Load Balancing in CORBA: A Survey

Response to the Aggregated Computing RFI

Document orbos/99-07-19

Thomas Schnekenburger
Siemens AG, Dept. ZT SE 2, D-81730 Munich, Germany
thomas.schnekenburger@mchp.siemens.de
Phone: +49 89 636 51223 Fax: +49 89 636 45450

Abstract

CORBA is an industrial standard for distributed object-oriented applications covering aspects such as heterogeneity and interoperability of products of different vendors. However, the CORBA standard does not define any methods for load balancing and there is a considerable confusion of terms describing associated load balancing techniques. Starting from the general platform- and language-independent CORBA object model, this paper presents a new hierarchical classification of possible load distribution methods for CORBA applications. The classification is a valuable starting point to identify the load balancing method that is appropriate for a given application. Furthermore, we show the relations to modern software design patterns and indicate the places in a CORBA application where load balancing components such as monitoring (obtaining load information), strategy (performing load distribution decisions), and control (executing the strategy decisions) may be integrated. Besides the discussion of the general object model as defined in the CORBA standard, we deal with concrete CORBA-compliant Object Request Brokers such as Orbix and VisiBroker. The presented techniques and examples are useful to systematically evaluate whether a given CORBA ORB can be used to implement a load balancing method as required by the application.

1 Introduction

CORBA (Common Object Request Broker Architecture) [OMG95], an industrial standard for distributed object-oriented applications, will be of increasing importance for the supercomputing community since it covers important aspects such as heterogeneity (with respect to hardware, operating systems, and programming languages) and interoperability of products of different vendors. CORBA also is an interesting candidate to serve as a bridge between scientific supercomputing applications and standard software. Although load balancing [Shirazi95] is a typical problem in all distributed environments, the CORBA standard does not define any methods for load balancing. Only a few CORBA-compliant products provide some proprietary extensions (as shown in Section 5). Similar to other distributed computing environments, one cannot expect any platform to provide a general load balancing technique that automatically solves the load balancing problems of a particular application. That means that at least during the design of an application, load balancing has to be considered. Thus, software developers using CORBA have to find suitable load balancing techniques for their individual applications. Considering the enormous confusion of terms in literature (and even worse: in commercial white papers), this may be a hard task for developers which are not experienced experts in this field. For example, there is inconsistent usage of the terms "load balancing" [Ferrari86, Zhou88], "load sharing" [Eager86, Kremien92], and "load distribution" [Beccard90, Kale88]. Similar inconsistencies can be found with "dynamic/static load balancing" [Zhou88, Becker95] or "adaptive load balancing" [Casas94, Kremien92, Eager86].

A typical CORBA application consists of a number of processes that are spread among many, often heterogeneous, computational nodes. In this context, load balancing generally means the assignment of "parts" of an application to computational nodes so that the distribution of load meets given requirements (e.g., even workload distribution or adaptive distribution according to the nodes' individual performance). Here, a "part" of an application may be any possible static or dynamic entity such as for example a process, an object (with respect to an object-oriented language), or a remote method invocation.

Generally, there are three aspects of load balancing:

1. The *load balancing method* (e.g., object migration) describes which entities of the application are considered for "balancing", i.e., the entities the load balancing problem deals with and which actions are to be taken on these entities in order to improve load balancing.
2. The *load balancing components* to be used. On the topmost architectural level, there are three load balancing components:
   - *Monitoring*: The component that delivers information about the actual state of the application with respect to load balancing. This component is obviously only required if the actual state is considered for load balancing.
   - *Strategy*: The component that performs the decision when and where entities are assigned. It is not within the scope of this paper to propose or compare specific load balancing strategies, we just assume that there *is* such a strategy. A practical load balancing strategy in CORBA environments has not only to consider the "load", but also aspects such as functional restrictions (not every object can run on every server), security aspects (for example, some objects should be assigned to "secure" nodes), or availability aspects (for example, an object and its replica should not be assigned to the same node to reduce the probability that both are not available).
   - *Control*: The component that controls load balancing. It implements the mechanisms that are necessary for load balancing.
3. The *integration* of load balancing into the system, i.e., the implementation of the load balancing components for a given application.

This paper serves CORBA users to identify which load balancing methods is appropriate for their given application. Furthermore, given a CORBA compliant ORB, it can be used to systematically categorize the support of individual load balancing methods of that ORB. Therefore, this paper may be a valuable aid in the decision which ORB should be used. Section 2 introduces some CORBA terminology and a general CORBA application model that will be used in the subsequent sections. Section 3 presents a hierarchical classification of load balancing methods for CORBA applications. Since load balancing methods may also be regarded as "design patterns" [Gamma95], Section 4 discusses the relationship between the possible load balancing methods for CORBA applications and published design patterns. In Section 5, techniques for the integration of load balancing components into CORBA applications are discussed. However, to keep the article clear, many technical details are just referenced. Sections 6 and 7 show related work and give some conclusions.

2 CORBA

Our terminology follows the definitions in the POA (Portable Object Adapter) specification [OMG97], [Schmidt98a]. The CORBA standard defines an abstract model of a distributed object architecture. A CORBA application consists of *servers* offering *services* of the application, and *clients* using these services. (CORBA allows a server to be also a client). In this context, a service denotes an application specific set of somehow related interfaces and their semantics. Services are implemented by *CORBA objects*. CORBA objects are programming entities with an identity, an interface, and an implementation. From a client's perspective, the object's identity is encapsulated in the object's reference. CORBA specifies how clients invoke requests on CORBA objects via an *Object Request Broker*. Interfaces of CORBA objects are specified using an *Interface Definition Language* (*IDL*). CORBA does not explicitly specify how CORBA objects are implemented. This depends on system characteristics, such as the operating system, and the programming language used for implementing CORBA objects. Nevertheless, we can describe the load balancing aspect for CORBA applications by assuming, that each CORBA object is implemented by a *servant* at a *server*. In practice, a servant is usually a C++ or Java object that is registered at the ORB´s object adapter. The servant may reside in an own server or it may share a server with other servants. Servers usually correspond to normal application processes (for example, if the ORB is implemented as a library that is linked to application programs). However, it is also possible that there is exactly one server at each computational node (e.g., if the operating system directly implements a CORBA ORB).

