UDC 681.3

A. S. BORANBAYEV

REFERENCE ARCHITECTURE FOR WEB APPLICATIONS

(Представлена академиком НАН РК М. О. Отельбаевым)

This article presents the reference architecture for building web based applications. This reference architecture can be used as the basis when designing a particular information system.

1. Overview

This article presents the reference architecture for web based application. The reference architecture should be used as the basis when designing a parti-cular system. It is assumed that architect can select from a set of well-known elements (standard parts) and use them in ways appropriate to the desired system.

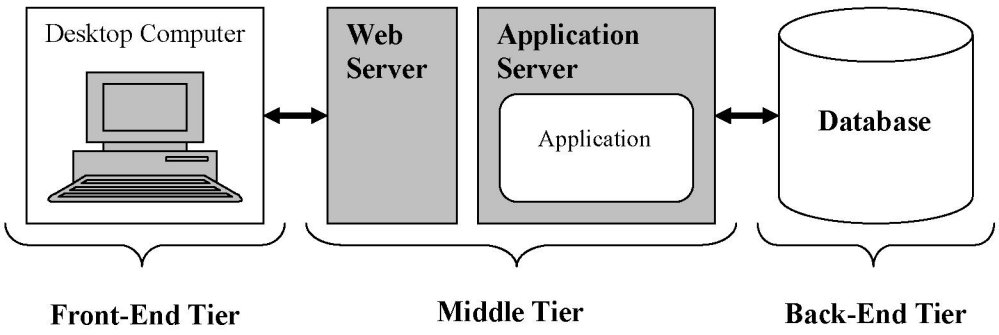
The economy and technology of today have intensified the needs for more efficient solutions of information management. If a company wants to build an e-business Website using a multi-tiered architecture, it should consider using Java 2 Enterprise Edition (J2EE) technologies. The J2EE specification provides a programming model that improves development productivity, and standardizes the platform for hosting enterprise applications.

Java was initially used for client-side development in the 1990s. Then in the late 1990s, it evolved and started to be used as a middle tier tool. Over time, Sun Microsystems added components to its middle tier, such as Enterprise JavaBeans (EJB) and Servlets. In 1999, Sun Microsystems released its J2EE specification which provided a full view of the Java middle tier solution. Since then, Java solution providers have embraced J2EE, which we will now explore [1].

Depending on the project requirements, web based applications come in two flavors - reporting or transactional. Transactional application performs a lot of business functions against incoming and outgoing data when managing transactions and controlling its state. Reporting application, on the other hand, does not apply extensive business logic to the tunneled data, its main task is to retrieve and to present big data sets to the end-user in the form of tables or lists of records. There are situations when transactional and reporting functionality need to be combined in the same web application.

2. Tiers

For both types, application functionality is distributed over three locations: user machines, the application server machine, and the database or any other resources at the back end.



For the front-end tier, there is a browser running on the desktop computer, which renders such interactive components as HTML or DHTML, with embedded scripting elements (JavaScript in most cases).

The back-end tier is constituted from one or several relational databases (such as Oracle) and/or multiple Tuxedo services (C based transaction processing system).

Application Server(s) presents the middle-tier. At this tier, the web container of WebSphere Application Server provides runtime environment for Java-based applications. The middle-tier is where the most of the architectural expertise and effort is usually applied.

