Satisfying the Graphical Requirements of Visual Languages in the DV-Centro Framework

Paul C. Brown
DataViews Corporation
47 Pleasant Street, Northampton, MA 01060
email: paul@dvcorp.com

Abstract

The graphical requirements for implementing a visual language include defining the visual elements of the language, the rules for connecting them together, and the graphical relationships that must be maintained once they are connected. The solutions for these graphical requirements are intricate and inherently difficult to implement. The resulting implementations tend to be so specialized as to be applicable only to a single visual language.

The DV-Centro Framework makes it possible to implement sophisticated visual languages without having to develop all of this machinery. In this paper we examine the patterns of interaction between the components of the DV-Centro Framework. We introduce the Supervisor-Agent pattern as a means of understanding these interactions. Through this pattern we see how it is possible to assemble sophisticated application-specific visual language elements from DV-Centro library objects. We also see how it is possible to provide the low-level user interface for interacting with the visual language elements as library components.

1. Introduction

Designers of applications incorporating visual languages have two difficult problems to solve: first, they must meet the complex graphical requirements of the visual language itself; and second, they must correlate the diagrams of the visual language with the information that those diagrams represent as well as maintain the correlation as the diagrams are edited. This paper is the first half of a two-part presentation of the DV-Centro Framework for implementing applications employing visual languages. In this paper we present a slightly simplified version of the DV-Centro Framework and discuss how it provides complete and reusable graphical machinery for designing and implementing complex visual languages. In [Br97], we complete the description of the framework, showing how it provides the machinery necessary to maintain the correlation between a diagram and the information it represents, and how the presence of this machinery enables standardized implementations of application operations such as cut, copy, paste, and delete.

The DV-Centro framework is based on ideas from a number of different user interface frameworks, including the Smalltalk Model-View-Controller (MVC) framework [BMRSS96] [KP88] [Sh89], the Seeheim framework [Gr85] [Gr86], InterViews [LCITV92], Unidraw [Vl90a] [Vl90b], and the MFC Document-View Architecture [Ms96]. While all of these earlier frameworks provide a reasonable structure for implementing visual languages, none of them promote the reuse of complex visual language code in implementing new visual languages. Since easily half of the total effort can go into designing this portion of the system [MR92], this has a significant impact on the cost of implementing visual languages.

In this paper we seek to explain how the DV-Centro Framework makes it possible to reuse code (i.e. the DV-Centro library) to implement new visual languages. We will begin by defining the Supervisor-Agent Pattern, a means of characterizing the relationships between architectural components. This pattern makes clear the design dependencies between the components. Next, we will present the DV-Centro framework as if it were a direct evolutionary descendent of the MVC framework. We will use the Supervisor-Agent Pattern to clarify the relationships between the MVC architectural components and motivate its evolution into the DV-Centro Framework. The resulting framework provides a straightforward solution for satisfying the graphical requirements of a visual language by reusing standardized components (e.g. standardized data structures and algorithms).

In this paper we will use the notation of the Unified Modeling Language (UML) [BJR97].

1.1 Visual Language Graphical Requirements

The graphical requirements of a visual language include defining the visual elements of the language and the graphical relationships that must be maintained when these elements are connected together. Algorithms must be provided for graphically editing these elements (e.g. moving, scaling, or rotating them or editing text) while maintaining the graphical relationships (i.e. graphical connections) between the elements. The editing operations themselves are event driven, so appropriate event interpretations for mouse and keyboard events must be provided.

The solutions for these graphical requirements are intricate and inherently difficult to implement. The data structures are complex, containing information about both visualization and connectivity. These data structures are often further complicated by the use of parallel hierarchies. The algorithms for maintaining graphical relationships are not only complicated but they also tend to be specialized to both the specific data structures being managed and the specific geometries involved. The consequence is that the solutions tend to be so specialized that they apply to only a single visual language. Each new visual language requires the re-development of this machinery.