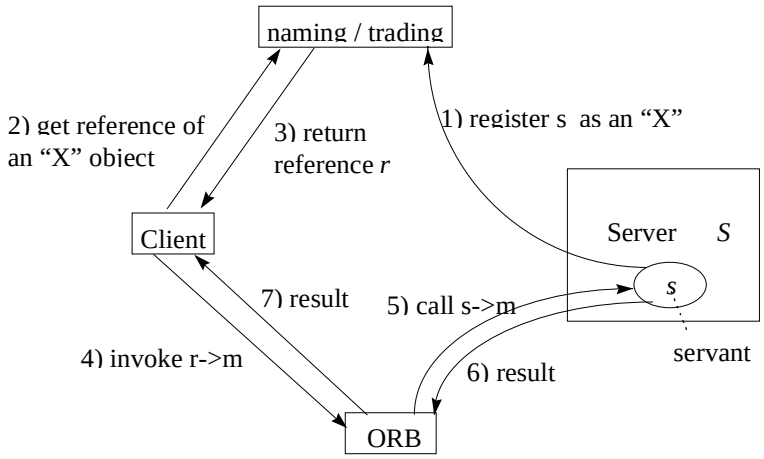

Figure 1: General CORBA application model

Figure 1 shows the general interaction between clients and servers according to the CORBA model. This interaction can be split up into seven steps:

1. The servant s, living in a server *S*, is registered at a naming / trading component as a servant implementing a CORBA object that offers a service "X". Note, that the CORBA standard does not specify a naming / trading component within the ORB. Naming or trading is offered by standard Naming/Trading CORBAServices. Furthermore, many ORBs have proprietary extensions to provide a naming / trading component within the ORB (e.g. the _bind() method in Orbix).
2. The client asks the naming / trading component for a reference to an object fulfilling certain requirements (e.g., a certain IDL-interface, etc.).
3. Assuming that "X" fulfills the given requirements, the naming / trading component will find the servant s and return an appropriate reference *r*. That means, the client receives a *connection* to a servant. Internally, the ORB provides a mapping of the CORBA object reference *r* to the servant s at server *S*.
4. The client invokes a method *m* on reference *r*.

5. The invocation *r->m* is delegated and transformed by the ORB into a method call *s->m* at server *S*.
6. The ORB receives the result (which consists of a return value and values of the out / inout parameters).
7. The ORB passes the result to the client.

This scenario generally fits to all CORBA systems. In the following section we discuss the load balancing possibilities implied by this model.

3 Classification

Traditional load balancing methods (e.g., [Eager86, Zhou88]) consider the assignment problem as the problem of assigning operating system processes to computational nodes with the objective to manage the load of computational nodes. In CORBA environments, servers are the entities for which load has to be managed. The usual goal is to avoid overloaded servers while other servers are idle. However, since clients invoke methods on server objects, the *implementation* of a load balancing method has to consider also the client side (cf. Section 5). Similar to other programming models, load balancing in CORBA environments basically consists of

- **Partitioning**: The application has to be partitioned into objects (servants) that can be distributed among servers and
- **Assignment**: Application workload has to be assigned to servers holding application objects (servants).

Partitioning is used to spread the application's services among servers. It is thus the prerequisite for the assignment, i.e. the distribution of the application workload. Consequently, we propose a classification of load balancing methods in CORBA environments that consists of two parts (Figure 2) which show partitioning and assignment methods, respectively.

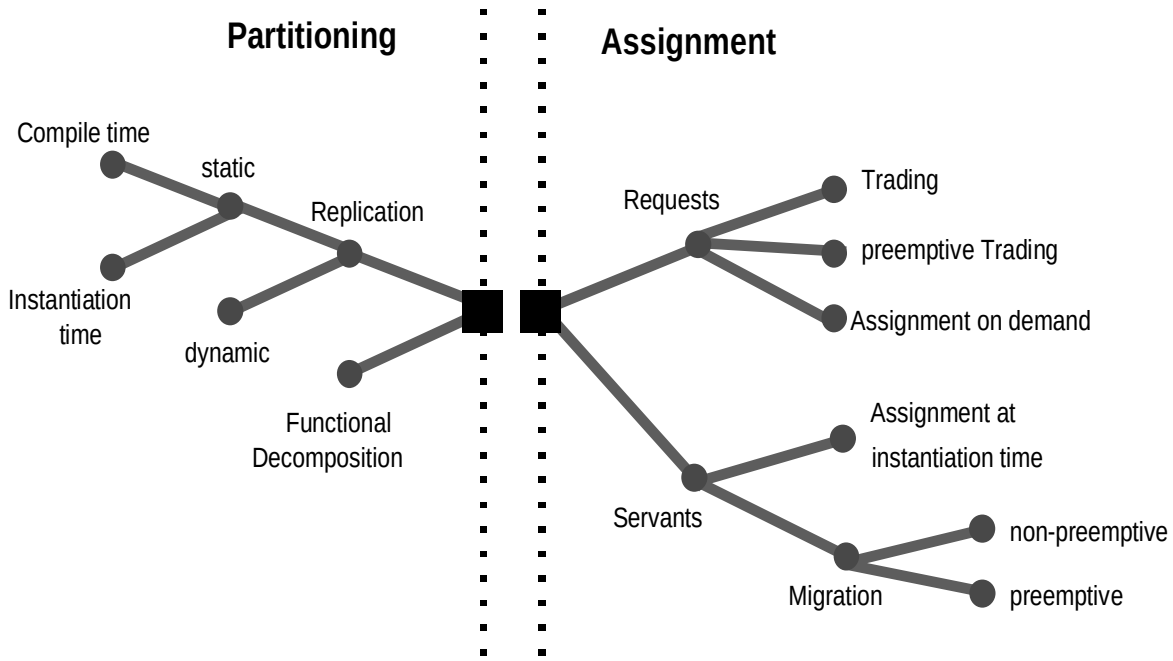

Figure 2: Hierarchical Classification of Load Distribution Methods

The following paragraphs discuss the different load balancing methods and illustrate each method by the structure of a typical CORBA application: A travel information system offering information about hotels, flights, car rentals, and weather to arbitrary clients in the Internet. We further assume that these services are distributed among several server nodes in the information provider's Intranet to provide high availability and short response times.

3.1 Partitioning

Partitioning comprises **functional decomposition** and **replication**. Functional decomposition means that different services of the application are implemented by different CORBA objects. Regarding our example, the travel information system may implement hotel information, flight information and weather information by one CORBA object each.

