A Pattern Language for Developing Web based Multi Source Data Acquisition Application[1]

Lei Zhen
Email: zhen_lei@263.net

Guangzhen Shao

Email: shaoguangzhen@salien.com.cn

Beijing Salien Computer Company

Abstract

This pattern language in progress presents patterns for developing web based multi source data acquisition applications. It deals with data acquisition applications that acquire and present data from multiple sources such as real-time databases, relational databases and other sources. The pattern language consists of four patterns. Display Component is the "thin-client" embedded in web browsers to present dynamic displays. Configuration Client is used to define and maintain dynamic displays and data sources. Remote Data Collector acquires and routes data. And finally Configuration Database stores the configuration data of dynamic displays.

1. Introduction

1.1 Intent

This pattern language in progress deals with developing web based multi source data acquisition applications. The data sources of the application may be of various types such as real-time database, relational database and other sources like dynamic web pages.

1.2 Scenario

Integrated and up-to-date information shared across the value chain is very important to the companies in process industries for optimizing their entire value chain and making fast and profitable decisions at all levels.

[1] Copyright © 2002, Lei Zhen and Guangzhen Shao
Permission is granted to copy for the PLoP 2002 Conference. All other rights reserved.

Consider the following scenario: when the leader of a refinery company opens his web browser, the main page of a Decision Assistant System is presented. In the center of the display, is the simulation of the production process. The main assemblies of the production process are presented, the relative numbers are colorized, e.g. with red indicating a warning. In the left of the display, he can see the important statistical data of the enterprise. In the right of the display, he can see the prices of petroleum and productions. If he wants to see details, he can click the relative target and go to the correlated displays.

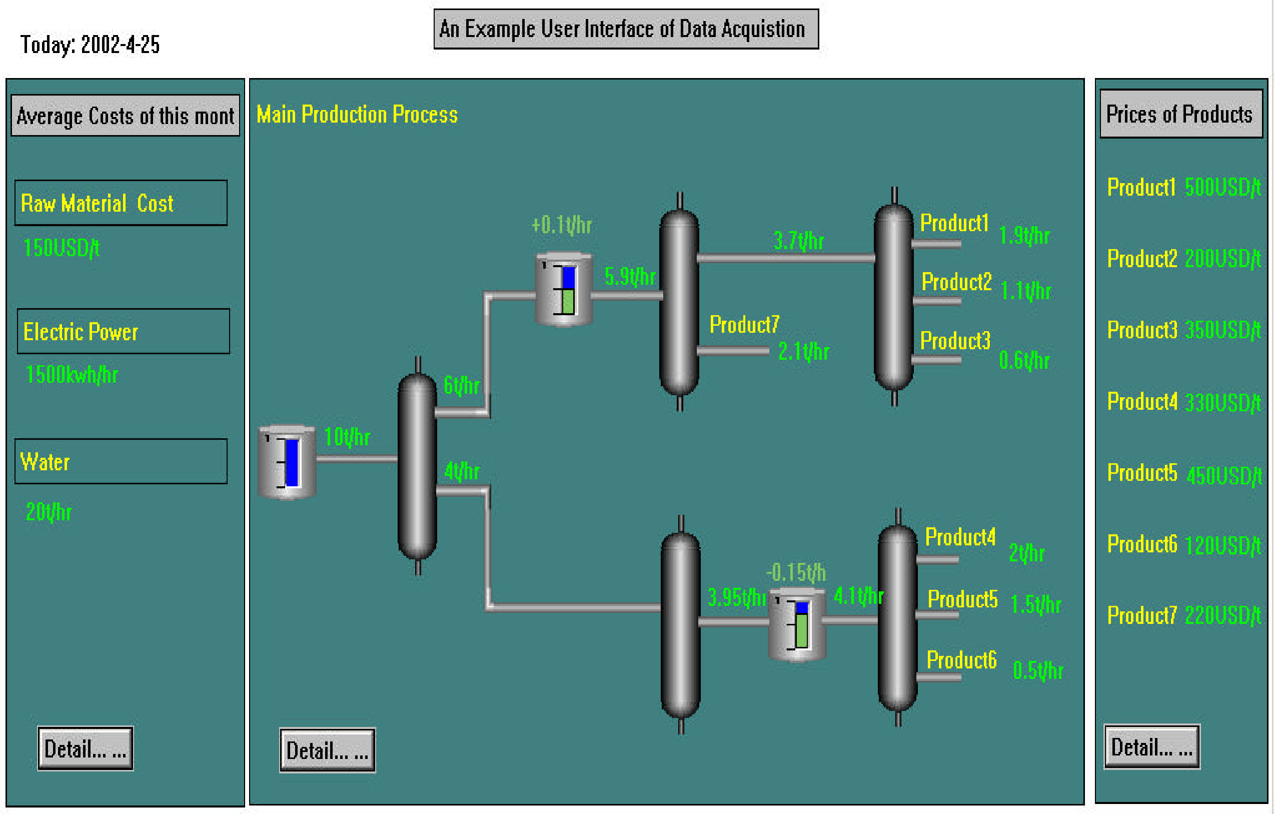

Fig 1: An example of user interface of data acquisition used in a Decision Assistant System

In the above case, all data is presented in a single web page, but the sources of the data are different. The real-time data of the production process is retrieved from a real-time database. The statistical data on the other hand is retrieved from the relational database of a Management Information System. And furthermore the market data is retrieved from the web site of a third party service provider.

Traditionally data acquisition systems collect real time data from a production process, store them into a process database and present them to engineers. Now it is expected to collect data from sources both inside the enterprise (MIS such as a storage management system) and outside the enterprise (such as a web site of stock prices), and its users include not only engineers but also enterprise leaders, managers, partners and other personnel of the enterprise. Now more and more vendors of data acquisition and real time database systems are developing new systems and updating their old system by integrating the data acquisition with the ERP and other information systems of their customers.

This pattern language in progress is based on our own experiences and studies of many systems of different companies. We began the work of integrating data acquisition into Decision Assistant Systems since 1995. And we have built web-based systems for several refineries and electric power companies in China since 1997. Furthermore there are some other productions that are used in this paper as examples:

- PI Interactive Configurable Environment (PI ICE) of OSISoft [OSISoft]
- iFixWebServer of Intellution[Intellution]

We got the information of these products from publicly available books e.g. [ZHA2001] and related web sites.