1.2 An Overview of Graphical Solutions in DV-Centro

As we shall see in the remainder of the paper, the DV-Centro Framework provides solutions that enable the implementation of visual languages without having to develop all of this machinery. It provides a rich standardized three-tier data structure for defining the logical elements (View Elements) of the Visual Language. The top tier contains the View Elements themselves. Each View Element is defined by one or more second-tier building blocks (Image Elements) whose visual appearance is defined by third-tier graphic primitives (Graphic Elements). The Image Elements and Graphic Elements are standard elements of the DV-Centro library. Only the View Elements are customized for the application.

Image Elements have connection regions (Pads) that are defined as arbitrary geometries. Graphical relationships between Pads are defined by Joints. Each Joint specifies a particular relationship (e.g. graphical containment of one pad within another) that must be maintained between the Pads. The relationship is defined independently of the geometries of the pads. Pads and Joints are standard elements of the DV-Centro library.

DV-Centro provides algorithms for graphically manipulating the Image Elements while maintaining the graphical relationships. These algorithms are completely independent of the visual appearance and graphical relationships of the elements. They simply direct image elements to perform transformations (e.g. translation, scaling, rotation), and the image elements report back what portion of the transformation is possible without violating the constraints. In the process, the affected image elements may generate transformation requests for other image elements (through the pads and joints) in order to maintain the graphical constraints. Only when all constraints have been satisfied is the transformation actually performed. DV-Centro also provides event interpretation (the Image Controllers) for mouse and keyboard events to implement in-place user-driven graphical editing of both graphical shapes and text. The Image Controllers are standard elements of the DV-Centro library, and the algorithms are embodied in the library's Image Controllers, Image Elements, Pads and Joints.

2. Pattern Analysis

We will begin by discussing the well-known Observer Pattern and defining a refinement of this pattern that we call the Supervisor-Agent Pattern. We examine the design dependencies between the Supervisor and the Agent in this new pattern and conclude that the Supervisor cannot be used independently of the Agent. We propose a design principle for encapsulating reusable functionality in an Agent.

2.1 The Observer Pattern

The Observer Pattern (Figure 1) [GHJV95], which is also referred to as the Publisher-Subscriber Pattern [BMRSS96], characterizes an Observer as an object that is notified by a Subject when changes in the Subject occur. In this pattern the Subject knows nothing about the Observer other than the Observer provides the interface required for notifications. The Observer Pattern says nothing at all about the Observer's knowledge of the Subject.

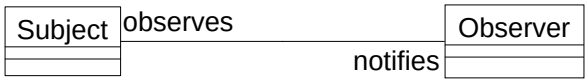

Figure 1: The Observer Pattern

2.2 The Supervisor-Agent Pattern

In the course of developing the DV-Centro Framework an interesting observation was made concerning the appearance of the Observer Pattern in earlier frameworks. In every case it was found that the Observer, besides observing the Subject, also

manipulated the Subject, whether recovering information from the Subject, making changes to the Subject, or both.

This pattern is distinct enough that we have given it its own name: the ***Supervisor-Agent Pattern*** (Figure 2). The ***Supervisor,*** a refinement of the Observer, is both an observer and a manipulator of the Agent, which is a refinement of the Subject. The Supervisor embodies the *policy* for how and when the Agent is to be manipulated. The ***Agent***, in addition to being the subject of the observation, provides functionality that is invoked by the Supervisor. In contrast with the Observer Pattern, in which no claim is made concerning the Observer's knowledge of the Subject, the Supervisor in the Supervisor-Agent Pattern must obviously have information about the Agent's interface in order to carry out its supervisory function.

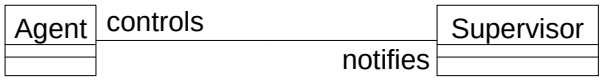

Figure 2: The Supervisor-Agent Pattern

2.2.1 Design Dependencies

Since the Supervisor must manipulate the Agent, it is clear that the design of the Supervisor is dependent upon the design of the Agent. On the other hand, since the Agent, as with the Subject in the Observer pattern, only knows about the Supervisor's notification interface, its design is only dependent upon the design of the notification interface. Given the assumption that the notification interface is stable (i.e. is a standard), then it is easy to see that the design of the Agent is independent of the design of Supervisor.