Replication means that an application service is implemented by several objects, called *replicas*. In other words, replication means that a certain service is available on more than one CORBA object (see [Kopetz93] for a thorough discussion of replication; see also [Isis95]). Regarding the example above, the hotel information service may be replicated, that means, information about all or a subset of all hotels is available at more than one CORBA object. Besides load balancing, replication is especially useful to guarantee availability of a service even if a server is shut down or disconnected. Replication of a service requires that all replicas have a consistent state, where the "state" is defined by the semantics of the application. There are many cases where replicas can be regarded as "stateless" with respect to client requests, that means, client requests do not change the state of application objects (for example, "read-only" queries). Generally, protocols ensuring replica consistency require implementation effort and consume resources at runtime. Therefore, there is a tradeoff between availability and degree of freedom for load balancing on one hand, and implementation effort and resource consumption on the other hand.

Regarding replication, there are different levels of dynamics. Using **static replication**, the replication degree (i.e., the number of replicas) is not changed as long as the service is instantiated. The replication degree may be fixed at **compile time** or at **service instantiation time**. Obviously, the first is less flexible than the latter. On the other hand, simplified (or optimized) consistency protocols may be used if the replication degree is determined at compile time. Regarding the travel information example, static replication would mean that the replication degree is not changed as long as the information system is running.

Using **dynamic** replication, the replication degree can be changed during *runtime*. For the travel information system, the number of replicas may be adapted to the current intensity of information requests to the information system. For example, if all servers running replicas of the weather service are overloaded, a weather service replica on a further server may be generated. Dynamic replication generally is more difficult than static replication since consistency protocols have to manage a dynamic set of replicas. Furthermore, if a replica is added at runtime, it has to obtain a consistent state.

3.2 Assignment

There are basically two kinds of entities that have to be assigned:

- **Requests**: When a client issues a request to a service, replication may imply a choice which servant handles the request. In that case, the servant may be selected according to a load balancing strategy.
- **Servants**: As depicted in the general model, each servant "lives" within a server (whatever this server corresponds to). Consequently, there may be a choice which server hosts the servant.

Assigning requests and assigning servants are orthogonal load balancing methods, i.e., in principle, they can be arbitrarily combined.

3.2.1 Assigning Requests

Assigning requests to servants can be classified according to the frequency a selection is performed. The most simple method is **assignment on demand** (also called "per-invocation" balancing). For each request, a servant is selected according to the given load balancing strategy. Obviously, this approach is not always practical: The application semantics often requires that semantically related requests are handled by the same servant. For this reason all distributed object oriented systems provide a "binding mechanism". That means, a client binds to a CORBA object and uses the received object reference for subsequent requests (cf. Section 2). In this case, the

selection of a servant is not performed at the time a request is issued. Instead the servant is selected when the client binds to the CORBA object implemented by the servant. This assignment method is usually called **trading** (sometimes this is also called "per-session" balancing). Using trading as a load balancing method requires management of *static* attributes (attributes that are not changed during the servant's lifetime, for example, the type of a service) as well as *dynamic* attributes (attributes that may change during the lifetime of the servant, for example, the "load" of the server where the servant is located).
Regarding the application example, we may use the assignment on demand method for "read-only" queries if the information services are replicated among several servers. For example, queries asking for hotel room rates, rental car rates, or actual weather data may be sent to the actually least loaded server. On the other hand, more complex queries such as transactions for booking a hotel room may use trading, i.e. the entire transaction is processed by a single object.
Summing up, we can say that there are basically two methods for request assignment. Assignment on demand can react faster on changes of the load situation since it is performed more frequently than trading. However, besides the limitations due to application semantics, there is another problem with assignment on demand: For every client request, a load balancing strategy has to perform an assignment decision. If the decision component is located at the server-side of the system (for example, to collect actual load information of servers), assignment on demand may lead to an intractable communication overhead and to an overloaded decision component at the server side. On the other hand, trading may produce the problem that the system cannot react fast enough on a changed load situation (If a server is overloaded, it has to "wait" until a sufficient number of clients close the connection). In that case we can use a mixture between trading and assignment on demand, that we call **preemptive trading**. With preemptive trading, clients bind to CORBA objects as usual. In contrast to normal trading, connections to a servant on an overloaded server (with respect to the given load index) can be redirected to another servant. That means, subsequent requests using the redirected connection are delegated to another servant. The effort to realize preemptive trading depends on the application semantics and on the flexibility of the ORB. Preemptive trading may be used in the application example as follows: If the travel agency starts a new session, the client program obtains an object reference that is used for subsequent queries. The connection is only closed and re-established when there is a significantly changed load situation on the server side.

3.2.2 Assigning Servants

According to the general CORBA model, servants are assigned to servers. When a new servant is generated, there may be a choice to which server the servant is assigned. That means, a server may be selected according to the current load situation. Furthermore, the assignment of servants to servers may be changed dynamically, i.e., servants may be **migrated** between servers.
We can classify migration according to its dynamics: Migration is called **non-preemptive**, if a servant is only migrated when it is currently not executing a method invocation, else migration is called **preemptive**. Similar to the dynamics with respect to request assignment, there is a tradeoff between implementation effort and runtime overhead on one hand, and load balancing possibilities on the other hand: Using preemptive migration, a servant may be migrated at any time, and hence, the load balancing strategy can immediately react on a changed load situation. The problem with preemptive migration is that the current "state" of the servant (including stack contents, etc.) has to be transferred to another server. This is impractical in most ORBs and operating systems. Non-preemptive migration is more easy since the servant can be expected to be in a well-defined state between method executions. However, migration can only take place after the actual operation is completed. This may be too late for strong load balancing requirements.
Regarding the application example, assignment of new servants may be used in combination with dynamic replication. For example, if all servers holding an instance of the weather information are overloaded, a new instance may be generated and assigned to a lightly loaded node. Using migration, an already started weather information object on an overloaded node may be migrated to a lightly loaded node.

3.3 Summary of the Classification

The classification has demonstrated that the CORBA object model implies a large range of possible load balancing methods. After showing the relations to actual software design patterns, Section 5 will give a deeper insight about the implementation of these load balancing methods.

4 Related Design Patterns

Modern software development processes include *design patterns*. Design patterns are an informal description of "good" solutions for recurring problems in software development. Design patterns can be used by software developers as a collection of expert knowledge about a specific problem. Furthermore, using design patterns can significantly improve the efficiency of information exchange in development teams. A broad range of design patterns can be found e.g. in [Buschmann96, Gamma95].

