Pre-print version

D. Ardagna, C. Francalanci, V. Piuri, "Designing and rightsizing the information system architecture", in *Information Systems Frontiers*, Springer, pp. 229-245, September, 2004. ISSN: 1387-3326. DOI: 10.1023/B:ISFI.0000037878.11967.50

Designing and Rightsizing the Information System Architecture

Danilo Ardagna and Chiara Francalanci
Department of Electronics and Information Technologies, Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
E-mail: ardagna@elet.polimi.it
E-mail: francala@elet.polimi.it

Vincenzo Piuri*
Department of Information Technologies, University of Milan, Via Bramante 65, 26013 Crema, Italy
E-mail: piuri@dti.unimi.it

Abstract. *Information system design and sizing constitute a complex, top-down process that tailors the technology architecture to application requirements. Practitioners usually tackle this top-down process by focusing on individual design aspects, such as data or specific applications, and by relying on their previous experiences to compare alternative architectural solutions. Acquisition costs are usually accounted for, but related operating and maintenance costs are often neglected or underestimated. The complexity of optimizing individual design problems leads researchers to avoid a global optimization perspective and, thus, the IS architecture is usually a result of the juxtaposition of multiple local optima.*

This paper takes an overall perspective on the cost minimization problem of information system design to achieve a better trade-off between cost and performance over the whole expected life of the technology architecture. A comprehensive design methodology is discussed as an integrating framework that accounts for all categories of costs, including design, implementation, maintenance, and operation, to achieve a globally cost-minimizing solution.

Key Words. *IS architecture, sizing, IT costs, resource allocation, optimization, distributed systems*

1. Introduction

Information System (IS) design constitutes a complex, top-down process which identifies the Information Technology (IT) architecture that satisfies performance requirements and at the same time minimizes costs (e.g., Zachman, 1987; Sowa and Zachman, 1992; Alter, 1996; Blyler and Ray, 1998; Aue and Breu, 1994). The efficiency of the design process and the cost-effectiveness of the resulting architecture are bound to a correct and complete understanding of organizational requirements and to the fit between requirements and technology. Technology components only represent the standard building blocks to construct an IS architecture that satisfies organizational needs and guarantees tangible benefits to the users' organization. System specification, design, and implementation are highly complex and interrelated tasks, but are critical to design a custom solution that satisfies requirements.

Practitioners usually tackle individual design issues and refer to previous sizing experiences to guide overall design. Theoretical researchers and practitioners (e.g., Zachman, 1987; Sowa and Zachman, 1992; Alter, 1996; Blyler and Ray, 1998; Aue and Breu, 1994; Gavish and Pirkul, 1986a; Jain, 1987; Distante and Piuri, 1989; Liu Shang, 1992; Lee, Shi, and Stolen, 1994; Piuri, 1994; Ahituv, Neumann, and Zviran, 1989; Boehm, 1981; Pressman, 1992; Breu et al., 1994) have often searched for specific techniques to optimize individual design tasks and extract experience-based rules to guide architectural design. Often, these efforts are positioned in different research areas and only seldom build on each other to create a cumulative research background that bridges across distinct IS communities

* To whom correspondence shpuld be addressed.

(Coad and Yourdon, 1991; Gavish and Pirkul, 1986; Banker and Slaughter, 1997).

The relationship between organizational requirements and architectural costs has been studied within design methodologies that incrementally translate requirements into technical choices (e.g., Alter, 1996; Aue and Breu, 1994). However, these methodologies focus on technical design issues by analyzing acquisition cost and performance of individual hardware or software components (e.g., Blyler and Ray, 1998; Aue and Breu, 1994; Jain, 1987; Distante and Piuri, 1989; Piuri, 1994). While pointing to relevant technical cost trade-offs, previous analyses have failed to consider management costs, which can be greatly dissimilar across different IT architectures. Over the lifetime of a system, management costs can represent over half of the total cost of ownership of IS components (Alter, 1996). Empirical results (Moad, 1994; Dec, 1996; Simpson, 1997a, 1997b) highlight that management costs can significantly shift the balance of technical cost trade-offs. As an example, the expected cost reductions from the adoption of distributed architectures, which involve low investment costs, have been demonstrated elusive. The main reason for this failure is that management costs are higher in distributed architectures and overall cost reductions depend on organizational requirements, which may or may not favor distribution (Guengerich, 1992).

The relationship between the overall costs of an IT architecture and organizational requirements has not been investigated. The few empirical results that have been published present absolute cost comparisons between centralized and client-server architectures and do not discuss the impact of organizational requirements on findings (Moad, 1994; Dec, 1996; Simpson, 1997a, 1997b). The IS literature has dealt with representational issues of organizational requirements, leaving costs as an issue for subsequent design phases more directly involved in technical choices (De Marco, 1978; Coad and Yourdon, 1991). An aggregate perspective evaluating the impact of organizational requirements on costs has rarely been taken.

As noted before, searching for local optima can easily lead to inaccurate or incomplete system specifications and, thus, to higher acquisition and management costs. This lack of accuracy may be a cause for low performance or for over-sizing to prevent performance degradation as the system is used. The unpredictability of the resulting IS architecture does not allow for planning on scalability and corresponding expenses. In turn, this raises difficulties in subsequent design phases, ultimately resulting into higher overall costs.

In order to achieve a more accurate trade-off between costs and performance, this paper proposes a comprehensive IS design methodology whose goal is the minimization of overall architectural costs. The methodology identifies a sequence of design steps, from requirements analysis to physical implementation, that allow designers to estimate the cost implications of architectural choices and, by evaluating multiple design alternatives, determine the minimum-cost architectural solution. Each design step has several degrees of freedom leading to different IT architectures, ranging from centralized to distributed architectures with different degrees of decentralization of information and processing capacity, with varying cost and performance levels. The methodology guides the designer through these choices to explore multiple alternatives and identify the solution that minimizes costs.

The paper is structured as follows. In Section 2, the overall design methodology is presented by discussing the objectives and design approach of four methodological phases, from organizational requirements to physical design. The four design phases are explained in Sections 3–6, respectively. Finally, Section 7 reports experimental results and discusses their methodological as well as practical implications.

2. *Overview of the Design Methodology*

The architectural design process (Aue and Breu, 1994; Francalanci and Piuri, 1998) ties the information requirements of an organization to the physical (both hardware and software) components necessary to satisfy those requirements. Acquisition, maintenance and operating costs are associated with these physical components (cost are further discussed in Section 6; Table 1 shows the classification of costs associated with physical components). The architectural design process has many degrees of freedom. In most cases, different architectures can be built to satisfy a given set of requirements, ranging from the centralized to distributed architectures with different degrees of decentralization. These degrees of freedom involve choices at different stages of architectural design, traditionally differentiating subsequent design phases.

The proposed methodology integrates these choices within an overall design framework. The information

Table 1. *Summary of cost categories*

Class of cost	Cost items included
Hardware and operating system	Hardware and operating system acquisition Hardware and operating system installation Hardware and operating system testing
Application software	Requirements analysis Acquisition and customization *or* development and verification Integration and installation Testing
Communication	Network lease (for wide area networks) Network installation (for local area networks)
Management	Hardware and operating system maintenance and upgrade Software maintenance and upgrade Network management User training on operating system User training on software applications User technical support (e.g. help desk)

requirements of an organization are translated into a corresponding physical architecture through a sequence of steps. At each step, the designer's attention is focused on a specific aspect of the technical design of the underlying IT architecture, through a top-down refinement process moving from the high-level organizational view of the IS architecture down to physical details.

At an organizational level, requirements are described as a collection of information processes, characterized by a processing intensity, a communication intensity, and a degree of networking among users. All these characteristics—either directly or indirectly—influence IT costs. For example, they typically influence the computing power, the technical features of processing units, the speed of the interconnection links and the amount of information sharing on storage devices and, consequently, IT costs.