2.2.2 Code Reuse Implications

The design of the Agent is only dependent upon the notification interface of the Supervisor. Consequently, the Agent can be used with any number of Supervisors, and, in particular, with Supervisors that are designed *after the Agent is designed*.

The design of the Supervisor, on the other hand, is dependent upon the design of the Agent that it is intended to work with. As a result, the Supervisor can *only* be used in conjunction with that Agent - it has no utility without its corresponding Agent.

Because the design of an Agent is independent of the design of its Supervisor, an Agent that embodies a useful unit of generic functionality is a good candidate for membership in a class library. Later an application-specific Supervisor can be designed to define the policy for when and how the Agent's functionality should be used. This leads to the following design principle: ***If functionality is to be reused, it should be embodied in the Agent part of a Supervisor-Agent Pattern***. Of course these Agents can only be used with the specific notification protocol that they are designed for. It is the framework's responsibility to define these protocols.

2.2.3 Examples of Reusable Agents

X-Toolkit Widgets and Microsoft Windows Controls are both examples of reusable Agents in the Supervisor-Agent Pattern. In both systems the Agent (Widget or Control) provides reusable functionality in the form of a building block for user interface design. In both systems there is a simple notification mechanism for the Agent to tell a Supervisor that a change has occurred, although the identification of the "supervisor" is not well defined in either system (i.e. notification occurs via callbacks in X-windows and Windows Event Generation in Windows Controls). In both systems the code that employs the agents plays the Supervisor role and determines the policies for when and where the Agent's functionality will be employed.

3. Analyzing The MVC Framework for Supervisor-Agent Patterns

The Smalltalk Model-View-Controller (MVC) framework (Figure 3) is one of the earliest frameworks for the construction of user interfaces [BMRSS96] [KP88] [Sh89]. For consistency with the remainder of the

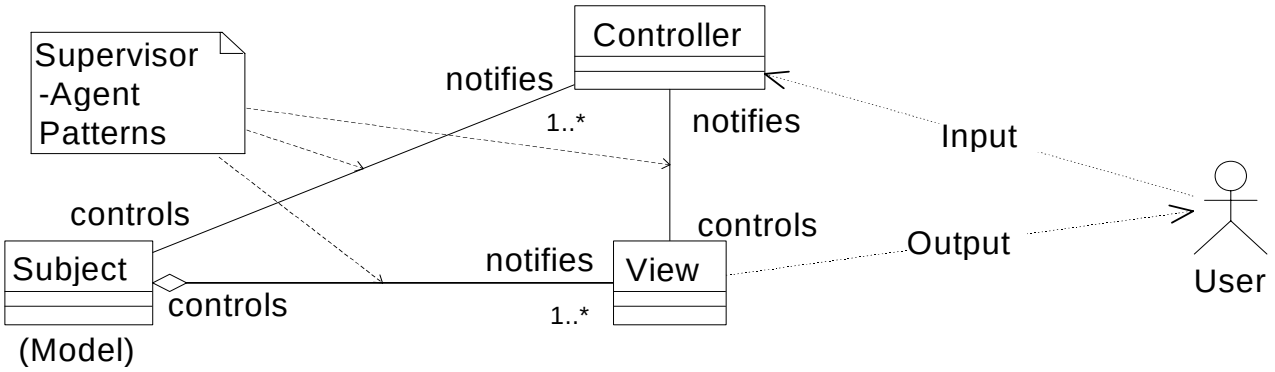

Figure 3: Supervisor-Agent Patterns in the Model-View-Controller Framework

paper we have replaced the term *Model* with the term *Subject* in the figure. The MVC framework provides a separation of responsibilities into three kinds of components: the Subject (Model), one or more Views, and one or more Controllers.

The ***Subject*** (MVC Model) comprises the information at the heart of the application. While this information may be presented in one or more views, it is kept in its pure informational form in the Subject, abstracted away from the visual syntax of its presentation in the View. The Subject plays the Agent role with respect to the Views and Controllers, providing notifications to the Views and Controllers when changes occur (as a result of other View or Controller manipulations of the Subject as described below) and providing an interface for the Views and Controllers to manipulate the information in the Subject.

