SPP-OL: An Object Library for Software Performance Engineering within SSADM[1]

Bálint Molnár
Information Technology Foundation
of Hungarian Academy of Sciences
29-33 Konkony-Thege Road,
H-1525 Budapest 114
HUNGARY

Abstract: Although, the structured methodologies are widely used in information systems applications, the performance prediction methods and algorithms are rarely used by system analysts and designers in their evaluation of the design alternatives. It is the intent of this paper to outline a comprehensive framework in the form of an appropriate methodology and to create an object library to support the decision making process.
The proposed methodology is interfaced to the SSADM (Structured Systems Analysis and Design Method) and has detailed instructions on what data should be collected and which source to set up a performance model from. Based on these facts, the design alternatives can be defined in the form of objects.
An object library and the related methods are developed to support the performance modelling of design options. The methods of the objects incorporate the best concepts from existing performance prediction algorithms and computational methods. The various objects provide ways for analysing distinct aspects of information systems as software *service demand*, transactions *resource requirement* depending on the data distribution, the average *response time* of the system based on patterns of functions usage. The object-oriented programming environment in which the object library is implemented, yields the opportunity for re-usability and enhancement to fit into some
particular requirements. A proposal for knowledge-based approach is outlined as well.
Keywords: Software Engineering, Information Engineering, SSADM, Performance Engineering, Capacity Planning, CASE

Bálint Molnár was born in 1956. He got his M.Sc. degree in Mathematics from the Lorand Eötvös University in Budapest. In 1981 after graduating, he started work at the Central Research Institute for Physics of Hungarian Academy of Sciences (MTA-KFKI), the Institute for Measurement and Computing Techniques. From 1981 to 1984 he participated in and led the development team for the Hungarian ADA compiler project. From 1985 to 1992 he was concerned with designing information systems for the Hungarian Government. He also started be interested in the Artificial Intelligence field, co-operating in this very direction with a Romanian research group. The use of AI in software engineering, more precisely in information engineering, has been paid special attention.
From 1988 on he has delivered special courses on AI and Information Systems Design at the Lorand Eötvös University, and partook in a regular software engineering course for informatics students at the Technical University of Budapest (TUB), and at the University of Economics in Budapest.
Currently, he is a principal consultant at the Information Technology Foundation of Hungarian Academy of Sciences, and Senior Lecturer at the Budapest University.
He is a member of the Board of the Specialist Group on Knowledge-Based Systems of John von Neumann Society for Computing Science. He represents his country in the Advisory Board of AI Communications , the European Journal on Artificial Intelligence. He is also a member of International Association of Knowledge Engineers (IAKE).
He acts as a referee for the " Software and Information Technology" Journal.
In 1989 and 1992 respectively, he got "Certificates of Proficiency in SSADM Version 3 and Version 4" from the Information Systems Examinations Board of British Computer Society . For the work carried out on software technology and artificial intelligence, in 1992 he was granted the László Kalmár Award from John von Neumann Society for Computing Science.
Early 1993 he benefited a three-month grant from Commission of the European Communities for "Applying AI Techniques in Supporting Tools for the Development of Responsive Information System".
Currently he prepares his Ph.D thesis on capacity planning, software performance engineering and SSADM. He participates in a joint research project -PEKADS- for developing and verifying an Integrated Knowledge Modelling Environment, carried out under the EC COPERNICUS Programme.
He has published several research papers.

1

This research was partially supported by the European Communities under the Community Action in Science and Technology with Central and Eastern Europe in the form of Individual Fellowship N° 7955. A part of the work was carried out under the PEKADS project within the COPERNICUS programme (CP 93: 7599 PEKADS).

1. Introduction

The software performance engineering can be considered as "lost knowledge" in the system analyst and designer community who works in the information systems field. As the performance of the hardware improved, the performance engineering and modelling did not pay enough attention just to those fields where the strict performance requirements made it cost-effective (e.g. flight-control, mission critical embedded systems).

In the information systems engineering field, the system tuning or "fix-it-later" approach proliferated; the early structured systems analysis and design methods (e.g. [Yourdon75], [Longworth86], [Brodie82], [Cameron83], [Jackson82], etc.) deferred the performance considerations to the technical and physical design or implementation stage. Claim can be made that to carry out a performance evaluation or a prediction exercise, more performance related data are needed and that at an early stage of the analysis they are not available with the required precision.

Responsive information systems can be defined as systems that meet the performance objectives regarding the response time and throughput following [Smith90]. With the real-time systems or *reactive* systems, the critical performance factor is the time required for responding to a *stimulus* or *event, and* fulfilling the performance requirement is absolutely necessary, so that this parameter strongly influences the quality of the end-product. With various information systems, there are not so strict demands, nevertheless there is a trade-off between the amount of processes executed and the response time, so that it bears on the effectiveness and the efficiency of users.

The other side of the problem is that relatively few experts are available and many analysts and designers who need their services; these services can be considered especially valuable in a country with an ageing equipment base and shortage of capital to renew that base or to procure the most sophisticated and advanced equipment, and with a shortage of the necessary expertise.

However, nowadays there are sophisticated information system analysis and design methodologies (e. g. [CCTA90A],[Matheron90], [Turner90]) that provide tools for collecting the non-functional requirements and, thereby, for supplying data for performance calculations. Although the performance of the required systems is of great interest to analysts and users, little research has been conducted in the field of the performance prediction of information systems of which design is being made in a structured analysis environment.

An appropriate programming environment, or supporting system would be an enormous helping hand for the analysts to evaluate the design alternatives. Such an environment can exchange data with various CASE (Computer Aided Software/System Engineering) tools, more exactly with their data dictionaries or repositories. As the variety and service level of the CASE tools differ, the supporting environment should have a stable, well-defined interface in that the necessary and available data can be flexibly imported. Furthermore, the CASE tools are evolving very rapidly so that a feasible solution for an appropriate environment fulfilling the above described requirement is an object library that provides the re-usability feature and relatively easy portability and customisation. Thus a fairly general facility would be available for the analysts to describe the technical environment in a primitive "language" and supply the parameters of the would-be software system.

However, before such a system can be devised, a methodology should be created to precisely define the data gathering and processing activities; furthermore, to fight the danger of generality, a particular systems analysis and engineering methodology should be chosen, and also a specific set of performance and capacity engineering methods to create a well-defined interface between them, for this reason the SSADM (***Structured Systems Analysis and Design Method***) (CCTA90A) is selected, and several theoretically well-grounded computing methods for performance evaluation have been chosen. So the description of the method can be fairly concrete (see [Molnár95]).

