Towards a Component-based n-Tier C/S-Architecture

Hans-Peter Steiert
Department of Computer Science, University of Kaiserslautern, P. O. Box 3049
D-67653 Kaiserslautern, Germany
Phone: +49 631 205 3279
e-mail: steiert@informatik.uni-kl.de

Abstract:

The world of C/S computing is changing fast. In times of global competition the companies require business information systems (BIS) supporting global computing in the internet. As internet applications are much more complex than conventional BIS new architectures must be developed. Additionally, new development methods (component-based system design), new middleware (CORBA, Java RMI, message-oriented middleware), and new database technology (object-relational database systems) have become applicable in the enterprise mainstream. Conventional client/server architectural models do not cope well with these new trends. This paper introduces a logical architecture for modern BIS as well as an implementation architecture exploiting the new technologies.

Topic:

Architectural Styles and Patterns

Towards a Component-based n-Tier C/S-Architecture

Hans-Peter Steiert
Department of Computer Science, University of Kaiserslautern, P. O. Box 3049,
D-67653 Kaiserslautern, Germany
Phone: +49 631 205 3279
e-mail: steiert@informatik.uni-kl.de

Abstract:
The world of C/S computing is changing fast. In times of global competition the companies require business information systems (BIS) supporting global computing in the internet. As internet applications are much more complex than conventional BIS new architectures must be developed. Additionally, new development methods (component-based system design), new middleware (CORBA, Java RMI, message-oriented middleware), and new database technology (object-relational database systems) have become applicable in the enterprise mainstream. Conventional client/server architectural models do not cope well with these new trends. This paper introduces a logical architecture for modern BIS as well as a implementation architecture exploiting the new technologies.

1 Introduction

Compared to the degree of heterogeneity and complexity of Internet applications the typical environments of conventional business information systems (BIS) have been quite simple. To support, for example, location independent access or, even, electronic commerce the services of BISs have to be globally accessible through the Inter-/Intranet. While conventional BIS, based on TP-Monitors and database technology, are able to fulfil the key requirements of mission critical systems (reliability, stability, availability, scalability, maintainability) current client/server systems often suffer from these crucial features [Ed98, Co98].

The challenge of global computing comes along with a set of new requirements, new development methods and new technology:

First, access to next generation BIS is not any longer limited to the local network. Customers and suppliers using mobile clients or a simple WWW browser must be able to use business functions via Internet.

Second, the use of prefabricated components will become more important, leading to shorter development times and easier customizing (*componentware*) [Sz97]. Also, object-oriented design methods are widely accepted today in order to cope with the complexity of current software systems.

Third, new technology has become applicable in the enterprise mainstream. Communication middleware like CORBA [OMG97, OHE97], Java RMI [Sun98] or MOM (message-oriented middleware) [BHM90] provides an infrastructure supporting the cooperation of heterogeneous components. Additionally, in the field of database management systems (DBMS) so called object-relational DBMS (ORDBMS) are to appear at the market [St96]. Their extensibility features allow for integrating application logic into the database engine.

In this position paper, we want to introduce a refined architectural model for C/S systems related but not limited to BIS (Section 4). We will give a short introduction into the logical architecture of a BIS (Section 2) and will discuss the pros and cons of actual implementation architectures (Section 3). Finally, Section 5 concludes the paper.

2 Logical Architecture of a BIS

Object-oriented software engineering (OOSE) approaches traditionally have focused on object-oriented programming and object-oriented data modelling [Bo94]. Nowadays, system design in the large takes also advantage of object-oriented design principles. The idea of *componentware* is to build systems using prefabricated objects, which provide the needed functionality, and a communication infrastructure, as for example CORBA, which serves as the clue. This may be seen as an incarnation of the object-oriented design principles of modularity, encapsulation and separation of concerns at a higher level of abstraction.

We consider the architecture of modern information systems consisting of three layers (Figure 1). The base layer provides *business objects* (BO) combining enterprise data and application independent object behavior. Together with the data management layer, used to store data persistently, this layer is often called **data layer**. The **application layer** incorporates application logic into the system. It offers *business functions* (BF) which are mapped to usually several operations on BOs. *Workflows* combine elementary business functions to more complex business functions. The **presentation layer** incorporates the user interfaces (UI) to present output information and, in turn, to route input parameters as well as user actions to the application layer.