Usually, more than one task is executed on each computer in any IT architecture (both centralized and distributed). Grouping tasks and related technological needs becomes necessary to identify the technical features of each computer and to optimize the cost of the overall architecture, while providing user-required performance levels. Mapping organizational requirements onto architectural components leads to designing and rightsizing the IT architecture, through the analysis of multiple alternative solutions with different degrees of distribution of processing, communication, and data storage activities. Our approach allows the comparison of costs across different types of IT architectures, for varying levels of networking and amounts of information exchanged in the IT-supported process. This provides the necessary information for the designer to perform his choices based on quantitative cost analyses.

The methodology considers four design phases, producing four corresponding models of the IT architecture (see Fig. 1):

Information requirements design, defining organizational requirements for information processing capacity.

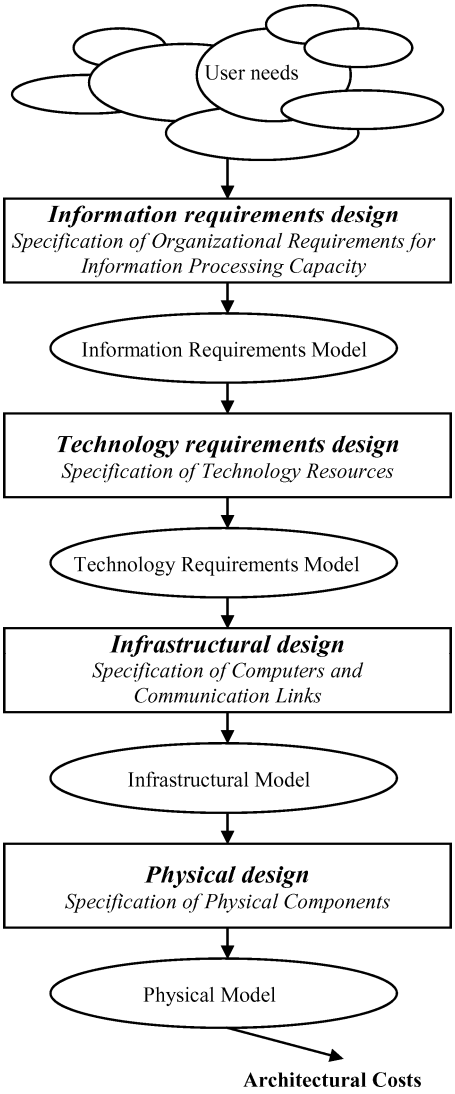


Fig. 1. *Design models and specification activities of the comprehensive design methodology.*

Technology requirements design, defining the information technology resources necessary to satisfy given requirements for information processing capacity.
Infrastructural design, allocating the required information technology resources onto separate and interconnected computers.
Physical design, associating computers and communication links with minimum-size commercially available devices.

These phases produce a corresponding model of the technology architecture, incorporating all the relevant information of the previous design phase and current design decisions. Each model is translated into the subsequent model by a specification activity involving design choices that, in turn, affect the overall cost of the resulting IT architecture. The following Sections describe in detail the goals, design choices, specification activities, and architectural models of each design phase.

3. Capturing Organizational Needs: The Information Requirements Model

Organizational needs can be studied and modelled in different ways according to design goals. In conceptual modeling of information requirements (e.g., De Marco, 1978), a detailed and descriptive representation of information is typically emphasized. Similarly, in software engineering (e.g., Pfleeger, 1998), requirements are specified from functional and operational points of view, with increasing detail through subsequent refinement steps. Functional requirements are critical to guarantee the conceptual completeness and correctness of software. Although including many of the representational constructs from the IS literature (e.g., tasks, information interdependences, complexity), this methodology relies on a more parsimonious representation of cost-differential characteristics of information processes, by limiting the modelling effort to organizational variables that are relevant to the identification of the cost-minimizing architecture.

The *information requirements model* summarizes the essential information needed to size the IT architecture. A preliminary version of this model was introduced in Francalanci and Piuri (1998). Organizational requirements for information processing capacity are modeled as *information processes*. An information process is defined as a set of activities to be performed in an organization whose overall output enables the achievement of an organizational goal. Organizational requirements are expressed as a set of information processes required to achieve all the goals of an organization.

An *information process* (subsequently referred to as *process*) is formally described as a set of *tasks* $\{T_i \mid i = 1 \ldots n\}$ with *input-output interdependences* $\{I_{ij} \mid i, j = 1 \ldots n\}$. A process receives the data to be processed (called *primary input*) from the external environment and generates the desired results (called *primary output*) to be delivered to the external environment and, possibly, used by other processes in the future. Tasks represent the activities to be performed within the process on the primary input to produce the corresponding primary output. Input-output interdependences represent the flows of information from the primary input to the primary output, which involve data dependencies among the tasks composing the process. This also defines the execution order induced by such dependencies. T_j is input interdependent with T_i if T_i produces information I_{ij} that is input to T_j: in this case T_j must be executed after information I_{ij} has been transferred to T_i. All dependencies among tasks are considered to be information interdependences. For the sake of simplicity, the primary input is supposed to be used by only one task of the process (called *entry task*); the input-output interdependence between the external environment and the entry task T_i is labeled with $I_{>i}$. Similarly, only one task (called *exit task*) generates the primary output of the process; the input-output interdependence between the exit task T_j and the external environment is labeled with $I_{j>}$. Information interdependences with entry and exit tasks represent information exchanges with the external environment.

On average a task T_i can be repeatedly accomplished by n_i human executors working in parallel, each of them completing k_i activations of the task in a time unit. This keeps the decomposition of a process into tasks at a conceptual level, while separately representing the number of a task's instances by the same or different individuals. Therefore, on average a task T_i has $n_i k_i$ instances concurrently executed in a time unit.

The quantity of operations performed by a task to accomplish its information processing activity is relevant to define the computational power of each computer in the IT architecture in order to satisfy functional, operational, and time requirements of corresponding

applications. The capacity demand of a task is measured by the *task complexity* χ_i, defined as the number of high-level instructions procedurally describing the task. This definition assumes that a task corresponds to a software application whose procedural high-level requirements are already available at this stage. This one-to-one correspondence simplifies the design process, but it does not affect the final relationship between the structural features of information processes and architectural costs. Possible inaccuracies in the number of high-level instructions can be reduced if they are provided by the same designer for all tasks or are derived in a homogeneous way from the documentation of off-the-shelf software.

The quantity of information exchanged between tasks and with the external environment is essential to understand the application demands influencing storage and networking requirements. Let an information unit be the smallest piece of information that can be exchanged in the organization (e.g., records or information objects). The quantity of information transferred from task T_i to an interdependent task T_j is measured by the *information transfer* ϕ_{ij}, defined as the number of information units exchanged in a time unit. The information transfer of the primary input entering an entry task T_i is denoted as $\phi_{>i}$, while the information transfer of the primary output leaving an exit task T_j is denoted as $\phi_{j>}$. A task T_i can receive input information from multiple output interdependent tasks and convey output information to multiple input interdependent tasks. The total input information transfer ϕ_{*i} of task T_i, i.e., the total quantity of information entering the task, is $\phi_{*i} = \sum_j \phi_{ji}$ Similarly, the total output information transfer ϕ_{i*}, i.e., the total quantity of information leaving the task, is $\phi_{i*} = \sum_j \phi_{ij}$.

If computational complexity and information exchanges are measured by the IT designer as the average, the worst, or the most optimistic values, our modeling approach will derive the IT architecture suitable for the average, the worst, or the best case scenario, respectively. The IT designer can make use of this intrinsic adaptability of the methodology to evaluate the dependency of costs and performance on the workload and the weight of information flows. A sensitivity analysis on these characteristics will allow the choice of a solution that balances costs with performance in case of fluctuations in workload and data exchanges. This assumption is realistic and acceptable in most cases. Analyses based on queuing theory or other similar techniques would be more accurate, but will result in a modeling effort that, in practice, is considered too complex at this level of abstraction.