This section analyses design patterns that are important with respect to load balancing methods. Several of them can be applied in more than one case. Therefore, they are not arranged according to their applicability for a specific load balancing method. Instead we start with the discussion of "low level" patterns that are useful to implement specific load balancing mechanisms and end up with "high level" patterns that describe the structure of entire applications.

- **Proxy Pattern:** [Buschmann96, Gamma95] Using the Proxy-Pattern, clients communicate not directly with an object, but with a representative ("proxy"). Regarding CORBA, proxies are used on the client side as programming language objects offering the interface (according to the given client-side IDL language mapping) of the represented CORBA object. When the proxy is invoked, it is usually responsible to forward the invocation to the actual servant via the ORB. Further processing such as access-control checking or caching may be added. Regarding load balancing techniques, proxies may be used to realize dynamic request assignment by hiding the actual object reference used for requests in order to keep load balancing transparent for the client program.
- **Client-Dispatcher-Server Pattern:** [Buschmann96] This pattern directly corresponds to a naming / trading component as shown in Figure 1. Clients ask the dispatcher for object references that will be used for subsequent requests. Therefore, this pattern can be applied to realize the dynamic request assignment based on trading.
- **Object Group Pattern:** The *Object Group* design pattern [Maffeis96] describes the implementation of object replication. The client has a CORBA object reference that "represents" several replicas. That means, when the client issues a request to the object group reference, this request is forwarded to the replicas. Although the object group pattern as described in [Maffeis96] assumes that client requests are multicast to all replicas, it can also be applied to implement state consistency when only one replica (in our case, the "least" loaded one) receives a request [Isis95,Maffeis95].
- **Migration Pattern:** The Migration Pattern [Schnekenburger97b] describes the implementation of non-preemptive object migration. An object is migrated to a target by generating a new instance on the target, transferring its state to the new instance, and deleting the original object. Migrations have to be coordinated with the communication subsystem in order to guarantee that requests to the object are always delivered correctly.
- **Broker Pattern:** The Broker pattern [Buschmann96] describes Broker systems consisting of clients, servers, and broker objects. Client requests are passed through proxies to the Broker (that means, the Broker pattern applies the Proxy pattern). The Broker looks for the location of the corresponding server object, and passes the request to that server object. Within a system, several brokers may be connected by so called "Bridges". The Broker pattern is obviously the base of the CORBA standard. Since Broker objects have control over the implementation of server objects and the communication between clients and servers, basically all load balancing methods may be realized (cf. Section 5).
- **Master-Slave Pattern:** This pattern [Buschmann96] describes the "divide and conquer" principle where a computational task is divided into a number of sub-tasks that are executed in parallel. The client that issues the computational task is not aware of this parallelism. This is realized by a "Master"-Object. The Master receives the computational task, divides it into sub-tasks, sends these sub-tasks to "Slaves", collects and combines the results, and returns the complete results to the client. The Master may use (transparently for the client) any of the load balancing methods mentioned above to realize load balancing within the computational task. This pattern is very interesting for supercomputing applications to separate a parallel computational algorithm from the user interface.

5 Techniques and Examples

The previous sections concentrated on the design aspect of load balancing methods for CORBA applications. In this section we consider how the three basic load balancing components monitoring, strategy, and control as described in Section 1 and the different load distribution methods as defined in Section 3 can be integrated into a CORBA

application. To give concrete examples, we often mention the two leading CORBA ORB implementations, namely Orbix [Orbix96] from Iona Technologies Ltd, and VisiBroker [Visigenic97] from Visigenic.

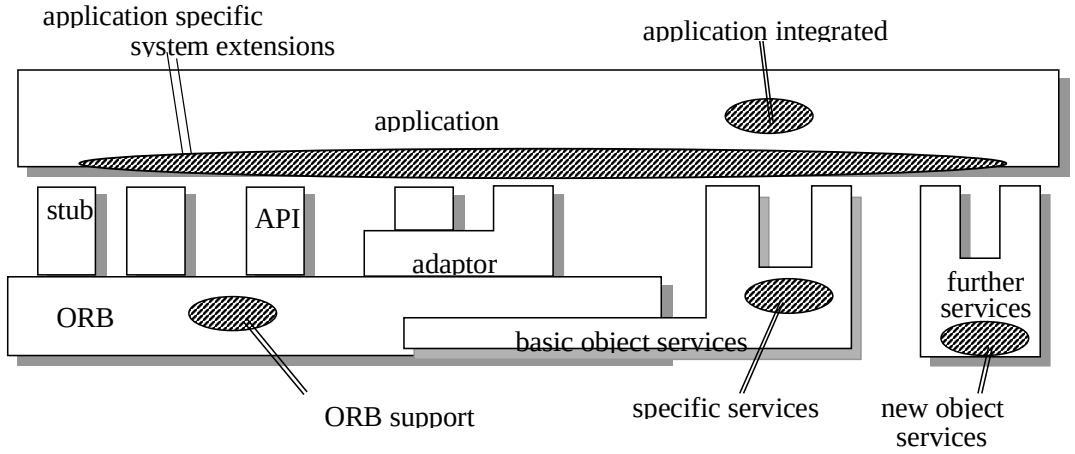

Figure 3: Integration points

Figure 3 shows a general model of a CORBA system similar to the CORBA model as defined by the OMG. For each IDL interface, an IDL compiler usually generates specific code (Stubs and "Skeletons") that implements the communication and synchronization necessary for an invocation of an operation of the IDL interface. The stub code allows the client program to invoke IDL operations fully transparent with respect to the underlying communication mechanisms. On the other hand, the object adapter hides the concrete implementation of CORBA objects from the ORB kernel. The application code furthermore uses the ORB's API (application programming interface) consisting of interfaces defined by the CORBA standard (for example, methods for registering a new server) and additional proprietary extensions (for example, Orbix provides methods for explicitly handling incoming messages). Figure 3 emphasizes the points where load balancing components may be integrated. We can identify the following system parts:

- ORB: The CORBA standard basically defines interfaces and language mappings that have to be adhered to by a compliant ORB whereas the concrete ORB implementation is left to individual vendors. Therefore, vendors may implement a CORBA compliant ORB with some load balancing functionality. Furthermore, there are some ORBs that are available with full source code (e.g., TAO [Schmidt98b], ORBacus [ORBacus], MICO [MICO], and JacORB [Brose97]). In that case, it's possible to modify / extend the ORB sources to implement load balancing functionality. However, this is impractical in most cases, because there would be no way to catch up with new versions of the ORB.
- Between the ORB interfaces (including stubs and skeletons) and the application code there may be a intermediate layer that is used to implement functionality specific for an application, but that is not directly related to the actual application functionality. For example, Orbix [Orbix96] provides such layers for application specific extensions of stubs ("smart proxies"), application specific modifications of requests ("Communication Filters"), and an application specific connection establishment ("locators"), etc. The benefits of such an intermediate layer is that the essential application code is not changed when communication mechanisms are customized. For example, smart proxies can be used to implement a caching mechanism for read-requests that is transparent to the essential client program.
- The OMG has defined several Basic Object Services [COSS95], now called CORBAServices, such as Naming Service, Life Cycle Service, Transaction Service, Event Service etc., that can be applied in many application domains. Some of these services are related to load balancing.