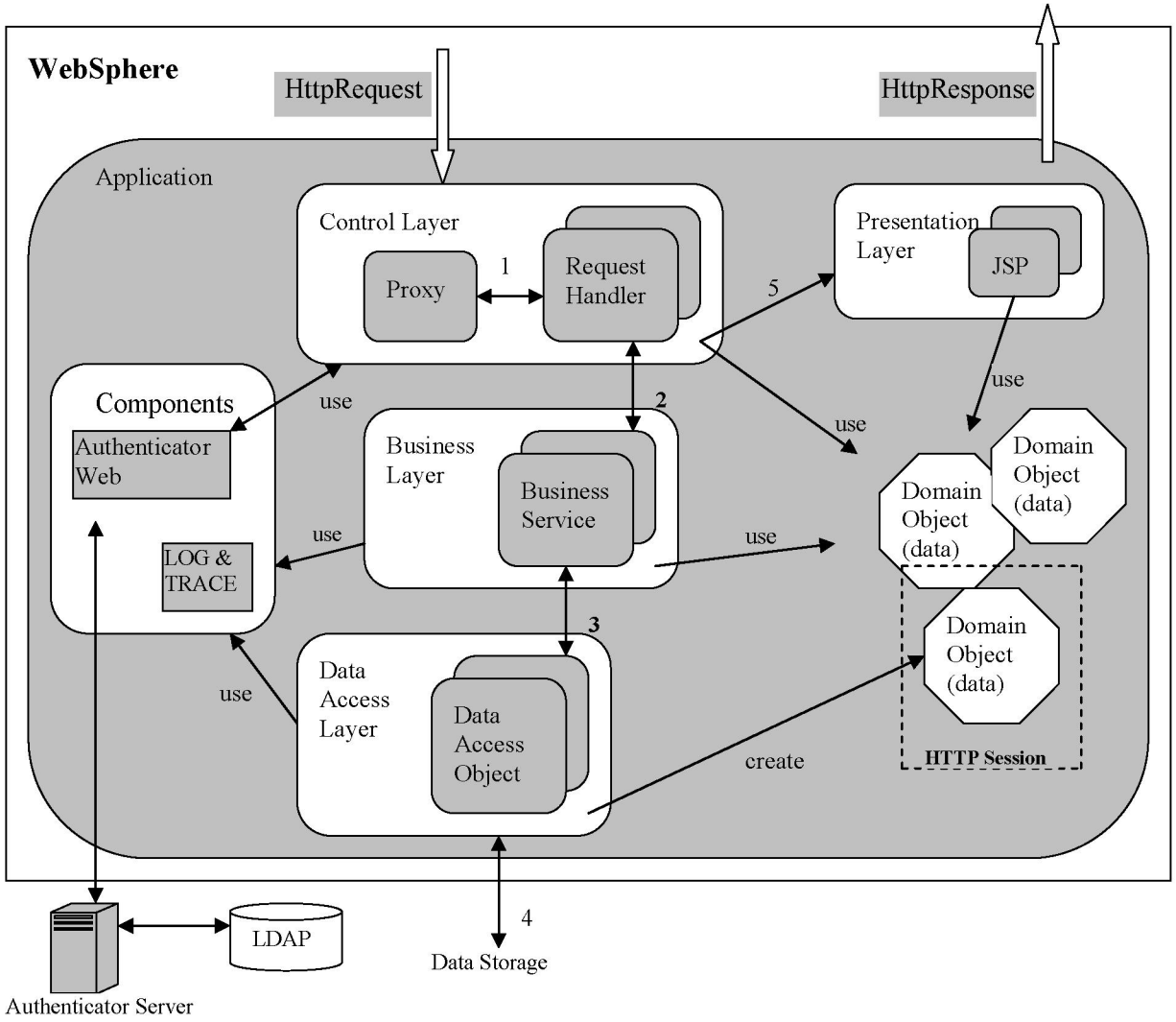
3. Transactional Application

The following diagram presents a typical architecture of a transactional application, its layers, components, and services. The segment of the diagram named «Application» portrays the web container JVM physical address space. Object instances on each layer, as well as across layers interact by executing methods on other object instances through their references. The outer communication utilizes network protocols:

Application to Desktop browser - HTTP

Application to Authentication system - HTTP

Application to Data Storage / Back End Systems (JDBC or HTTP)



The numbers 1 through 5 illustrate sequence of steps that are executed by server thread when processing user request:

1. Based on the HTTP request (URL's extra path information) framework proxy automatically routes the incoming request to the appropriate request handler through doRequest() method.

2. Request Handler disassembles HTTP request data into any of the following: java object(s), java primitives, collections, xml documents, etc., and invokes method on the business object applying received data as method arguments.

3. Business layer performs any business-related logic if necessary, but in most cases for incoming data the control is immediately transferred to the data access layer for data retrieval. Business logic will be applied to the retrieved data later - when execution flow returns back to the business layer after the data access layer creates all required data objects.

4. The data access layer retrieves data from the data storage and creates data objects in accordance with the use case requirements

5. After current thread returns back to the control layer, request handler forwards (not redirects) execution to the presentation layer for the final processing.