According to the definitions above, a process can be represented as a labeled directed graph, called *process graph*, as shown in Fig. 2. Nodes and arcs represent tasks and interdependences, respectively. An arc's direction indicates the orientation in which information is exchanged between two interconnected tasks. Task complexity and information exchanges are represented as labels attached to nodes and arcs, respectively. An information system is therefore represented by a collection of process graphs. As an example, Fig. 3 shows the definition of a data entry process implemented by a call center. The process is composed of four basic tasks: call center operators fill an order form to supply goods/services to customers, the order is then verified, information on products and customers are updated and, finally, the result of the operation (failure/success) is notified to the operator. This simple example is taken as a benchmark case study and it will be further

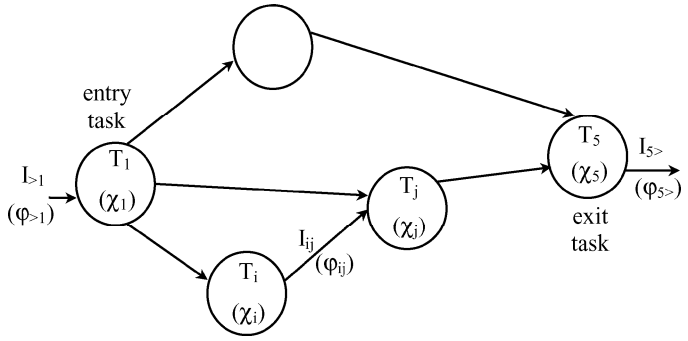


Fig. 2. *The Information Requirement Model: Graphical representation of an information process.*

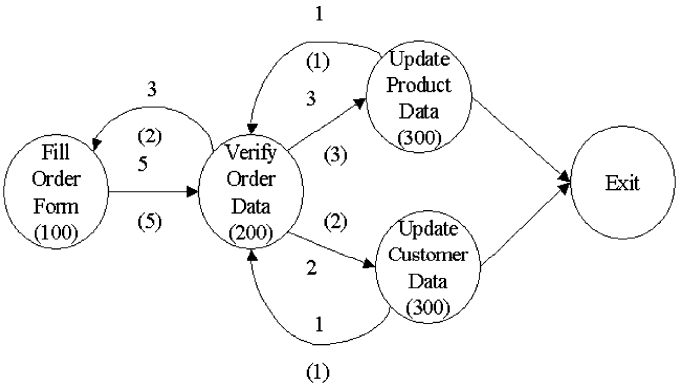


Fig. 3. *Information Requirement Model of a call center. Operators fill an order form, data are then verified and information on products and customers are finally updated. A dummy exit task is introduced, which, in a real system, would link the call center with the production or delivery organizational units. It is assumed that there are n concurrent users and that each user completes only one activation of tasks at a time, that is* $k_i = 1$ *for all i.*

analyzed in the following sections. In Section 6 empirical verifications will show the cost savings that can be obtained by applying the cost-minimization methodology to this case study.

4. Measuring Organizational Requirements: The Technology Requirements Model

Once organizational needs have been represented with the information requirements model, their impact on technology requirements can be identified by translating information processes into a detailed specification of the underlying IT architecture, that is, the set of computers and network components that will support the execution of applications. At this stage, each task's requirements for a technological support are taken into account separately, irrespective of the computer on which the task will be physically executed. This allows for abstracting from the actual architecture that will support the IS and for a more comprehensive analysis of all feasible architectural solutions.

Technology requirements design associates the tasks and the information interdependences of the information requirements model with corresponding technology resources. Namely, we focus on computing, storage (both central and mass memory), and communication resources.

To build the technology requirements model, the information process, composed by tasks and their mutual interdependences, is transformed into the corresponding set of *applications* $\{A_i \mid i = 1 \ldots n\}$ and *data exchanges* $\{D_{ij} \mid i, j = 1 \ldots n\}$. An application describes the computer-supported activities to be performed by a task, while a task is a high-level description of information-processing requirements. Similarly, data exchanges characterize the hardware support and possible network infrastructure among applications, while interdependences model generic needs for sharing data. Applications are classified as client, monolithic and server. The latter can also play the role of client while interacting with other server applications according to the client-server paradigm.

Each task T_i is associated with a software application A_i by a one-to-one correspondence. Entry and exit applications correspond to entry and the exit tasks, respectively. The computational demand of a task, measured by the task's complexity, is translated into a technology-related measure, referred to as *application complexity* α_i. The application complexity is the number of machine instructions to be executed in a time unit by the application. Task complexity is translated into application complexity by a conversion factor β_i from high-level operations to machine-level instructions; the application complexity α_i is therefore given by $\beta_i \chi_i$. If task complexity can be assumed to be homogeneous across tasks, β_i can be considered constant.

The second technological feature that characterizes applications is the amount of memory that is required for their execution. This requirement concerns both the primary (or central) memory and the secondary memory (or mass storage). Each type of memory

requirement is measured in bytes. For application A_i, the quantity μ_i of primary memory includes the space occupied in central memory by the executable code, the data of the current execution, and the execution stack. Similarly, the quantity σ_i of secondary memory includes the storage space in mass storage devices for the executable code, all configuration and support files, archives and possible database components used only—or, at least, mainly—by that task. Memory requirements are tied to application complexity, generally growing with it. However, since the amount of local data may vary, memory requirements are not univocally determined by complexity. These requirements can be obtained by analyzing the characteristics of software applications, for example, as they appear in technical manuals.

From the definition of applications discussed above, the total number of applications evaluates to the total number of tasks n. A one-to-one correspondence between tasks and applications also involves an equal number of executors n_i and activations k_i for tasks and corresponding applications.

A data exchange D_{ij} between applications A_i and A_j corresponds to the information interdependence I_{ij} between tasks T_i and T_j, but represents the quantity of data transferred from the generating to the receiving application through a technology-based communication channel. The interdependences between the entry and exit tasks and the external environment (i.e., $I_{>i}$ and $I_{j>}$) are translated into data exchanges $D_{>i}$ and $D_{j>}$, respectively.

Information exchanges among tasks measured by information transfers ϕ_{ij} are also translated into a machine-related measure, that is a *memory data transfer* δ_{ij} and a *storage data transfer* τ_{ij}, if they occur in primary or secondary memory, respectively. A data transfer is the number of bytes to be transferred from application A_i to A_j in a time unit. The input data of the entry application A_i are denoted as $\delta_{>i}$ and $\tau_{>i}$, for primary and secondary memory, respectively. Similarly, the output data generated by the exit application A_j are denoted as $\delta_{j>}$ and $\tau_{j>}$.

An information interdependence is transformed into a data transfer by specifying a conversion factor γ from the high-level information unit to the machine-level data unit (e.g., from records to bytes). Since information interdependences are defined as multiples of the same unit of information, γ is supposed to be constant across interdependences, both between tasks and with the external environment. If the information transfer occurs in primary memory, $\delta_{ij} = \gamma\phi_{ij}$, $\delta_{>i} = \gamma\phi_{>i}$ and $\delta_{j>} = \gamma\phi_{j>}$. Similarly, for data transfers occurring in secondary memory, $\tau_{ij} = \gamma\phi_{ij}$, $\tau_{>i} = \gamma\phi_{>i}$, and $\tau_{j>} = \gamma\phi_{j>}$. The total input data of application A_i receiving data through primary memory from two or more applications is

$$\delta_{*i} = \sum_{A_j \in M_i} \delta_{ji} = \gamma \sum_{A_j \in M_i} \phi_{ji},$$

where M_i is the set of applications which are output interdependent with A_i and exchange data through primary memory with A_i.

The total output data leaving application A_i towards two or more applications through the primary memory is

$$\delta_{i*} = \gamma \sum_{A_j \in N_i} \phi_{ij},$$

where N_i is the set of the applications which are input interdependent with A_i and exchange data through primary memory with A_i. Similar definitions hold for data transfers performed through secondary memory.

The characterization of applications and data exchanges is a critical activity to be performed by an IT designer. An accurate choice of conversion parameters is essential to obtain an accurate model of the IT architecture and, consequently, correct cost/performance evaluations. As for the information requirements model, the designer can adopt an average, worst or best-case perspective on system sizing.

