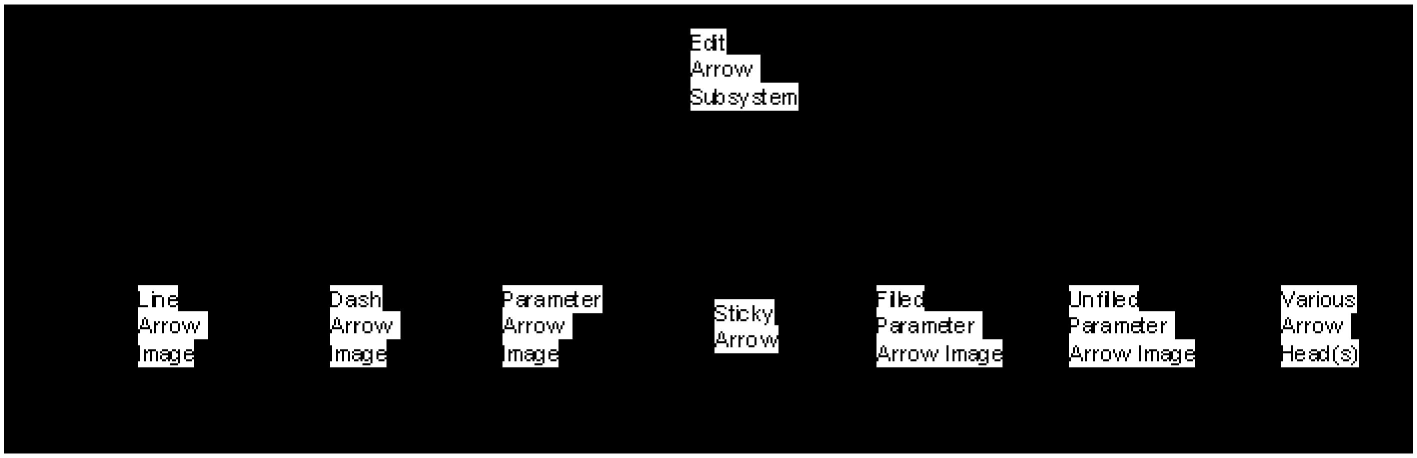
Subchart C:



Subchart D:



Subchart E:


Edit Arrow Subsystem
Line Arrow Image
Dash Arrow Image
Parameter Arrow Image
Sticky Arrow
Filed Parameter Arrow Image
Unfiled Parameter Arrow Image
Various Arrow Head(s)

5.2.2. Stable Modules

- Stable modules will appear in all applications within the graphical editor domain.
- These modules are:

  - ✓ *file subsystem* - controls all user requests of file menu options by calling the appropriate file module to fulfill the request.
  - ✓ *file open* – provides a file open dialog box for the user to make a selection from, and processes the selection to open the requested file.
  - ✓ *file close* – closes the currently opened file if there are no unsaved changes, otherwise prompts the user to save the changes prior to closing.
  - ✓ *file new* - creates and opens a new file, possibly providing options to the user of the types of templates available within the specific application based upon variable requirements.
  - ✓ *file save* - saves the unsaved changes of the currently open file.
  - ✓ *file save as* – provides a dialog box to prompt the user with the name to save the currently open file.
  - ✓ *file print* – prompts the user with the current printer configuration and spools the currently file to a printer device.
  - ✓ *file exit* - checks for any unsaved changes in currently open files, and, if there are any, prompts the user to save the changes before exiting the program.
  - ✓ *edit subsystem* - controls all user requests of edit menu options by calling the appropriate edit module to fulfill the request.
  - ✓ *edit undo* - undoes the last action(s) within the undo buffer.
  - ✓ *edit cut* - cuts the selected object(s) to the clipboard buffer.
  - ✓ *edit copy* - copies the selected object(s) to the clipboard buffer.
  - ✓ *edit paste* - pastes the selected object(s) from the clipboard buffer.
  - ✓ *edit text* – provides a text cursor within the selected text object, allowing the user to insert, delete, modify text.
  - ✓ *edit image subsystem* – controls all user requests of edit image options by calling the appropriate edit image module to fulfill the request.
  - ✓ *edit image* - provides "handles" on the selected image object, allowing the user to insert, delete, modify the image
  - ✓ *edit arrow subsystem* - controls all user requests of edit arrow options by calling the appropriate edit arrow module to fulfill the request.
  - ✓ *interface subsystem* – controls all user requests to change the current interface by calling the appropriate interface module to fulfill the request.
  - ✓ *selector arrow* - provides a method for graphical object on screen allowing the user to select objects for application action.
  - ✓ *ruler* - displays a ruler on the screen xy borders, providing a visual interface of vertical and horizontal measurements.
  - ✓ *inches* – a method to display the dimensions on ruler in inches.

