RESEARCH REPORT ISIS-RR-96-11E

Functional Analysis of DUI Systems: Toward PizzaSystem Development

Kazuo Misue Tao Lin*

August, 1996

Institute for Social Information Science (*ISIS*)
at Numazu

FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
Telephone: +81-559-24-7210 Fax: +81-559-24-6180

Functional Analysis of DUI Systems: Toward PizzaSystem Development

Kazuo Misue Tao Lin*

Institute for Social Information Science (*ISIS*)
at Numazu

FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan

Email: misue@iias.flab.fujitsu.co.jp

Abstract

A diagrammatic user interface (DUI) allows users to interact with computers through diagrams. A DUI platform is a flexible base on which we can develop application systems with various DUIs quickly and efficiently. We aim at developing a total system of a DUI platform and component modules on it. A formed idea of the system is called the "PizzaSystem." To realize the idea, a general DUI system should be analyzed form a viewpoint of functions. In this document, intuitive definitions of several terms related to the "PizzaSystem" are given, and then several requirements for functions of DUIs and implementation of DUIs are listed. Last, we give a functional architecture of DUI systems in object-oriented manner.

Key words: Diagrammatic User Interface (DUI), DUI platform, PizzaSystem

*CSIRO, Division of Information Technology, GPO Box 664, Canberra, ACT 2601 Australia

1 Introduction

A *diagrammatic user interface* (DUI) [1] supports users to interact with computers through diagrams (visual representations of structural information). Since the DUI determines the usability of the system, a DUI is one of the most critical part of an interactive system. However, implementation of interactive systems with such the DUIs tends to require complex mechanisms and high cost. To solve these problems, a *DUI platform* [2], which is a flexible base on which we can develop various DUIs quickly and efficiently, is required.

We aim at developing a complete DUI platform. Such a platform is called "PizzaSystem" [3] and has several features; the platform is a slim kernel for easy maintenance, component modules can be chosen like pizza toppings, and this platform allows potential users to have the tastes of the application system with different DUIs by using the different combinations of the developed modules in this platform.

To support high flexibility and extensibility, the architecture of this platform should well decompose the functions in the sense of that we can easily change the implementation of a module without disturb other modules. Therefore, an architecture which well specifies the interfaces between the possible functional modules is required for the development of this platform. Due to the features supported by object-oriented methodology, such as encapsulation and polymorphism, we use this methodology for this project.

In this document we analyze a general DUI system to decompose its functions; decomposed functions are represented as networks of objects. The platform can be decomposed into a groups of object components: "Manager," "Data Handler," "Raster Handler," "Graphical User Interface," "Layout Generator," and "Application Filter." These group of objects are further decomposed in so detail that they are available to design the architecture of the "PizzaSystem."

In the following part of this document, Section 2 gives intuitive definitions of several terms related to the idea of a "PizzaSystem." Section 3 lists the requirements from the point of view of the functions and implementation of DUIs. Section 4 presents a functional analysis of DUI systems. Decomposed functions are represented as diagrams of object-networks. Section 5 gives concluding remarks and a plan of developing the "PizzaSystem."

2 Terminology

This section gives the terminology related to the "PizzaSystem." Section 2.1 gives explanation of the concept of "PizzaSystem" and its constituents. Section 2.2 gives classification of people concerned with the "PizzaSystem."

2.1 PizzaSystem

PizzaSystem is a comprehensive concept including a DUI platform, its components, and technologies on them. The concept of PizzaSystem is outlined by "PlainPizza," "Toppings," and "MixedPizza."

PlanPizza: a DUI platform, that is, a flexible base on which we can quickly and efficiently develop various DUIs.

Toppings: function modules, which are *exchangeable* according to user's preference. The users may develop their original toppings. Toppings are classified by functions. For example, a Topping class for generating layout is called "layout generator." Such a class is called a *functional Topping*. The implementation of a functional Topping is called an *implementation Topping* , for instance, Sugiyama algorithm [4] is one of the implementation Toppings for the functional Topping: layout generator.

MixedPizza: an interactive system with a DUI. A MixedPizza consist of a DUI part and applications. We should not call an "application system" to avoid confusing it with an application (described below). To strictly specify only a DUI part of a MixedPizza, a term "pure MixedPizza" should be used. In the following part of this document, however, the term "MixedPizza" is used to mention a DUI part of an interactive system (see Figure 1).

Applications: application programs or application systems without DUIs. For example, a database management system can be an application. A PlainPizza combined with a Topping that is an application interface to the database system is a MixedPizza and can provide a DUI of the database system.

2.2 Users and Developers

It is possible to regard people except platform developers as "users." Here, however, we classify such the users into three levels: "MixedPizza users," "MixedPizza developers," and "Topping developers." This classification is important to consider measures to cover various user levels.

1. **MixedPizza Users:** use a MixedPizza and may customize the MixedPizza. They are often called "end users."

2. **MixedPizza Developers:** develop a MixedPizza using the PlainPizza and Toppings. They choose the Toppings ready developed and do not need to develop new Toppings. They know the architecture of the PizzaSystem to make a MixedPizza.

3. **Topping Developers:** develop new (implementation) Toppings.

4. **PlainPizza Developers:** develop the PlainPizza.

These four classes can be considered to represent a hierarchy in terms of the degree of providing skills in the PizzaSystem. MixedPizza users should not be required high level of skills, while PlainPizza users must be required highest level of skills.