- Further services may be added for adding specific functionality or for the support of specific application domains. This includes services offered by a single ORB vendor, but also services that may be standardized by the OMG in the future.
- Application: Obviously, load balancing components can be directly integrated into the application code.

Except for the last point,. all other points offer the possibility to more or less expose load balancing to the application. Similar to other environments, there is a tradeoff between simple usage with probably inefficient load balancing and more complex usage with the possibility to adapt load balancing to specific requirements of the application.
The application, the ORB, as well as the services can be divided into a client part (the part of the program used for making a request) and a server part (the part of the program used for processing a request). Using this model, we can identify several possible points were load balancing components may be integrated.

5.1 Monitoring

For every load balancing method that requires knowledge about the actual system state, a monitoring sub-system is required. The job of this sub-system is to collect load information and to pass it to the point where it is needed by the load balancing strategies (and probably other system parts such as a performance visualization tool). The suitability of an integration point for monitoring depends on the kind of information monitoring should deliver, the so called *load index* [Ferrari86, Ferrari87]. It is often cited in the literature that the performance benefits of load balancing are strongly dependent upon the selection of representative load indices [Hac90, Kunz91]. Regarding CORBA environments, there are several possible load indices:

- Load of system resources: For example, the load index may be the actual ready queue length of the CPUs used. Load of system resources is (if available at all) delivered by the operating system. Therefore, this kind of load index can be gathered independent from a specific application or a specific ORB. Consequently, the monitoring subsystem may be realized as an additional service (the service offers resource load information), as part of the ORB kernel (the ORB‘s API has an interface to request load information) or within the application itself (by using the operating system information directly). Load of system resources may include also server specific information such as the "average idle time" of a server.
- Number of active threads: In case of a multi-threaded server, the load of a server may be described by the number of threads that are currently active. Either the ORB kernel or the object adapter is responsible to generate threads for incoming messages, and thus, it obviously knows the number of active threads (this applies for example to VisiBroker), or the application may generate threads by itself (this applies for example to Orbix). In the first case, information about the number of threads may be provided by the ORB’s API. If not, it may require a considerable effort to implement such a "thread counter": For each method implementing an IDL operation the thread counter has to be increased when the method is started as a thread. The counter has to be decreased if the method terminates (e.g., by a return-statement or an exception). Depending on the ORB’s API, obtaining this kind of load information directly within the application is quite costly and error prone, too.
- Message queue length: The load of a server may be described by the length of its "message queue", i.e., the number of requests waiting for being processed by the server. The ORB or any other part of a CORBA program can provide this information only if the underlying transport layer provides it too. The experiences reported for example in [Schnekenburger97a] and an implementation following [Schiemann96,Schiemann97] show that this information is not available in many frequently used CORBA implementations (for example, Orbix and VisiBroker).
- Operation monitoring: Whenever a servant’s operation is called by a client, the monitoring system may use that information to compute operation-related load indices such as the "average processing time" of a method or the "throughput" with respect to a method. Some ORBs such as IONA’s Orbix provide APIs that allow easy integration of this kind of monitoring. For example, Orbix provides "Filters" that are called whenever an operation is called "from outside" or when results are returned to clients.
- Servant monitoring: The number of active threads, message queue length, and operation monitoring may be also applied with respect to a given *servant*. Here we have similar limitations as for applying these load indices with respect to a given *server*.

Actually, there is no CORBAService that deals with monitoring, that means, there are currently no standard interfaces that can be used to obtain load information.

Up to now we have only discussed how information about a server may be obtained. However, most load balancing methods require efficient exchange of this information. Basically, a CORBA environment may use an arbitrary load exchange protocol out of the huge number of protocols proposed in the literature. For example, [Schnekenburger97a] shows how popular load exchange protocols can be implemented for CORBA applications using Orbix.
A typical problem with single-threaded CORBA servers (cf. [Schnekenburger97a]) is that it is not sufficient for the server to provide an IDL-interface that can be used to request the server's load information: Since a load information request is processed as a normal CORBA operation, the result may be not up to date any more. Furthermore, it is not practical to block the caller until it obtains load information. The solution in [Schnekenburger97a] and also in an implementation according to [Schiemann96,Schiemann97] solves this problem by implementing a "load server" residing at the same node as the original server. The original server sends its load information to the load server. The load server provides an IDL interface to request that information. Since the load server is only used for providing this information, the results are returned immediately and can be expected to be up to date since local communication between the original server and the load server can be assumed to be quite fast.

5.2 Strategies

The literature describes a huge number of possible load balancing strategies, ranging from simple round-robin and random strategies to complex distributed protocols following economical concepts [Ferguson88] or distributed information exchange according to a physical model [Heiss95]. See [Casavant88] and [Schnekenburger97c] for surveys and classifications of load balancing strategies. One problem with dynamic load balancing strategies is that load information should be provided as up to date as possible, but exchange of load information should not produce too much overhead (cf. previous section). Another problem is the tradeoff between achieving a "good" load balance and the overhead implied by the execution of load balancing methods (e.g., by process migration). This is often solved by introducing "thresholds", i.e., a re-distribution is only taking place if the load difference in the system exceeds a certain threshold.
Regarding the structure in Figure 3, a strategy may be integrated into any system part shown. However, existing CORBA products include rather simple strategies. Orbix OTM (Orbix Transaction Monitor) [OrbixOTM], Orbix+Isis [Isis95] as well as VisiBroker only provide simple round-robin load balancing strategies that do not consider actual load information. [Schnekenburger97a] shows how common threshold-based strategies can be implemented with Orbix at application level. The existing CORBAServices do not define such a "load-exchange" service. However, some load balancing schemes are tightly related to the Trading Service (as shown in [Schnekenburger97a]).