1.3 Overview of this pattern language

There are four patterns in this pattern language, showed below:

No.	Name	Description	Related Patterns [RK1997]
1	Display Component	The "thin-client" embedded in web browsers. It can be a Java Applet or ActiveX Document.	Remote User Interface Remote Database Distributed Application Kernel
2	Configuration Client	To define and modify the data sources and the displays for end-users.	Remote Database
3	Remote Data Collector	It acquires and routes data from different data sources.	Remote User Interface Remote Database Distributed Application Kernel
4	Configuration Database	To store the configuration data of the system.	Remote Database Distributed Database

1.3.1 Deployment View

Figure 2 is the typical deployment view of this pattern language.

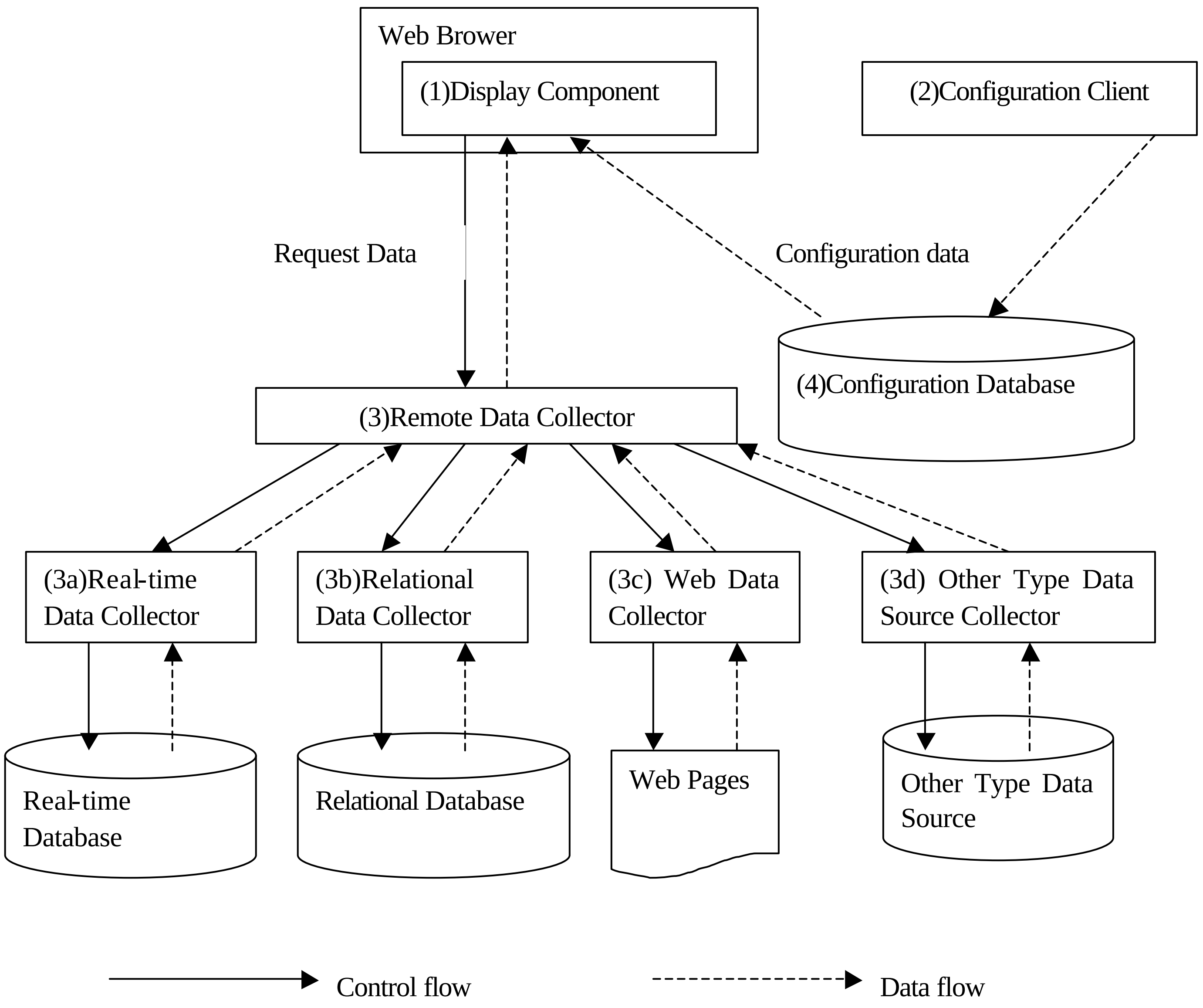

Fig 2: Deployment View

(1) The Display Component is a thin-client component, which is embedded in the web browser. The browser invokes the Display Component when the end-user opens the web page of the data acquisition system.

(2) The Configuration Client may be either a web based client or a normal desktop client. By using it, one can define and modify the displays for end users and the data sources for data acquisition. It stores the configuration data in the Configuration Database.

(3) The Remote Data Collector and its extended collectors are deployed in the application server.

(3a) The Real-Time Data Collector deals with the request for acquiring data from real-time database (RTDB). Because of the variety of RTDBs, there may be different Real-Time Data Collectors for each RTDB involved.

(3b) The Relational Data Collector deals with the request for acquiring data from a relational database.

(3c) The Web Page Data Collector deals with the request for acquiring data from dynamic web pages.

(3d) There may be other kinds of data sources, for example, the data from control systems. Special data collectors should be built for these data sources.

(4) The Configuration Database is a kind of relational database.