3 Requirements

The roles played in a PizzaSystem can be divided into four categories: MixedPizza users, MixedPizza developers, Topping developers, and PlainPizza developers. Here, we investigate the requirements from the view point of these four roles respectively.

3.1 Requirements of MixedPizza users

The functions of the MixedPizzas mainly decides the requirements of MixedPizza users. A MixedPizza has two major roles: one is as a user interface of applications, and the other is as a provider of diagram handling facilities. Requirements for functions can be also classified according to these two roles.

3.1.1 As a User Interface of an Application

A DUI system must or should provide the facilities as a user interface to an application.

Communication with an Application Module: A user interface must be able to communicate with an application modules. Communication should be used to control and to retrieve the information from application modules.

Providing Graphical User Interface: Menus and dialog boxes should be available. It is also desirable that MixedPizza users should be allowed to customize menus and menu items according to their preferences.

Communication with Other MixedPizzas: Communication with other MixedPizzas enables the MixedPizza users to share information with other users of other computers. The MixedPizza should be a groupware.

Undo for Every Possible Operations: Every possible operations could be canceled by the "Undo" command. Diagrams used by DUIs are sometimes very delicate and difficult to restore by ordinal operations. The DUI system should take and keep every snapshot of diagrams and should provide for user's cancellation requests.

Recording Working History: A history of working with the MixedPizza should be recorded. Recorded history should be machine-readable to be able to replay. The history should also be human-readable or be able to convert into human-readable format. Analysis of such the working history is an important research topic for certain applications and is useful to improve the MixedPizza.

Revision Control of Diagram Data: Multiple revisions of diagram data should be managed. Tasks with some kind of applications need trial and error on editing diagram. Revision control is useful for the users to perform such tasks.

Customizable: A DUI should be customizable to user's custom, habit, preference, environment and so on.

3.1.2 As a Provider of Diagram Handling Facilities

The most important feature of a DUI system is to provide diagram handling facilities such as the followings.

Powerful Visual Formalism: Diagrams used by user-interfaces should provide powerful visual formalism adaptive to a certain application. Some applications might have conventional styles of visual formalism. It is desired that such styles shall be supported.

Layout Generation / Adjustment: Facilities to generate layout of diagrams are important because that such the facilities are necessary to visualize logical information provided by applications. Some applications might also have conventional styles of diagram layout. It is desired that such the styles shall be supported. Layout generation generates layout from scratch and does not care for the current layout, while layout adjustment is sensitive to the current layout. Layout adjustment might be more useful in most interactive systems.

Fisheye Views: It is desired that fisheye views [5,6] of diagrams should be available. One of the most important feature of diagrams is to be able to show the total structure of information represented by the diagram at once. However, it is difficult to show the whole of a large diagram in detail. Fisheye views enable to show the whole diagram and local details of viewpoints at the same time.

Multiple Views: It might be required multiple views of the same logical information, that is, information provided by applications. Different views can have different advantages, so combination of two or more different views enable to exploit various advantages at once [7].

Mental-Map Preservation: The mental map of diagrams should be preserved [8]. Layout generation and sometimes layout adjustment can destroy the mental map and tend to decrease efficiency of tasks with diagrams.

Direct Manipulation: Some operations should be allowed to be performed directory by using pointing devices like a mouse. For example, selecting, moving, and resizing are suitable to direct manipulation.

Saving and Restoring Diagrams: Diagram data should be able to saved and restored. Not only topological information of diagrams but also their geometry and appearance might be desired to be saved.

3.2 Requirements of MixedPizza developers

The implementation of MixedPizzas determines the requirements of MixedPizza developers. In another word, the requirements of MixedPizza developers depends on the functions of the PlainPizza. The PlainPizza, that is, a DUI platform should be adaptable and flexible to various applications and various developers. These are essential aspects of platforms.

Flexible Visual Formalism: Visual formalism of diagrams used by user-interfaces should be suitable to a certain application, but suitable diagrams are various depend on applications. It is desired that DUI platforms should deal with general diagrams to be applicable various applications or should provide exchangeable and extensible visual formalism of diagrams.

Extensible Functions: It should be easy to add new functions. For this purpose, the functional Toppings should be well encapsulated and thus the implementation Toppings are exchangeable. The users should be allowed to develop new implementation Toppings.

Non-Programming Development: Development of a MixedPizza should be easy. Users develop a MixedPizza without programming However, the users who like programming should be allowed to write programs.

Automatic Configuration: Some MixedPizza developers do not want to take care to maintain their system. For such people, Toppings should be added or exchanged automatically.

Dynamic Configuration: It should be allowed to modify and exchange function modules in run time.

3.3 Requirements of Topping developers

Functional Independence: Toppings should be developed independently. Topping developers are ought to desire to concentrate on the development of the implementation Topping for certain functional Toppings without the concerns of the interfaces of these Toppings to others.

Language Independence: The programming language to develop original function modules is not limited to a certain language, so the users can employ a suitable and a familiar language to develop new modules in the easiest way.

3.4 Requirements of PlainPizza developers

Software systems should be economical of development and maintenance costs.

Slim Kernel System: The kernel system, that is, the PlainPizza should be so slim that it is easy to develop and to maintain.

Clean Architecture: The architecture of the PlainPizza should be clean. Clean architecture makes system maintenance easy.