- ✓ *centimeters* – a method to display the dimensions on the ruler in centimeter.
- ✓ *background grid* - provides a background of grid type such as xy plane of user screen is, allowing for precision placement of objects.
- ✓ *grid spacing* - provides a method of changing the background grid spacing
- ✓ *snap to grid* – a method which locks the image onto background grid system as a graphical object is repositioned on the screen, providing the freeing the user from having to pay close attention to the vertically or horizontal alignment of objects.
- ✓ *grid lines* – draws the current background grid with dotted lines at each horizontal and vertical gridline.
- ✓ *grid minor dots* – draws vertical and horizontal lines of dots on the screen to indicate incremental positions between grid lines.

5.2.3. Variable Modules

- Variable modules may appear in some applications within the graphical editor domain.

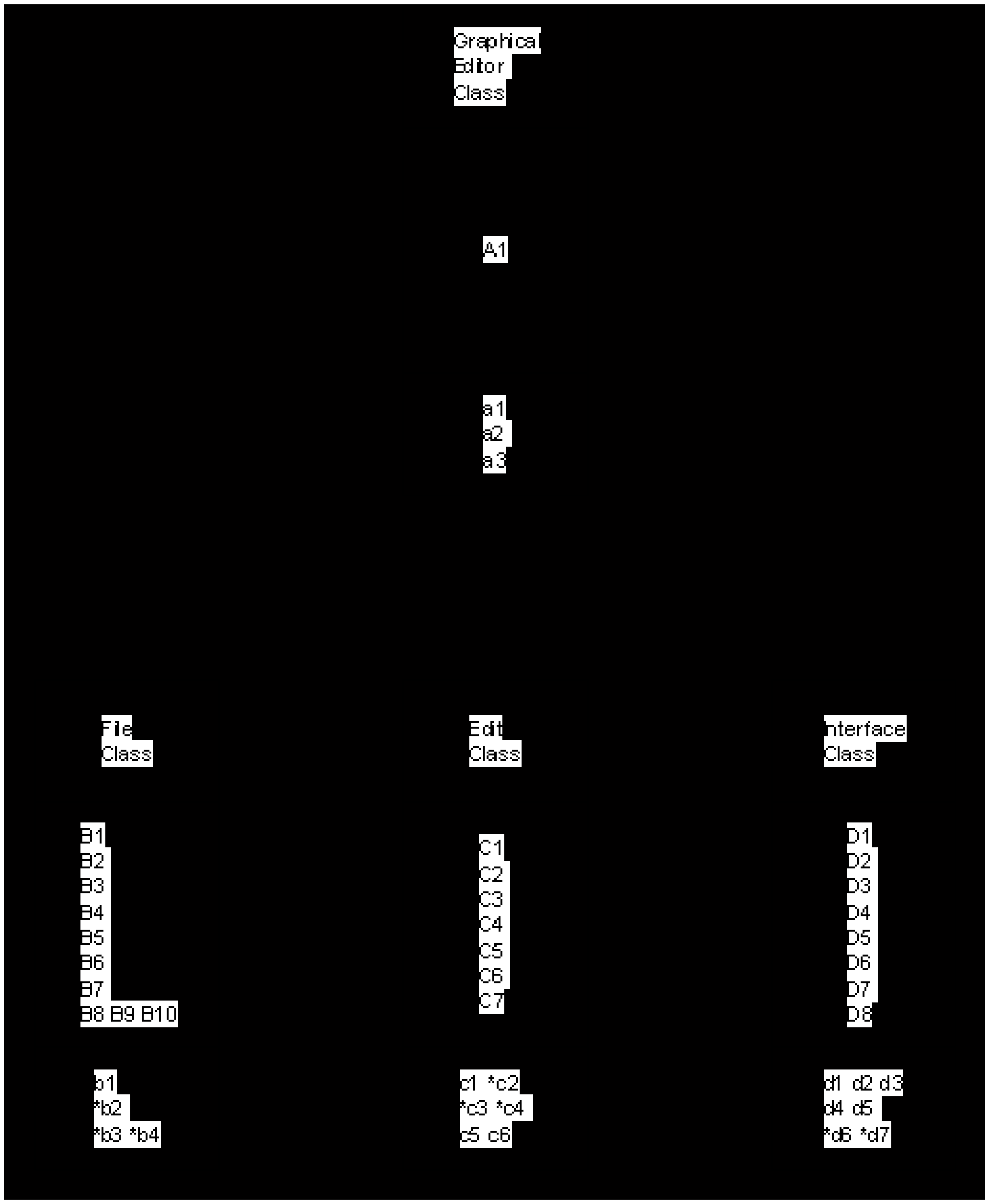
- These modules are:

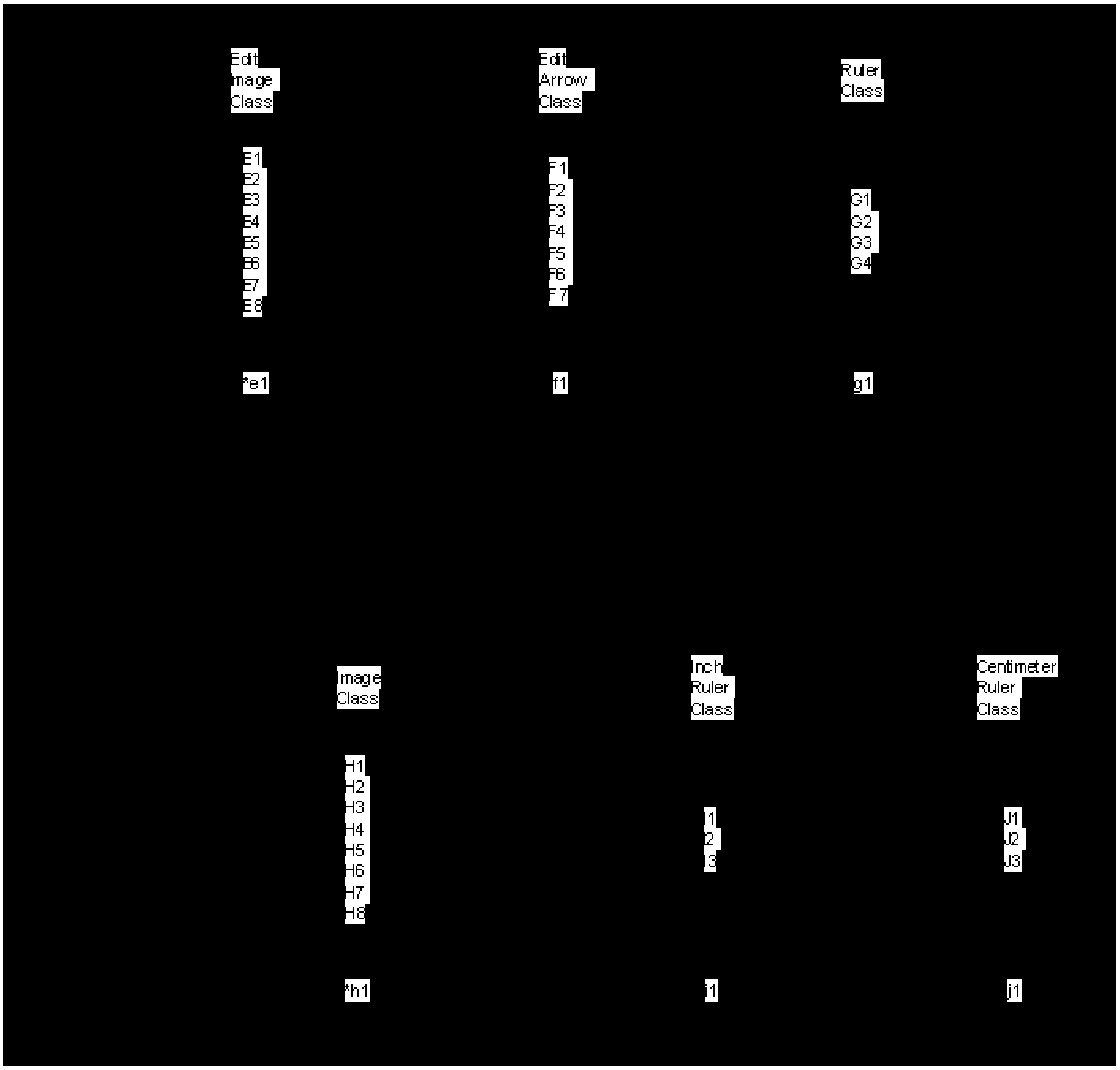
  - ✓ *square image* - draws a square object on grid system with the size and position specified by the user.
  - ✓ *circle image* - draws a circle object on grid system with the size and position specified by the user.
  - ✓ *triangle image* - draws a triangle object on grid system with the size and position specified by the user.
  - ✓ *line image* - draws a line object on the grid system with the size and position specified by the user.
  - ✓ *line arrow image* - draws a line object with an arrowhead on the grid system with the size and position specified by the user; indicates direct function call between modules of a structure chart.
  - ✓ *dash arrow image* - draws a dashed line object with an arrowhead on the grid system with the size and position specified by the user; indicates an asynchronous function call between modules of a structure chart.
  - ✓ *parameter arrow image* - draws an arrow image to show direction of parameter being passed between modules in a structure chart.
  - ✓ *sticky arrow* - draws arrows connecting graphical objects and will "stick" with the objects as the repositioned.