Presentation layer builds HTML page and commits the HTTP Response to the user. Domain objects sometime are stored in HTTP session for future reference - for example for paging.

NOTE: Developers sometime abuse HTTP Session usage by storing container managed objects, such as HTTP Request for example, there. We must be aware that according to the servlet specification [3]:

«Each request object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method. Container might recycle request objects in order to avoid the performance overhead of request object creation…»

Developers must be aware that maintaining references to request objects outside the scope described above may lead to the non-deterministic behavior of JVM.

4. Layers

4.1. Control Layer

The Control Layer translates user requests into actions to be performed by the application.

The framework provides major control for this layer through the means of MyProxy and MyRequestHandler java classes. Application specific proxy and request handler classes must extend from them. The proxy acts as a controller, responsible for delegating the requests to the appropriate request handlers. It also initializes (once, at start-up) and manages all available services and resources for the application. Framework logic around MyProxy and MyRequestHandler also interacts with AuthenticatorWEB component on behalf of the application for authentication and authorization purposes.

NOTE: Each specific application action (view, create, submit, delete, calculate etc.,) should have one corresponding request handler on the server side. This approach helps to define fine grained control structure for the application, which is easy to understand, manage and support.

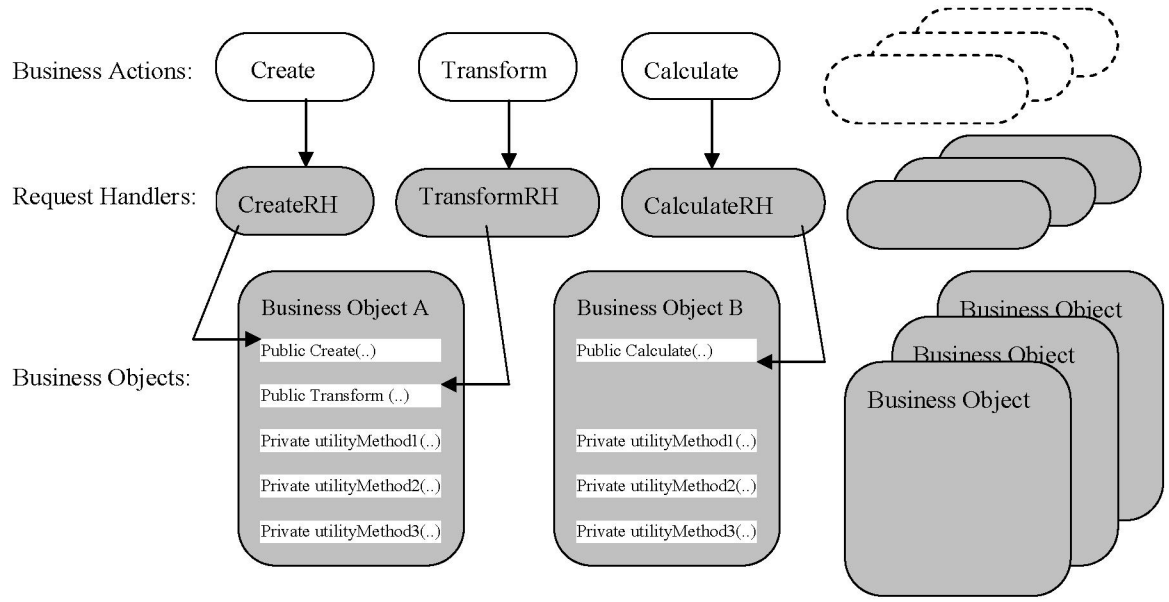
4.2. Business Layer

Business Layer serves the purpose of encapsulating all necessary business logic that needs to be performed against incoming and out-going application data, whatever form it might take - primitive Java types, Domain Value Objects, Collections, etc. [2]

Each business service on that layer is supposed to contain only objects that are logically related to each other - either because they are consistent with business process flow (use case) or they operate on the same set of data.

NOTE: A Business service, either presented as one single object or collection of objects, executes business logic - it is a processing unit! - not a Java Bean with numerous get..() and set...() methods.

