BIBSYS System Architecture

Jan Erik Kofoed <Jan.Kofoed@bibsys.no>

Revision History
Revision 1.0 2004-06-03

Table of Contents

1. Introduction

Architecture can be defined as *The art and technique of designing and building, as distinguished from the skills associated with construction.* [1]

The importance of analysis and design is always stressed in computing literature. Is it possible to identify some common principles that apply to the system architecture of most library systems?

This article is based upon a project specifying a new system architecture for BIBSYS Library Automation. Many of the principles that hold for BIBSYS may also apply for most library systems or large database software systems in general.

This document is written in XML using DocBook. [2] Versions formatted in XHTML, PDF or PostScript may be produced from the manuscript source in XML.

2. Models

The central part of the system architecture is the model: The main principle for how the system is designed.

1 Encyclopædia Britannica. http://www.britannica.com/
2 http://docbook.org/

2.1. Components

A system can be described as assembled of components. A *component* is a set of data structures and procedures that it is natural to split out as a part of the system. The purpose of defining components is to achieve simplicity both inside the component itself and in its interface to other components. A component should be well defined: It shall be easy to find out how the component works, what functions it can do and which data structures it handles.

The components must both be independent of each other and be able to cooperate. They should be reusable and easy to substitute when needed. This is more easy to achieve when components are organized in separate layers or tiers. A *tier* contain components with similar or related functions. Each tier has an interface to the other tiers that makes communication and interoperability possible between the tiers.

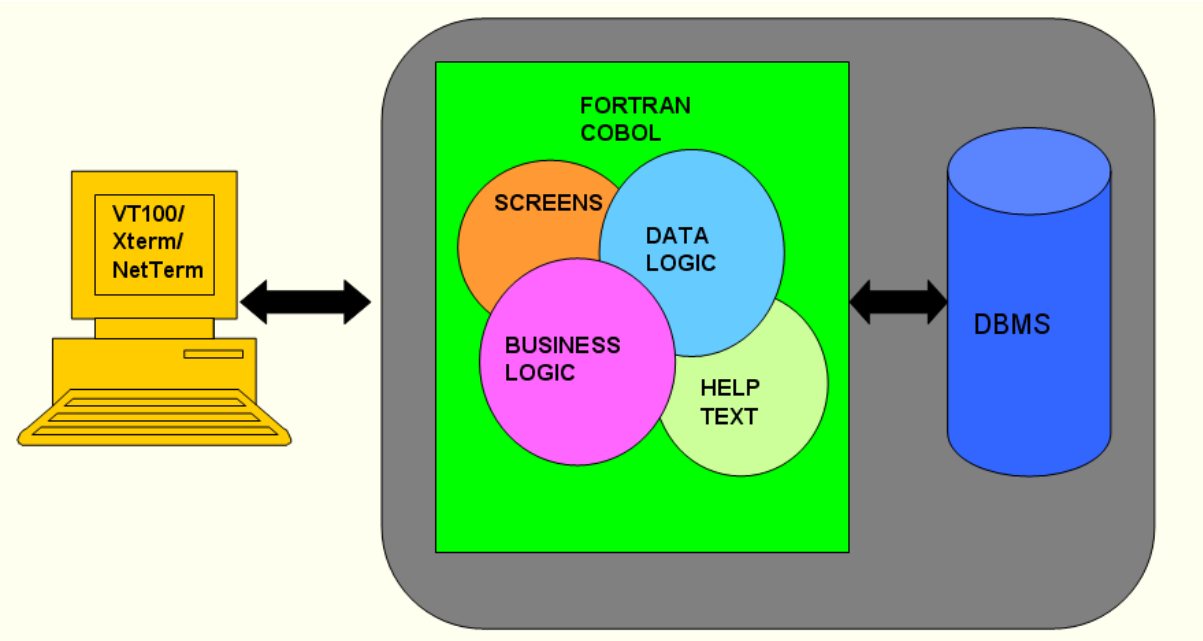
Some examples of components may be:

XML to ISO-2901 converter for MARC records.
MARC to Dublin Core translator.
ISBN/ISSN syntax checker.
Builder of FRBR relations from MARC records.
Access control.