- ✓ *filled parameter arrow image* - draws a special parameter arrow, which is filled in order to represent data couple between modules in a structure chart.
- ✓ *unfilled parameter arrow image* - draws a special parameter arrow, which is unfilled in order to represent control couple between modules in a structure chart.
- ✓ *various arrow head(s)* - draws variations on the type of pointers on the arrow.
- ✓ *sharpen* - draws a sharper image.
- ✓ *smooth* - draws a smoothing of the image using a splining algorithm.
- ✓ *rotate* - provides handles and the selected image for the user to rotate the image; clockwise or counterclockwise.
- ✓ *invert* - redraws the image 180 degrees with respect to the current position.
- ✓ *RGB controls* - provides three slider bars for the user to control the color scheme of the selected object.
- ✓ *mirror* - draws a mirror image of the selected object.
- ✓ *left-right justify* - redraws the selected objects, justifying them respect to left and right margins.
- ✓ *various baseline symbols* - provides drawing objects to show such properties as aggregation and inheritance in a class model diagram.
- ✓ *iterative function* - provides a recursive type graphical drawing object (square with arrow returning to itself) in a class model diagram.
- ✓ *module* - draws a rectangular object with double borders to denote a module in a structure chart.
- ✓ *data store* - draws an object to represent a type of data store in a data-flow diagram.
- ✓ *external entity* - draws a representation of an external person, place, or thing in a data-flow diagram.

5.2.4. Reference Class Model

- A "generic" class model for the graphical editor domain that can be used for all possible applications.


