1. ABSTRACT
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.

1.1 Keywords
client/server-architectures, componentware, ORDBMS

2. 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 [4][2].

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) [12]. 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 [9][8], Java RMI [11] or MOM (message-oriented middleware) [1] 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 [10]. 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 5). We will give a short introduction into the logical architecture of a BIS (Section 3) and will discuss the pros and cons of actual implementation architectures (Section 4). Finally, Section 6 concludes the paper.

3. LOGICAL ARCHITECTURE OF A BIS
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 object-oriented software engineering, 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 layers in a little more detail.

3.1 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 [6] 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.

### 3.2 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.

### 3.3 Business Functions

While BOs are the smallest domain-specific building blocks, they are still application-independent. BF s 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 BF s 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.

### 3.4 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.

### 3.5 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 BF s, 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.

### 4. TODAY’S C/S 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 [3].

#### 4.1 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).

✓ System development is ‘easy’ (free choice of development method and implementation technology, considering the skills of the developers, strategic partnerships and costs for tools and education).

✗ System tend to perform bad and does not scale (data shipping, high network traffic, high conflict rates in multi-user environments, server manages a lot of resource-consuming client connections).

✗ Not useful w. r. t the WWW (application code and data needs to be transferred to the client).

✗ Maintenance is difficult (a large number of clients must be serviced).

#### 4.2 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 (functional programming style, proprietary tools and implementation technology, developers need special skills, not platform independent).
- Fat Servers provide better performance (function request shipping (FRS), low network traffic, applications inherit the reliability and scalability from the server platform).
- WWW integration is easy (thin clients, WWW-clients behave as every other client).
- This architecture is easy to maintain, because everything is concentrated at a single service point.

### 4.3 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 (3-tier).

- Application servers are difficult to implement (support of many clients, intra transaction parallelism should be exploited).
- Free choice of technology (exploit object-oriented design methods and any implementation technology providing appropriate database access).
- Application servers achieve high performance (use of inter-transaction caches by exploiting semantic knowledge, cache coherency protocols in order to reduce database accesses, accessing the application server via FRS while itself loads the data set-oriented from the database).
- Easy integration of WWW-client support.
- Maintenance is less complex (just a few service points).

### 5. COOPERATIVE COMPONENTS

In Section 2 we have already mentioned that the world of C/S computing is in change because global computing comes along with higher complexity, a higher degree of heterogeneity and raising communication costs. 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 4 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 DBMS, which allow for increasing system performance by evaluating complex predicates involving method calls at the server instead at the middle tier.

![Figure 2. Component-based N-Tier Architecture](image)

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 3 (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.

#### 5.1 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, code can be replicated and cached in a component. So, afterwards, 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 its 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 ORDBMS). 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. An implementation of the behavior must already exist in every component.

5.2 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 at every tier. The system designer should be able to route the function call to the component with the lowest execution costs or, alternatively, migrate the object to the component, where the method call appeared. 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. 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. If several components access the same BO, it may be replicated in several caches. So, cache coherency protocols are needed. 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.

6. CONCLUSIONS
New technologies, as the Internet, CORBA and ORDBMS, allow for development of highly distributed BIS. Conventional 2/3-tier architectures are not sufficient to cope with the new complexity. 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 3), separating 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 5) 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 analyzing the needs of modern applications in C/S environment.

7. ACKNOWLEDGEMENTS
The author would like to thank Th. Härder and N. Ritter for their helpful comments on an earlier version of this paper.

8. REFERENCES