5.3 Control

Even when monitoring and strategy is implemented, there has to be a control component that actually can re-distribute the application's load. The following paragraphs discuss techniques and examples how such control may be integrated into CORBA environments. The paragraphs are arranged according to the classification in section 3.

5.3.1 Functional Decomposition

CORBA was invented to support object-oriented client programs accessing servers that are distributed among several nodes. Any practical ORB implementation is supporting multiple servers on different machines. Therefore, obviously all practical ORBs can be used to implement functional decomposition. There is no need for extra support by specific Object Services. However, we can not expect any system to perform such a decomposition automatically, that means, the application developer has to decide how the application is decomposed into servers and CORBA objects.

5.3.2 Replication

On the server side, replication means to keep a number of server objects in a consistent state (cf. Section 3). If the application semantics allows the objects to be regarded as stateless, there is no need for a specific support by the

CORBA environment; the application can just generate several replicas like ordinary CORBA objects. If objects are not stateless, it has to be ensured that all replicas perform the same state changes. To achieve this is a complex and error prone task. Implemented individually and within an application, it very likely performs quite poor. So, support by the CORBA environment is needed. Orbix+Isis [Isis96], VisiBroker [Visigenic97], and Electra [Maffeis95] directly support replication mechanisms within the ORB. For example, the application programmer can use a "group" object adapter. It is guaranteed by the internal transport protocol of the ORB that all requests are submitted to the replicas in the same order.

5.3.3 Trading

The CORBA standard does not define any functionality to obtain object references within the ORB kernel. This is left to the CORBAServices or individual products. [OMG96] defines a trading service for CORBA. Orbix OTM [OrbixOTM] implements the CORBA Transaction Service using a simple round robin strategy (see above). If this service is not available or if a general trading service would induce too much overhead, trading may also be implemented within the application. For example, [Schnekenburger97a] implements an application-specific trading service on top of Orbix. Some ORB implementations allow to customize the underlying mechanisms for obtaining object references. For example, Orbix allows to customize its "locator". The locator is used internally by Orbix when the application tries to bind to an object. [Orbix97] shows how the locator may be customized in order to implement load balancing based on trading.

5.3.4 Assignment on demand

There are several places where assignment on demand mechanisms can be integrated.

- The application can implement this method on client side by replacing the ordinary CORBA object pointer (pointer to a proxy object) through a pointer to a list of proxies that may be used for the request. When a request is performed on that pointer, one proxy out of the list is selected for the request. Using inheritance and operator-overloading as provided in C++, the application can hide this list-representation within a class in order to keep the application specific code unchanged. Only those methods that are used for assignment on demand load balancing have to be redefined by the list class. Orbix supports this by *smart proxies*. A smart proxy class inherits from the original proxy class. Therefore it is possible to hide a proxy list as mentioned above within the smart proxy class.
- Assignment on demand may also be supported by the ORB kernel. For example, Orbix+Isis [Isis96] provides an assignment on demand load balancing mechanism in combination with replication. Thus the ORB can automatically select an object reference for a request out of the set of all replicas associated with the given object reference. However, as mentioned above, the strategies used by actual products for selecting replicas are rather primitive.
- Assignment on demand may also be supported by a specific service. For example, a service similar to the CORBA Naming Service may be used to obtain an object reference. Subsequent requests are not invoked directly at the object reference, but are passed to the service. Thereupon, the service selects the actual object reference and performs the invocation.
- There is currently no standard CORBAService that corresponds to assignment on demand.

5.3.5 Preemptive Trading

Preemptive trading is an extension of ordinary trading. Two mechanisms have to be implemented: Firstly, a client's object reference has to be invalidated, that means, there must be a possibility to disconnect a client from a servant. Secondly, a client has to automatically re-connect to a suitable servant. Again we consider different integration points.

- Application: Invalidation of client references can be implemented using specific CORBA user exceptions: If the load balancing strategy decides to disconnect a client, it causes the server to raise an appropriate user exception when it is called (e.g., by setting a global variable that is checked at the beginning of each operation in the servant). When the client receives that exception, it re-connects to another servant using an ordinary trading

method as mentioned above. Again, some CORBA ORBs such as Orbix provide *communication filters and smart proxies* that can be used to separate this load balancing functionality from the application specific code.
- Up to now neither any actual CORBA ORB, nor any CORBAService supports preemptive trading.

5.3.6 Assignment of generated servants

When the application wants to generate a new servant, there may be a choice where to locate it. The CORBA LifeCycle Service uses a Factory pattern for creating CORBA objects. That means, for a given interface there is an associated factory interface that can be used to create object instances of the interface. Regarding the actual implementation of this mechanism, there are basically two possibilities:
- Firstly, a servant may be generated at an already running server. This corresponds to the *shared* activation mode as defined in the CORBA BOA specification. It can be used for load balancing among already running servers.
- Secondly, the application may activate a new server instance that will host the new servant. This corresponds to the CORBA BOA's definition of the *unshared* activation mode and the *per method call* activation mode. This technique corresponds to the traditional, operating system based method "load balancing by process generation". Therefore it may be used in combination with popular load sharing tools supporting load balancing by process creation (see [Kaplan94] for a survey).

5.3.7 Non-preemptive object migration

The CORBA LifeCycle Service defines interfaces for "moving" CORBA objects. A new object instance is created (using a Factory pattern similar to the object generation pattern), the object's state is copied to the new instance, and the original instance is removed. Consequently, if the Life Cycle Service is available for a given ORB, it will be used to implement object migrations. However, the realization and application of the Life Cycle Service is quite complicate if relationships between the object to be migrated and other CORBA object are considered. On the other hand, object migration may be implemented within the application. The effort for implementing this mechanism depends on application semantics. There are two approaches to implement object migration: Firstly, objects can be migrated between servers, or secondly, entire servers may be "migrated". Generally, application based migration of objects can be implemented as follows: (cf. Migration Pattern in [Schnekenburger97b]):
- Generate a new instance of the object using a factory pattern similar to the LifeCycle service.
- Transfer the object's state to the new instance using extensions of the object's interface.
- Force clients that have invoked the original instance after the beginning of the state transfer to re-issue the request (e.g., by returning a user exception).
- After the state transfer is completed, delete the original instance.
- Clients may obtain exceptions after the original instance is deleted. In that case, the they should re-connect to the object.