Graphical Editor Class
A1
a1
a2
a3
File Class
Edit Class
Interface Class
B1
B2
B3
B4
B5
B6
B7
B8 B9 B10
C1
C2
C3
C4
C5
C6
C7
D1
D2
D3
D4
D5
D6
D7
D8
b1
*b2
*b3 *b4
c1 *c2
*c3 *c4
c5 c6
d1 d2 d3
d4 d5
*d6 *d7



⇴ Graphical Editor class

- ✓ A1 - Graphical_Editor; Graphical Editor object constructor
- ✓ a1 - an object of the File class
- ✓ a2 - an object of the Edit class
- ✓ a3 - an object of the Interface class

⇴ File class

- ✓ B1 - File; File object constructor
- ✓ B2 - File_Finished; a function to return the file_finished_flag of FALSE to the Graphical_Editor object until the program is exited, then returns TRUE
- ✓ B3 - File_New; a function to open up a new file and provide the user with the available graphical editor option, i.e. structure chart, data-flow diagram, state-transition diagram …
- ✓ B4 - File_Open; a function to open an existing file; provides a file dialog box to prompts the user to make a selection
- ✓ B5 - File_Close; a function to close the active file; if there are any unsaved changes, provides a dialog box to prompt the user to save the changes before closing
- ✓ B6 - File_Save; a function to save the active file
- ✓ B7 - File_Save_As; a function to save the active file with a different name; provides a file dialog box to prompts the user to type the new file name
- ✓ B8 - File_Print; a function to print the active file; provides a print dialog box to prompts the user to review the current printer and provides a button to change the current printer
- ✓ B9 - File_Current_Printer; a function to change the current printer; provides a dialog box to prompt the user to change the current printer; called with a button press from the print dialog box
- ✓ B10 - File_Exit; a function to change the file_finished_flag from FALSE to TRUE for return to the Graphical_Editor object; if there are any unsaved changes, provides a dialog box to prompt the user to save the changes before changing the finished_flag
- ✓ b1 – file_finished_flag; a TRUE/FALSE value return to the Graphical Editor object when exit is performed
- ✓ *b2 - file name; dynamically allocated name and path for the users file name to save with
- ✓ *b3 - printer location(s); dynamically allocated storage of locations currently available to print from, either locally or on a network
- ✓ *b4 – active printer; dynamically allocated storage of the active printer location

⇘ Edit class

- ✓ C1 - Edit; Edit object constructor
- ✓ C2 - Edit_Finished; a function to return the edit_finished_flag of FALSE to the Graphical_Editor object while the current editing activity is underway, then returns TRUE
- ✓ C3 - Edit_Undo; a function to undo the users recent commands; uses the edit_undo_buffer data member
- ✓ C4 - Edit_Cut; a function to cut the selected text and/or object(s) to the edit_clipboard_buffer

- ✓ C5 - Edit_Copy; a function to copy the selected text and/or object(s) to the edit_clipboard_buffer
- ✓ C6 - Edit_Paste; a function to paste the edit_clipboard_buffer
- ✓ C7 - Edit_Text; a function to provide editing of the selected text
- ✓ c1 - edit_finished_flag; a TRUE/FALSE value return to the Graphical Editor object for all edit activities
- ✓ *c2 - edit_undo_buffer; a dynamically allocated buffer to provide a history of the users recent commands
- ✓ *c3 – edit_clipboard_buffer; a dynamically allocated buffer to hold cut and copied text and image objects
- ✓ *c4 - edit_text; a dynamically allocated buffer to hold the current text being edited
- ✓ c5 - an object of the Edit Image class
- ✓ c6 - an object of the Edit Arrow class

⇘ Interface class

- ✓ D1 - Interface; Interface object constructor
- ✓ D2 - Selector_Arrow; a function to display a selector arrow which provides a method for the user to select image and text objects
- ✓ D3 - RGB_Controls; a function to dynamically allocate and display three sliders for control of red, green, and blue colors of the display
- ✓ D4 - Background_Grid; a function to display the background grid
- ✓ D5 - Grid_Spacing; a function to change the spacing of the current background grid; provides a dialog box to prompt the user to adjust the grid
- ✓ D6 - Snap_To_Grid; a function to toggle the snap_to_grid flag and display or remove the objects in alignment with the grid based upon whether the flag is TRUE or FALSE
- ✓ D7 - Grid_Lines; a function to toggle the grid_line_flag and display or remove the grid lines based upon whether the flag is TRUE or FALSE
- ✓ D8 - Grid_Minor_Dots; a function to toggle the grid_minor_dots_flag and display or remove the grid minor dots based upon whether the flag is TRUE or FALSE
- ✓ d1 - selector_arrow; a structure containing the current status
- ✓ d2 - border_ruler; a structure containing the current status
- ✓ d3 - snap_to_grid_flag; a flag to indicate whether grid snap is selected; TRUE or FALSE
- ✓ d4 - grid_line_flag; a flag to indicate whether grid lines are selected; TRUE or FALSE