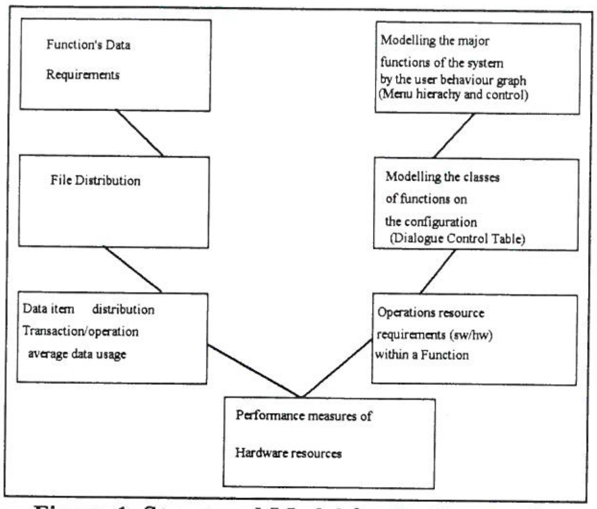

Function's Data Requirements
Modelling the major functions of the system by the user behaviour graph (Menu hierachy and control)
File Distribution
Modelling the classes of functions on the configuration (Dialogue Control Table)
Data item distribution Transaction/operation average data usage
Operations resource requirements (sw/hw) within a Function
Performance measures of Hardware resources

Figure 1. Structural Model for the Proposed Detailed Methodology

2. Description of SPP-OL: A Comprehensive Performance Modelling Library

In this section, we present and briefly refer to several algorithms that can be used to calculate the expected performance of an application system designed in SSADM, as these algorithms have already proved their capabilities and applicability. Then we describe the tool set, the relevant objects that implement and contain the algorithm depicted previously, and some examples for creating the objects and sending the messages to execute the required computation (a more detailed description and further references can be found in [Molnár93c], [Molnár95]). The examples demonstrate that this object library can be used as a language extension of an object-oriented language that makes the performance modelling and the alternatives description and evaluation fairly flexible. A user- friendly interface can be built above it to assist the parameter gathering but in this case we lose a part of the flexibility. We built as a demonstration prototype, a simple window-based, interactive data-gathering interface for the exact mean value analysis, for one of the facilities of the object library. But otherwise, we kept the more flexible programming interface. In further development, the alternative interfaces can be defined for each object and the end-user analysts may alternate among them depending on the requirement and on personal capability and knowledge.

Our main purpose is to demonstrate the feasibility and viability of this approach in a structured method environment.

2.1 Questions To Be Answered

At the Business System Options and the Technical System Options (SSADM concepts) several performance and capacity related questions should be answered. The Project Board (CCTA91) needs estimate how much will the application cost, whether it will meet the service level agreements, which hardware and software configuration fits to the requirements at the best. The database designer and the database administrator are interested in how large disk space a database will use, what the optimal or sub-optimal data distribution will be. The capacity planner during the operation of the system - which is already beyond the scope of the SSADM - would like to know how much growth in transaction volume can be sustained, how an increase in database size will affect the performance, what the impact will be if we upgrade the CPU, buy more disks, enhance the network, whether it is worth buying more memory, and how the new application will affect the existing systems.

The modelling tool allows analysts and designers to clearly see the effect on performance of each option. So we can call this tool as ***SSADM Performance Predictor Object Library*** (***SPP-OL***).

2.2 A Structural Model for the Proposed Detailed Methodology

Those who are unfamiliar with the notion of SSADM and the related Capacity Planning proposal are referred to (CCTA90, CCTA90A, David92), because of lack of space we cannot undertake to presenting them here. In this section, we summarise the most important steps of the proposed techniques, the proposed order and the interrelationship between them. The following sections will describe the algorithm and the coupled objects in more detail.

The coreconcept for describing the data processing activities in SSADM is the function and there exists a "Universal Function Model" to depict the main components or fragments of the function. The update or enquiry process within a function is represented in a Jackson or Jackson-like diagram.

Function's Data Resource Requirement. The available Jackson-like diagrams (Effect Correspondence Diagram and Enquiry Access Path) associated with the specific function explicitly show the data resource requirements through the entities referred in the diagrams. Entities (represented in the Logical Data Structure) have volumetrics containing static and dynamic aspects. The main goal of this step is to collect basic information on the function what and how many data items, records are needed from the affected or retrieved entities, and to prepare a recommendation for data distribution among the available secondary storage devices.

File Distribution. If there are several available disks within a centralised environment we should decide where to place records of a particular entity depending on the function data requirements. A similar question arises in a distributed environment (e. g. client-server architecture) and the communication costs all through the interconnection network should be taken into account, not only the costs of the secondary storage. There is a heuristic algorithm that provides a sub-optimal answer and does not require too much computing (Motzkin90, Motzkin90b). This calculation is cost-conscious, that means it cares for the real cost factors but can only concentrate on the response time optimisation with suitable parameters. This procedure provides more exact answers for data optimisation than the rough optimisation recommended in SSADM Version 4 and even takes into consideration the distributed environment, if necessary. The subtlety of the distributed data design can be found in more detail in (Martin81).

Average Usage of Distributed Data Items of a Function. If the technical environment is a distributed one we need analyse not only the file distribution but also determine the usage pattern, the frequency of usage of the data items by the database transactions belonging to a particular function. The distribution of the data item has been established in the previous step by defining the file/relation distribution. The data item required by a function can be deduced from the functional requirements, i.e. Effect Correspondence Diagrams, Enquiry Access Path (Jackson-like) and Update Process Model, Enquiry Process Model (Jackson diagrams). The transactions are the logical success units or atomic success units within a function. The necessary input into this analysis is the data item distribution and grouping related to the nodes and the data item demand by a transaction of function or by the entire function.

Performance Metrics of Hardware Configuration. In order to create simple spreadsheets for trivial calculations (see CCTA90A, Molnár95, Smith90), we have to collect the basic performance parameters of the given configuration. The source could be the Technical Environment Description (SSADM product), the hardware vendors' data, and data from experimental measurement.

We have to decide on how to map the given hardware architecture to the concept of queueing network modelling:

- disks, secondary storage devices can be modelled as queueing service centers,
- CPU, FESC (Flow Equivalent Service Center) can be modelled as load-dependent service centers, (see Lazowska84)
- terminal stations, hardware elements causing delay (e.g interconnection network), as delay centers.