2.2. Single tier model

Old systems based on access via terminals was built in one tier as a monolithic system.

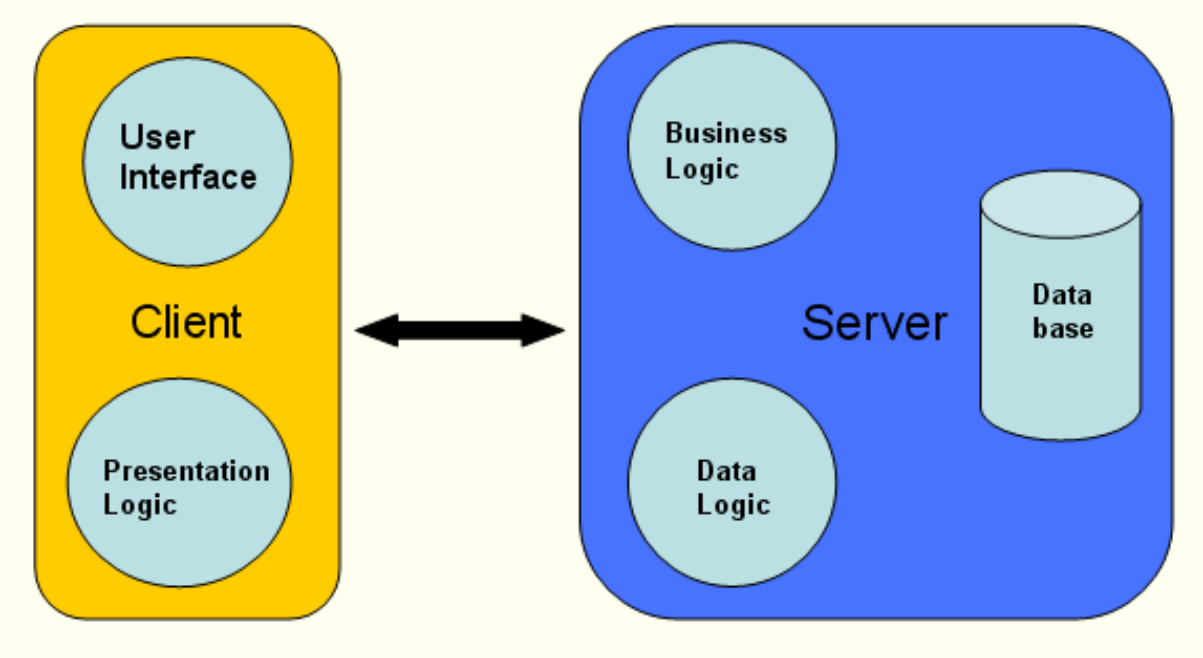
Figure 1. Terminal based applications



2.3. Two tier model (Client Server model)

On the approach of the Internet and other networks a new concept was developed: *The client server model*. It can be viewed as a two tier model where data handling and most logic are handled at the server side while presentation is done at the client. Typical clients was generalized and standardized software such as web-browsers, gopher-clients, wais-clients a.o.