The technology requirements model can be represented as a labeled directed graph, called *application graph*, as shown in Fig. 4. Nodes represent applications, while arcs represent interdependences. An arc's orientation indicates the direction of a data exchange between two applications. The application complexity and the quantities of primary and secondary memory are used to label nodes, while arcs are labeled with data transfers. Fig. 5 shows the technology requirements model for the call center example (Fig. 3). The fill order task is associated with a browser, the verify order task is implemented by a web server; the update task maps into two different DBMS systems. It is assumed that all information transfers are implemented as primary memory transfers.

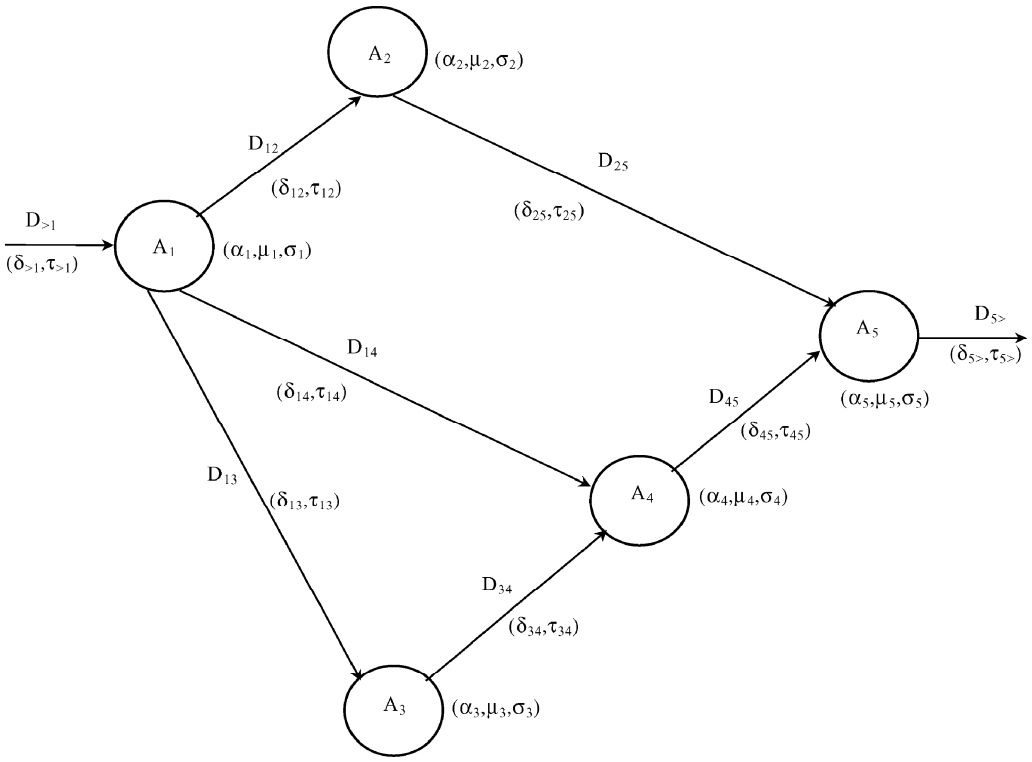


Fig. 4. *The Technological Requirements Model: Graphical representation of applications.*

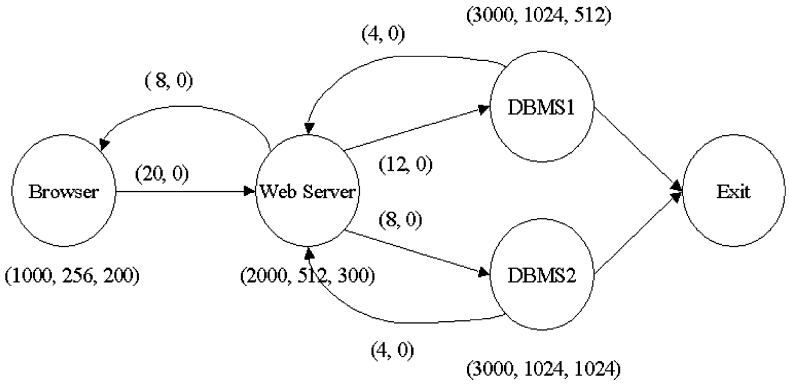


Fig. 5. *The Technological Requirements Model of the call center example (Fig. 2a). It is assumed that β evaluates to 10 for Web and DBMS applications. Data transfers are derived from the Information Requirements Model under the assumption that $\gamma = 4$.*

5. Allocating Tasks on Computing Resources: The Infrastructural Design Model

As noted before, the technology requirements model is independent of the physical computers executing applications. This high-level view is similar to the perspective often adopted to describe requirements for the task allocation problem in distributed computing systems (e.g., Distante and Piuri, 1989; Piuri, 1994). To define the technical features of computers, the set of applications that should be executed by each computer must

be specified. This allows the definition of the hardware resources that should be provided by each computer in order to fulfill requirements in terms of information processing capacity and performance.

For this purpose, the applications defined in the technology requirements model must be allocated onto the computers of the IT architecture. Allocation must be performed so as to minimize the overall cost of the IT architecture, while providing the required performance in order to guarantee the practical usability of the system. In the literature concerning software allocation, this optimization problem is usually tackled by assuming that the IT architecture has already been defined in terms of number and type of computers and related network infrastructure. As a consequence, the performance with which applications are executed on different computers is the only decision variable. Unfortunately, in our case—as well as in Distante and Piuri (1989) and Piuri (1994)—this assumption cannot be made. We are building the IT architecture *while* allocating software applications. Thus, the correct sizing of the IT architecture *and* the simultaneous software allocation are a more complex optimization problem with more degrees of freedom with respect to a traditional software allocation problem. Our problem involves not only choices on software allocation, but also decisions on the computers composing the IT architecture, including their number, type, primary and secondary memory, and possible network interconnections, in order to minimize the overall cost with the required performance.

Infrastructural design is the phase of our methodology aimed at allocating software applications and, at the same time, rightsizing the IT architecture by identifying the optimum size of its components. In Distante and Piuri (1989), a similar rightsizing problem for real-time applications in the area of industrial plant control was shown to be NP-complete. By analogy, the rightsizing of the IT architecture of this paper belongs to the class of NP-complete problems. As a consequence, the identification of the global optimum solution cannot be obtained in a reasonable time, since the number of configurations to be analyzed increases more than polynomially with the number of applications and computers. Thus, an accurate, but efficient heuristic strategy must be adopted. For example, the approaches analyzed in Distante and Piuri (1989) (encompassing branch&bound algorithms, hill-climbing heuristics) can be effectively used and extended to deal with IT architectural design.

At each step, any optimization strategy that simultaneously sizes the IT architecture and allocates software applications must generate a solution, evaluate its performance against requirements, and compare costs with the lowest-cost solution identified until that moment during the exploration of the solutions' space. To achieve this goal, the technology resources specified in the technology requirements model for the whole information system are grouped into subsets. The computer on which a subset of applications is mapped will have to execute all these applications. The combined requirements of each one of these subsets specify the minimum characteristics that such a computer should possess. This will allow the identification of the smallest computer with such characteristics and to evaluate its costs. Client and monolithic applications are assigned to PCs, while server applications are assigned to either independent servers or server farms, depending on costs.

The minimum characteristics defined for each of the subsets above are the *processing capacity* and *memory capacity* of each computer as well as the *communication capacity* of each interconnection link between computers. The processing capacity E is the minimum number of instructions that the processor of a computer must be able to execute in a time unit in order to provide the required performance for the set of applications allocated on it. The memory capacity is the minimum amount of primary (P) and secondary (S) memory that must be present in the computer in order to run its applications and to store related data, according to the applications' specifications. The communication capacity L is the minimum amount of data that should be transferred through a link to execute the applications that use that link.