After the modelling decision, the relevant performance metrics should be determined to

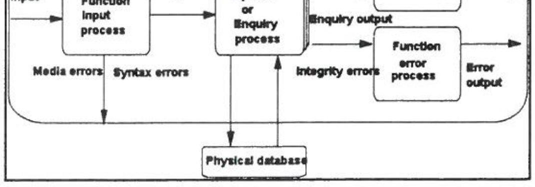


Figure 2. Universal Function Model (CCTA90A)

conclude the average service time for operations, tasks, transactions, functions, etc.
The parameters are the *service times*, *think times* at terminal stations, chain-independent *capacity function* for load-dependent station. At load-dependent service centers, the average service time depends on the loading level of the center, that can be given by a function or table showing the dependency between the number of jobs awaiting the service and the actual service time. For example, this formula can be valid for a CPU, $C(n) = 1/(0,95 + 0,05n)$ where n is the job number or total population waiting for the service.
Resource Requirements per Function. SSADM Version 4 (and the new Version 4.2) generates several important diagrams that contain significant characteristics of the designed software and the concept of operations, some logical database primitives, and plays crucial roles in this step. Using the results of the previous steps, to fill in spreadsheets, firstly for operations as they require the DBMS or File Handler services, then later on we try to determine the other software resource requirements and deduce the hardware resource requirements, finally concluding the hardware resource demand of each operation associated with a Jackson diagram and with the related function and thereby the function's. After this data gathering, we can decorate the Jackson-like or the rigorous Jackson structures with the hardware resource requirements coupled to operations. The Jackson-like structures reflect the non-procedural aspects of the software execution, the rigorous Jackson structures mirror rather procedural aspects. Anyway, the diagrams enhanced by the resource requirements will be subject to an algorithm to get an overall resource requirement for the associated function. The algorithm is an amended and adapted version of Smith's original one (Smith90). The enhanced diagram can be analysed and modified using Smith's heuristic design rules. An extra parameter that we need for this computation is the probability of the selection branches. There are facilities in SPP-OL to carry out the calculation.
Modelling the Classes of Functions. If we want to analyse the performance behaviour of particular functions in a multi-programmed environment or where there is contention for resources, we can start on the Dialogue Design coupled to the function. We can transform the Dialogue Control Table (SSADM product) into User Behaviour Graph that represents the calling sequences within a function and the conditional probability invoking one single command after another, with various functions being modelled by different User Behaviour Graphs. The danger is that too many classes of functions is are defined and they become computationally unmanageable.
Modelling the Critical Functions. After selection of the critical functions, we have already the basic performance parameters and resource requirements for the entire function of the previous steps. If we want to evaluate the response time of the critical functions we have to build up a suitable User Behaviour Graph based on the menu structure of application (SSADM product) and to collect the necessary probabilities describing the users' usage patterns concentrating on the critical functions. We have implemented the facilities based on Calzarossa's model in the SPP-OL (Calzarossa86, Calzarossa90). There are an exact mean value analysis for single-chain, an algorithm for multiple-chain queueing network, for supporting Calzarossa's algorithm.
Modelling Impact of the New Application System. We have to carry out a similar data gathering exercise as in the previous step and set up User Behaviour Graphs depicting the users' behaviour when they use several application systems on the same platform. The available algorithms and objects can be used the same way as we did before.

2.3 File Distribution

Motzkin's algorithm provides a tool for creating a sub-optimal proposal for data distribution with some restricting assumptions (Motzkin90, Motzkin90b). In the design phase, we need an algorithm in the initial state of the design for data distribution in order to evaluate the performance and responsiveness of the system, while the data distribution should obey some constraints and get close to the requirements.

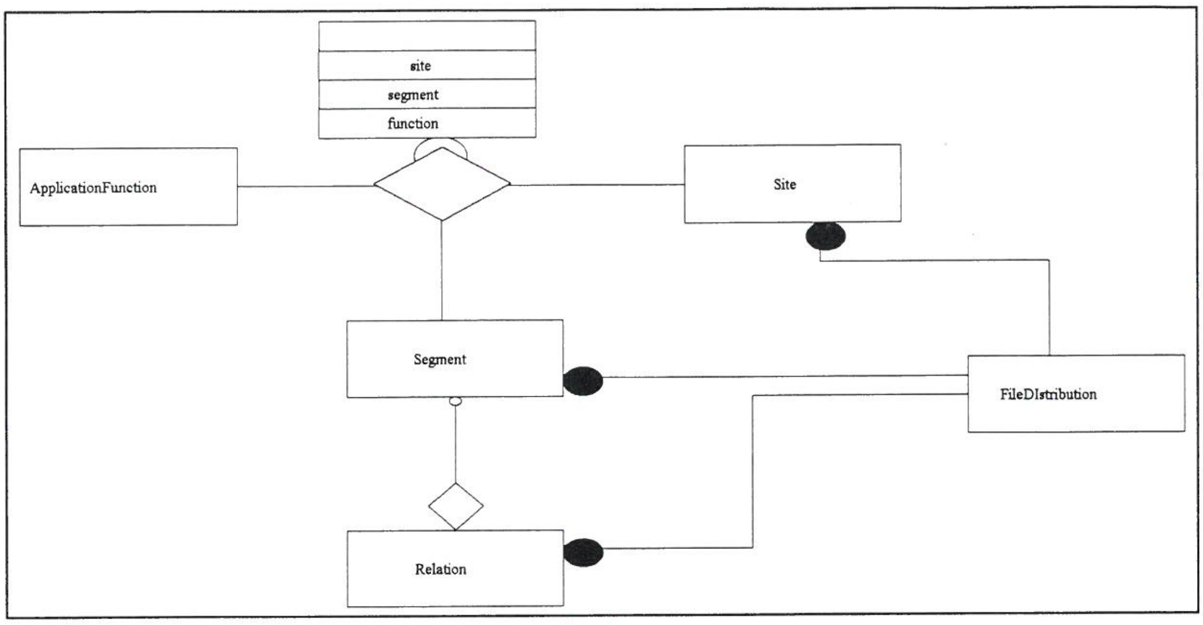


Figure 3. Structure of Cost Optimisation of File Distribution

2.3.1 Algorithm for Proposing a File/Relation Distribution (Data Perspective)

The structure of the implemented objects for Motzkin's algorithm can be seen in Figure 3. (The OMT notation is used [Rumbaugh91]). In the following section, we summarise the main steps and demonstrate the use of the objects, the parameter assignment mechanism and the message calling pattern.

In an SSADM environment, we can assume that a global relational database for the distributed environment is fully defined since we have the logical and physical database design. We concentrate on the relational databases as they are very popular, commercially available and they are based on a sound mathematical background. The entity descriptions can be transformed into relations using the rules of methodology. We can assume that at each node there are functions that plan to use the DDBMS (distributed database management system) and each application is assumed to have a home site where it is run and where it can access the DDBMS from. A function can be denoted by f_j.

Motzkin defines the following concepts needed for the algorithm:

- *fragment* is a portion of a global relation that is allocated one or more nodes, only horizontal fragmentation is considered since it has practical relevance
- a *horizontal fragment* consists of a subset of records of a relation (or tuples of relation).

Data allocation to the nodes in a distributed database happens in two phases, (1) the *fragment definition phase,* (2) the *fragment allocation phase*.

An auxiliary notion:

- *segments*, a subset of records in a relation that is required by some function or part of the function but not necessarily disjointed.