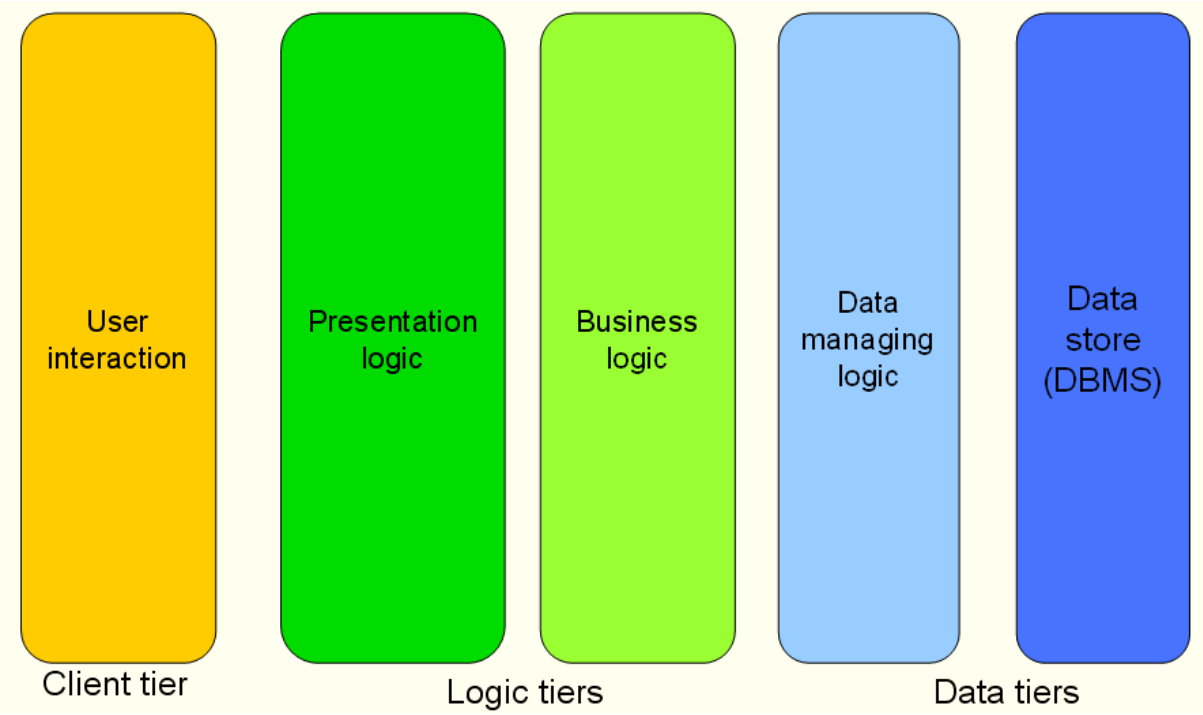
Figure 2. Two tier model (client server model)



2.4. Multiple tier model

The two tier model can be developed in several steps adding more layers. A three layer model has a middle layer containing common logic for both business and data handling. A model with five tiers is locating presentation logic, business logic and data logic in different layers.

Figure 3. Multiple tier model

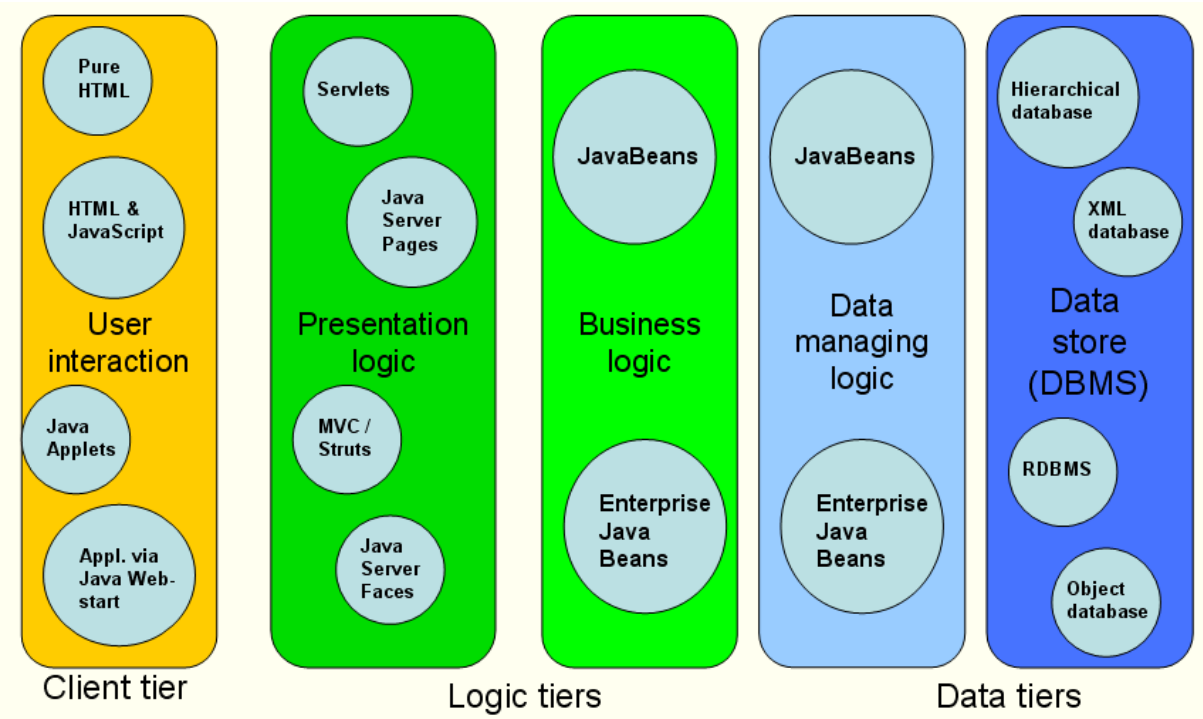


2.5. Placing components inside the tiers

Components are located inside the tiers.