Exploiting Existing Resources: There are many programmers and many programs in the world. We should attempt spontaneous distribution of tasks and reusage of programs like in the UNIX world.

Design of the PizzaSystem must take account of these requirements mentioned above. Requirements of MixedPizza users influence what functions should be provided by MixedPizzas. Other requirements, especially requirements of MixedPizza developers influence how to organize provided functions.

In Section 4, we analyze functions of a typical MixedPizza according to the requirements of MixedPizza users. A MixedPizza is decomposed into functional parts to prepare reorganization of them for new architecture of the PlainPizza and Toppings.

4 Functional Analysis

According to the requirements discussed in the previous section, we aim to develop a MixedPizza development environment which can support two development phases:

MixedPizza Development: This environment phase is to support the activities of MixedPizza developers. This phase should provide the mechanisms to guide the MixedPizza developers to easily find the implementation Toppings and to develop a new MixedPizza by integrating the PlainPizza and the implementation Toppings chosen by MixedPizza developers. The interfaces to applications are treated as Toppings as well.

Topping Development: This environment phase is to support the activities of Topping developers. This environment phase should provide the mechanisms to manage the implementation Toppings by Topping developers and the criteria to organize the implementation Toppings.

The development of a MixedPizza development environment is a long term task. The most important task for the development of such an environment is to give a functional architecture which can be applied to a wide range of MixedPizzas. This functional architecture lists the possible functional Toppings of MixedPizzas and specifies the interfaces between the possible Toppings.

Section 4.1 analyses the functionality of a MixedPizza, and divides the objects into six functional groups ("parts"): "Manager", "Raster Handler", "Data Handler", "Application Filter", "Layout Generator", and "Graphical User Interface". From Section 4.2 to 4.7, we will further divide the objects in these functional parts.

4.1 A MixedPizza

A MixedPizza handles a diagrammatic representation which maps the relational information (which can be modeled as "graphs" or "networks") into the graphical forms (which are displayed on the screen). A MixedPizza is a DUI system which represents the information retrieved from application into pictures and supports users to modify the picture by direct manipulations to control the application. Figure 1 illustrates these basic functions of a MixedPizza.

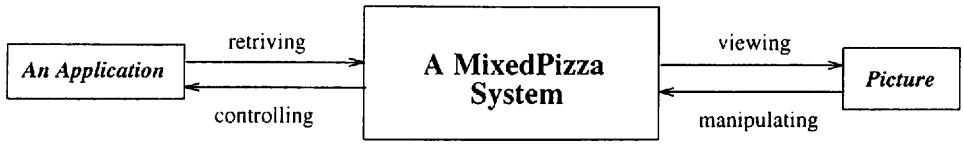

Figure 1: Basic Functions

A MixedPizza can be decomposed into several functional parts illustrated in Figure 2.

- A *Data Handler* maintains the data entities of the data model in a MixedPizza. The data entities contain the following information: logical attributes (information related to the applications), topological attributes (structural relationship), geometric

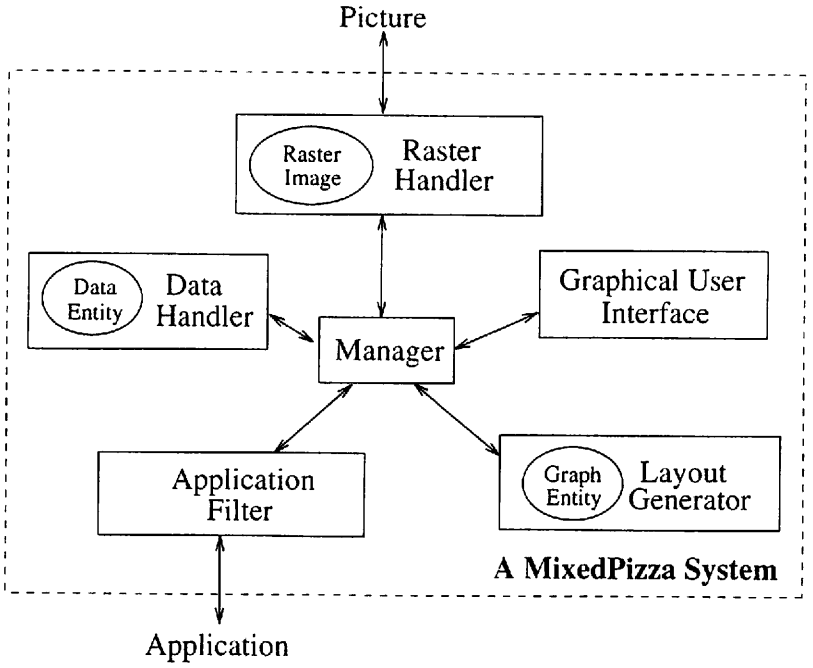

Figure 2: Functional Parts of a MixedPizza

attributes (shape and location) and appearance attributes (color and texture). These entities create a raster image for the entire data entities which are handled by "raster handler".

- A *Raster Handler* handles the raster image and displays it on the screen. Users can deal with the raster image by direct manipulations. The cursor positions, operation types are sent to the data handler to be interpreted. The data handler modifies the data entities and the image handled by the raster handler is changed accordingly.
- The *Graphical User Interface* provides the interfaces to users to invoke operations.
- An *Application Filter* converts the formats between application data and the data entities if these data formats are different.
- A *Layout Generator* creates the layout for the data entities of the data model. The layout generator contains a set of graph entities which provide methods for layout generation. The graph entities require topological relationships for generating layout. The topological relationships between the data entities in data handler and graph entities in the layout generator could be different. For example, the data entities are structured as a directed graph and the graph entities in the layout generator can be structured as a tree.
- The *Manager* coordinates the above functional parts.