1.3.2 The interaction of the patterns

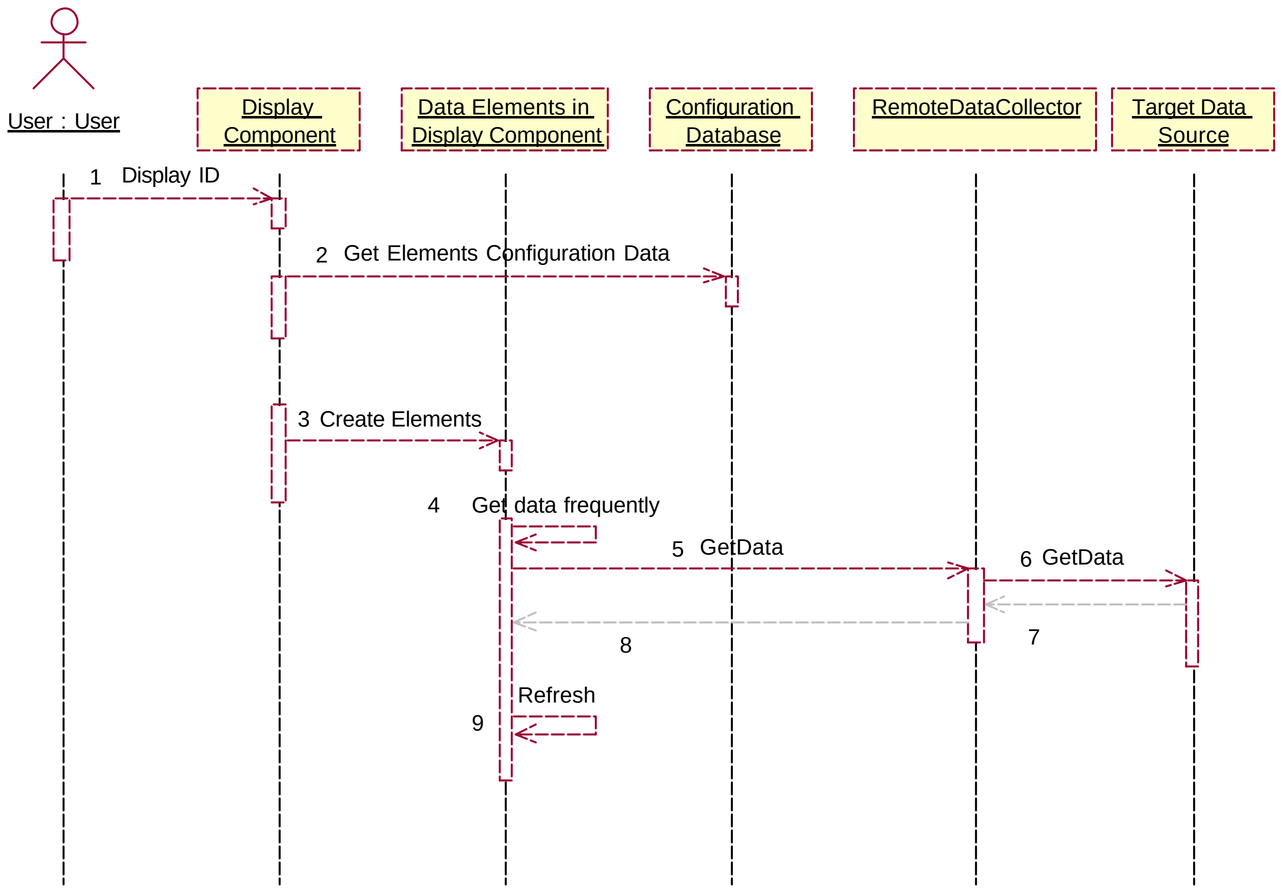

Fig 3: Sequence Diagram

(1) The display ID is provided as a parameter for the Display Component by the web page.

(2) The Display Component gets data from the Configuration Database. The data contains the properties of the Display Component (such as title and background image)

and the properties of the displayed elements.

(3) The Display Component creates and displays elements that include data elements, normal elements and control elements.

(4) There is a timer that controls the frequency of refreshing the data.

(5) Each data element requests data from the Remote Data Collector frequently.

(6) The Remote Data Collector gets the data from the target data source.

(7) Return the data to the Remote Data Collector.

(8) Return data to the data elements.

(9) The Data Elements refresh the data.

2. Pattern 1 - Display Component

2.1 Intent

For integrating data from different sources into one display, the Display Component presents dynamic displays based on configuration data as a “thin client” component embedded in a web browser.

2.2 Context

You are developing the client part of a web based data acquisition system. It has to integrate the data from different sources into one display. The data may include real-time data, relational data and other types. The refresh frequencies may be various from seconds to minutes.

2.3 Forces

- **Interactive and Configurable:** The dynamic displays depend on the business needs of the end users. They should be changeable according related business needs. But the users also want to gain more information about dynamic data in the display by simple operation. For example, when they move the mouse on the dynamic data, they can see the upper and lower warning limits on the tip text; when they click on a dynamic data, the display can show the historic data trend of this data.
- **Data integrity and consistency:** You have to ensure data integrity and consistency. Even if a data source has problems (e.g. network problems) and some data cannot be acquired in a defined period (timeout period).
- **Interactive and Configurable vs. Complexity**: The dynamic displays should be both interactive and configurable. To ensure a uniform interface, you want to present all information in one display. However generating one dynamic web

page as an asp or jsp for each information is technically much easier and the structure is less complex.

- **Multithreading vs. asynchronous**: Using asynchronous transmission is easy to implement. However, using multithreading transmission will make your system more robust.

2.4 Problem

How to integrate dynamic data from different sources into one display?

2.5 Solution

Provide a Display Component, which is a "thin client" component embedded in a web browser. It retrieves the configuration data from the Configuration Database, and then presents the dynamic display based on that data. Each data element connects with (or creates) a Remote Data Collector and acquires data through it.

Use multithreading to avoid a data transmission block. For each data element, there is a separate thread for acquiring data. Furthermore define a timeout period for each data element. If the data element does not acquire data from the data source after this period, the value of it will be set to "N/A".

2.6 The Structure of the Display Component

A display presents a background image and several elements on it. The background image may be a gif or jpeg image. The elements include data elements that display data from the Remote Data Collector and refresh frequently, normal elements that are independent of dynamic data source such as "Text Label" and "Time Label", and control elements that provide more interactive functions such as links between displays. The properties of the elements are stored in the Configuration Database.