It is possible to standardize on technology inside the tiers. This is based on the fact that some technologies are more suited for certain tasks than others. BIBSYS has chosen Java as technology platform for new applications. This opens up for use of *Open Source components* and ability to utilize standards and modern techniques. The figure below shows which Java technology that will fit into each tier.

Figure 4. Tiers with components using Java technologies



3. Frameworks

A framework can be described as a collection of *methods*, *tools* and *rules*. The methods can be specific algorithms or task descriptions. The tools may be some type of building blocks, technologies and libraries with software routines. The rules are ensuring that everything is interacting correctly with each other.

3.1. Jakarta Struts

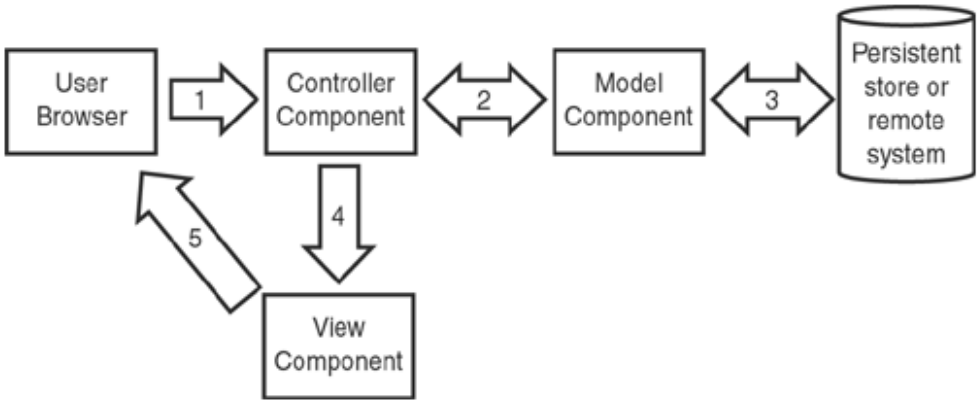
The concept of framework can be well illustrated with the Jakarta Struts [3] framework. Struts is an implementation in Java of the methods, principles and patterns called *MVC (Model View Controller).*

MVC is a method for dividing presentation logic from business logic. The *model* consist of data stores, data logic and business logic. The model maintains the state of the system. The *view* is the user interactions (screens, printouts, sound, keyboard input) and presentation logic. The *controller* is responsible for the flow of data and control between the model and the view.

MVC was invented of the Norwegian professor Trygve Reenskaug i 1979 when he visited the Smalltalk group at Xerox PARC. Reenskaug's idea was to separate presentation from information.

Figure 5. Model - View - Controller

3 http://jakarta.apache.org/struts/



1. The client contacts the application.
2. Controller receives the request and decides how it shall be handled. The decisions are taken based on rules built into the controller.
3. The model manages data and communication with other systems. The state of the system is updated and a message is given to the view.
4. The controller decides which part of the view that must be updated and sends control to the view.
5. A updated view is sent back to the user.

3.2. Access control

Although providers of library services tries to keep their system open for free access to the public, there is some need for access control. Some services are based on paid subscription and others are limited to certain user categories. The services may be free to access, but the user needs to be authenticated in order to give her personalized service.

Access control is often divided into three parts: *authentication*, *authorization* and *accounting* (AAA).

Authentication means secure identification of a user. Most common is username and password. Some publishers allow shared access based on IP address or domain name of the user's computer. Today there exist quite sophisticated methods based on PKI (Public Key Infrastructure), certificates and smart cards.