Definition of segment:

- Let f_j an application function or a part of it that requires data from the database; all the tuples (or records) are needed by the f_j function,
- Let us denote by $SEG_w(r_j\ f_j)$ the segment being part of the r_j relation, used by the f_j function; and no record in $r_1 \backslash SEG_w(r_j\ f_j)$ (set theoretical difference) is required by f_j.

As it can be seen from the definition, a segment is the smallest subset of records of a particular relation $r_{j,}$ that are used by a given application function f_j. We can assume that every f_j function is associated with one relation since - as we have seen in the Universal Function Model - we can decompose the function into enquiry and update parts. Both parts can further be decomposed into more elementary operations, into logical success units, and can be considered as elementary application functions (transactions). The relations used by such an application function can be linked to each other by the relational operators that formally describe the data requirements of each application function. The fragment can be determined using a simple algorithm based on set theory and implemented in ***SPP-OL***.

The Motzkin cost optimisation algorithm can be applied for minimizing the response time of the system if we are interested in the *responsiveness* of our interactive system in a DDBMS environment. In this case, we should use unit of time instead of unit of costs in dollars, pounds sterling or Forints. The time unit at a node or server can be gained from the mean service time for update and retrieval operation at a node. The total time of retrievals and updates can be minimized using the algorithm with zero storage space cost so we will get a response time minimization regarding the fragment allocation. Naturally, conflict should be resolved if there are contradictions between the fragment allocation and the available storage space, however we have a good approximation and compromise of a response time that may be achieved and an initial fragment configuration. On this basis, we can refine and adjust to other requirements.

2.3.1.1 Cost and Benefit Calculation of Fragment Allocation

After the Logical Design and Physical Design we know how much storage space is available at each node, and if the storage space is not limited, practically we can consider it as infinite and denote it as ∞. Firstly, all the nodes are examined whether they have enough storage space to store all the fragments allocated to them. If a node does not have sufficient storage space, all its fragments are arranged by calculated benefits in an ascending order, beginning with the least beneficial fragment and cycling through all the fragments until none of them has remained, in the following way: if the fragment is allocated elsewhere, its allocation to node n_i is cancelled. If the fragment is not allocated elsewhere for all the nodes, the benefit of the fragment should be reviewed in a descending order and the fragment should be allocated to the first node n_j that has enough storage space and the allocation to node n_i is obliterated. If the space constraint cannot be met, an error message is generated. The algorithm has two parts after the determination of fragments:

1. Allocate fragments to all sites where *the benefit of allocating fragment to the node* >0
2. Adjust to meet space requirements

The details of the algorithm can be found in (Motzkin90, Motzkin90b, Molnár93c) and the explanation of the necessary formulas and computation methods.

At the Physical Design Stage in SSADM, there are some obscure references that the reconciliation between various non-functional requirements should be carried out in a loop, this algorithm makes that procedure a little more precise.

The communication cost for a function, or its appropriate tasks can be estimated from the local node to the remote node participating in the transaction. If we know exactly which node will be accessed, we can calculate these values more precisely for a given transaction. However, when we have an update and retrieval mechanism, we can further refine that estimation using Mukkamala's algorithm ([Mukkamala90]), we also have the tool to (Mukkamala90), we also have the tool to estimate the average accessed nodes and data items in ***SPP-OL***.

2.4 Transaction Evaluation in a Distributed Environment

The next step is to create an estimation for the transactions in the case of distributed data. The problem can be formulated mathematically and there is a so-called closed formula with some very specific presumptions, and with more general presumptions there is an algorithm for calculating the number of participating nodes and the data items accessed at each of these nodes. These data can provide a sound basis to determine the communication overhead, the impact of the number of copies of data items on transaction performance. The results produced by the algorithm can be used to assist in the approximation of a transaction execution time, of the probability that a transaction requires data items from the co-ordinating node and the average number of data items accessed (locally) by the co-ordinating node. We briefly summarise the presumptions and the procedure of Mukkamala's algorithm (Mukkamala90), a detailed interpretation in the SSADM environment can be found in (Molnár93c), (Molnár95). The placement of each data element and the file distribution can be concluded from the Requirement Catalogue and from the results of the previous steps calculating the costs of the file distribution regarding the functional requirements and the space constraints for each node.

When the distributed database contains replications of data items, the number of copies of a particular data item may be determined by the application environment or by the analysis of the enquiry access paths in the distributed environment. The presumption that all data items have the same number of copies does not hold, nevertheless the decision, in practice, is not arbitrary and the data item copy distribution is not a random choice at all. There are several clustering and de-clustering techniques to assist the designer in decision making on the application system (Martin81).

We bring together the elements with the same number of replicas into a data group and each data group G_k ($y_k=|G_k|$ the number of data items in the group) is allocated to a set of nodes Q_k ($x_k=|Q_k|$, the number of replicas of each data item), containing the replicas of data items belonging to G_k and they are disjoint sets of data items. The number of data groups in a database with the general data distribution is determined by the number of replicas and the distribution of the copies among nodes. We can indicate this number by *ng*.

We introduce the concept of the *group access vector*, $GA=\langle g_1, g_2, \ldots, g_{ng}\rangle$ where g_k represents the number of data items references by a transaction from group G_k.

Given the general data distribution assumption, the grouping of the data items and the distribution of the groups are arbitrary so that a closed-form mathematical formula is not to be derived for $\overline{n}_s$ and $\overline{d}_s$ (the average number of nodes accessed and the average data size).

The data grouping ($G_1, G_2, \ldots, G_{ng}$) and the *s* value (number of data items accessed by a transaction) given, the algorithm generates all possible values for $GA=\langle g_1, g_2, \ldots, g_{ng}\rangle$ so that $0 \le g_i \le y_i$, $\sum_{i=1}^{ng} g_i = s$, the generated vectors are marked as $GA_1, GA_2, \ldots, GA_{d'}$, i.e. $\Phi=\{GA_1, GA_2, \ldots, GA_{d'}\}$, $d'=|\Phi|$. For each *GA*, the corresponding node access vector *J* is determined, where $J=\langle a_1, a_2, \ldots, a_n\rangle, \sum_{j\in N} a_j = s$, a_j represents the number of data items accessed by the transaction at the jth node. To understand the algorithm, we should keep in mind that for all G_k there is an associated Q_k containing the nodes that have one copy of all data items in G_k, and the query transaction follows the read-anyone policy, i.e. it reads the first copy of data item because the consistency of the database is always maintained. With a simple program, we can determine the associated access vector (it is implemented in ***SPP-OL***). Using that algorithm, we may compute the value of J_k for the corresponding GA_k, and calculate the value of A_k (the index set of nodes accessed in the transaction), $|A_k| = n_{GAk}$ represents the number of nodes accessed by a particular query depicted by the given GA_k, the average number of data items accessed per node in the set A_k is denoted by $d_{GAk}(=s/n_{GAk})$. The probability of the occurrence of a given *GA* can be computed following Mukkamala's mathematical analysis.

```
Algorithm for computing the averages
  Given    s, n, ng, and    $\{ x_k , y_k \}$ k=1,....,ng
    Generate a GA vector
    Compute the probability of GA
    Generate the access vector J to the vector GA
    Determine    the index set of accessed nodes (A) from the access vector
    Determine    $n_{GA} = | A |$ corresponding to $GA$ vector
    Calculate    $d_{GA} = \frac{s}{n_{GA}}$
  Calculate the averages using equation
    $\overline{n}_s$ , $\overline{d}_s$
```

Figure 4. The Main Structure of Mukkamala's Algorithm

So, the average number of accessed nodes and data items can be calculated for each particular transaction, thereby the transaction communication costs and the average resource requirements (hardware/software) can be estimated.