Here is the visual representation of the relationship between the application's action, request handler, and business service methods («Business Object» means «Business Service»):



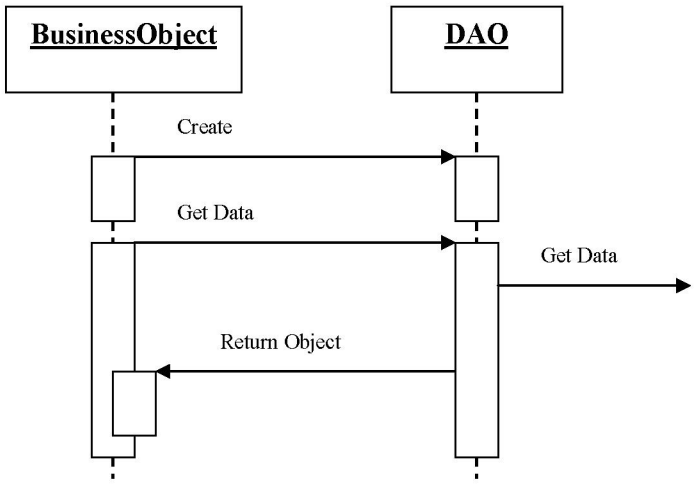
The rule of thumb here is - for each request handler to execute only one method on the business service. This method might be seen as an entry point into carefully designed and tuned process that serves a particular business purpose (use case). This way the business service (primary objects and utility objects associated with it) become a course grained processing entity that easily can be converted to stateless EJB's or a Web Service if needed [4]. This approach also provides for complete de-coupling between control and business layers of the application.

NOTE: There are cases when a request handler might execute several methods on the business service sequentially. Those kinds of situations are usually related to the application flow control (not a business flow). For Example, the request handler might require making sequential method calls involving one or several business services when handling some exceptional condition.

4.3. Data Access Layer

The data access layer separates application business logic from system data storage's [2]. The Data Access Object (DAO) is the primary object of this layer. The data access object abstracts the underlying data access implementation for the business layer to enable transparent access to the data resource.

The data resource could be a persistent storage like an RDBMS, or a repository like an LDAP database. The DAO completely hides the data access implementation details. Because the interface exposed by the DAO is not supposed to change when the underlying data resource implementation changes, this allows the DAO to adapt to different storage schemes without affecting application business layer. Essentially, the DAO acts as an adapter between the business logic and the data resource.