Authorization describes the rights given to an (authenticated) user. A right can be access to a given resource, often differentiated in write or read access where appropriate. Users are often organized into groups. Authorization can use these groups assigning rights to roles instead of individual users. User management can be much easier that way.

Accounting is often handled together with access control since it is often applied to individual users or group of users. Accounting can also be useful even when there is no payment involved. Accounting systems can be used for collecting valuable statistics on use of the systems.

3.2.1. The shared access system FEIDE

FEIDE [4] (Federated Electronic Identity for Education) is a Norwegian project in the higher education (HE) sector designed to allow identification of users based on a distributed system of user databases and a shared authentication mechanism.

BIBSYS is committed to provide FEIDE as a method for authentication for it's users.

3.2.2. Single Signon (SSO)

Single Signon (SSO) is a method to share access information between heterogeneous applications. A user often needs to jump between many different sources and applications in order to collect information. It is rather

4 http://www.feide.no/index.en.html

tedious to log in and out several times and keep track of all passwords needed. SSO minimizes all those logons. If the applications trust each other it is possible to login once and share the access information.

The Shibboleth project [5] can be used in a SSO solution:

> *Shibboleth, a project of Internet2/MACE, is developing architectures, policy structures, practical technologies, and an open source implementation to support inter-institutional sharing of web resources subject to access controls. In addition, Shibboleth will develop a policy framework that will allow inter-operation within the higher education community.*

Shibboleth is based on trust between peers, and is mostly spread in the higher education domain.

SSO is appropriate when the systems have about equal demand for security, and when the security level needed is not too high. SSO may be a good solution for library applications when sharing access to reference databases and full text resources.

3.3. Identificators

Identificators have come quite necessary, especially with the origin of digital documents. An identificator is a unique, permanent and unambiguous label which specifies an entity.

Printed documents are using standard numbers such as ISBN and ISSN. They can also be ordered for digital documents, both the large number of digital documents require a more easy and flexible system.

Many systems have been defined for identificators. Several origin from the publisher domain. Some examples are SICI (Serial Item and Contribution Identifier), BICI (Book Items and Contributions Identifier) and DOI (Digital Object Identifier).

3.3.1. URN

IETF (The Internet Engineering Task Force) has defined an identifier called URN (Uniform Resource name) especially constructed to function on internet. URN has these functional requirements defined in RFC 1737 [6] :

- Global scope
- Global uniqueness
- Persistence
- Scalability
- Legacy support
- Extensibility
- Independence
- Resolution

All URNs have the following syntax: `URN:<NID>:<NSS>` where <NID> is called the Namespace Identifier, and <NSS> is called the Namespace Specific String.

Of special interest is the NID *NBN* (National Bibliographic Number). It is defined in RFC 3188 [7]. The NSS is prefixed with ISO-3166 country codes to ensure that they are unique between countries. A Norwegian URN may have the following syntax: `URN:NBN:no-2420` .

URN and other identificators should be backed up of a resolver service. An identifier is only a label for a resource. In order to locate items a resolver service should be established. In the case of electronic documents accessible on the internet the resolver service should map the URN to a URL. The National Library of Norway has a such service with the syntax `http://urn.nb.no/<urn>` e.g. http://urn.nb.no/URN:NBN:no-6883

3.4. Metadata

Metadata is rather the core of a library system than a framework. A library system has one or more central

5 http://shibboleth.internet2.edu/index.html
6 http://www.ietf.org/rfc/rfc1737.txt
7 http://www.ietf.org/rfc/rfc3188.txt

databases containing metadata. The content is organized according to a data model which may or may not involve systems for metadata.

Metadata may emerge as result of queries and with exchange of data, regardless of how data actually is stored inside the databases. The well known MARC system is an exchange format, not really a format for cataloguing, although most library systems also catalogues directly in MARC syntax.

Library systems should support several metadata systems. The data model used for storing data should be rich enough to allow exchange of data based on many metadata systems: MARC, Dublin Core a.o. Exchange of data can in this context mean presentation to the user, export to the users personal reference manager, export to library systems, import form other systems, cataloguing records directly or by copying data from other sources.

3.5. Multilingual support

Modern library systems must have support for many languages. The system must be constructed in such a way that it is easy to add a new language and easy to edit phrases in different languages.