First, let us consider the processing and memory capacity of a computer. Being assigned a number n_i^t of users of an application A_i from the technology requirements model, the computer underlying the subset containing A_i should guarantee that:

- each activation of A_i is attributed μ_i and σ_i shares of primary and secondary memory, respectively;
- $n_i^t k_i$ activations of A_i can be executed in a time unit.

In order for $n_i k_i$ activations of A_i to be executed in a time unit, processing capacity E should be at least $n_i^t k_i \chi_i$ machine-level instructions per time unit. If multiple applications $\{A_1, \ldots, A_q\}$ are assigned to the same computer, processing capacity must be at least $E = \sum_{i=1\ldots q} n_i^t k_i \chi_i$. If k_i measures the number of

activations per second, $n_i^t k_i \chi_i$ expresses the processing capacity in MIPS (millions of machine-level instructions per second). If different types of processors are adopted in the IT architecture (in particular, if RISC processors are adopted), a suitable conversion factor φ_i should be introduced to deal with a possibly different complexity of machine-level instructions among processors. Therefore, processing capacity should at least evaluate to $n_i^t k_i \varphi_i \chi_i$. For a multi-processor computer, the minimum processing capacity represents the minimum value of the effective summation of the capacity of individual processors (i.e., without possible operating interferences among processors that can be estimated by means of benchmarks such as SpecIntRate (Menascé, Gomaa, and Kerschberg, 1995; Menascé and Almeida, 1998; Hennessy and Patterson, 1994).

Similarly, the memory requirements for the computer underlying the subset of applications $\{A_1, \ldots, A_q\}$ is the summation of the primary and secondary memory requirements of all activations of these applications, that is $P_A = \sum_{i=1..q} n_i^t k_i \mu_i$ and $S_A = \sum_{i=1..q} n_i^t k_i \sigma_i$, respectively. Secondary memory requirements are supposed to include an estimate of the data created by applications and possibly stored in a common database. By referring to data created as opposed to data accessed by applications, the summation of secondary memory requirements across applications involves minimal redundancy.

Data exchanges are used to size the communication infrastructure. If applications A_i and A_j are allocated on the same computer, data exchange D_{ij} is *local*, that is, it takes places within a computer. Otherwise, it is *remote*, that is it requires a communication infrastructure among computers. Local and remote data exchanges involve different memory requirements. A local data transfer in primary memory can take place in an area of central memory—having size δ_{ij}—shared between the two applications; δ_{ij} must be added to the minimum requirements of primary memory for the underlying computer. Similarly, a (possible, but rarely used) local data transfer in secondary memory induces the need for τ_{ij} additional secondary memory on that computer. A remote data transfer requires a quantity of memory δ_{ij} on both communicating computers since one transmission buffer and one reception buffer are necessary for A_i and A_j, respectively. Remote data transfers in secondary memory are supposed to avoid the duplication of data since the operating systems can manage the low-level block transmission with a small (and negligible) need for central memory for temporary buffers. In this case, τ_{ij} must be taken into account in the secondary memory of only one of the communicating computers (the one on which data are assumed to be permanently stored).

In summary, for the communication involving applications belonging to the subset S allocated on computer C_l, the required primary memory is:

$$P_{C_l} = \sum_{i \mid A_i \in S} \delta_{ij} + \sum_{j \mid A_j \in S, A_i \notin S} \delta_{ij}.$$

Similarly, the additional secondary memory required for communication is:

$$S_{C_l} = \sum_{i \mid A_i, A_j \in S} \tau_{ij} + \sum_{i \mid A_i \in S, A_j \notin S, D_{ij} \subset C_l} \tau_{ij} + \sum_{j \mid A_j \in S, A_i \notin S, D_{ij} \subset C_l} \tau_{ij},$$

where $D_{ij} \subset C_l$ indicates that memory for data exchange D_{ij} is allocated on computer C_l. The previous formulas take into account the data transfers both from and to the external world (i.e., $\delta_{>i}$, $\delta_{j>}$, $\tau_{>i}$, and $\tau_{j>}$), when the symbol $>$ is considered as one of the application identifiers. The total primary memory capacity is $P = P_A + P_C$, while the total secondary memory capacity is $S = S_A + S_C$.

If a distributed architecture is adopted, the communication among applications allocated on different computers determines the capacity of the interconnection links among computers. In turn, this determines the cost of these links. To support the data exchange between subsets S_1 and S_2 of applications allocated on computers C_1 and C_2, respectively, the minimum capacity of the communication link between these computers must be:

$$L_{12} = \sum_{i \mid A_i \in S_1, A_j \in S_2} \delta_{ij} + \sum_{j \mid A_j \in S_2, A_i \in S_1} \delta_{ji} + \sum_{i \mid A^i \in S_1, A_j \in S_2} \tau_{ij} + \sum_{j \mid A_j \in S_2, A_i \in S_1} \tau_{ji}$$

The IT architecture resulting from the allocation of technology resources can be represented as a labeled undirected graph associated with the infrastructural model, called *infrastructural graph*, as shown in Fig. 6. Nodes represent the computers of the distributed system. Arcs represent communication links between computers. Nodes are labeled with their processing

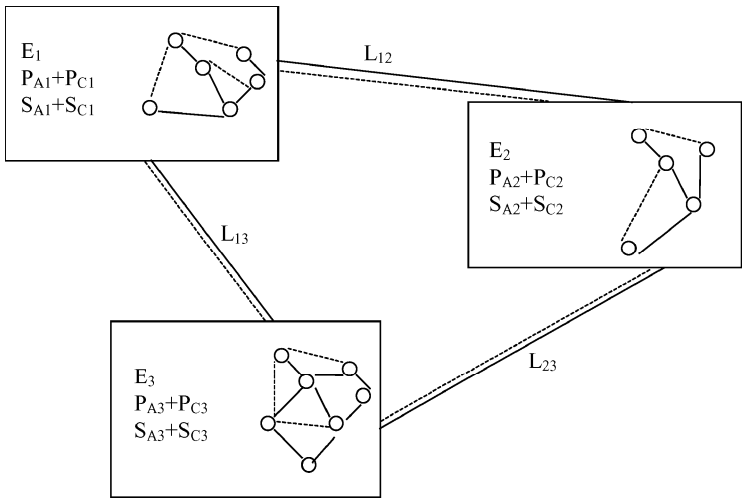


Fig. 6. *The Infrastructural Model: Graphical representation of the IT infrastructure.*