The estimation can be re-used in Motzkin's algorithm in a following cycle of refinement, i.e. there is a clear synergy between the two algorithms and thus we gain a more precise picture of the required system, on the data storage requirement side and the interdependency with.

2.5 A Graph Reduction of Diagrams Representing Software Design

There is an algorithm defined on the so -called software execution graphs (Smith86b, Smith90) which associates a concrete numerical value with each diagram. The problem with this procedure is that it is dedicated to certain types of graphs that are not generally used and not too many tools and software development environments support it. In information systems development, the -state- of -the -art is represented by the structured methods and the object-oriented technology in the analysis and design stage.

A lot of varieties of structured methodologies obtained significant support from the CASE tools that made the structured methodology/technology feasible and practically usable. A majority of these methodologies use Jackson diagrams, methods and/or Jackson-like notations, especially SSADM:

- ELH (Entity Life History) (Jackson-like diagrams),
- Diagrams coupled to functions:
    - ECD (Effect Correspondence Diagram), EAP (Enquiry Access Path) (Jackson-like diagrams),
    - UPM (Update Process Model), EPM (Enquiry Process Model) (standard Jackson structure).

It is worth investigating how we can apply Smith's reduction algorithm to such an environment and adjust it to the given circumstances; furthermore to alleviate the analyst/designer work, an implementation is made in ***SPP-OL*** for various Jackson structures that are strictly coupled to functions since we are interested in the performance prediction of

critical functions. Our approach is conform with the latest version of SSADM making it possible to use the proposed procedure in this case as well (Hedges94, Slater94)
After allocating the operations to the appropriate node within the structure, we have to determine the resource requirements, first the software, secondly the hardware resource requirements for each operation. At this point, we have to define the service centers from the hardware specification and to take into account the characteristics of the data distribution. The result of previous steps of the method can be used to determine the number of visits or resource requests to a particular service center regarding the file and data distribution and the implication for the data usage pattern of transactions. Thus, we have a vector for each operation, representing the average service time and the number of visits for each resource.
Without going into detail, we provide an illustrative example for the Effect Correspondence Diagram in Figure 5, to give a feeling of the algorithm, and the computation procedure. Having associated with each logical database operation (SSADM concept) the aforementioned tuple, every single node in the diagram has the set of the related operations so that we can associate a tuple or an accumulated value with each of them. On computing the performance parameter for the whole diagram, we take into account the elements of a *sequence* by adding them together, the *iteration* by multiplying the average number of cycling, and the *selection* by regarding the probabilities of the appropriate selection paths. The algorithm is also valid, naturally, for the rigorous Jackson structure (represented as an inverted tree) and decorated with the operations (Update and Enquiry Process Model are described in this form within SSADM).
The result of the algorithm is a tuple for the entire function representing for each modelled hardware resource the number of visits and the average required service time. The average service demand can be gained with a simple multiplication per each resource, and summing them we can get an overall resource requirement for the function. That detailed analysis provided the opportunity to apply Smith's heuristic rules for software performance design (Smith90).
Thereby, we deduced a stand-alone resource requirement for each function and when necessary, we could investigate certain parts of the function in detail as well and improve the performance characteristics.

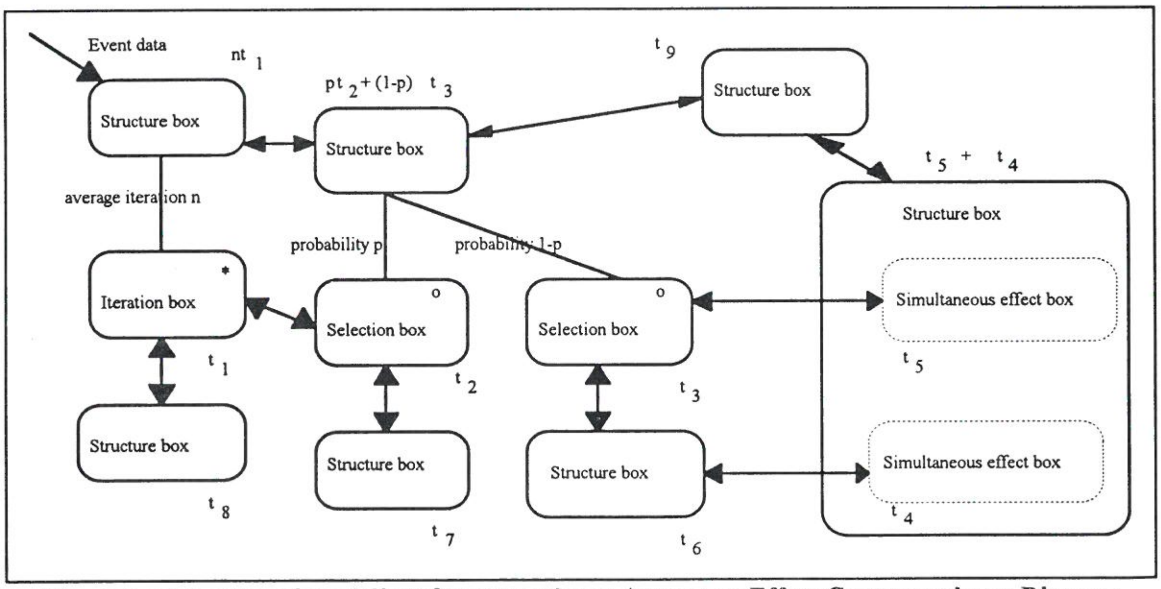


Figure 5. Reduction of the Effect Correspondence Arrows on Effect Correspondence Diagram

Logical Grouping of Dialogue Elements ID	**Occurrences**			**Default Pathway**	**Alternating Pathways**		
	Min	**Max.**	**Ave**		**Alt1**	**Alt2**	**Alt3**
LGDE1	1	1	1	×	×	×	×
LGDE2	1	1	1	×	×	×	×
LGDE3	0	10	2	×	×		
LGDE4	0	10	3	×		×	
% Path usage				70	5	5	20

Table 1. A Dialogue Control Table

2.6 Analysing the Impact of User Behaviour on the Performance of the System

2.6.1 The Concept of User Behaviour Graph

The workload of an on-line, interactive system can be represented in the form of user behaviour graphs, so it can be used as a basis for creating workload models. A user behaviour graph (ubg) consists of a set of commands or command types represented by nodes or vertices, the switching from one node to another (from one command to another) being represented by the arcs linking the nodes together; self-referencing of a node is allowed. The probability of issuing the next command is determined by the transition probabilities of *ubg* attached to the arcs. The sojourn time that a user spends with a command at a node in *ubg* is the sum of *typing* or *selecting* the *command*, the *response time of the system*, and the *user think time*.