Each data element has properties for displaying data such as data source type, data source name, and data refresh frequency. The data elements acquire data from the Remote Data Collector frequently. All data elements have the same properties. But the values of some properties depend on the data source type that the data element binds. If the data source type is a kind of RTDB, then the data source may be the server name of the RTDB and the target name is the data point number in the RTDB. If the data source type is a relational database, then the data source may be the ODBC name built in the Application Server and the target name may be an SQL statement. The data elements themselves do not deal with these properties. They pass the values of these properties on to the Remote Data Collector.

Figure 4 shows the concept model of display with dynamic data elements.

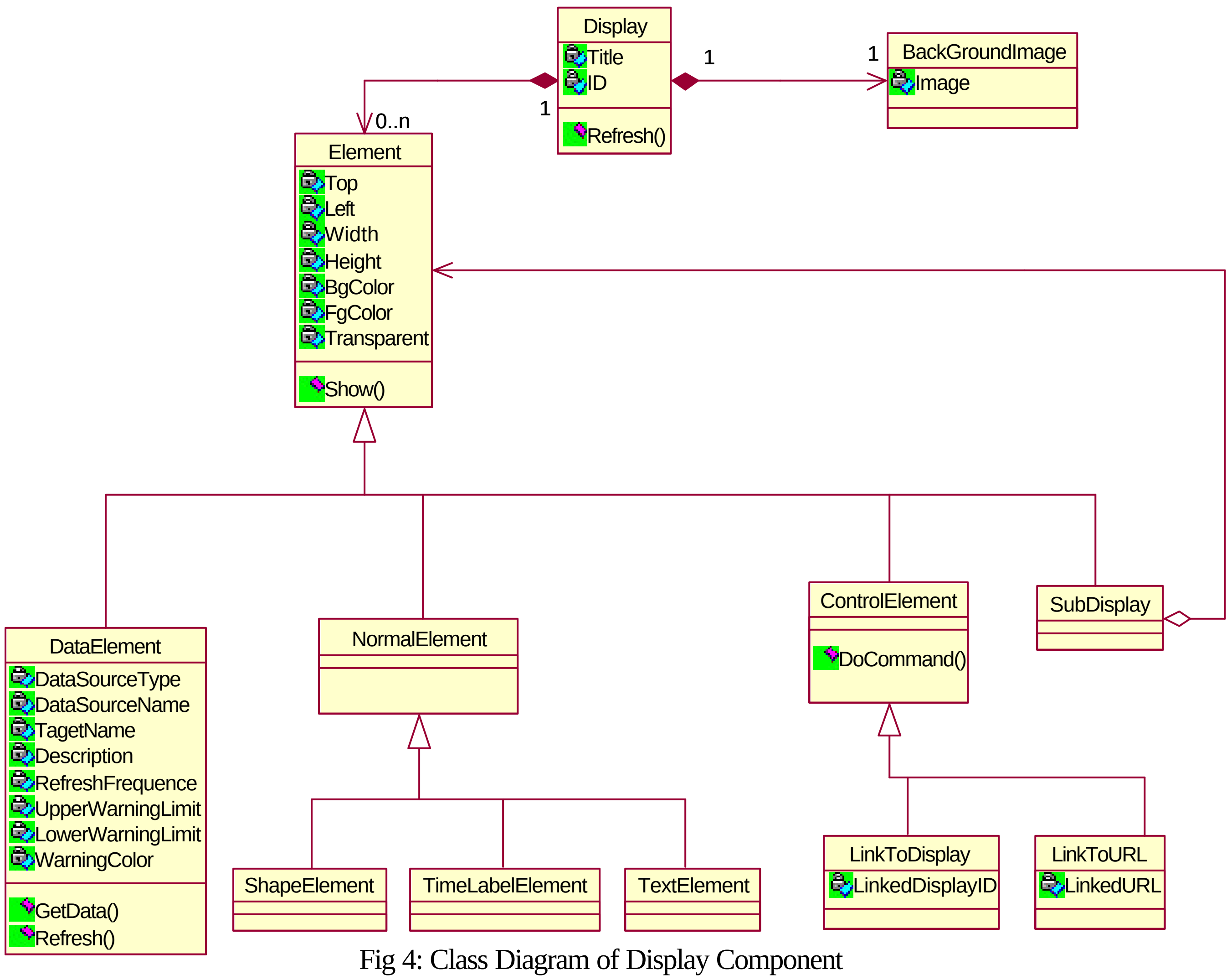

Fig 4: Class Diagram of Display Component

2.7 Consequences

This pattern has several benefits:

- You can present as many dynamic displays as you want by using a single component.
- Because the Display Component is independent of data sources, it can be used in many situations.
- The user interface can be more powerful and interactive.

It has also some drawbacks:

- If there were a lot of data coming from several relational databases involved in one display, this would cost more execution time to retrieve the data.
- Because the interaction between Display Component and Remote Data Collector

cannot be implemented by off-the-shelf system components and a standardized communication protocol, the communication issues have to be addressed by the application developer.

2.8 Related Patterns

The Display Component uses some patterns in [RK1997]. They are:

Remote User Interface: The Display Component is a thin client embedded in web browser.

Remote Database: The Display Component retrieves configuration data from the Configuration Database.

Distributed Application Kernel: Each data element connects with (or creates) a Remote Data Collector and acquires data through it.

Many design patterns [GHJV95] can be used to implement this pattern. There are:

Composite pattern: A display is a composite of elements.

Facade pattern: Data elements get the data from Remote Data Collector that separates data elements from data sources.

Iterator pattern: The elements in the display are stored in a collection that is implemented as an iterator.

2.9 Known Uses

1) iFixWebServer of Intellution [Intellution]. The client side of this production is a Java Applet embedded in a web browser to present dynamic displays.

2) PI Interactive Configurable Environment (PI ICE) of OSI Soft [OSISOFT]. The Active View of PI ICE is an ActiveX embedded in a web browser to present dynamic displays.

3. Pattern 2 - Configuration Client

3.1 Intent

The Configuration Client defines and maintains the displays presented by Display Component.

3.2 Context

You are developing a data acquisition system and you decide using a Display Component as the interface for end users. Your users want to define and maintain the dynamic displays themselves.