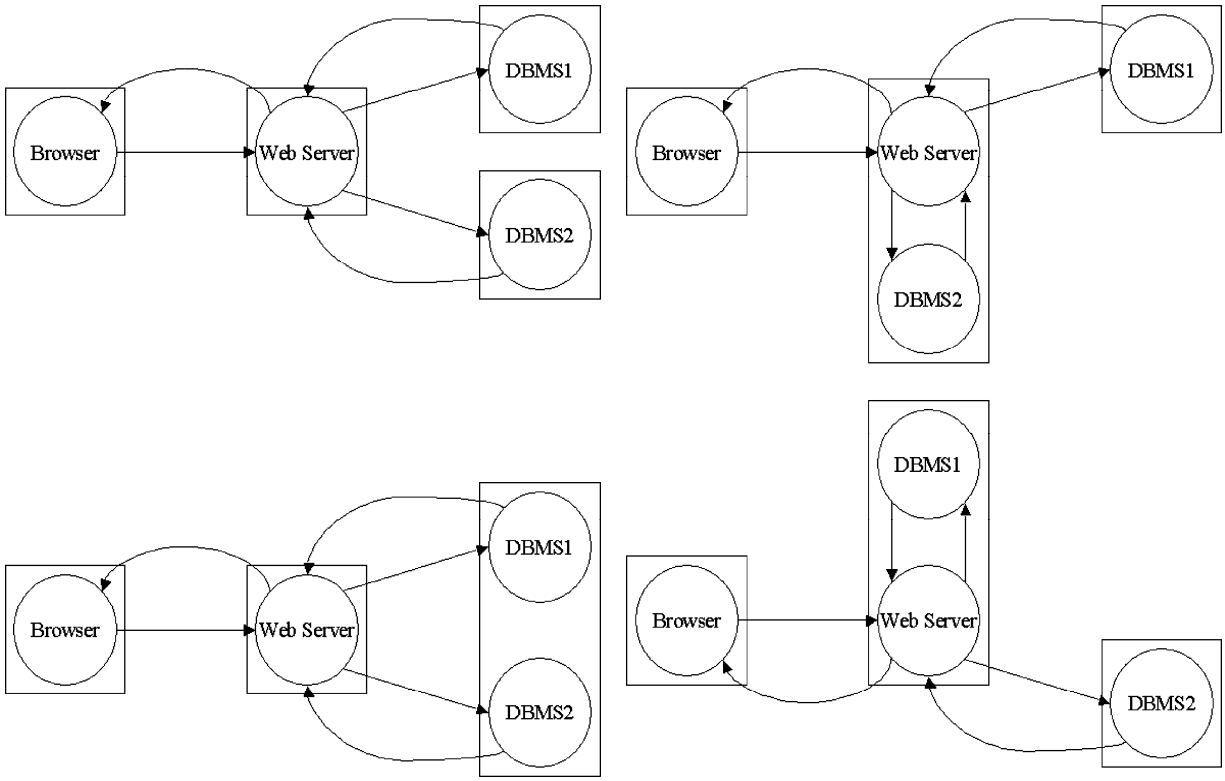


Fig. 7. *Alternative infrastructural solutions evaluated for the call-center case study.*

capacity E, their primary memory P, and their secondary memory S. Arcs are labeled with their communication capacity L. The infrastructural graph can be viewed as a hierarchical graph in which each node corresponds to a portion of the application graph that represents the subset of applications allocated onto that node.

Alternative solutions are built by considering all possible groupings of applications. Fig. 7 shows the

set of alternative solutions that is evaluated for the call center example. The browser is always executed by personal computers of call center operators. Vice versa the Web server and DBMS server applications can be allocated on the same server or on multiple servers. Note how this simple example generates multiple four-node solutions.

6. Designing the Physical Architecture: The Physical Model

The infrastructural model specifies the minimum technological resources that must be provided by each computer of the IT architecture. This model implicitly guarantees that performance requirements are satisfied, since processing capacity describes the speed at which an application must be executed. To identify the cost-minimizing solution among alternative infrastructural models, the total cost of ownership must be calculated. The total cost of ownership encompasses the acquisition cost of hardware, operating system, system software, application software, maintenance, management, personnel, and training. This represents the set of costs used by the optimization algorithm to explore the space of possible solutions.

The infrastructural model specifies the *minimum technology resources* that the computers and the network must provide, while it does not indicate the *exact* amount of resources provided by commercial computers and network components that will be used to implement the architecture. The actual technological resources of each computer may be greater than minimum requirements, since commercial resources have a discrete distribution over requirements dimensions. For example, commercial personal computers have a discrete distribution over computing capacity. Therefore, the physical components to be associated with requirements specified by the infrastructural model are the smallest hardware configuration with computing, storage and communication capacities greater than (or, in a few cases, equal to) requirements.

Note that, if a designer needs to take into account the possibility of expanding the architecture to support increasing requirements, the smallest hardware configuration may not represent the cost-minimizing solution over time. This version of the methodology does not support the specification of scalability requirements. However with this version of the methodology, a first rough estimate of scalability can be easily obtained by increasing current technology requirements by a scalability factor and, thus, oversizing the architecture.

Physical design is the methodological phase that identifies the smallest physical configuration of the IT architecture for a given infrastructural model. Infrastructural design associates with infrastructural components the smallest commercially available computers and communication resources that satisfy requirements. The result of this design activity is the *physical model* of the IT architecture. This model includes commercial components and, thus, supports the evaluation of the total cost of the IT architecture.

The result of this phase is typically vendor dependent, since each manufacturer offers a different set of components with distinct basic configurations, as well as different versions of similar off-the-shelf application software. The physical model and the corresponding evaluation of costs is also time dependent, since technology is offered at continuously decreasing costs.

We consider the following technological and management costs for each component (computers, communication links, and software applications) of the IT architecture (see also Table 1):

hardware costs, encompassing all costs related to the acquisition and installation of the hardware architecture, namely, hardware acquisition, operating system acquisition, hardware installation, operating system installation, hardware testing, operating system testing, and user training for hardware and operating system;
software costs encompassing all costs related to application software acquisition and installation, including, requirements analysis, software acquisition or software development and verification, software installation, software integration, software testing, and user training for software applications;
management costs encompassing all costs related to running and maintaining the system, including, technical support for hardware and operating system, technical support for applications, hardware maintenance and upgrade, operating system maintenance and upgrade, application software maintenance and upgrade, operating system management, network management, and application software management.

Hardware costs are associated with computers, possible expansion components and communication links.

Software costs are associated with applications. Management costs are associated with computers, communication links, and applications.

Fig. 8 shows the set of alternative physical models that are evaluated for the call center example. The inclusion of all costs in the evaluation of the total cost of the IT architecture is the main innovation of our approach. Total costs are reported in Fig. 8 for all alternative physical models. Note that practical choices are often based on initial acquisition costs, which are tangible and can involve a considerable investment. However, in large ISs, the cost of operating and maintaining the IT architecture can be comparable with initial acquisition costs, even if it is distributed over the system's life time. This reflects into a substantial cost variance across alternative physical models in Fig. 8.

7. Experimental Results

To verify the feasibility and effectiveness of the proposed methodology, a vendor-independent database of commercial components has been built and a prototype software application has been developed to support requirements specification and cost minimization. The following sample of physical components and related cost data has been considered:

1100 PC configurations from 4 major vendors.
9000 server configurations from 4 major vendors.
1000 switch configurations form 3 major vendors.
50 backbone link node router configurations form 3 major vendors.

The technical characteristics and costs of these components have been obtained from official technical brochures and price lists of vendors. Interviews have been conducted with vendors to complement their official documentation.

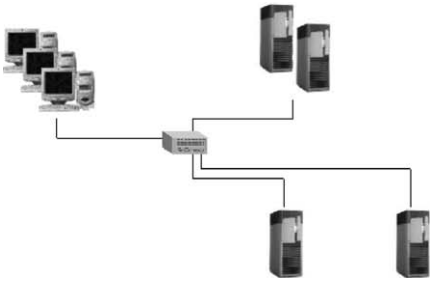
Management costs of hardware components have been evaluated as a percentage of acquisition costs (Redman, Kirwin, and Berg, 1998). Numerous benchmarks of PCs' annual management costs have been published in the literature, ranging between 5 and 15% of acquisition costs. The average value of these professional benchmarks, that is 10%, has been considered for empirical verifications. Annual management cost of servers have been estimated as in Meta Group (1999). Once again the variability across empirical benchmarks discussed in Redman, Kirwin, and Berg (1998) has been resolved by considering an average 20% value of acquisition costs to assess annual management costs for empirical verifications.

Standard off-the-shelf applications have been considered. The sample of applications is comprised of 15 largely used applications for office automation, accounting, sales, customer service, reporting, and management, produced by Microsoft, Oracle, SAP, IBM, Digital Equipment, Hewlett Packard, and Bull. Technical characteristics and costs have been obtained from the vendors' web sites and through ad hoc interviews.

The complexity of design according to our methodology is mostly related to the amount of information that must be processed and stored. Besides, the search for an optimum solution requires automatic support due to the number of alternative solutions to be explored and compared. Sample windows of the prototype software tool implementing the methodology are shown in Fig. 9. As noted before, the cost-minimization problem is computationally complex, as the degrees of freedom in building an infrastructural model satisfying technology requirements grow exponentially with the number of optimization variables. Due to this complexity, optimization is based on the *tabu search* meta-heuristic algorithm (Glover aand Laguna, 1997).