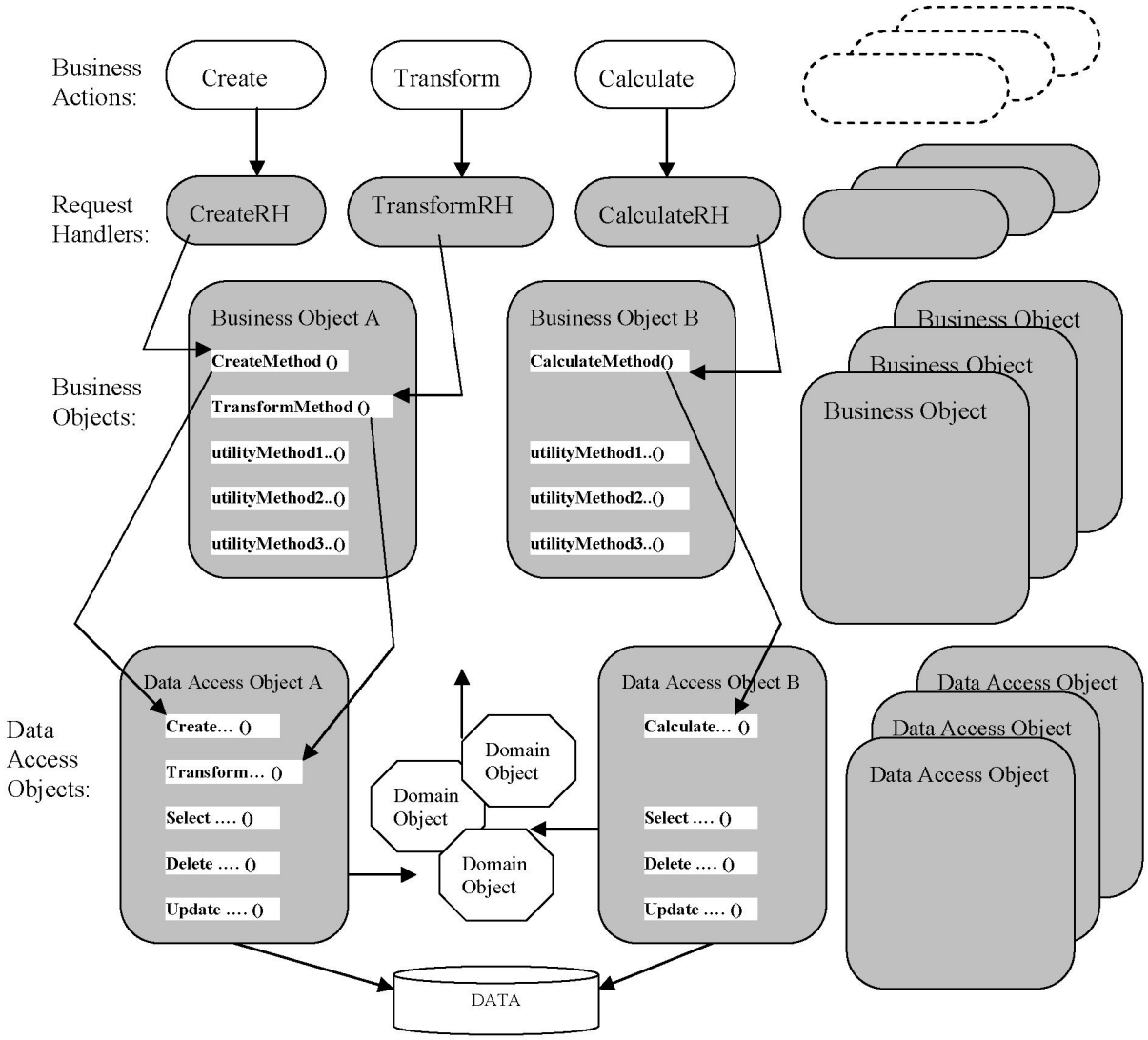


The functionality (set of methods) for DAO should be centered on business domain entities. It means, for example, that for such business entity as «Order» all DB operations (object methods) for updating, selecting, or deleting order related data should be placed into one DAO (one java class).

In some cases, DAO will return such business domain entity (in the form of Java Bean or Value Object) to the business layer for further processing. In other cases, it might return only the final condition of the performed operation (success or failure).

There are several frameworks available for usage on the data access layer. When application data resource is RDBMS - the Database Access Framework provides seamless database connection pooling and simplified set of functions for retrieving connections, executing queries, and working with the query results set. When there is a need to persist data between several requests - the Persistence Framework provides the storage of objects in any JDBC or JNDI compliant databases. It helps in relating the object model to the data model through the set of simple Java API's.

Here is the visual representation of the relationship between application's action, request handler, business object, and data access object.



NOTE: Normally an application is able to manage the transaction as long as it references a JDBC connection object that originally started that transaction. There are numerous situations when transactional boundaries must be extended beyond one DB connection or the transaction must be maintained across multiple methods (or even across multiple DAO's).

4.4. Presentation Layer

Presentation Layer handles the layout and formatting of the data before it is sent to the front-end tier. It relieves the other application layers of any concerns regarding data represented to the end-user.

The presentation layer contains set of JSP's. Usually one JSP corresponds to one HTML page displayed by the browser. JSP obtains all necessary data from Domain Objects that had been previously created by the data access layer and processed by business layer [2].

The presentation layer does its work after the control layer of the application transfers the execution flow to it.

4.5. Business Domain Entities (Domain Objects)

Before discussing the concept of Business Domain Entities I would like to spend some time describing two fundamental approaches that are often applied when designing web applications. In IT industry, they are commonly referred to as top-down and bottom-up [5].

The top-down approach assumes that the very first thing that happens during application design is the design of graphical user interface. The GUI prototype (screen mockups) is the outcome of analyzing business requirements - during project development lifecycle. From that moment all other decisions that application designers are making are driven by the GUI, in other words application design is «screen driven». The objects of the control layer are designed to take care of any action issued by GUI. The business layer usually contains objects that encapsulate business functionality related to the particular screen. Data access layer also serves the purpose to deliver screen specific data to the layer above it. This means that data objects (Java Bean, Value Object, etc.) created by data access layer, processed by business layer, and used by presentation layer contain very specific sets of data required by particular screen.