The ***MVC View*** presents the visual language diagrams. It creates the images for the user to observe, and transforms the Subject's representation of the information to and from the diagram. There may be many views associated with a single Subject, each showing a representation of a subset of the Subject's information. The MVC View plays the Agent role with respect to the Controller, providing an interface for it to manipulate the View, and plays the Supervisor role with respect to the Subject which it manipulates to keep its information consistent with the diagram presented in the View.

The ***MVC Controller*** interprets external events arising from user actions. It establishes the policy for interpreting mouse and keyboard activity and utilizes the interfaces of the Subject and View to implement application functionality such as editing, storage and retrieval. The MVC Controller plays the Supervisor role with respect to both the Subject and the View. It manipulates the View (to edit the view) and the Subject (to save and restore) in response to external events. These events usually arise from user actions, but in data-driven applications, in which there is an active external data source, these events may come from the external data source as well.

3.1 Implications of the Supervisor-Agent Patterns in the MVC Framework

In Figure 3 we have labeled the relationships between the MVC components with the labels of the Supervisor-Agent pattern. As the diagram shows, there are three instances of the Supervisor-Agent Pattern in the MVC framework. The Controller plays the Supervisor role with respect to both the Subject and the View, while the View plays the Supervisor role with respect to the Subject. In terms of design dependencies, this means that the design of the View depends upon the design of the Subject, and the design of the Controller depends upon the design of both the Subject and View. Consequently the View cannot be used independently of the Subject, nor can the Controller be used independently of either the Subject or View.

Since the visual language is embedded in the MVC View and MVC Controller, the consequence of the design dependency is that the visual language cannot be used in another application with a different Subject. In addition, since the graphical data structures and algorithms supporting the visual language are embedded in the Controller and View, they are not usable in support of other visual languages either. This means that each visual language requires a new View and Controller, largely written from scratch. These conclusions are consistent with the MVC limitations concerning the "intimate connection between view and controller" and the "close coupling of views and controllers to a model" noted by Buschmann et. al. in [BMRSS96].

DV-Centro solves this problem by dividing the MVC View into two components, DV-Centro View (an application-specific Supervisor) and a DV-Centro Image (a standardized Agent), and a by dividing the MVC Controller into DV-Centro View Controllers (application-specific Supervisors) and DV-Centro Image Controllers (standardized Agents).

4. The Simplified DV-Centro Framework

We now perform three decompositions on the MVC framework to arrive at a simplified DV-Centro framework: 1) we decompose the MVC View component into the DV-Centro View component and the DV-Centro Image component; 2) we further decompose the information content of the View and Image into View Elements and Image Elements; and 3) we decompose the event interpretation of the MVC Controller into the DV-Centro View Controller and the DV-Centro Image Controllers. The resulting simplified DV-Centro Framework is shown in Figure 4.

4.1 The Image

The ***Image*** is responsible for the visual presentation of a diagram, defined as an assemblage of building blocks known as *Image Elements*. The Image provides an interface for creating Image Elements and for establishing the spatial relationships that must be maintained between these Image Elements[1]. It provides

[1] The Image Elements and the spatial relationships between them are the Spatial Relations Graph of Rekers and Shürr [RS96]. The algorithms in the Image correspond to the Constraint Solving described in this