By using our design environment, we have configured ISs with a wide variety of characteristics of complexity, users, and performance requirements. Overall 10.000 tests were generated. Each test is comprised of a varying number of tasks (ranging from 4 to 100) with task complexity varying from 100 to 350. The topology of input-ouput interdependences was generated, with I_{ij} ranging between 0 and 10. MIPS and RAM requirements for applications were derived from software vendors documentation and web sites. Costs were evaluated over a 3-year system lifetime. To evaluate the cost savings obtained through the methodology, each optimum solution was compared with a corresponding solution that can be obtained by applying empirical guidelines published in the professional literature that is:

All application servers should be centralized in one site, to minimize management costs (The Robert Frances Group, 1999; HP, 2002; IBM, 2002).
Applications should be allocated with the maximum number of tiers, to minimize hardware acquisition costs (Dewire, 1997; Umar, 1997).



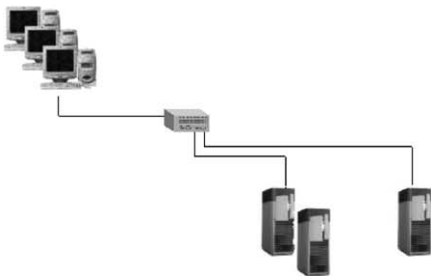
PC:
Ntrnbcr of clients 30
Pill 800 RAM 256MB, HD 20GB

Web Server:
Ntrnbcr of servcrs 3
1-way PIII 833 RAM 512MB, HD40 GB

DBMS 1:
Ntrnbcr of servcrs 1
1-way PIII Xeon 700RAM 1024MB, HD 40GB

DBMS2:
Ntrnbcr of servers 1
1-way PIII Xeon 700 RAM 1024 MB, HD 40 GB

Overnll cost $442764

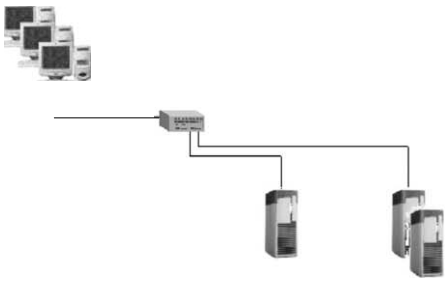


PC:
Nwmer of clients 30
l'III 800 RAM 256MB, HD 20GB

Web Setvcr- DBMS 1:
Nwmer of servcrs 4
2-way Pill Xern 700 RAM 1536MB, HD 40GB

DBMS2:
Nwmer of servcrs 1
4-way Pill Xern 700 RAM 1024 MB, HD 40 GB

Overnll ca;t: $416.254

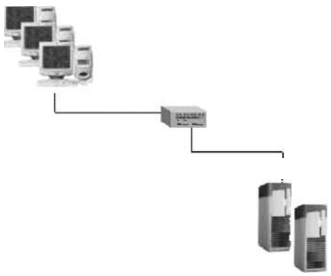


PC:
Ntrnbcr of clients 30
Pill 800 RAM 256MB, HD 20GB

DBMS 1:
Ntrnbcr of servcrs 1
4-way PIII Xeon 700 RAM 1024MB, HD 40GB

Web Server - DBMS 2:
Ntrnbcr of servers 4
2-way PIII Xeon 700 RAM 1536MB, HD 40GB

Overall cost $416.254



PC:
Nwmer of clients 30
l'III 800 RAM 256MB, HD 20GB

Web Setvcr - DBMS 1 – DBMS 2:
Nwmer of servcrs 6
2-way Pill Xern 700 RAM 2560 1\IIB, HD 40 GB

Overnll C<J3t: $ 250.208

Fig. 8. Alternative physical models evaluated for the call-center case study. Costs are evaluated over a 3-year period. The minimum-cost architectural solution delivers over 70% savings.

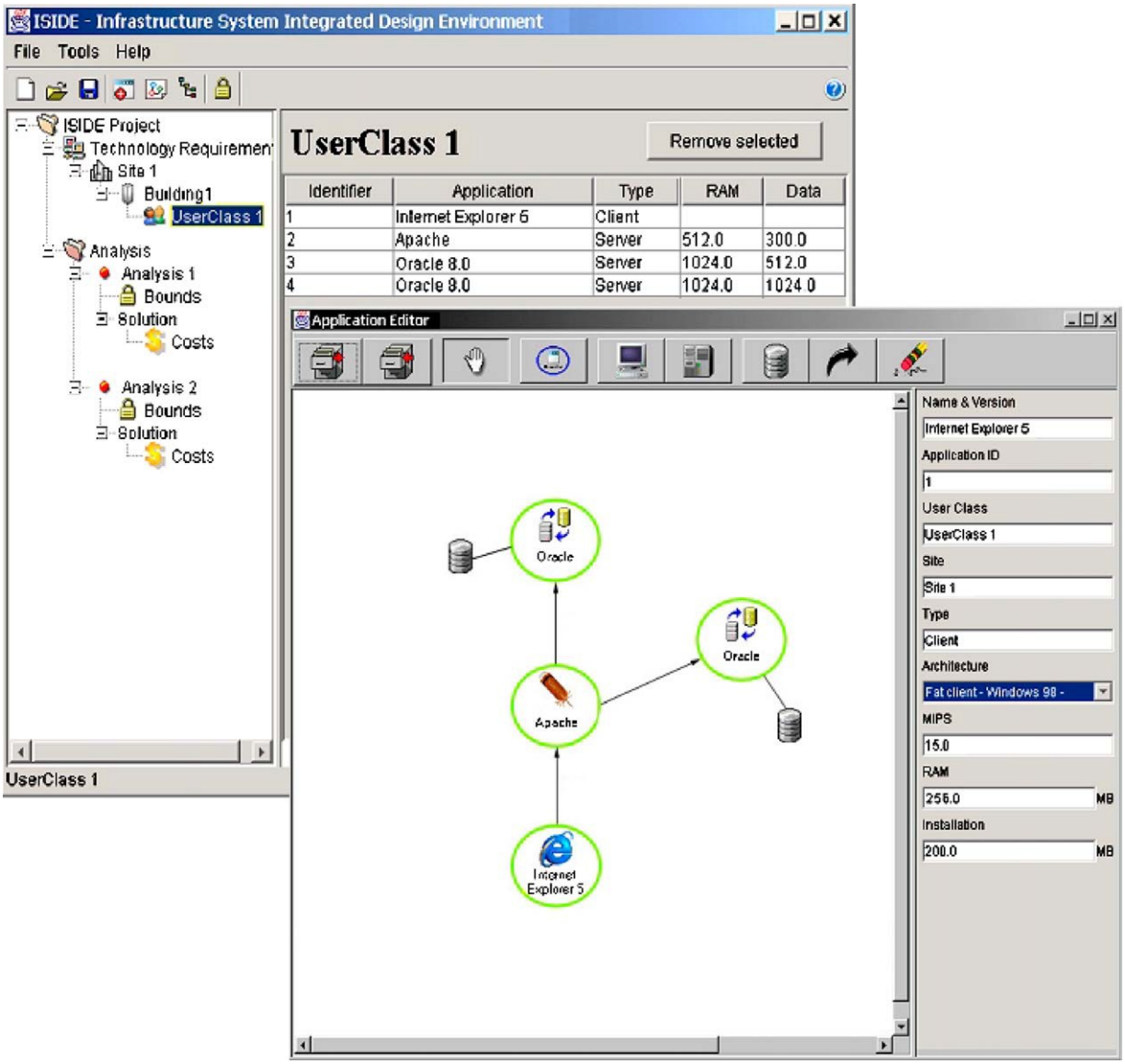


Fig. 9. *Main window and sample representation of the call-center case study.*

Server farms should be built with the smallest server in the database that can support the execution of applications, to minimize hardware acquisition costs (Scheier, 2001).

Average savings are calculated as the mean value of cost variations between the minimum-cost solution and the solution obtained by applying empirical design guidelines. Empirical tests have shown that on average 30% cost reductions can be obtained.