- ✓ d5 - grid_minor_dots_flag; a flag to indicate whether grid minor dots is selected; TRUE or FALSE
- ✓ *d6 - RGB_sliders; dynamically allocated sliders for control of the red, green, and blue colors of the display
- ✓ *d7 - background_grid; a dynamically allocated structure containing the current status

⇘ Edit Image class

- ✓ E1 - Edit_Image; Edit Image object constructor
- ✓ E2 - Sharpen; a function to sharpen the display
- ✓ E3 - Smooth; a function to smooth the redisplay of the currently selected object
- ✓ E4 - Rotate; a function to rotate and redisplay the currently selected object 90 degrees to the left
- ✓ E5 - Mirror; a function to mirror and redisplay the currently selected object
- ✓ E6 - Invert; a function to invert and redisplay the currently selected 180 degrees from its position
- ✓ E7 - Left_Right_Justify; a function to justify and redisplay the currently selected objects with respect to the left and right hand margins
- ✓ *e1 – current_objects; a dynamically allocated linked-list containing all current objects

⇘ Edit Arrow class

- ✓ F1 - Edit_Arrow; Edit Arrow object constructor
- ✓ F2 - Line_Arrow_Image; a function to add and display a line arrow with a single head
- ✓ F3 - Dashed_Arrow_Image; a function to add and display a dashed arrow with a single head
- ✓ F4 - Parameter_Arrow_Image; a function to add and display a parameter arrow with a single head
- ✓ F5 - Sticky_Arrow; a function to add and display a sticky arrow with a single head
- ✓ F6 - Filled_Parameter_Arrow_Image; a function to add and display a filled parameter arrow with a single head
- ✓ F7 - Unfilled_Parameter_Arrow_Image; a function to add and display an unfilled parameter arrow with a single head
- ✓ F8 - Arrow_Heads; a function that provides the user with a dialog box to make a selection of various arrow heads

- ✓ *f1 - current_arrows; a dynamically allocated linked-list containing all arrows

➯ Ruler class

- ✓ G1 - Ruler; Ruler object constructor
- ✓ G2 - Change_Increments;a function to change the displayed increments of the rulers based upon the current_zoom
- ✓ g2 - current_zoom; storage of the current zoom of the display

➯ Image class

- ✓ H1 - Image; Image object constructor
- ✓ H2 - Square_Image; a function to create and display a square image object
- ✓ H3 - Circle_Image; a function to create and display a circle image object
- ✓ H4 - Triangle_Image; a function to create and display a triangle image object
- ✓ H5 - Line_Image; a function to create and display a line image object
- ✓ H6 - Base_Line_Image; a function to display the various base line image objects
- ✓ H7 - Iterative_Image; a function to create and display a iterative image object
- ✓ H8 - Module_Image; a function to create and display a module image object
- ✓ *h1 - image; a dynamically allocated structure containing the type, position, and size of image object

➯ Inch Ruler class

- I1 - Inch_Ruler; Inch Ruler object constructor
- I2 - Display; a function to display the rulers in inches based upon the current_zoom inherited from the Ruler class object
- I3 - Display_Cursor; a function to display the movement of the cursor with dotted lines
- i1- cursor_position; a projection of the current position of the cursor on the xy axis of the inch ruler

➯ Centimeter Ruler class

- ✓ J1 - Centimeter_Ruler; Centimeter Ruler object constructor
- ✓ J2 - Display; a function to display the rulers in inches based upon the current_zoom inherited from the Ruler class object
- ✓ J3 - Display_Cursor; a function to display the movement of the cursor with dotted lines
- ✓ j1- cursor_position; a projection of the current position of the cursor on the xy axis of the centimeter ruler

➯ Stable Classes

- This identifies the stable classes and modules that will appear in all applications within the graphical editor domain.