A prerequisite for multilingual support is to use a universal character set. Practically there is only one alternative: UNICODE. [8] UNICODE makes it possible to write text in literally all known languages both alive and historical. Modern software are also supporting UNICODE so it is not a problem any more to use the vast amount of characters defined in UNICODE. A limitation is access to fonts, but this constraint is becoming smaller and smaller as time pass by.

Building multilingual systems requires a structured approach. Not a single letter may be hardcoded into the program code. All text must be written into special structures (e.g. Resource Bundles in Java) and accessed using special methods. If one just follows the rules, making multilingual systems becomes both safe and easy.

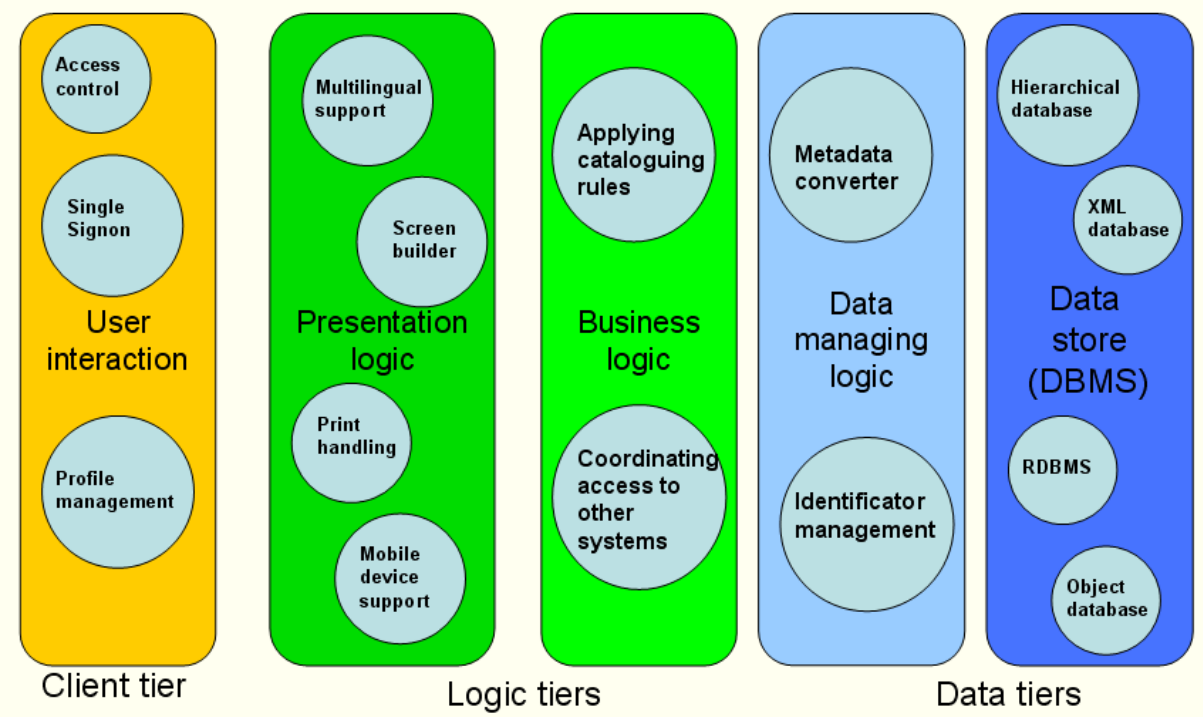
3.6. Model, framework and components

We may now try to place components inside the multi tiered model based on type of framework. Components made from same type of framework may be placed into different layers based on its function. The important issue is to make the components small, well defined and replaceable.

In the figure below is indicated use of framework related to tier. Components may be specified solely on function, and may also exist without any relation to framework.

Figure 6. Tiers with components made from different frameworks

8 http://www.unicode.org/



4. Conclusion

A good architecture should be simple and valid also over a time span. It should contain a model which allows replaceable components. When requirements change it must be possible to replace components without changing the basic model. The components defined must then be small with well defined interfaces. Also the interface between the tiers in the model must be precisely specified.

BIBSYS has chosen technologies based on Java. This choice ensure the possibility to add components available from open sources. Java also contains technologies that is scalable and well suited for building multilingual enterprise systems.