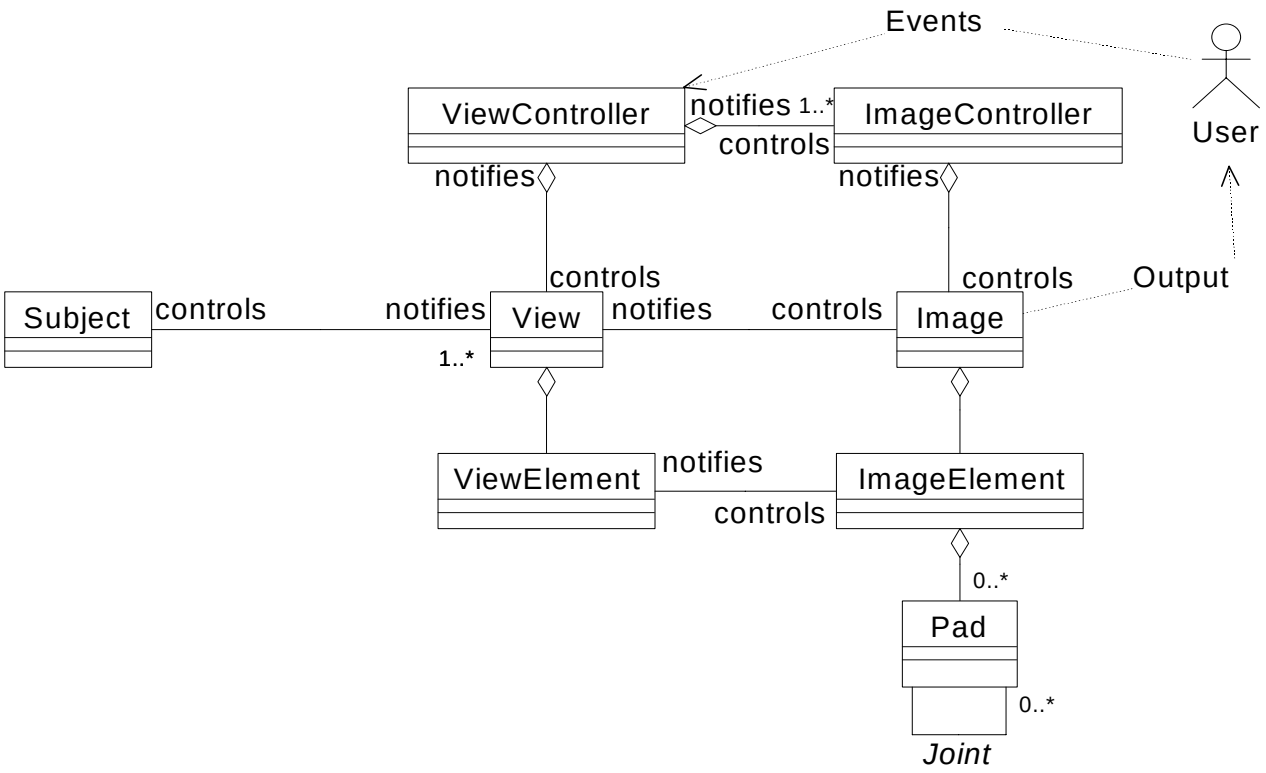

Figure 4: Simplified DV-Centro Framework

interfaces for moving, scaling, and rotating these Image Elements (in actuality, the interface allows an arbitrary two-dimensional affine transformation to be applied to the Image Elements), and for altering their visual properties. Support for textual image elements, including in-diagram editing of the text, is provided. The Image also contains the algorithms necessary to maintain the spatial relationships and the physical layout of the diagram during the modification of the elements.

The Image plays the Agent role with respect to the View, providing notifications to the View when graphical or textual changes occur to existing Image Elements. The Image provides an interface for the View to add, remove, connect, and disconnect Image Elements. It notifies the View when existing Image Elements are geometrically modified or textually edited.

The Image also plays the Agent role with respect to the Image Controllers (typically a graphical editing controller and a text editing controller), whose function it is to provide event interpretation during in-place graphical and textual editing within a drawing. The Image Controllers are the primary manipulators of existing Image Elements. As the Image Controllers manipulate the Image, the Image provides feedback as to whether the manipulations are allowed by the current graphical constraints.

4.1.1 Image Elements

Image Elements (Figure 5) are the building blocks of diagrams. They are logical entities that are assemblies of two types of components: *Graphic Elements* and *Pads*. ***Graphic Elements*** define the visualization of the Image Element. An arbitrary number of Graphic Elements, each