Following the lessons learned in OOSE, dependencies between presentation layer, application layer and data layer should be avoided in order to achieve flexibility, extensibility and reuse. In the following we want to consider the mentioned layers in a little more detail.

2.1 Data Management

The key features of a BIS depend on the data management layer, which is the foundation for reliability, stability, availability and scalability. Today's database management systems are known to meet these requirements. Additionally, the concept of ACID transactions [HR83] with their clear and simple failure model provide a solid base for the higher layers and allow to build reliable systems. The main task of the data management layer is to store data and to protect it against failures and inconsistencies.

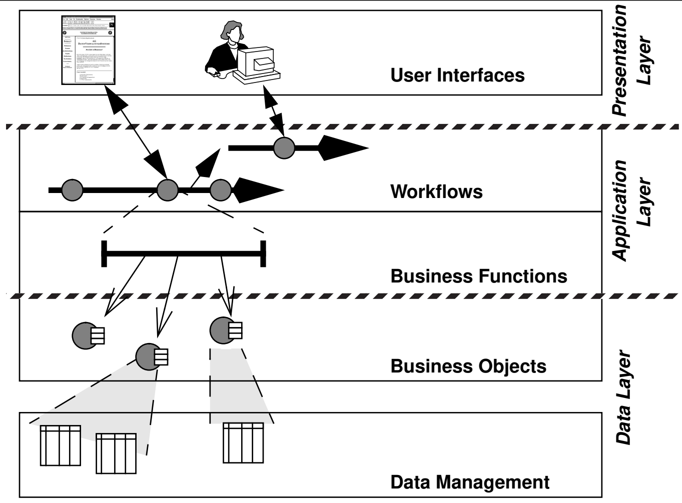

Figure 1: Architecture of a Business Information System

2.2 Business Objects

BOs provide a certain degree of abstraction for the higher levels. They hide all details of storing the state persistently, as for example the mapping of objects to relations or the use of one or several databases and external information systems. Hence, BOs may also be used to integrate legacy systems, which means they encapsulate heterogeneity if necessary. Manipulation of BOs is performed through defined interfaces, which ensures consistent state transitions.

2.3 Business Functions

While BOs are the smallest domain-specific building blocks, they are still application-independent. BFs integrate business logic into the system. They group together semantically interrelated state transitions of (several) BOs in an application-dependent manner. Hence, a BF provides an abstraction from state transitions of single BOs to a consistent sequence of state transitions usually concerning several BOs. A set of BFs and their related BOs can be used to establish a component, which may be (re)used in other BIS or exchanged if new customers have different needs. To use a BF as a part of the interface for a strictly encapsulated component it has to provide specific functionality and, additionally, a assured quality of service, as for example atomicity of the state transitions.

2.4 Workflow Layer

In order to use components in a flexible manner, the control flow of the business logic should be isolated from the functionality. Therefore, the Workflow Objects (WfO) control both the execution of business transactions involving more than one BF and the control flow between different components.

WfO also coordinate the work of the users engaged in the business transaction. Thus, the WfO need to manage organizational data like employees and business rules.

As reliability is a key feature of a BIS, a WfO is also responsible for application recovery in case of a failure. Workflows need to be restarted after a crash (forward recovery) and in the case of business conditions have changed compensation of previous steps may be necessary (backward recovery).

Note that workflows may cross department boundaries as well as combine activities of separate organizations. Hence, the workflow layer is the backbone of the BIS cluing together the different components of a highly distributed system.

2.5 Presentation Layer

Users should be able to access a modern BIS from different places all around the world using mobile clients, simple WWW browsers or their desktop PC. Therefore, different user interfaces need to be supported at the presentation layer, which all access the same application functionality. In order to support all these types of access to the BFs, the application layer should be strongly separated from the presentation layer. Furthermore, the requirement to provide access to other information systems leads to a strong separation of UI and application logic.

3 Today's Client/Server Architectures

The logical architecture of a BIS presented above can be mapped to an implementation architecture in different ways. Here we restrict our discussion to the state of the art of C/S systems. A bunch of properties may be used to rank C/S systems but we want to concentrate on development technology, implementation technology, expected performance and scalability, enabling for Web-based access and maintainability [ED97].