Application based migration of a server can be implemented as follows:
- Create a new server instance (e.g., using the CORBA BOA activation methods).
- Transfer the state of the original server to the new instance.
- Terminate the original server.
- Invocations to the original server usually should not get lost during the state transfer. This is accomplished either by forwarding incoming requests to the new server instance or by raising an exception that forces the client to re-connect to the new server (similar to preemptive trading).

If the application already provides a recovery mechanism for servers this can obviously be used for migration. In that case, migration can be implemented by terminating the original server and starting the new server process afterwards.

5.3.8 Preemptive object migration

Depending on the type of servant and server activation mode, there are different possibilities for implementing preemptive object migration:

- Single threaded, shared servers: In this case the system has to interrupt the execution of an operation and transfer the state (including stack etc.) to another server. Obviously this is rather impossible in most practical systems.
- Multi-threaded, shared servers: When invocations of an object are implemented as threads (for example, one thread per method invocation, or one thread per object within the server) preemptive object migration means to migrate all threads associated with an object to a new server. Obviously, this requires some support from the operating system and is not generally applicable.
- Unshared servers: If an object resides on its "own" server, preemptive object migration means to preemptively migrate the entire server. That means, the server is migrated to a new destination independent of whether it is currently executing an operation or not. The literature proposes a large number of concepts and techniques for preemptive process migration (e.g., [Douglis87]). However, process migration has also to consider communication connections used by the ORB. (For example, an ORB's internal representation of object references may include actual host names of other servers). Consequently, it is rather unlikely that a general process migration method can be used with an ORB. In other words, the ORB itself has to provide mechanisms for process migration on the given platform. Up to now we do not know about such ORB's.

Summing up, preemptive object migration appears to be a rather exotic load balancing method that is unlikely to be applied in any serious software project.

5.4 Summary

The following table summarizes the possibilities for integrating load balancing methods within individual parts of a CORBA system. The following possible integration points are considered:

- Application: Load balancing methods are directly implemented in the application specific code.
- Intermediate layer: Load balancing methods are implemented using an intermediate layer (see above)
- ORB kernel: Load balancing is performed by the ORB kernel. As state above, this comprises also the possibility to modify the ORB's sources, although this is impractical in most cases.
- Actual CORBA service: Load balancing method is defined by an existing CORBA service
- Possible future service: Load balancing method could eventually be defined by a future CORBA service

Abbreviations:

"+" already defined by CORBA standard,
"yes" possible to implement within the application, or possibly available with an individual ORB or service, etc.,
"dep." possible, depending whether underlying operating system / ORB provides suitable mechanisms (e.g. process migration),
"?" may be possible, but unlikely to be applied in practice,
"--" impossible, (or "not yet defined" in case of the actual CORBAServices)
" " not needed, already defined in CORBA standards

	Application	Intermediate Layer	ORB kernel	actual CORBA Service	poss. future service
Monitoring Resource Load (1)	yes	yes	yes	--	yes
Monitoring # threads (2)	?	yes	yes	--	? (3)
Monitoring Message queue	dep.	dep.	dep.	--	? (4)
Operation monitoring	yes	yes	yes	--	--
Strategies	yes	yes	yes	-- (5)	yes
Functional Decomposition			+		
Replication	yes	?	yes	--	yes
Trading	yes	yes	yes	+	
preemptive Trading	yes	yes	yes	--	?

Assignment on demand	yes	yes	yes	?	?
Object Assignment	yes	yes	yes	yes (6)	
Non-preemptive Migration	yes	yes	yes	yes (6)	
preemptive Migration	--	--	?	--	--

Remarks:
(1) Some resources can only be monitored if the operating systems provides interfaces for accessing the load index.
(2) Obviously, this is applicable only for multi-threaded systems.
(3) Rather unlikely, since threads are not included in the CORBA programming model.
(4) Unlikely, since there is no possibility to implement such a service for all platforms.
(5) The trading service may be used in some cases.
(6) The LifeCycle service defines interfaces for generating and migrating objects. However, the mechanisms have to be supported by the object implementation, i.e., the application or the ORB itself.

6 Related Work

The previous chapters referenced several papers covering some specific aspects of load balancing in CORBA environments (e.g., [Maffeis95], [Orbix97], [Schiemann96], [Schiemann97], and [Schnekenburger97a]). Furthermore, there are two other important object oriented distributed computing platforms, namely Microsoft's DCOM and Java RMI. However, although the classification in Section 3 and the patterns in Section 4 can be applied more or less also to these platforms, the techniques as presented in Section 5 are strongly related to OMG's CORBA architecture.

7 Conclusions

Our discussion of the different load balancing methods and their possible integration points into CORBA environments has revealed why there is no "standard" load balancing concept for CORBA. Several reasons are identified:

- The general CORBA model implies a variety of rather different load balancing methods, ranging from general trading mechanisms to rather low-level object migration methods.
- CORBA does not define the implementation of the ORB and of CORBA objects. In this paper we have assumed that "servants" are hosted by "servers" and that the goal is to balance load among servers.
- It is well known from experiences with load balancing for parallel applications that there is no "universal" load balancing strategy, that means, the suitability of a load balancing strategy depends on the characteristics of the application (see for example [Schnekenburger97c]). However, the CORBA standard, actual ORB's such as Orbix and VisiBroker, as well as the CORBAServices are not intended for specific application areas.

Summing up, we conclude that applications in the near future have to integrate mechanisms and strategies into the application code. It is unlikely that future ORBs and standard services will give sufficient support for load balancing in all cases. From the area of parallel computing we know that a *transparent* load balancing is not practical in many application areas. However, application programmers may be supported in their task to realize load balancing within their specific applications, for example, by "load balancing frameworks" that offer pre-manufactured basic mechanisms and load balancing strategies.

References

[Amaral92] P. Amaral, C. Jacquemot, P. Jensen, R. Lea, and A. Mirowski, A., "Transparent Object Migration in COOL-2", In *Proc. of Workshop on Dynamic Object Placement and Load-Balancing in Parallel and Distributed Systems, ECOOP'92,* Utrecht, The Netherlands, 1992

[Beccard90] R. Beccard and W. Ameling, "From Object-Oriented Programming to Automatic Load Distribution". In Proc. *CONPAR 90, Zurich, pages* 502-512, 1990.

[Becker95] Wolfgang Becker, "Dynamic Balancing Complex Workload in Workstation Networks - Challenge, Concepts and Experiences". In *High-Performance Computing and Networking*, Springer, LNCS 919, pages 407-412, 1995.