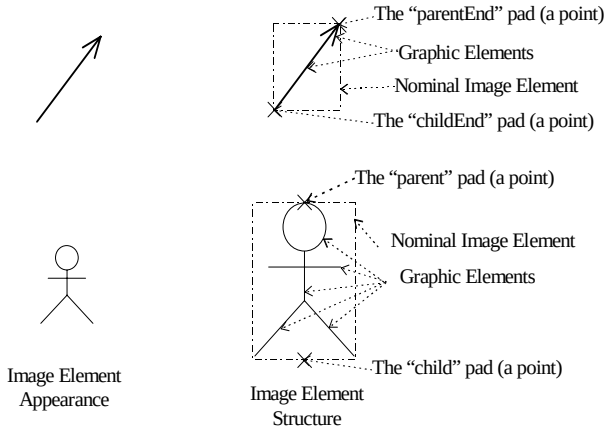

Figure 5: Image Element Examples

reference, and the visualizations of the Image Elements correspond to the Physical Layout. In DV-Centro there is no need for a Graphical Scanning component as the Physical Layout cannot be edited directly in the DV-Centro framework.

with its own shape and visualization properties, can be added to an Image Element to provide as sophisticated a visualization as required. DV-Centro Graphic Elements include scaleable text, bitmaps, and pixmaps as well as geometrically defined graphics.

Pads are connection regions that are used to establish graphical relationships between Image Elements. Each Pad is simply a geometric shape, either a point, a line, or one of a number of two-dimensional shapes. An arbitrary number of Pads may be associated with an Image Element. A *graphical connection* between two Image Elements (a *Joint*) is just a graphical relationship that must be maintained between two Pads. Note that since the Pads have no inherent visualization, the positioning of the Pads with respect to the Graphic Elements is independent of the appearance of the Image Element.

When Image Elements are manipulated (e.g. moved, scaled, or rotated) all of their component pieces (Graphic Elements and Pads) are manipulated uniformly. Thus the relative positions the Graphic Elements and the Pads is maintained during Image Element manipulation.

4.1.2 Image Element Connections: Graphical Relationships

A graphical connection between two Image Elements is really a graphical relationship that must be maintained between a Pad on one Image Element and a Pad on the other. These graphical relationships are established and maintained through the use of an intermediary known as a ***Joint***. Each type of Joint defines a particular graphical relationship that must be maintained between the Pads (e.g. the geometry of one Pad must be contained within the geometry of the other).

The key to the flexibility of Joints is that they are able to enforce their constraints regardless of the geometry of the Pads that they connect. However, there are some conditions that must be met. For example, if the Joint specifies that one pad is to be contained within another, the pads must meet two requirements: 1) the containing pad must be of the same or higher dimension than the contained pad (it does not make a lot of sense to contain a rectangle within a point); and 2) it is a requirement of the algorithms used that the containing pad must be convex in shape (if it is two-dimensional).

Each Image Element has two special Pads that represent the "front" and "back" of the Image Element. These pads are used to implement the containment of one Image Element within another. For example, connecting the back pad of a text Image Element to the front pad of a rectangle Image Element with a containment Joint will keep the text inside the rectangle even as the text is edited.

Other Pads, usually referred to as "edge" pads can be used to connect *Paths* (lines) to other Image Elements. If the point pad at one end of a path is connected to the edge pad of a rectangle, then the end of the path will always remain connected to the edge of the rectangle. In Figure 6 we see the result of connecting the "child" Pad of John with the "parentEnd" pad of the Father relation, the "parent" Pad of Tim with the "childEnd" pads of both the Father and Mother relations, and the "child" Pad of Mary connected to the "parentEnd" pad of the Mother relation.

Joints not only define the graphical relationships between the Pads (and hence the Image Elements that the Pads belong to), but they also enforce these relationships. When two Pads are connected via a Joint, the Joint ensures that the relationship can be initially satisfied or the connection operation is not allowed. Once the Pads have been connected, if one Pad is manipulated (via its Image Element), the Joint attempts to manipulate the other Pad (and its Image Element) to maintain the graphical relationship. If the second Pad is unable to accommodate, the manipulation operation is not allowed. Thus the Image Element - Pad - Joint structure forms a distributed algorithm for maintaining graphical relationships. In the example of Figure 6, if Tim were to be moved, the "childEnd" ends of both the Father and Mother relations would be moved, thus keeping the line of both relations "attached" to Tim.