This type of information is collected in SSADM in the form of Dialogue Control Table linked to each Function Definition and in the User Role/Function Matrix (Table 1). The 'User Role' is a set of users carrying out tasks in common, the dialogues used by one User Role are represented in the User Role/Function Matrix as an intersection or cross (×) between rows and columns. These dialogues are depicted by a Jackson-like diagram (Figure 6) representing the data items manipulated during the terminal session; the volumetric information, frequencies, and path usage collected in the Dialogue Control Table (Table 1).

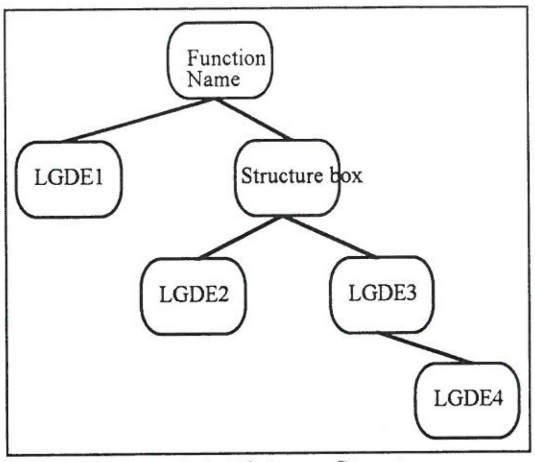
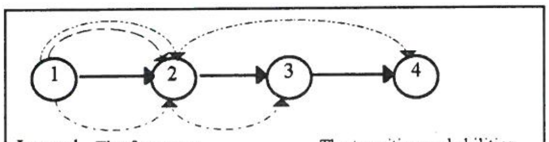


Figure 6. A Dialogue Structure

We have acquired an overall picture of the stand-alone resource requirements of critical functions in the form of a tuple associated with the relevant Jackson diagram. The fragments or tasks of functions represented by a Jackson diagram can be mapped onto the logical dialogue element, i. e. the processes carrying out some well-defined database transactions and the necessary input and output activities can be associated. Depending on the granularity of the analysis, the analyst/designer can decide what level of the performance data is required, i. e. function level or task level associated with the fitting logical dialogue elements. The trade-off is between the granularity and the computational complexity. If we are interested in the detailed behaviour of some functions we should have a rough estimation of other functions and applications, and we should model in this environment by approximating the real-life situation so that a prediction of the response time of definite part of the functions, should exist. This is a detail of modelling either the



Legend	The frequency of path usage	The transition probabilities	
- - - -	20%	1-4	75%
———	70%	1-2	100%
-·-·-·	5%	2-3	75%
-··-··-	5%	2-4	75%
		3-4	75%

Figure 7. A User Behavior Graph for the Dialogue Control Table

decision of the project management or the analyst.

Menus are tools that provide access to the permitted on-line application functions for users (representing certain User Roles). Generally menus are hierarchically structured devices where dialogues, application functions, common functions, other menus can be called from. With the assistance of Menu Structures, we can create the appropriate *ubg*s, the nodes representing commands (menu, function, dialogue call, etc.), but we have to collect statistical data about the function, menu, and dialogue usage or directly the transitional probabilities between the nodes; in SSADM, it is not prescribed to gather the transitional frequencies from one command to another, but these data are required in Calzarossa's model, so we have to complete our information base with these data as well. Summarizing, the SSADM analysis and design stages are able to supply the data that are suitable for defining the appropriate *ubg*s at function or dialogue level as well as at application level .

The detail of the model, the theoretical and mathematical background can be found in (Calzarossa86, Calzarossa90).

Having reduced the complexity of the problem using Calzarossa's model, we can feed the reduced model parameters into the appropriate object of ***SPP-OL,*** namely:

- the transition probability matrix,
- the population vector signifying the expected multi-programming level, the maximum number of functions for a type of ubg cycling in the system at the same time,
- the service demand and number of visits to each resource providing these tuples for different types of service centers, namely, for the queueing centers (secondary storage devices, disks), load-dependent centers (e. g. CPU), delay centers (e. g. terminal)
- the capacity functions associated with load-dependent centers,
- the number of the ubg used in common.

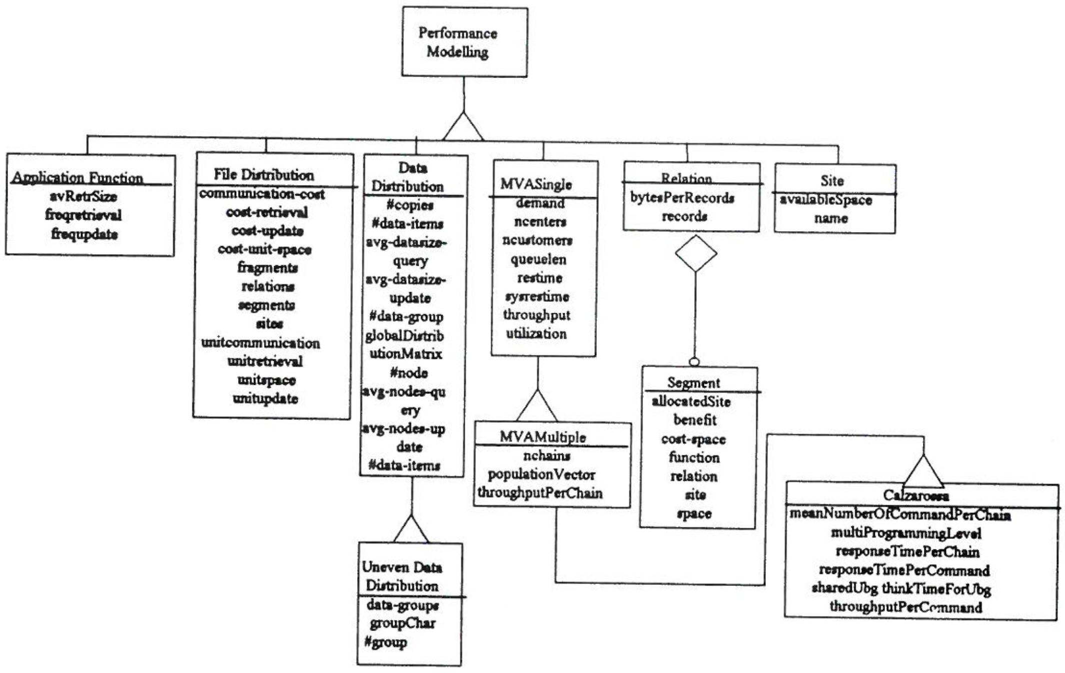


Figure 8. Structure of the Performance Modelling Object Library

Having carried out the performance calculation, the average response time and the expected throughput per *ubg*s are provided, and furthermore for each command (or function call) within a particular *ubg*, the same parameters are yielded.