It is allowed to have multiple raster handlers, multiple layout generators, and multiple data handlers. However, in a MixedPizza there can only be a single manager part, a single

application filter part and a single graphical user interface. It is also possible that a new part may be required to be added into the system. For example, to allow several MixedPizzas working in a groupware environment, a communicator part should be added into the system. This part is responsible to receive and send messages between this MixedPizza and other MixedPizza. Figure 3 illustrates a MixedPizza with a communicator part. However, in this design, we just focus on a stand alone MixedPizza, but keep the flexibility to add new part.

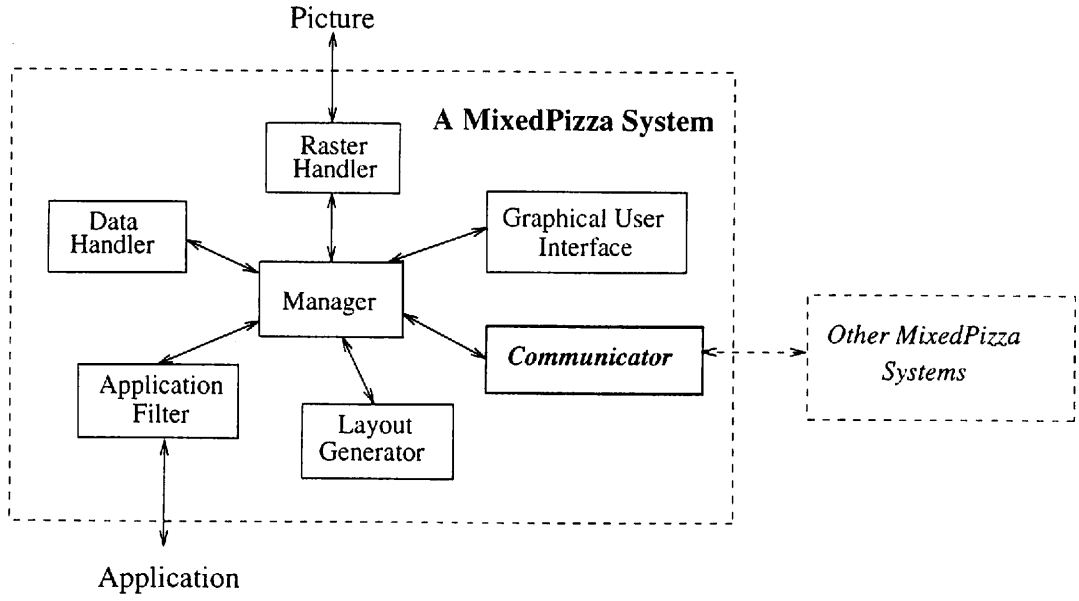

Figure 3: Extra Component in a MixedPizza

Some operations for a MixedPizza are listed below to illustrate the functionality of these parts:

- *Application Input.* To conduct this operation, application filter reads the data from the application and converts the data into the data format of data entities, and then data handler creates the data entities.
- *Application Output.* To conduct this operation, data handler send the data to application filter to convert the data format into the application data format, and then application filter sends the converted data into application.
- *Dressing Assignment.* This operation assigns color and texture values for the appearance attributes of the data entities. This operation is conducted by data handler.
- *Animation.* This operation aims to provide a smooth change after the geometric attributes of data entities been modified. This operation is performed by data handler.
- *Rasterization.* This operation lets part or all of the data entities to generate their raster data to form an image handled by raster handler. This operation is conducted by data handler.

- *Topological Connection Setting.* Data handler conducts this operation which sets up the object links according to the text-index links.

- *Raster Input.* After a user makes some changes by direct manipulations, the raster handler sends the requests of the operations and cursor positions to the data handler through manager. Data handler interprets the operations and change the data entities accordingly. The change is broadcasted by manager to inform the relevant parts to change their contents.

- *GUI Input.* A user can invoke an operation through GUI to change data entities and other object attributes in a MixedPizza, such as "color mapping rules." A user can change data entity attributes by using a dialog box or a combination of GUI input and raster input.

- *Size Determination.* This operation is a preprocessing operation for automatic layout generation. This operation determines all the sizes of nodes and the widths of edges. Layout generator conducts this operation.

- *Layout Generation.* This operation is an automatic layout generation method which decides the values of the geometric attributes for data entities. Obviously, layout generator performs this operation. The layout generation method decides the geometric values of graph entities in the layout generator part first. Then geometric values of the relevant data entities are assigned accordingly.

- *Viewpoint Focusing.* This operation modifies the raster image in order to allow users to emphasize their own focuses. The major technology is fisheye presentation which can enlarge the image of the focused points and reduce the rest of the image. It is allowed to have more than one focus. This operation is performed by raster handler.

- *Display.* This operation displays the raster image on the screen. This operation is performed by raster handler.

4.2 Data Handler

Data handler is a group of object components for handling data entities. The attributes for data entities can be divided into four categories:

- *Topological attributes* give the connection relationship between the entities of the relational information, such as tree, directed graph, undirected graph, compound digraph, and compound mixed graph.