4.2 The View

The ***View*** is a manager of a diagram in a visual language. Each instance of a View contains a single diagram. The diagram comprises *View Elements* that are the logical elements of the visual language. A connection between two View Elements is implemented by connecting two Pads from their associated Image Elements. These connections provide the graphical relationships between the logical elements of the visual language[2]. The View provides the user interface for creating and editing a diagram.

4.2.1 View Elements

View Elements are the logical elements of the visual language. To form the actual language element, each View Element creates an assemblage of one or more Image Elements, configuring their Graphic Elements and Pads to give the visualization and connection regions required for this element of the language. In effect, the View Element becomes the Supervisor of this collection

[2] This is the Abstract Syntax Graph and its mapping into the Spatial Relations Graph of Rekers and Shürr [RS96].

of Image Elements. View Elements are notified by their associated Image Elements when there is a modification to the Image Element. In Figure 6, the Person view element has the single Image Element shown in the bottom of Figure 5, and the Father and Mother relations each have the single Image Element shown in the top of Figure 5 (for simplicity we have not shown the text for either Image Element in Figure 5)

Each View Element in a View represents a particular unit of information from the Subject known as a *Subject Element*. For example, a Subject Element might represent a person. Each View Element reflects the details of its Subject Element's information in the visual appearance of the View Element, such as making the person's name appear in the drawing. More information concerning Subject Elements can be found in [Br97].

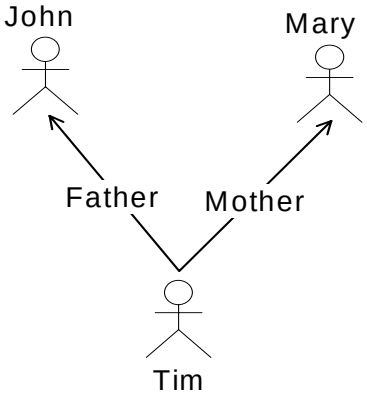

Figure 6: Diagram Example

4.2.2 Relating the Subject and View

The View embodies the policies for maintaining consistency between a diagram and the information that it represents. For completeness, we give a brief summary of this topic here, but the management of the relationship between the Subject and the diagram is the main topic of [Br97].

The View plays the Supervisor role with respect to the Subject. The View interprets notifications from the Subject in terms of manipulations required to View Elements, which in turn manipulate the Image Elements to reflect the information change in the Subject.

The View also plays the Supervisor role with respect to the Image. It is notified by the Image when an Image Element is modified, and in turn calls a method (`recoverData` or `recoverGeometry`, depending upon the nature of the modification) on the View Element that supervises the Image Element. Within these methods lies the logic for retrieving information from the Image Element and updating the associated Subject Element information. In the example given above, if the name "Tim" were selected and edited (using the Image Controllers discussed later), the `recoverData` method would be called on the Person View Element. This method would take the new name and update its associated Subject Element.

The View plays the Agent role with respect to the View Controller. The View notifies the View Controller when specific changes to the View occur (generally this is just notifying the View Controller that the selection has changed so that user interface state can be updated). It presents an interface to the View Controller that allows the View Controller to add, remove, connect and disconnect View Elements consistent with the rules of the visual language.

4.3 Controllers

If we were to leave the framework with a single Controller per view (as in the MVC framework), this Controller would have to provide the interpretation of all user actions in the View. In particular, the single Controller would have to handle the complex interpretation of mouse and keyboard events associated with the graphical and textual editing of the diagram. DV-Centro divides these responsibilities between Image Controllers and the View Controller, with the Image Controllers interpreting the detailed mouse and keyboard events associated with the graphical and textual editing of the diagram.

4.3.1 View Controllers

Each ***View Controller*** is the manager of a single View. It provides the user interface necessary to display the View's diagram, and it provides the interpretation of events arising within that interface. The View Controller in turn uses the Image Controllers to implement the graphical and textual editing operations. Upon recognizing an event that initiates a graphical or textual editing operation (such as a mouse button 1) the View Controller activates the appropriate Image Controllers (if a text Image Element is selected, both the graphic and text editing Image Controllers will be activated) and passes subsequent keyboard and mouse events to the Image Controllers for interpretation. The View Controller plays the Supervisor role with respect to the Image Controllers.