Summing up, the modelling granularity should be decided, then the *ubg*s and the commands within the *ubg* should be determined. The performance indices computed in the previous steps of the methods can be used for the commands (function calls), performance parameters of Jackson diagrams can be mapped onto the resource requirements of commands. An *ubg* can be a cluster of interdependent functions, or represent separate applications running on the same platform. If we used correct data the difference between the real situation and the predicted one would be at most 10-30%. This provides a good chance to comparing different Technical Design Options that influence the average service time and the average number of visits to resources. The above described procedures of the method make the evaluation more precise, measurable, documentable and yield a good opportunity of discussion and conflict resolution in a clear environment and the arguments can be underpinned.

3. The Structure of the SSP-OL

In this section, we furnish the structural description of the object library containing the algorithms previously presented in the style of OMT (Object Modelling Technique) notation (Rumbaugh91) .

There are objects defined for each important performance calculation and for an entity of the performance modelling. There are some objects that have not been described in detail since their facilities (variables and messages) are used by some others and the use of them will be allowed only for 'knowledgeable-user' (analyst/designer), and some of them represent simpler versions or requires several restrictions.

In Figures 8 and 9, all the objects can be seen that are to be applied for performance calculation in the proposed method, and have been described previously in a brief form.

A part of the input parameters may be transferred automatically from a data dictionary or repository of CASE systems, but some of them should be collected independently and fed

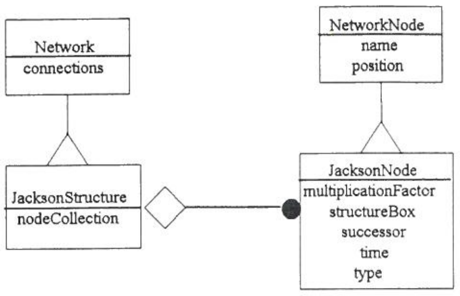


Figure 9. Description of the Jackson Structures in SPP-OL

into the performance modelling library. Nevertheless, the stable interface of object library offers the chance to be interfaced to repositories storing at least a part of the necessary input parameters, and specifies precisely the parameters that are requested for.
Correct Jackson diagrams may be stored in repositories and basic data can be delivered, however, some essential parameters needed for the software performance calculation, for the graph reduction should be collected (the average number of iterations, the probability of selection paths).
The present implementation language is Smalltalk-80 or more precisely one of the dialects Smalltalk/V of Digitalk Inc. (Digitalk86).
The object library serves as versatile and flexible tool for performance evaluation of information systems in an SSADM environment. As the examples demonstrated, it is easy-to-use. The difficult part is to find the input parameters and model the hardware and software environments but the proposed method provides clues and rule of thumb how to use the facts collected by SSADM and request for further data.

4. Analysing the Performance Prediction for Information Systems Using KADS

We use the diagram techniques of KADS (Schreiber93, Wielinga92) to depict a part of the inference mechanism and knowledge base that may provide further help for analyst/designer. At the present stage of the research, we cover only a little part of the whole cognitive and knowledge intensive activity but the purpose is to cover as much as possible and to operationalize that knowledge through the specification in CML language of KADS and later on transform it into an appropriate programming language (PROLOG, Smalltalk, etc.) by a structure preserving design. Now, the assessment part, i. e. the comparison between the design options, is analysed and implemented using the available object library (***SPP-OL***) and the object-oriented environment (Smalltalk/V of Digitalk Inc. [Digitalk86]).

4.1 Application of the KADS Method for Analysing the Knowledge Structure of Software Performance and Capacity Engineering

The goal of the performance evaluation is to support the selection process among the design options. Each of the design options has to conform to some resource requirements as response time, throughput and perhaps utilisation specified in the non-functional requirements.

4.2 Proposals for a Knowledge Based Approach

An integrated system that would support the analysts/designers would consist of several parts. We need a database about the available technology and about the required system functional and non-functional specification. The knowledge base would comprise rules for assessing the design options, proposing solution, rules for designing and configuring, methods for problem-solving and among them conflict resolution methods.
In our research, we have taken only the first

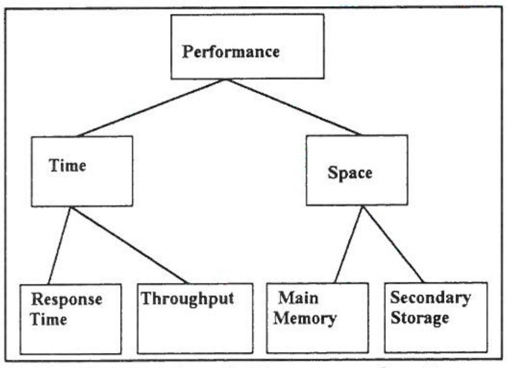


Figure10. Nixon's Taxonomy for the Performance Goals (sorts)

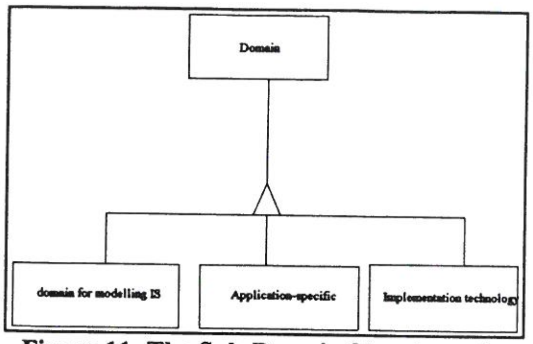


Figure 11. The Sub-Domain Structure of an SPE Knowledge Base.

steps towards such a system so that we tackle some simple inference and tasks mechanisms supporting the assessment of design options regarding the requirements. In future research we plan to enhance it with more complex ones using the CommonKADS Library.

We have to split up the domain knowledge into three large sub-domains and we should define some databases or object-bases to be interfaced to the knowledge base (Figure 11.). We have to enhance *SPP-OL* with some extra objects, however, several already defined objects fit into or can be mapped onto the domain ontology.

Nixon proposed a goal taxonomy that can be used to define the goal structure and the associated task structure. (see Figure 10, [Nixon94]).

The global aim of the knowledge-based system will be to provide active assistance for the system designer/analyst during the design option evaluation stages and along with the logical and physical design steps. To limit the scope to realistic aim , firstly we tackle *the evaluation or assessment of the design option* within an information system analysis and design from the performance perspective using SSADM.