- *Applicational attributes* provide application related information except topological attributes. In a MixedPizza, the application attributes can be used to decide the values of geometric attributes and appearance attributes.

- *Geometric attributes* include size and coordination. Geometric attributes determine the layout of a diagram.

- *Appearance attributes* determine the color, texture and the shape for the graphical entities to be displayed.

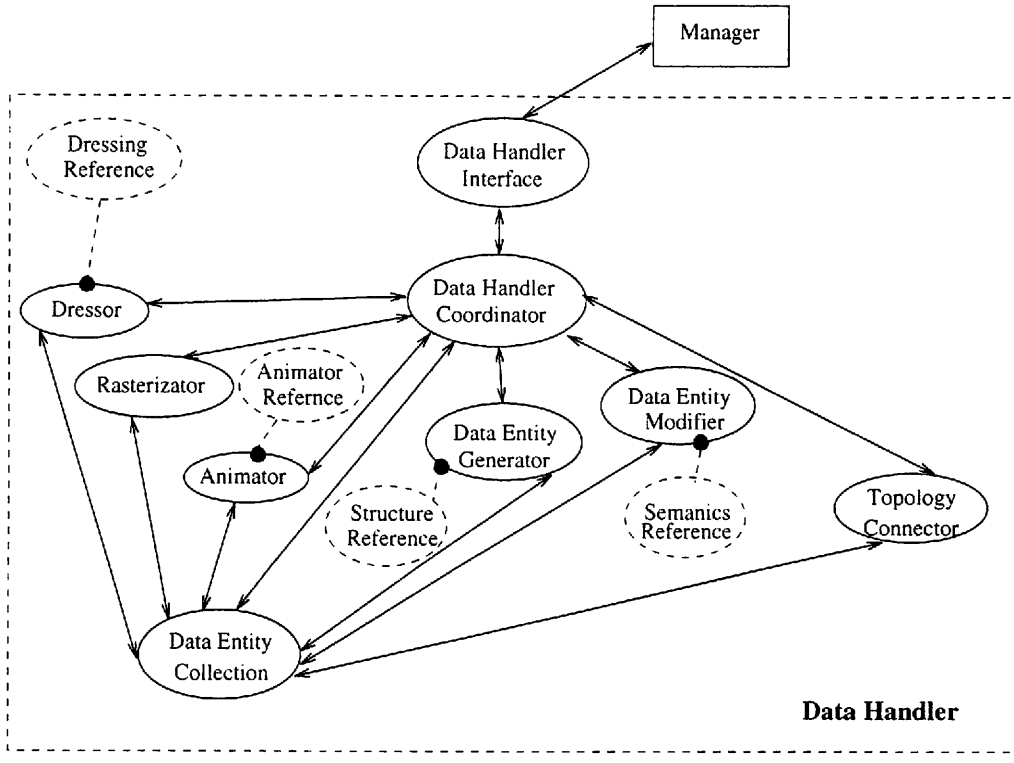

Figure 4: Components of Data Handler

A data handler consists of the following components (illustrated in Figure 4):

- A *Data Handler Interface* provides an interface between manager part and data handler part. This interface passes commands, data and object reference between these two parts.

- The *Data Entity Collection* organizes the data entities and provides the object reference of the data entities to other components in data handler part.

- A *Data Entity Generator* receives the commands for generating data entities from application filter or graphical user interface parts. Data entity generator provides a generic mechanism for creating data entities. The "structure reference" provides the information for generating the specific data entities.

- A *Topology Connector* sets up the topological connection by using object references. When generating data entities from application data, the topological connection are represented by the text identifiers of the data entities.

- A *Dresser* assigns the appearance attributes for data entities. Dresser provides a generic mechanism for the dressing of data entities and "dressing reference" provides the information for the dressing of the specific data entities.

- A *Data Entity Modifier* is responsible to modify the data entities. Data entity modifier checks the consistency of semantics after adding, deleting data entities, or modifying the attribute of data entities. Data entity modifier does not check the consistency of semantics when loading a batch of application data. Data entity modifier only provides a generic mechanism and "semantic reference" provides detail information for a specific MixedPizza system.

- An *Animator* is responsible to create the intermediate frames between the original state and final state of the layout after the geometric attributes of the data entities are modified. According to the operations for modifying the geometric attributes, different approaches can be used to create the frames. The "animator reference" provides the information for choosing the approaches for the geometric modification operations in a specific MixedPizza.

- A *Rasterizator* is responsible to generate the raster image for the data entities to be viewed. Each data entity creates its own raster image.

- A *Data Handler Coordinator* is responsible to coordinate the activities of the above components in data handler part.