3.3 Forces

- **Complexity vs. Usability:** A full functional WYSIWYG client will help your users to define and modify the dynamic displays easily and quickly, but it is more costly and complex than a simple one. However, an extremely simple client can be developed easily and quickly, but the usability of it may be poor.
- **Balance of the cost between development and deployment:** The deployment cost of data acquisition is usually high because there are lots of dynamic displays to be configured. A powerful tool such as a full functional WYSIWYG client can make the job easy and reduce the deployment cost, but the development cost of this kind of tool will be reasonable high.

3.4 Problem

- How to store configuration data of the data acquisition system correctly and easily?
- What kind of solution can reduce the total cost of your application?

3.5 Solution

Use a Configuration Client, which is a combined solution, because it defines configuration data and stores them. It is more like a WYSIWYG image editor. But it does not include complex functions that may cost more time and money to implement. It can load a background image and allows defining the elements easily. In this case, you can use some parts of the Display Component in the Configuration Client.

3.6 Consequences

This pattern has several benefits:

- You can develop a Configuration Client that suits the requirements and the conditions of your application.
- You can balance the cost between development and deployment and then reduce the total cost.

3.7 Related Patterns

It stores the configuration data into the Configuration Database that is a ***Remote Database*** [RK1997].

Many design patterns [GHJV95] can be used to implement this pattern, e.g. ***Composite pattern, Iterator pattern***.

3.8 Known Uses

1) iFixWebServer of Intellution [Intellution]. iFixWebServer presents the graphical dynamic display defined by iFix system, so the Configuration Client of iFixWebServer is iFix Client.

2) PI Interactive Configurable Environment (PI ICE) of OSI Soft [OSISOFT]. PI ProcessBook is an easy-to-use graphics package that allows users to create dynamic, interactive graphical displays.

4. Pattern 3 - Remote Data Collector

4.1 Intent

The Remote Data Collector collects data from various data sources and sends the data to a Display Component.

4.2 Context

You are developing a data acquisition application. It should treat several different types of data sources. You could not presuppose the number of data sources. And it is possible that a new type of data source could be added into the system.

4.3 Forces

- **Independence of data sources vs. Complexity:** Independence of both the number and the type of data sources can make your system better maintainable and scalable. However, the independency requires a higher complexity.

4.4 Problems

- How to separate the data sources from other parts of the system?

4.5 Solution

Use a Remote Data Collector, which acquires and routes data from various data sources. Figure 5 shows the architecture of the Remote Data Collector.

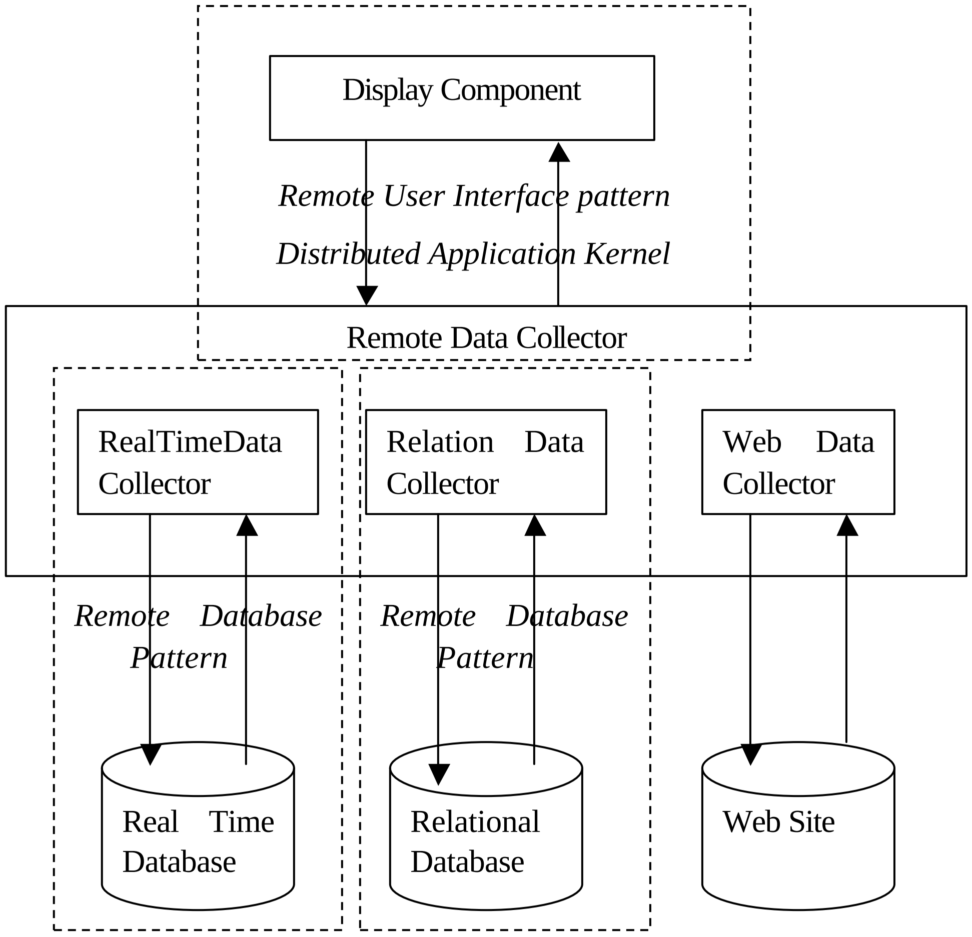

Fig 5: Architecture of Remote Data Collector

For each type of data source, you can develop a component that implements a uniform interface to acquire the data. Typically, there are three kinds of remote data collectors: the Real-Time Data Collector, Relational Data Collector and Web Page Data Collector.

The Real-Time Data Collector retrieves the data from a real-time database (RTDB). Because there are various kinds of RTDBs and no unified API, you have to develop a special Real-Time Data Collector for each RTDB involved.

The Relational Data Collector retrieves the data from a relational database. It can use standard SQL as the protocol. You can build one Relational Data Collector for different types of relational databases. But you must consider the different flavors of the SQL they support. For example, the descriptions of the "date field" in SQL statements of various relational databases are different. You can further build a Relational Data Collector for each type of relational databases. This way, you can use the special API to access the database that will improve the performance of your system.