3.1 Fat Client

In a C/S architecture leading to fat clients, presentation and application layer are associated with the client side and a database server is used to store the data (2-tier).

One of the advantages of this approach is ease of system development. Any development method and any implementation technology can be used as long as access to a database server is provided. The decision for a particular technology can consider the skills of the developers, strategic partnerships and costs for tools and education. Unfortunately, these systems tend to perform badly and do not scale. BFs must be supplied with data by the server and results must be propagated back (data shipping), leading to a lot of network traffic. In a multi-user environment with many concurrent clients, inter-transaction caching is not advisable, because of high conflict rates. Additionally, the server has to manage a lot of resource-consuming client connections and therefore will not scale. Furthermore, the fat client approach is not useful w. r. t. the WWW, because a lot of application code and data needs to be transferred to the client. Last but not least, maintenance is difficult, since a large number of clients must be serviced.

3.2 Fat Server

In contrast to the fat client approach in a fat server architecture all application logic has been integrated into the (database) server. Only the presentation remains on the client (2-tier).

Development of server applications is a hard task. The functional programming style does not allow to exploit the advantages of object-oriented development methods at all. Additionally, tools and technology of the particular database vendor must be used, because of the proprietary implementation technology. Hence, developers need special skills. Also, large parts of the system need to be reimplemented, if more than one platform is to be supported.

On the other hand, this approach promises to provide better performance. Only function calls and small result sets are transferred between client and server, which is called function shipping, or, more exactly, function request shipping (FRS). The application is able to take advantage of the properties of modern database servers leading to a reliable system. However, fat servers must also manage a lot of connections, which limits scalability. WWW integration is easy, since clients are "thin" and WWW-clients behave as every other client. This architecture is easy to maintain, because everything is concentrated at a single service point.

3.3 Application Server

The application server approach tries to combine the advantages of the architectures discussed above: Thin clients accessing a server via function calls, a DBMS-vendor independent technology for application development, performance and scalability of database servers, easy integration into the WWW, and low maintenance costs. This goal is reached by using application servers as a middle tier between client and server (3-tier). Usually, the middle tier covers the business object layer, the business function layer and the workflow layer.

From the database server's point of view an application server is a database client. Hence, object-oriented design methods as well as every implementation technology providing appropriate database access may be used. However, application servers are difficult to implement. They must be able to support many clients and, therefore, they have to run either replicated (under the control of a TP-Monitor for load balancing [GR93]) or must be multithreaded themselves so that parallelism can be exploited.

The application server brings together data and functions at the middle tier in order to achieve high performance. Specialized application servers are able to take advantage of inter-transaction caches by using semantic knowledge. Even cache coherency protocols reducing database accesses may be feasible as far as the number of servers stays small. Thus, the business logic at the application server is accessed by clients via FRS while the application server itself loads the data from the database.

In application server architectures thin clients are used, allowing for easy integration of WWW-client support. Maintenance is less complex than in fat client environments, because there are just a few service points.

4 Cooperative Components

In Section 1 we have already mentioned that the world of C/S computing is in change because of new requirements, new development methods and new technology. Global computing comes along with higher complexity, a higher degree of heterogeneity and raising communication costs. Object-oriented design methods may be able to cope with the complexity of developing such systems and corresponding tools may help managing them. Also, heterogeneity may be bridged using an adequate technology as a communication infrastructure, for instance CORBA. However, performance of a C/S system depends very much upon a well chosen architecture reducing communication costs for FRS and data shipping.

Hence, in this position paper we want to concentrate on enhancing the simple C/S architectural models in order to cope with the new challenges of a highly distributed and heterogeneous transaction processing environment.

The architectural models discussed in Section 3 do not cope well with these new trends. In the world of componentware the simple 2/3-tier models are too static. The idea of a monolithic block of application logic completely implemented in a single tier does not fit into the concept of cooperative components, where the application logic is partitioned into several components. Additionally, the conventional C/S architectural models consider only one or two C/S relationships, while component-based systems come along with much more complex relationships.

Furthermore, the 2/3-tier models do not cope with the new generation of relational DBMS. ORDBMS allow for more complex data structures and the integration of object behavior as user-defined functions (UDF) into the DBMS kernel. Hence, system performance may increase, if complex predicates involving method calls are evaluated by the DBMS instead in the middle tier.