When application is designed this way, logically there is a firm, very solid vertical axis that functionally not only ties together all layers but also expects that data transferred through the layers will be strictly limited to the subset required by the screen. This is the way majority of consulting teams usually approach application design process. They target their design towards satisfying «screen GUI». One screen, one use case, one logical axis.

The other way to approach application design is through detailed analysis of business domain. Understanding the nature of the business operations, functions, and most importantly data that support those functions might lead to the bottom-up design approach. This approach differs from top-down in the way that screens are not central to the application design anymore. Instead, data objects created by data access layer, become the main point of attention during design process. They create the model of application business domain - business domain entities (Order, User, Item, etc). One such object might be used across several use cases (accessed by multiple business objects) satisfying multiple screens GUI. This will provide much more flexible layered application, allowing for clear separation between layers and effective future reuse. From the architects and designers this approach requires the excellent understanding of the nature of the business, its required operations, functions, and existing or proposed database schema. In order to apply this approach successfully, the architect must be involved into the project at the earlier stages in the project lifecycle. The extensive exchange of the information between business domain experts and project architect must precede the design process.

The top-down approach is very often used in projects. There is nothing fundamentally wrong with this. Just keep in mind that there will be the situations when you might consider that designing business domain entities is the better choice for achieving ultimate goal of reuse and flexibility.