In [Br97] we see that the View Controller itself can make use of subordinate View Controllers, and that virtually all of the functionality necessary for visual language editing can be provided in a reusable Agent controller known as the *Network Editor*.

4.3.2 Image Controllers

Image Controllers are manipulators of Image Elements. Each controller interprets specific mouse and

keyboard events and performs a particular kind of manipulation of the Image Elements. The DV-Centro ***Image Editor***, for example, is an Image Controller that interprets mouse buttons and mouse motion to provide the ability to move and scale arbitrary Image Elements (the Image Elements themselves maintain the graphical constraints as this is happening). The DV-Centro ***Text Editor*** is an Image Controller that interprets mouse buttons and motion to provide text cursor placement and text selection, and interprets keyboard input as editing operations on the text.

Image Controllers play the Supervisor Role with respect to the Image and Image Elements. They directly manipulate the Image and Image Elements, particularly with respect to their graphical position and appearance.

Image Controllers play the Agent Role with respect to the View Controllers. They provide an interface for the View Controllers to initiate Image Controller operations. Once these operations are initiated, the View Controllers pass events to the Image Controllers for interpretation. Finally, the Image Controllers notify the View Controllers when operations are completed or aborted.

Note that since the Image and Image Elements are already standardized library elements, the Image Controllers can also be added to the library and used in the construction of applications.

5. Summary

The DV-Centro Framework, with its library of Image Controllers, Image, Image Elements, Graphic Elements, Pads, and Joints provides all of the graphical machinery necessary to implement complex visual languages. In this framework, the application-specific View Controller, View, and View Elements determine how and when these capabilities are employed. The Supervisor-Agent pattern was used to identify the design dependencies and support the contention that the Image Controller, Image, Image Elements, Graphic Elements, Pads and Joints can truly be used with a wide variety of View Controller, View and View Elements without modification.

In explaining the DV-Centro Framework, we introduced the Supervisor-Agent Pattern, a refinement of the well-known Observer Pattern. An analysis of design dependencies within this pattern led to a design principle that generic functionality can be appropriately packaged as an Agent in this pattern and reused with an arbitrary number of Supervisors.

6. Bibliography

BMRSS96 F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, 1996

BJR97 G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language for Object-Oriented Development, Version 1.0*, Rational Software Corporation, www.rational.com, (1997)

Br97 Brown, Paul C., "Correlating Diagrams with Information in the Centro Framework," Technical Note 97-2, DataViews Corporation, Northampton, MA (1997)

GHJV95 E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

Gr85 Green, Mark, "Report on Dialogue Specification Tools," in *User Interface Management Systems*, Gunther E. Pfaff, Editor, Springer-Verlag (1985)

Gr86 Green, Mark, "A Survey of Three Dialogue Models," *ACM Transactions on Graphics*, Vol. 5, No. 3 (1986)

KP88 Krasner, Glenn E., and Pope, Stephen T., "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming* Vol. 1 No. 3 (1988)

LCITV92 M. Linton, P. Calder, J. Interrante, S. Tang, and J. Vlissides, *InterViews Reference Manual*, CSL, Stanford University, 3.1 Edition, 1992

MR92 Myers, Brad A., and Rosson, M. B. "Survey on User Interface Programming," *Proceedings SIGCHI'92: Human Factors in Computing Systems*, ACM (1992)

Ms96 *Microsoft Foundation Classes Version 4.2*, Microsoft Corporation, 1996

RS96 J. Rekers and A. Schürr, "A graph based framework for the implementation of visual environments," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*, IEEE Computer Society Press (1996)

Sh89 Shan, Yen-Ping, "An Event-Driven Model-View-Controller Framework for Smalltalk," *Proceedings of OOPSLA '89* (1989)

Vl90a Vlissides, John M., *Generalized Graphical Object Editing*, A Dissertation Submitted to the Department of Electrical Engineering and the Committee on Graduate Studies, Stanford University (1990)

Vl90b Vlissides, John M., "Unidraw - A framework for building domain-specific graphical editors," *ACM Transactions on Information Systems*, Vol. 8, No. 3, pp. 237-268, July 1990