The Web Page Data Collector retrieves the data from dynamic web pages. You should develop it using the structure of the dynamic web page.

There may be other kinds of data sources, for example, the data from control systems. Special data collectors should be developed for these data sources.

4.6 Structure

You can develop the Remote Data Collector as an interface for various data sources, see figure 6. For each type of data source, you have to develop a component that inherits this interface. The component is deployed in the application server. The data elements in Display Component create a remote object that is an instance of the component on the application server to acquire the data.

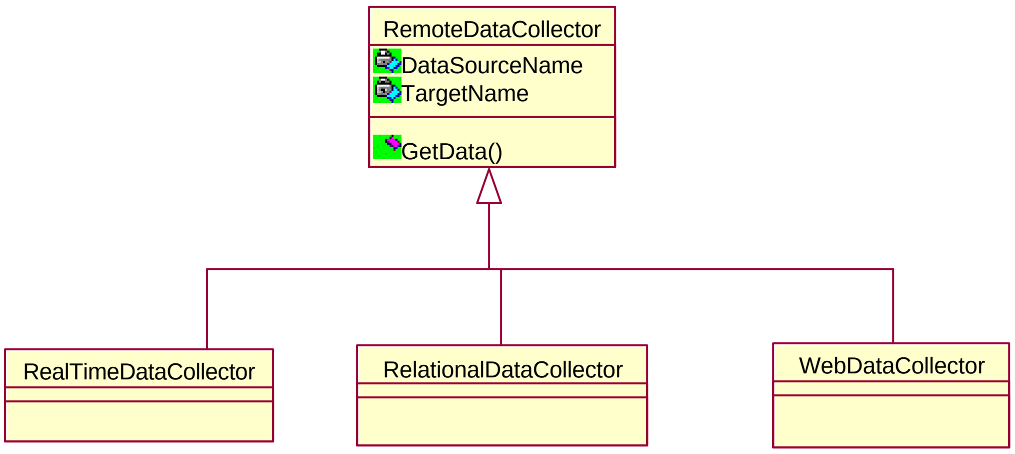

Fig 6: Remote Data Collector Interface

You can also develop the Remote Data Collector as a server. As shown in figure7, the Remote Data Collector Server includes Collectors for every type of data sources. In this case, the Remote Data Collector Server creates the concrete data collectors. It acquires data through these collectors and sends them to Display Component.

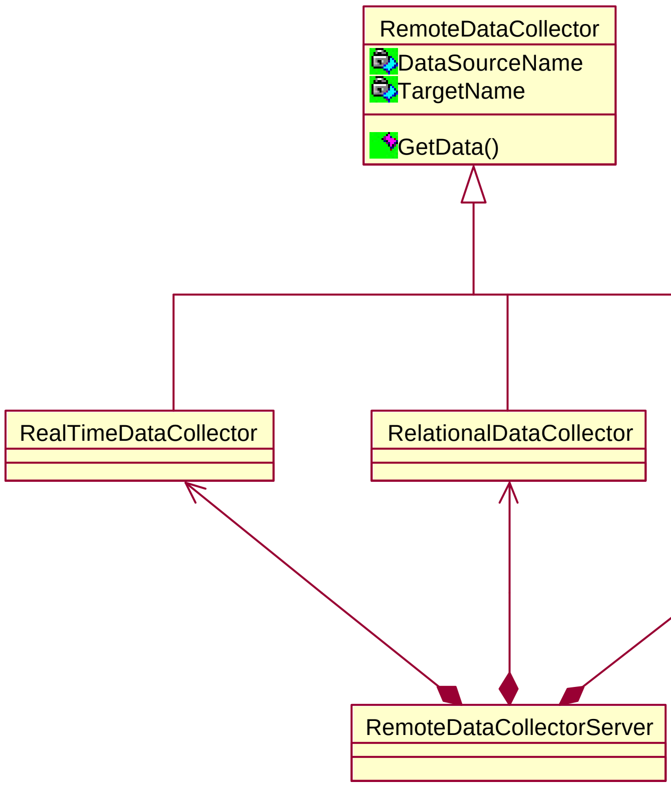

Fig 7: Remote Data Collector Server

4.7 Consequences

This pattern has several benefits:

- There will be less maintainability: the users can define and modify the interface themselves easily.
- The system will be more scaleable: the data sources can be added easily.

However:

- The structure of the Remote Data Collector is complex. And you should consider the data transmission block and exceptions to maintain the data integrity and consistency.
- If there is a new data source, you have to provide a new Data Collector.

4.8 Related Patterns

As shown in figure 5, the relational databases and real time databases are ***Remote Databases*** [RK1997]. It also uses ***Remote User Interface*** and ***Distributed Application***

Kernel [RK1997] as shown in 2.8.

Many design patterns [GHJV95] can be used to implement this pattern. There are:

Facade pattern: Data elements get the data from the Remote Data Collector that separates data elements from data sources.

Adapter pattern: Each kind of Remote Data Collector is an adapter between the Remote Data Collector interface and the concrete data source API.

4.9 Known Uses

1) iFixWebServer of Intellution [Intellution]. iFix Server is the Remote Data Collector of iFixWebServer. It acquires data from supported data sources through iFix interfaces.

2) PI USD of OSI Soft [OSISOFT]. UDS provides open access to all supported data sources. UDS seamlessly connects to PI Data Storage, to non-PI data sources, such as OLEDB providers (e.g., SQL Server or Oracle), to proprietary systems (e.g., PHD and CIM/21), or to any other supported database.

5. Pattern 4 - Configuration Database

5.1 Intent

The Configuration Database stores the configuration data that defines the displays, the elements in the displays and the data sources.

5.2 Context

You are developing a web based multi source data acquisition application. You should find a way to store the configuration data.

5.3 Forces

- **Integration with existing systems:** Most data sources of your application are application systems of your customers. Your application should keep the configuration data integrity and consistency with these existing systems instead of letting your users define them again.
- **Data consistency, security vs. complexity**: Use a relational database to store configuration data can keep the data consistent and make the system more secure. However storing configuration data in files is less costly and less complex.
- **Structure of the configuration data model:** A formalized data model will enforce data integrity and consistency naturally. However, ensuring data consistency is really difficult, if the data has to be integrated from different

systems.