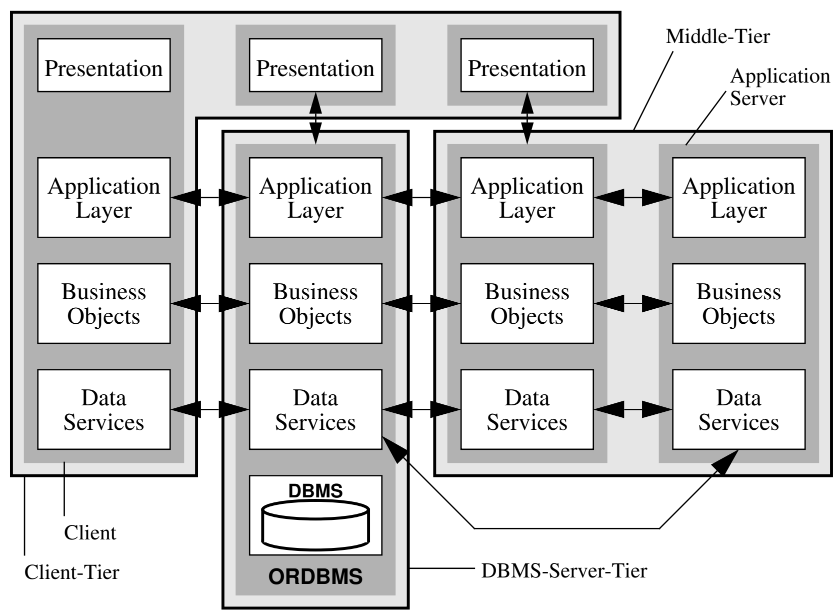

Figure 2: Component-based N-Tier Architecture

We want to overcome these drawbacks by enhancing the simple 2/3-tier models towards a n-tier C/S architecture of cooperative components (Figure 2). In our, architectural model there are still three tiers: First, a client-tier, which serves as the front-end and comes along with the presentation logic. Second, the middle-tier with a set of components acting as specialized application servers and, third, the DBMS-tier, where the database servers reside. In contrast to the conventional models, the application logic is not exclusively contained in one tier. Instead, every tier is populated with components and every component may include parts of the application logic. The components are divided into five layer, according to the logical architecture of a BIS introduced in Section 2 (presentation, application layer, business objects, data services, database server). A workflow layer is not considered, because workflow management may either be part of the application layer or provided by a separate component. The data management layer is refined into two layers, a data service layer and the DBMS itself. Functionality in the data service layer may reach from the pure API of the DBMS to a main memory database system providing the whole database functionality in that component. Mapping functionality to bridge the impedance between an object model and the data model of the DBMS including the mapping of methods to UDFs is also implemented in this layer. According to our logical architecture, the application layer implements the BFs and the business object layer manages BOs. In this architecture, the C/S relationships between the tiers are refined to relationships between components, more precisly they are refined to relationships between the layers of the components. Also, the architecture allows for consideration of C/S relationships between components at the same layer.

Examining C/S relationships

The refinement of C/S relationships allows for a closer look upon the cooperation at the different layers. In the application layer, BFs, implementing large-grained application logic, are executed. Hence, FRS is used for the cooperation between components. In contrast to BFs the methods provided at the interface of a BO are much more fine-grained. So, accessing a BO's methods via FRS tends to high communication costs, because of the overhead of the communication protocol. If a BO is referenced more than once during BF-processing, migration or replication and caching of objects may be useful. Note that this kind of object shipping is a extension of data shipping. Object shipping means to exchange the state and the behavior of an object between two components, while data shipping means only to transfer the object's state. Usually, the code implementing object behavior does not change at run time and is invariant for all instances of a particular class. Hence, a component needs not to migrate code more than once. It can be cached in a component and afterward exchanging the object's state is sufficient.

Although object shipping is the desired approach, data shipping is often necessary, because object shipping is not adequately supported today. Only Java allows for dynamically transferring both, the object's state and the code implementing it's behavior. This requires either all components being implemented in Java or the integration of a Java virtual machine into a component (which is already planned by database vendors of ORDMBS). CORBA allows the exchange of object states between heterogeneous platforms and applications implemented in different programming languages but not the exchange of code. So, CORBA does only support data shipping and an implementation of the behavior must already exist in every component. Data shipping occurs at the data service layer.