4.3 Raster Handler

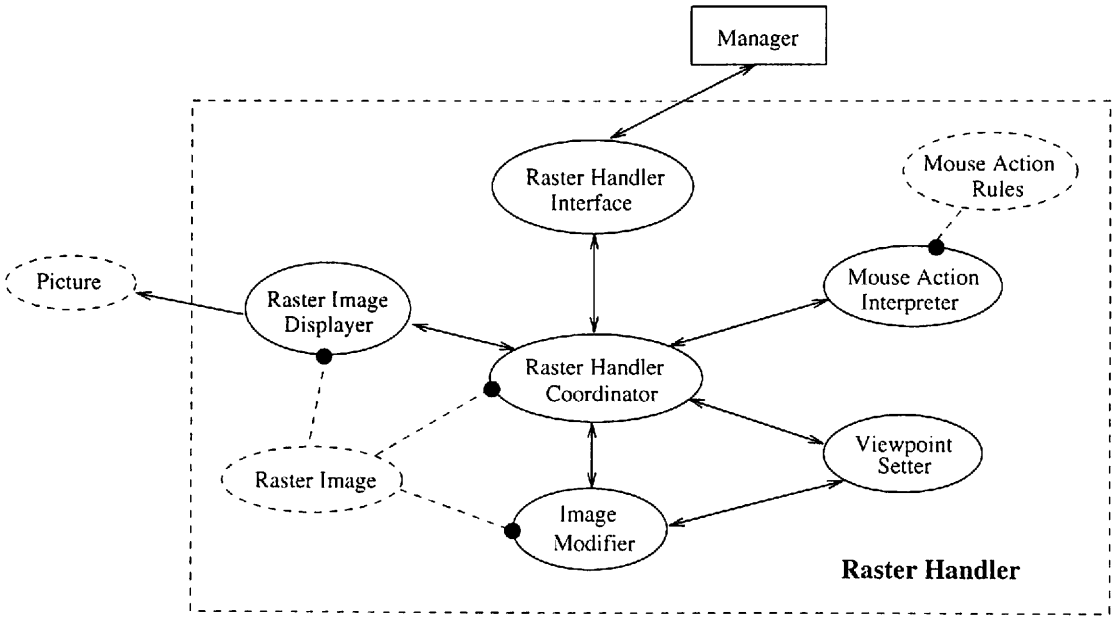

Figure 5: Components of Raster Handler

A raster handler consists of following components (illustrated in Figure 5):

- A *Raster Handler Interface* provides an interface between manager part and raster handler part. This interface passes commands, data and object reference between these parts.

- The *Raster Image Display* displays the raster image on screen. The raster image can be created by rasterizator in data handler part, and can be further modified by image modifier defined below.
- A *Image Modifier* mainly uses fisheye view technology to enlarge some parts of the image to emphasize some information.
- A *Viewpoint Setter* helps the user to choose the emphasized points in the image.
- A *Mouse Action Interpreter* takes "mouse action rules" to interpret the mouse actions. The mouse operations are passed to data handler part for further processing.
- A *Raster Handler Coordinator* is responsible to coordinate the activities of the above components in raster handler part.

4.4 Application Filter

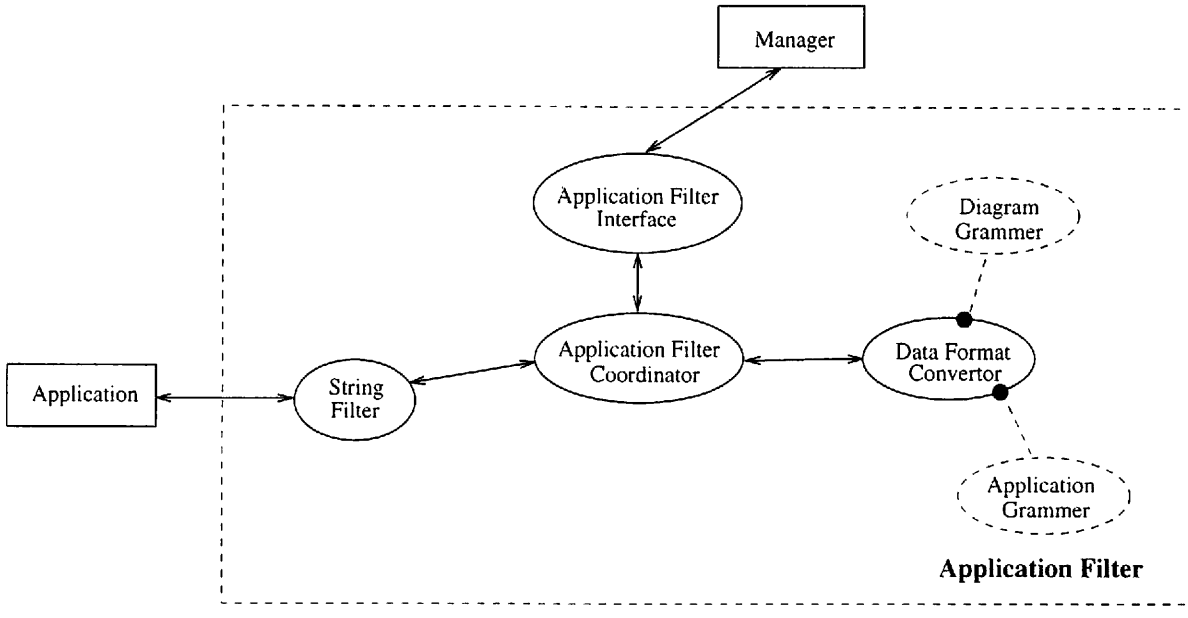

Figure 6: Components of Application Filter

An application filter consists of the following components (illustrated in Figure 6):

- An *Application Filter Interface* provides an interface between manager part and data handler part. This interface passes commands, data and object reference between these two parts.
- A *String Filter* maps the data from the application file and the data entities in the MixedPizza. If the data formats are different, a data format converter is required.
- A *Data Format Converter* is responsible to convert the data between formats of application data and data entities if these two formats are different.

- An *Application Filter Coordinator* is responsible to coordinate the activities of the above components.