5.4 Problems

- How to store the configuration data?
- What is the structure of the data model to satisfy the requirements of integrating existing systems?

5.5 Solution

To keep the configuration data consistent, use a relational database to store the configuration.

This Configuration Database has two logic parts: the Display Database and the Integration Database. The Display Database is implemented based on a formulized data model to store the configuration data of dynamic displays. The Display Component gets the configuration data of dynamic displays from it. The Integration Database stores the data elements of existing systems, which are the candidates presented on the dynamic displays.

Use separate stored procedures or processes to maintain the configuration data consistency between existing systems and the Integration Database. When the user configure the dynamic displays, they picks the data elements from the Integration Database, define the properties for display and stores them to Display Database by using the Configuration Client. The architecture of the Configuration Database is shown in Figure 9.

If the configuration data of the existing system is stored in a relational database that is of the same type as the Configuration Database, we use stored procedures to maintain the configuration data consistency. Otherwise we have to develop separate processes to fulfill this job.

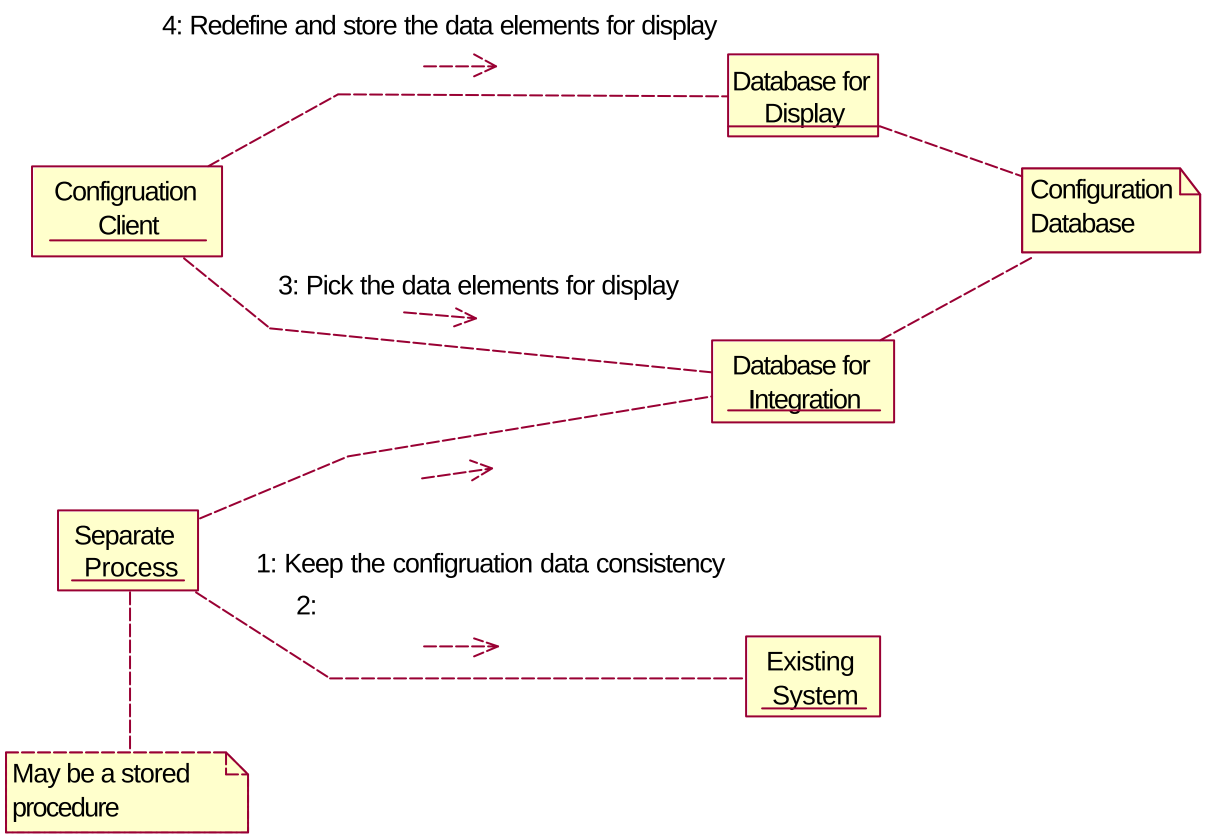
4: Redefine and store the data elements for display
Database for Display
Configruation Client
Configruation Database
3: Pick the data elements for display
Database for Integration
Separate Process
1: Keep the configruation data consistency
2:
Existing System
May be a stored procedure