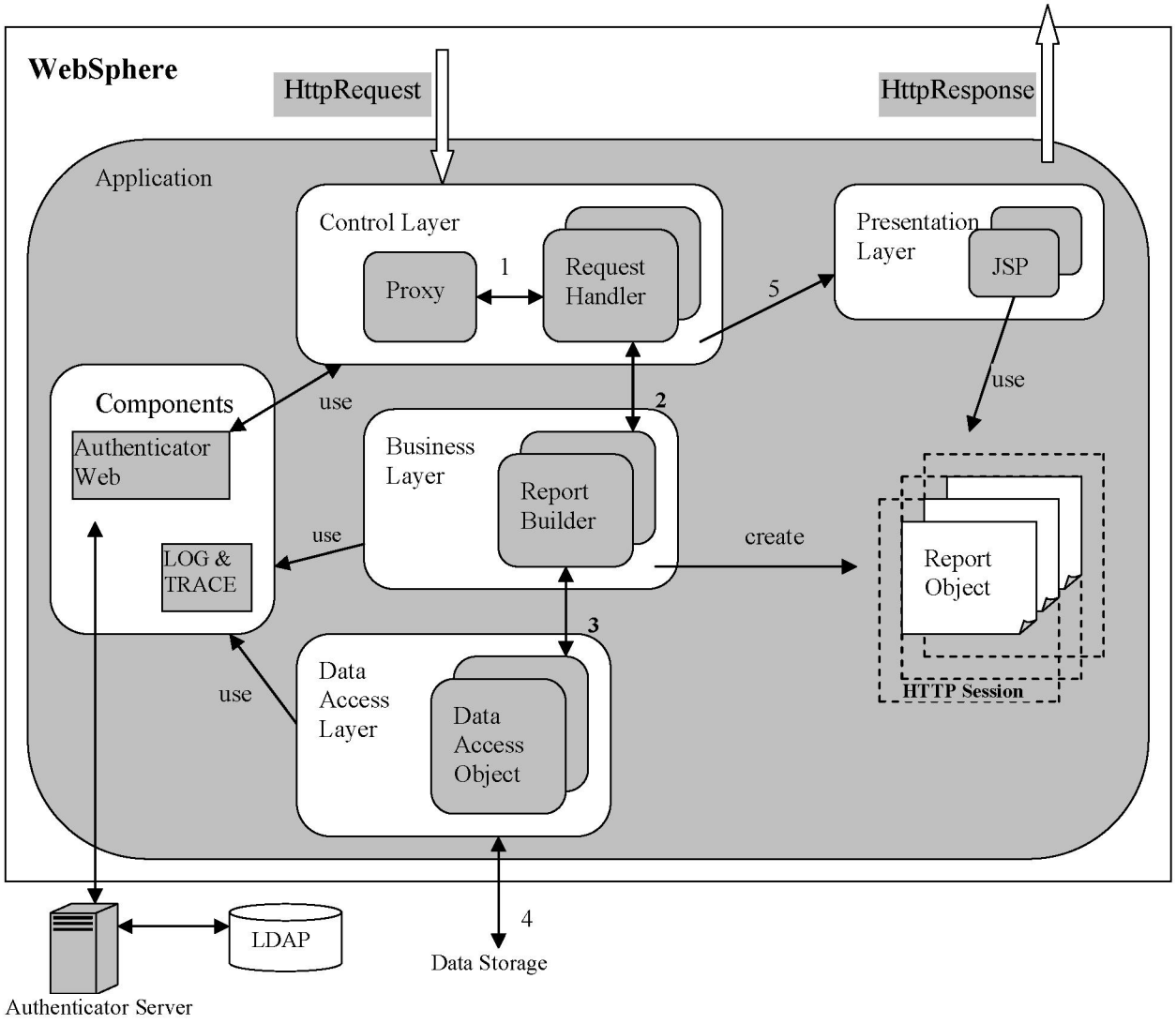
5. Components

Two, most commonly used components are Authenticator Web and Log&Trace. Authenticator Web can be used by framework on behalf of the application without any involvement from the development team. Authenticator Web provides authentication and authorization functionality by accessing the Authentication service. Log&Trace component is also part of the typical architecture solution.

Provided by the Authentication System the authentication service is commonly used by many applications.

6. Reporting Application

Most of what has been mentioned in section 3 about layers, components, and services can be successfully applied for reporting type applications as well. At the same time a reporting application differs from a transactional application in two major aspects; first - it works with big (sometime huge) data sets, second - it does not apply extensive business logic when processing those data sets.



In the reporting application there is no need for creating domain entities, because there is no business logic that needs to operate on those entities. The functionality that is applied to the retrieved data is usually limited to the structuring, formatting, and sorting tasks. This leads us to the idea of storing retrieved data in the form of columns and records in plain data structures, such as arrays and collections. The absence of sophisticated business logic usually restricts the business layer to the set of simple responsibilities of assembling and arranging retrieved data, so it can be easily embedded into HTML pages by the presentation layer.

7. Conclusion

In this paper, I provided an overview of J2EE architecture at a high level. I have addressed important architecture topics, technologies and development steps that one should consider in a J2EE project. The information is taken from real-world experiences, and is intended to help developers build J2EE systems. This, however, is just the tip of the iceberg, as no short paper could begin to do justice to J2EE's potential impact on an enterprise application.

I focused my attention on the technologies that people are most likely to encounter when working with J2EE: Servlets, EJBs, and JSPs. Whether you are a developer, business analyst, or project manager, you now should have a good idea of what J2EE has to offer you and your enterprise applications.

In a world of constantly changing technology requirements, J2EE is the way to enable the Enterprise to integrate the latest technology standards while still leveraging existing IT investment.

Nowadays software developers need to be able to develop truly scalable and distributed enterprise server applications. The ultimate goal is to deliver a complete J2EE solution that satisfies our customer's requirements.

REFERENCES

1. Java Platform, Enterprise Edition from Wikipedia: http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
2. Sun Developers Network (SDN) J2EE 1.4 Specification: http://java.sun.com/javaee/index.jsp
3. Sun Microsystems Servlet specification: http://java.sun.com/products/servlet/
4. Sun Microsystems EJB specification: http://java.sun.com/products/ejb/docs.html
5. Top-down and bottom-up design, from Wikipedia: http://en.wikipedia.org/wiki/Top-down#Software_development.

Резюме

Web-қолданбалы бағдарламалар жасайтын сілтемелі архитектура ұсынылған. Осы сілтемелі архитектураны ақпаратты жүйелерді жобалаудың негізі түрінде қолдануға болады.

Резюме

Предлагается архитектура-ссылка, позволяющая строить Web-приложения. Эта архитектура-ссылка может использоваться как основа для проектирования информационной системы.

L. N. Gumilyov Eurasian national universit *Поступила 2.07.07г.*