4.5 Layout Generator

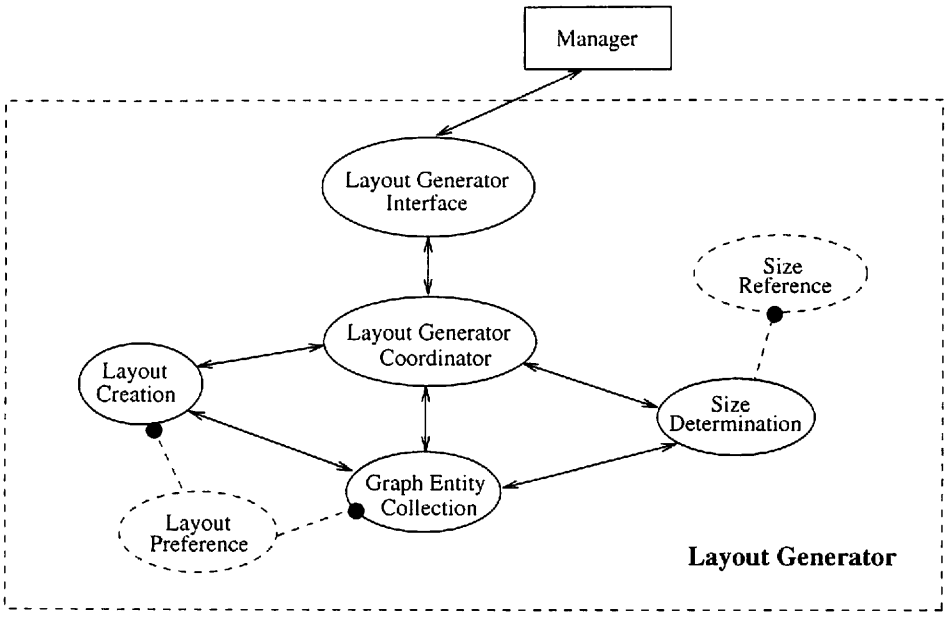

Figure 7: Components of Layout Generator

Layout generator is responsible to generate the layout for the data entities. The graph entity set can have the same topological structure as the data entity set in data handler part. However, it is also allowed to have different topological structures. For example, the topological structure for data entity set is a directed graph and for graph entity set is a tree. The graph entity also differentiates from the data entity in that graph entity provides local layout operations. A layout generator consists of the following components (illustrated in Figure 7):

- A *Layout Generator Interface* provides an interface between manager part and layout generator part. This interface passes commands, data and object reference between these parts.
- A *Graph Entity Collection* organizes the graph entities and provides the object reference of graph entities to the data entities in graph handler part.
- The *Layout Creation* in layout generator provides global operations and graph entities provide the local operations for layout generation. The geometric values of the graph entities are decided by the operations provided by both sides.
- The *Size Determination* determines the size for each 2-dimensional graph entity and the width for each 1-dimensional graph entity.

- A *Layout Generator Coordinator* is responsible to coordinate the activities of the above components.

4.6 Graphical User Interface

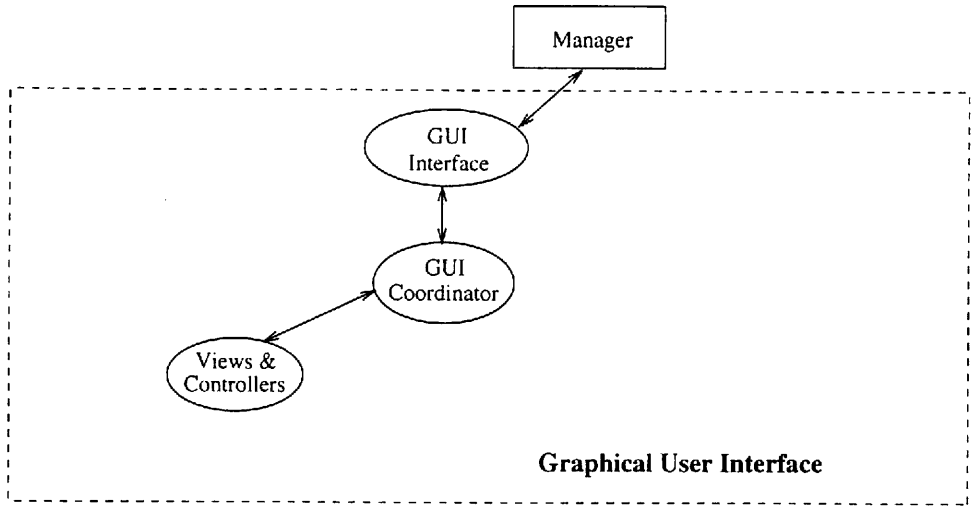

Figure 8: Components of GUI

Graphical user interface consists of the following components (illustrated in Figure 8):

- A *GUI Interface* provides an interface between manager part and graphical user interface part. This interface passes commands and data between these parts.
- *Views & Controllers* provides graphical widgets and controlling mechanisms.
- A *GUI Coordinator* passes the messages between manager part and views & controllers. Users' operations on the views and controllers are converted into text form in GUI coordinator. The GUI coordinator is the model in model-view-controller or model and control-view architectures.