Fig 9: The Architecture of the Configuration Database

5.6 Structure

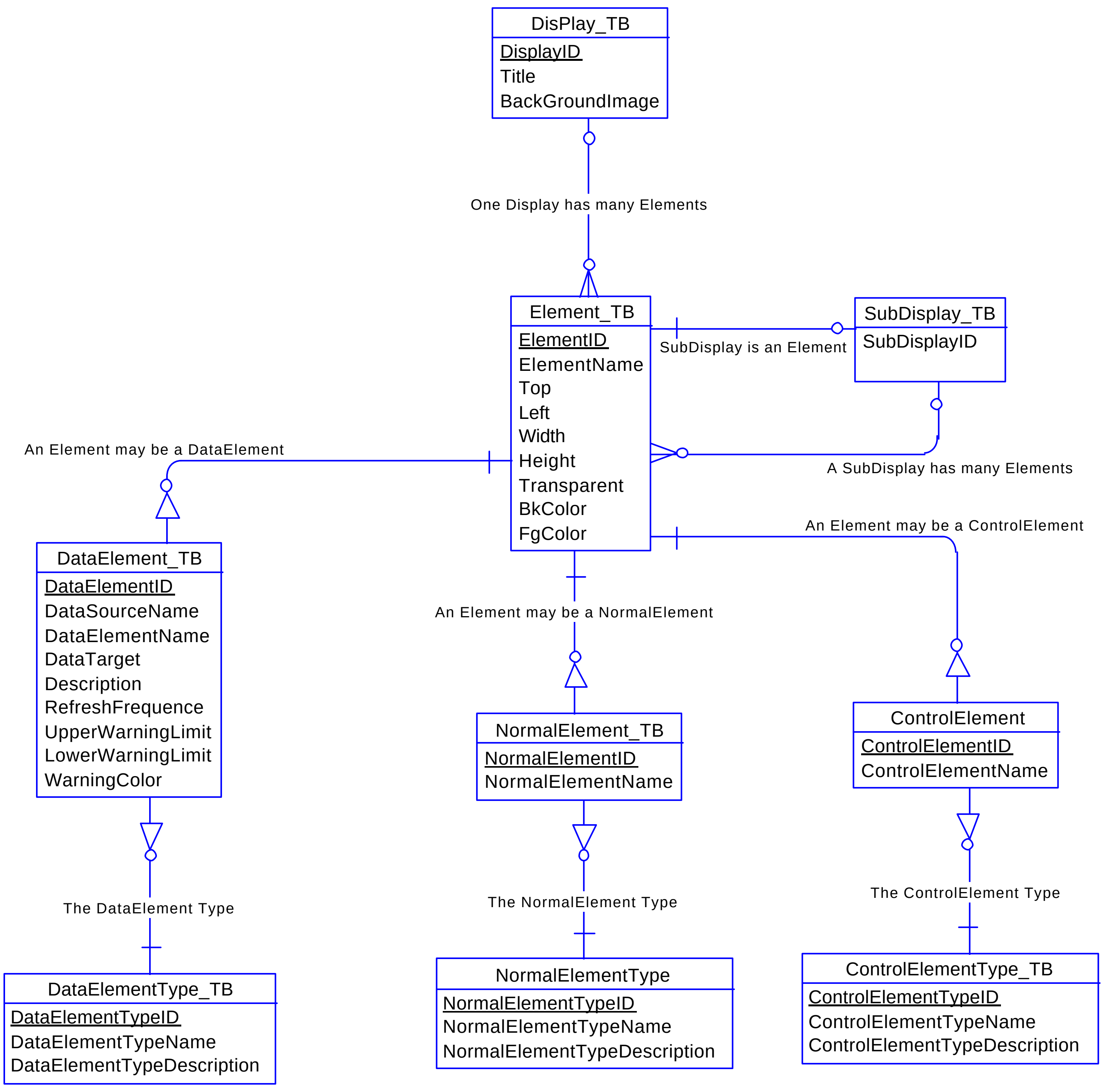

Fig 10: The conceptual data model for display

We use ERA to develop the conceptual data model for display, as in Figure 10.

Because most relational database management systems support long binary fields, the background image of the display can be stored as a field of the display.

The relationship between “Element” and its sub classes (data element, normal element and control element) is a “one to one”, and these sub classes depend on “Element”. You can create a “trigger” in the database to maintain this relationship automatically. The “trigger” is an Observer [GHJV95].

To keep the data model stable, we create a type Entity for each type of Element. So you can add the element’s type by adding records and there is no need to modify the data model.

5.7 Consequences

- Using a relational database to store the configuration data makes the system better maintainable and more robust. However, if the application is small, using a relational database is more costly and complex. Then you can use files to store the configuration data. In this case, each display’s configuration data is stored in a file. You can use the id or name of the display as the file name. The Display Component gets configuration data via ftp or http.
- You can use this pattern to integrate existing systems, but the structure is more complex and you may have to develop separate procedures or processes to keep the configuration data consistent between your application and existing systems.

5.8 Related Patterns

The pattern ***Connecting Business Objects to Relational Databases*** [YJW1998] can help you to connect the Object Oriented design model with the relational data model.

The Configuration Database is a ***Remote Database*** [RK1997] accessed by Display Component and Configuration Client. If the Configuration Database and the database of existing system that stores configuration data are same kind, they constitute a ***Distributed Database*** [RK1997].

The triggers that maintain data consistent are ***Observer Pattern*** [GHJV95].

5.9 Known Uses

1) iFixWebServer of Intellution [Intellution]. The configuration data of iFixWebServer are stored in the iFix process database.

2) PI Interactive Configurable Environment (PI ICE) of OSI Soft [OSISOFT]. PI ICE stores personalization and component information within a Microsoft SQL Server 2000 database.

6. Summary

There are more and more patterns about web based applications, such as [RIC2001] and [PK1999]. Because of the maintainability and usability, the web-based applications are widely used in both Internet and intranet. More and more process industry companies want to integrate the data acquisition systems into their ERP systems. The data integration can transform data into information and knowledge. This pattern language in process is based on our own experience and studies of many systems of different companies. Besides the known uses presented in this paper, there are more systems related to these patterns, e.g. Web 2.1 of AspenTech [AspenTech] [ZHA2001] and the VisualPHD of Honeywell [Honeywell] [ZHA2001].

Developers can easily apply this pattern language. Each pattern in this language can be implemented by various technologies, based on the scope and size of your application.

7. Acknowledgments

Special thanks go to our PLoP shepherd Jutta Eckstein for her insightful and valuable comments during the revision of this paper.

8. References

[ZHA2001] Zhilin Zhang. The Principle and Application of Real Time Database Management System: published by Sinopec Press, China, 2001
[RK1997] Klaus Renzel, Wolfgang Keller. Client/Server Architectures for Business Information Systems, PloP'1997.
[GHJV95] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-oriented Software. Reading, MA: Addison-Wesley, 1995.
[RIC2001] Chris Richardson. A Pattern Language for J2EE Web Component development: PloP'2001.

[YJW1998] Joseph W. Yoder, Ralph E. Johnson, Quince D. Wilson. Connecting Business Objects to Relational Databases: PloP'98

[PK1999] Kimberly Perzel, David Kane. Usability Patterns for Applications on the World Wide Web: PloP'99

[Intellution] http://www.intellution.com/products/fixwebserver/default.asp

[Honeywell] http://www.honeywell.com

[AspenTech] http://www.aspentech.com

[OSISoft] http://www.osisoft.com/