[Brose97] Gerald Brose, "JacORB: Implementation and Design of a Java ORB". In *Procs. of DAIS'97, IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, Cottbus, Germany, Chapman & Hall 1997. See also `http://www.inf.fu-berlin.de/~brose/jacorb/`

[Buschmann96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, *Pattern-Oriented Software Architecture A System of Patterns*, Wiley, 1996

[Casas94] Jeremy Casas, and Ravi Konuru, and Steve W. Otto, "Adaptive Load Migration systems for PVM". In Proc. *Supercomputing'94*, pages 390-399, 1994

[Casavant88] Thomas L. Casavant and Jon G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", *IEEE Transactions on* Software *Engineering*, 2(14), pages 141-154, 1988

[COSS95] *CORBAservices: Common Object Services Specification* OMG (Object Management Group), Technical Report 95-3-31, 1995

[Douglis87] Fred Douglis and John K. Ousterhout, "Process Migration in the Sprite Operating System", In *Proceedings of the 7th* International *Conference on Distributed Computing Systems*,1987

[Eager86] Derek L. Eager, and Edward D. Lazowska, and John Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems". In *IEEE Transactions on Software Engineering*, 12(5), pages 662-675, 1986

[Ferguson88] Donald Ferguson, Yechiam Yemini, and Nikolaou Christos, "Microeconomic Algorithms for Load Balancing in Distributed Computer Systems", In *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, California, pages 491-499, 1988

[Ferrari86] Domenico Ferrari, "A Study of Load Indices for Load Balancing Schemes", In *Workload Characterization of Computer Systems and Computer Networks*, pages 91-99, 1986

[Ferrari87] D. Ferrari and S. Zhou, "An Empirical Investigation of Load Indices for Load Balancing Applications", In *12th Int. Symp. On Computer Performance Modeling, Measurement and Evaluation*, pages 515-528, 1987

[Gamma95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[Hac90] A. Hac and T.J. Johnson, "Sensitivity Study of the Load Balancing Algorithm in a Distributed System". *Journal of Parallel and Distributed Computing*, 10, pages 85-89, 1990

[Heiss95] Hans-Ulrich Heiss and Michael Schmitz, *Decentralized Dynamic Load Balancing: The Particles Approach* Information Sciences, Vol. 84, No. 1+2, pages 115-128, 1995

[Isis95] *Orbix+Isis Programmer's Guide*, Isis Distributed Systems, Inc., and IONA Technologies, Ltd., 1995

[Kale88] L.V Kale, "Comparing the Performance of Two Dynamic Load Distribution Methods", In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 8-12, 1988

[Kaplan94] Joseph A. Kaplan and Michael L. Nelson, "A Comparison of Queueing, Cluster and Distributed Computing Systems", NASA Langley Research Center, Hampton, Virginia, Technical Memorandum 109025 (Revision 1), 1994

[Kopetz93] Herrmann Kopetz and Paulo Verissimo, "Real Time and Dependability Concepts", In Sape J. Mullender, editor, *Distributed Systems*, chapter 16, pages 411-446, ACM Press, New York, 1993

[Kremien92] Orly Kremien and Jeff Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms". In *IEEE Transactions on Parallel and Distributed Systems*, 6(3), pages 747-760, 1992

[Kunz91] Thomas Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", IEEE Transactions on Software Engineering, 17(7), pages 725-730, 1991

[Maffeis95] S. Maffeis, "Adding Group Communication and Fault-Tolerance to CORBA", *In Proc. of the 1995 USENIX Conference on Object-Oriented Technologies*, Monterey, CA, USA, 1995

[Maffeis96] S. Maffeis, "The Object Group Design Pattern", In *2nd USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, Toronto, Canada, 1996

[MICO] `http://www.vsb.informatik.uni-frankfurt.de/~mico/`

[OMG95] OMG (Object Management Group), *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995

[OMG96] *OMG RFP5 Submission Trading Object Service*, OMG (Object Management Group), Document orbos/96-05-06, 1996

[OMG97] OMG (Object Management Group), *ORB Portability Joint Submission (Final)*, Technical Report orbos/97-05-15, 1997

[ORBacus] See `http://www.ooc.com/ob/`

[Orbix96] *Orbix 2 Programming Guide, Version 2.2*, IONA Technologies Ltd., October 1996

[Orbix97] "An Orbix Load Disperser", In *orbix Journal* Q3 1997, IONA Technologies Inc., pages 12-13, 1997

[OrbixOTM] See `http://www.iona.com/support/whitepapers/orbixotm/otm_wp.html`

[Schiemann96] Björn Schiemann and Lothar Borrmann, "A New Approach for Load Balancing in High Performance Decision Support Systems", In Future Generation Computer Systems 12(5), pages *345-355*, 1997

[Schiemann97] Schiemann, B.: *Exploiting Interface Definition Languages for Load Balancing*; Journal on Information Sciences; Issue 1-2, pages 221-231, 1997

[Schmidt98a] Douglas C. Schmidt and Steve Vinoski*: Using the Portable Object Adapter for Transient and Persistent CORBA Objects;* C++ Report, SIGS, Vol. 10, No 4, April, 1998.

[Schmidt98b] Douglas C. Schmidt, David L. Levine, and Chris Cleeland, *Architectures and Patterns for High-performance, Real-time CORBA Object Request Brokers*, Advances in Computers, Academic Press, Ed., Marvin Zelkowitz, to appear. See also `http://cs.wustl.edu/~schmidt/TAO.html`

[Schnekenburger97a] Thomas Schnekenburger and Günther Rackl, "Implementing Dynamic Load Distribution Strategies with Orbix", *International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'97), Las Vegas, Nevada, CSREA, 1997

[Schnekenburger97b] Thomas Schnekenburger, "The Migration Pattern", In *Component Users Conference 1997*, Munich, Germany, to appear 1998 in SIGS publications.

[Schnekenburger97c] Thomas Schnekenburger and Georg Stellner (eds.), *Load Distribution for Parallel Applications*, Teubner, 1997, ISBN 3-8154-2309-0

[Shirazi95] Behrooz A. Shirazi, Ali R. Hurson, and Krishna M Kavi (eds.*), Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, 1995

[Visigenic97] *Distributed Computing in the Internet Age* Visigenic Software Inc., February 1997

[Zhou88] Songnian Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing". *In IEEE Transactions on Software Engineering* 14(9), pages 1327-1341, 1988