Flexible C/S computing

The idea of cooperative components implementing application logic and BOs at every tier makes the system much more flexible but, in turn, the design much more complex. Now, a designer has to decide in which components a particular BF is to be implemented and where the BOs have to reside.

A static separation of business logic may not be sufficient. Assume a complex predicate including method calls, which is needed to be evaluated by the presentation logic, by specialized application servers, as well as by the database server. The system designer should provide such application logic at all three tiers in order to be able to route a function call to the component with the lowest execution costs. In a transaction processing environment factors relevant for this decision are the costs to access BOs, to make their state durable at the end of a transaction and to synchronize concurrent access.

We have already discussed that either objects must migrate to the component or their methods must be called via FRS. Which alternative to choose depends on costs for object migration, communication and reference locality.

In transaction processing environments the object's state has to be stored persistently in the database at the end of a transaction. Hence, not only run-time costs for object migration or FRS have to be considered in the decision, but also the costs of propagating a changed object's state to the database server at the end of the transaction.

If several components access the same BO, it may be replicated in several caches. So, cache coherency protocols are needed to ensure access to the current state of an object. Furthermore, access to BOs must be synchronized so that the isolation property of transactions is not violated. Cache coherency as well as synchronization will lead to expensive communication, if a lot of replicates exist, which has to be considered if objects are accessed by remote components.

5 Conclusions

The world of C/S computing is changing fast. New technologies, as the Internet, CORBA and ORDBMS, allow for development of highly distributed BIS. In times of global competition the companies require BIS, supporting their world wide trading partnerships. As the Internet is a much more complex computing environment, a more detailed architectural model of C/S systems is needed.

Conventional 2/3-tier architectures are not sufficient to cope with the complexity of modern BIS. The implementations based on these architectural models lack on flexibility needed to perform well in all scenarios appearing in new C/S applications. Therefore, we introduced a logical view upon modern BIS (Section 2), seperating five layers of functionality, every layer providing an abstraction leading towards a component-based system, whereas a workflow layer is used to coordinate the cooperation of different components. We have also provided a refinement of the conventional 2/3-tier architectural model of C/S systems (Section 4) which overcomes the strict separation of functionality in two or three tiers. Instead, we presented a model of cooperative components, whereas each component consists of 3 to 5 layers itself. This allows for a much more detailed consideration of the application scenarios, the C/S relationships and their impact to system performance. Both models can serve as a foundation for analysing the needs of modern applications in C/S environment.

Literature

BHM90 P. A. Bernstein, M. Hsu, B. Mann: "Implementing Recoverable Requests Using Queues" Proc. ACM SIGMOD, 1990

Bo94 G. Booch: "Object Oriented Analysis and Design with Applications" The Benjamin/Cummings Publishing Compony Inc., 1994

Co98 E. E. Cobb: "Issues when making object middleware scalable" MiddlewareSpectra, May 1998

ED97 J.Edwards, D. DeVoe: "3-Tier Client/Server At Work" John Wiley & Sons, Inc., 1997

Ed98 J. Edwards: "Let's Get Serious about Distributed Objects" Distributed Computing, February 1998

GR93 J. Gray, A. Reuter: "Transaction Processing: Concepts and Techniques" Morgan Kaufmann Publishers Inc., 1993

HR83 T. Härder, A. Reuter: "Principles of Transaction Oriented Database Recovery" ACM Computing Surveys, Vol. 15, No. 4, 1983

OHE96 R. Orfali, D. Harkey, J. Edwards: "The Essential Client/Server Survival Guide" John Wiley & Sons, Inc., 1996

OHE97 R. Orfali, D. Harkey, J. Edwards: "Instant Corba" John Wiley & Sons, Inc., 1997

OMG97 OMG: "The Common Object Reques Broker: Architecture and Specification" Revision 2.0, July 1995, Updated July 1996, Object Management Group, formal document 97-02-25, http://www.omg.org

St96 M. Stonebraker: "Object-Relational DBMSs - The Next Great Wave" Morgan Kaufmann Publishers, Inc., 1996

Sun98 Sun Microsystems, Inc: "RMI - Remote Method Invocation" Available at: http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi, March 1998

Sz97 C. Szyperski: "Component Software" Addison Wesley Longman, Ltd., 1997