➯ Stable Classes and Modules

- ✓ Graphical Editor
- ✓ Graphical_Editor

- ✓ File
- ✓ File_Finished
- ✓ File_New
- ✓ File_Open
- ✓ File_Close
- ✓ File_Save
- ✓ File_Save_As
- ✓ File_Print
- ✓ File_Current_Printer
- ✓ File_Exit

- ✓ Edit
- ✓ Edit
- ✓ Edit_Finished
- ✓ Edit_Undo
- ✓ Edit_Cut
- ✓ Edit_Copy
- ✓ Edit_Paste
- ✓ Edit_Text

- ✓ Interface
- ✓ Interface
- ✓ Selector_Arrow
- ✓ Background_Grid
- ✓ Grid_Spacing
- ✓ Snap_To_Grid
- ✓ Grid_Lines
- ✓ Grid_Minor_Dots

- ✓ Edit Image
- ✓ Edit_Image

- ✓ Image
- ✓ Image

- ✓ Edit Arrow
- ✓ Edit_Arrow

- ✓ Ruler
- ✓ Ruler
- ✓ Change_Increments

- ✓ Inch Ruler
- ✓ Inch_Ruler
- ✓ Display

- ✓ Centimeter Ruler
- ✓ Centimeter_Ruler
- ✓ Display

⇘ Variable Classes

- These are all the variable classes that will appear in all applications within the graphical editor domain.

⇘ Variable Modules of Stable Classes; Variable Classes and Modules

- ✓ Interface(stable class)
- ✓ RGB_Controls

- ✓ Edit Image(stable class)
- ✓ Sharpen
- ✓ Smooth
- ✓ Rotate
- ✓ Mirror
- ✓ Invert
- ✓ Left_Right_Justify

- ✓ Image(stable class)
- ✓ Square_Image

- ✓ Circle_Image
- ✓ Triangle_Image
- ✓ Line_Image
- ✓ Base_Line_Image
- ✓ Iterative_Image
- ✓ Module_Image

- ✓ Edit Arrow(stable class)
- ✓ Line_Arrow_Image
- ✓ Dashed_Arrow_Image
- ✓ Parameter_Arrow_Image
- ✓ Sticky_Arrow
- ✓ Filled_Parameter_Arrow_Image
- ✓ Unfilled_Parameter_Arrow_Image
- ✓ Arrow_Heads

⇘ Configuration Decision Diagram

- The configuration decision diagram will be used to configure a specific application.
- Configuring a system (application) is the process of selecting a subset of the reference requirements.
- The application in this case will be named System X, and it mainly handles class models.
- Since it is in the Graphical Editor Domain, it has all the stable functional requirements and it will have a select subset of the variable requirements.

System X – Variable Functional Requirements of the application

Vx	Variable Functional Requirements
V1	*square image* - draws a square on grid system; object representation
V2	*line image* - draws a line on the grid system; object representation
V3	*line arrow image* - direct function call; shows image referencing on screen
V4	*dash arrow image* - asynchronous call; dashed arrow image
V5	*sticky arrow* - arrows connecting graphical objects will reposition themselves in accordance with object location as objects are edited
V6	*various arrow head(s)* - variations on the type of pointers on the arrow
V7	*left-right justify* - justify the image according to margins
V8	*various baseline symbols* - used to show such properties as aggregation and inheritance
V9	*data store* - graphical object used to represent a type of data store
V10	*external entity* - graphical representation of an external person, place, or thing
V11	*iterative function* - recursive type graphical object(square with arrow returning to itself)

6. Summary:

- The requirement analysis and the software architecture of a domain of applications.
- Examples: Wed browsers, Web servers, Word processors, etc.
- There are three main elements of a DSSA:
    - Domain Model
        - Complete description of the domain
        - Achieved by experts in the domain, users, developers who have experience in the domain, etc.
    - Reference Requirements
        - Stable (or Fix)
        - Variable (or optional)
        - Requirements can also be broken into:
            - Functional
            - Non-functional
            - Design
            - Implementation
        - List a reference of each requirement of the domain.
    - Reference Architecture
        - Make sure to state the right architecture style for the domain.
        - List a reference of each architectural element.