4.7 Manager

A manager consists of the following components (illustrated in Figure 9):

- A *Component Interface Coordinator* is responsible to coordinate the activities of the interfaces of other functional parts.
- A *Command Interpreter* is responsible to interpret the commands from the interface of each functional part. The "command reference" defines the rules for the interpretation of the command for a specific MixedPizza.
- A *Operator Coordinator* is responsible to invoke the operations after the command is interpreted by the command interpreter. The "operator reference" defines the rules for the invoking of the operators.

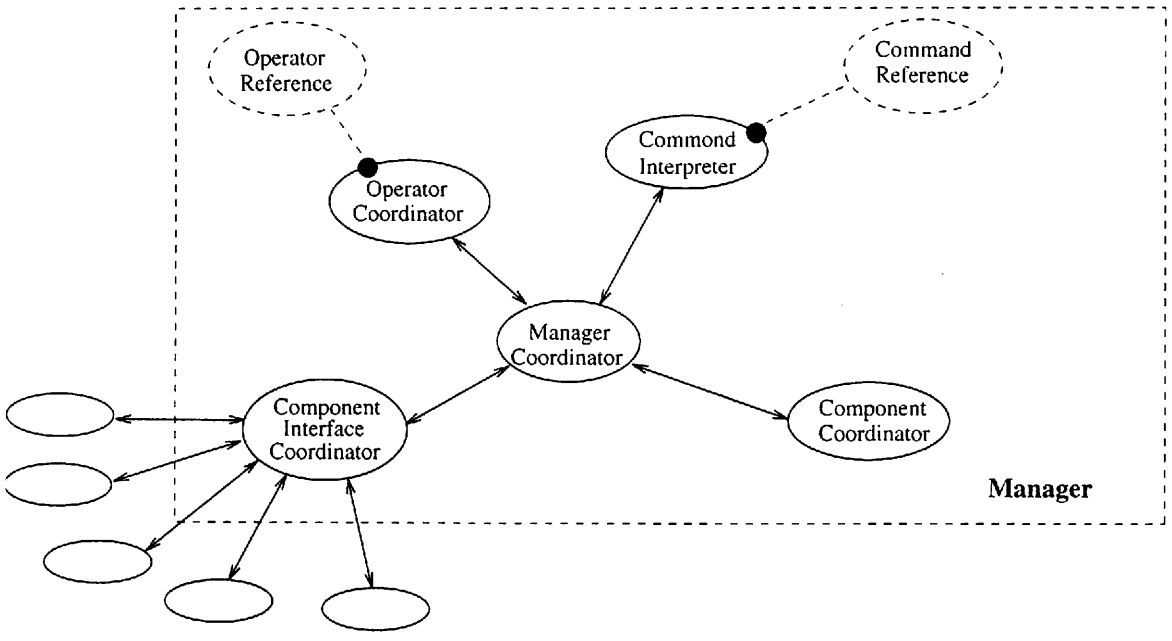

Figure 9: Components of Manager

- A *Component Coordinator.* It is required that there should be multiple raster handlers, multiple data handlers and multiple layout generators. The component coordinator is responsible to coordinate the components in this situation.
- A *Manager Coordinator* is responsible to coordinate the activities of the components in manager part.

5 Conclusions

The functions of possible MixedPizzas were analyzed and decomposed. Decomposition was performed based on object-oriented approach, so decomposed functions are represented as networks of objects. These objects should be reorganized to construct a platform architecture adequate to the fundamental idea of the PizzaSystem. These object networks should be useful to develop a DUI platform in some object-oriented languages.

We have the following plans to develop the PizzaSystem.

1. Design architecture of the PizzaSystem; Objects decomposed in this document will be reorganized in line with the fundamental idea of the PizzaSystem [3].
2. Decide specification of the PlainPizza and some Toppings; Their specification as objects and messages among them will be described in writing.
3. Implement the PlainPizza and some Toppings; Based on the specification, program of the PlainPizza will be written and some Toppings will be also written as examples.

4. Combine the PlainPizza and some Toppings to develop MixedPizzas; some MixedPizzas will be developed for a tasting of the PizzaSystem.

Acknowledgment

The authors would like to thank Mr Kiyoshi Nitta of Fujitsu Laboratories, ISIS for his useful comments.

References

[1] Tao Lin. *A General Schema for Diagrammatic User Interfaces.* PhD thesis, Department of Computer Science, The University of Newcastle, 1993.

[2] Kazuo Misue, Kiyoshi Nitta, Kozo Sugiyama, Takeshi Koshiba, and Robert Inder. Techniques for DUI platforms: Developing graph drawing applications on D-ABDUCTOR. Research Report ISIS-RR-96-7E, FUJITSU LABORATORIES, ISIS, 1996.

[3] Kazuo Misue and Kiyoshi Nitta. Basic idea of the hyper diagram platform. ISIS-BW-MISU9510B (Fujitsu Laboratories Ltd., ISIS internal document, in Japanese), October 1995.

[4] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. SMC*, 11(2):109–125, 1981.

[5] George W. Furnas. Generalized fisheye views. In *Proceedings of CHI'86, Human Factors in Computing Systems*, pages 16–23, 1986.

[6] M. Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of CHI'92*, pages 83–91, 1992.

[7] Kiyoshi Nitta, Robert Inder, Kazuo Misue, and Kozo Sugiyama. GOL: Graphical outline processor — simultaneously using a text view and a graph view. In *Asia Pacific Computer Human Interaction (APCHI'96)*, pages 469–478, June 25-28 1996.

[8] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.