8. *Conclusions*

This paper has presented a comprehensive methodology to support the cost-oriented design of information technology architectures. The main goal was the definition of a single design framework in which all design activities and choices can be performed in an integrated way, with clear relationships among design decisions and their effects on the resulting architecture and corresponding costs. In particular, the methodology

proposed in this paper takes into account all architectural costs, including acquisition, development, maintenance, management, and training, spanning over the whole expected operating life of the system. Preliminary empirical results indicate that general design guidelines are difficult to infer from empirical results and current professional rules are challenged by methodological findings. With respect to previous results in the literature, a systematc approach to cost minimization seems to allow a finer comparison across architectures.

The computer and cost database built as part of this research includes a rather large set of components and corresponding costs. Due to the natural and intrinsic evolution of both IT and related costs, this database contains information which is subject to obsolescence. However, the methodology and the software tool are general and less subject to change over time.

Validation with practical cases is an essential activity in future developments of this research. Additional testing for a broader range of process features is also required in order to generalize results. The tool should be extended to allow for the specification of scalability requirements and related optimization activities. The modularity of the proposed approach will also allow for further extensions dealing with organizational and user related aspects, such as legacy systems, organization restructuring, systems' acceptance, and productivity. In turn, this will support a more accurate analysis of real cases and prediction of costs so as to support more conscious and far-sighted design decisions.

References

Ahituv N, Neumann S, Zviran M. Factors affecting the policy for distributing computing resources. *MIS Quarterly* 1989;Dec.:389–400.

Alter S. *Information Systems: A Management Perspective*. The Benjamin/Cummings Publishing Company, Inc.

Aue A, Breu M. Distributed information systems: An advanced methodology. *IEEE Trans. on Software Engineering* 1994;20(8):594–605.

Banker R, Slaughter S. A field study of scale economies in software maintenance. *Management Science* 1997;43(12):1709–1725.

Blyler JE, Ray GA. What's size got to do with it? *Understanding Computer rightsizing*. IEEE Press, Understanding Science & Technology Series.

Boehm, *Software Engineering Economics*. Prentice Hall, 1981. Breu M, Aue A, Hall J, Robinson K. *Distributed Systems: Application Development*. HSMO Information Systems Engineering Library, London, 1994.

Coad P, Yourdon E. *Object-Oriented Analysis*. Yourdon Press Computing Series, 1991.

Dec K. Gartner view: Client/server payoff. *CIO* April 15, 1996:72–76.

De Marco T. *Structured Analysis and Systems Specification*. Englewood Cliffs, N. Y. Prentice Hall, 1978.

Dewire DT. *Second-Generation Client/Server Computing*. McGraw Hill, 1997.

Distante F, Piuri V. Hill-climbing heuristics for optimal hardware dimensioning and software allocation in fault-tolerant distributed systems. *IEEE Trans. on Reliability* 1989;38(1):28–39.

Francalanci C, Piuri V. Designing information technology architectures: A cost-oriented methodology. *Journal of Information Technology* 1999;14:181–192.

Gavish B, Pirkul H. Computer and database location in distributed computer systems. *IEEE Trans. on Computers* 1986;35(7):583–590.

Gavish B, Pirkul H. Computer and database location in distributed computer systems. *IEEE Trans. on Computers* 1986;35(7):583–590.

Glover FW., Laguna M. *Tabu Search*. Kluwer Academic Publishers, 1997.

Guengerich S. *Downsizing Information Systems*. SAMS-Prentice Hall Computer Publishing, 1992.

Hennessy JL, Patterson DA. *Computer Organization and Design*, Morgan Kaufmann Publishers, 1994.

HP. (2002), *The Scope of HP Systems Consolidation*. http://www.hp.com/products1/unixservers/solutions/sc/scope.html. IBM. (2002), *Improve the Return on Your IT Investment with Server Consolidation*, http://www-1.ibm.com/servers/eserver/iseries/australia/serv.htm.

Jain HK. A comprehensive model for the design of distributed computer systems. *IEEE Trans. on Software Engineering* 1987;13(10):1092–1104.

Lee H, Shi Y, Stolen J. Allocating data files over a wide area network: Goal setting and compromise design. *Information & Management* 1994;26:85–93.

Liu Sheng OR. Optimization of file migration policies in distributed computer systems. *Computers Ops. Res.* 1992;19(5):335–351.

Menascé DA, Almeida VAF. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice-Hall, 1998.

Menascé DA, Gomaa H, Kerschberg L. A performance oriented design methodology for Large-Scale distributed data intensive information systems. In: *IEEE International Conference on Engineering of Complex Computer Systems*, 1995.

Meta Group. *Next-Millennium Platform Infrastructure Strategies: Evaluation, Consolidation, and Costing*, 1999. www.metagroup.com.

Moad J. Client/server costs: Don't get taker for a ride. *Datamation* Feb. 15, 1994:34–41.

Pfleeger SL. *Software Engineering: Theory and Practice*. Prentice Hall, 1998.

Piuri V. Design of fault-tolerant distributed control systems. *IEEE Trans. on Instrumentation and Measurements* 1994;43(2):257–264.

Pressman R. *Software Engineering A Practitioner's Approach*, 3rd ed. McGraw-Hill, Inc., 1992.

Redman B, Kirwin B, Berg T. *TCO: A Critical Tool for Managing IT*, Gartner Group, 1998. www.gartnergroup.com.

Scheier RL. Scaling up for e-commerce. *Computerworld*, 2001:http://www.computerworld.com/softwaretopics/software/-appdev/story/0,10801,59095,00.htm.

Simpson D. Cut desktop management costs!. *Datamation* Jan. 1997a:102–105.

Simpson D. Will NCs save you a bundle?. *Datamation* May, 1997b:100–105.

Sowa JF, Zachman JA. Extending and formalizing the framework for information system architecture. *IBM Systems Journal* 1992;31(3):590–616.

The Robert Frances Group. (1999), *Server Consolidation Strategies*, http://www-1.ibm.com/servers/solutions/serverconsolidation/-pdf/robertfrancesgroup.pdf.

Umar A. *Object-Oriented Client-Server Internet Environments*. Prentice-Hall, 1997.

Zachman JA. A framework for information system architecture. *IBM Systems Journal* 1987;26(3):276–292.

Danilo Ardagna is a Ph.D. student at Politecnico di Milano, where he also graduated in Computer Engineering in December 2000. He has worked for a 6-month period at IBM T.J. Watson Research Center in the System Optimization Department. His research interests include IT infrastructural design and cost minimization, capacity planning and scalability.

Chiara Francalanci is an associate professor of information systems at Politecnico di Milano. She has a master's degree in Electronic Engineering from Politecnico di Milano, where she has also completed her Ph.D. in Computer Science. As part of her post doctoral studies, she has worked for two years at the Harvard Business School as a Visiting Researcher. She has authored articles on the economics of information technology and on feasibility analyses of IT projects, consulted in the financial industry, both in Europe and the U.S., and is a member of the editorial board of the Journal of Information Technology.

Vincenzo Piuri obtained the Ph.D. in Computer Engineering in 1989, at Politecnico di Milano, Italy. From 1992 to September 2000, he was Associate Professor in Operating Systems at Politecnico di Milano. Since October 2000 he is Full Professor in Computer Engineering at the University of Milano, Italy. He was Visiting Professor at the University of Texas at Austin during the summers from 1993 to 1999. His research interests include distributed and parallel computing systems, computer arithmetic, application-specific processing architectures, digital signal processing architectures, fault tolerance, neural network architectures, theory and industrial applications of neural techniques for identification, prediction, control, signal and image processing. Original results have been published in more than 150 papers in book chapters, international journals, and proceedings of international conferences. He is Fellow Member of the IEEE and member of ACM, INNS, AEI. He is Associate Editor of the IEEE Transactions on Neural Networks and the Journal of Systems Architecture. He is Vice President for Publications of the IEEE Instrumentation and Measurement Society, Vice President for Members Activities of the IEEE Neural Networks Society, and Member of the Administrative Committee both of the IEEE Instrumentation and Measurement Society and the IEEE Neural Network Society.