The most important feature of the application specific layer is described by Nixon's taxonomy that demonstrates the hierarchy of performance goals. The performance goals are specified in the Function Definitions and Requirements Catalogue for the dynamic and static aspects of an IS.

The performance goals that should be attained

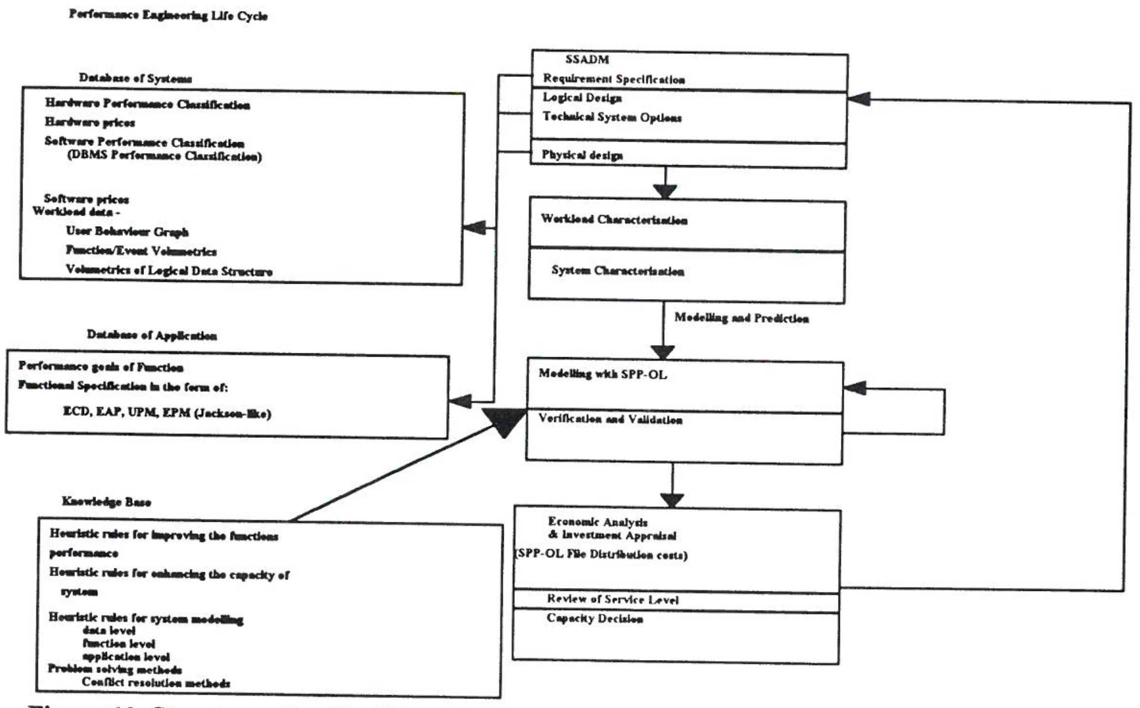

Performance Engineering Life Cycle
Database of Systems
Hardware Performance Classification
Hardware prices
Software Performance Classification (DBMS Performance Classification)
Software prices
Workload data -
User Behaviour Graph
Function/Event Volumetrics
Volumetrics of Logical Data Structure
Database of Application
Performance goals of Function
Functional Specification in the form of:
ECD, EAP, UPM, EPM (Jackson-like)
Knowledge Base
Heuristic rules for improving the functions performance
Heuristic rules for enhancing the capacity of system
Heuristic rules for system modelling
data level
function level
application level
Problem solving methods
Conflict resolution methods
SSADM
Requirement Specification
Logical Design
Technical System Options
Physical design
Workload Characterisation
System Characterisation
Modelling and Prediction
Modelling with SPP-OL
Verification and Validation
Economic Analysis & Investment Appraisal
(SPP-OL File Distribution costs)
Review of Service Level
Capacity Decision

Figure 12. Structure of an Intelligently Supported Performance Engineering and Capacity Management Life Cycle

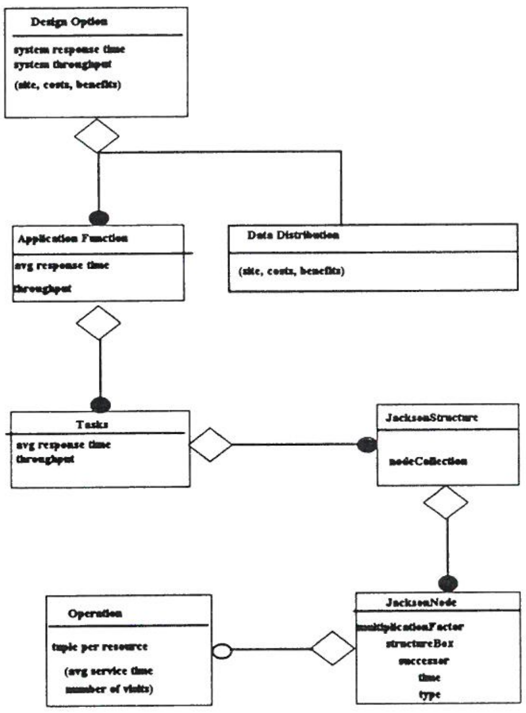


Figure 13. IS Modelling Domain

are related to concepts of the IS modelling domain that contains the concepts used in the software performance modelling and performance predicting. The static, data storage aspect appears in the *data distribution* or file placement calculations and in its end-results, while the dynamic aspect becomes visible in the *application functions* that make use of the basic data of data (distribution) and of the parameters of the implementation technology domain as well, through the operations attached to the appropriate Jackson structures (Figure 13.).

The diagram shows the significant concepts and their interdependency, to model the performance aspects of an IS described by SSADM concepts. This model finds two-fold use: (1) the above-mentioned performance goal should be associated with the appropriate concepts providing a 'norm' that should be achieved or approached (2) provides the performance prediction of a given option related to a peculiar hardware and software configuration.

Löckenhoff's analysis of the assessment task (Löckenhoff93) can be used. A case (the given design option) is assessed against and in terms of a system model (the IS modelling domain). Assessment consists of two classification tasks: (1) finding the appropriate concepts and their attributes in the case description for assessment, (that is the abstraction inference), (2) classifying this new description according to the norms of the system model (classification). A **case** is a structured description which consists of observables (the predicted performance data), in fact an instance of the *system description.* A **decision** is a classification or grading, that is made about the case. The **system model** comprises *system description* and *measurement system. System description* is an abstract description of the system, the *measurement system* is described in terms of the system model and contains possible decisions that are abstract terms.

The most important inferences and knowledge roles are depicted in Figure 15. The assessment inferences are described in detail in Figure 14. We firstly select the so-called critical functions for performance modelling, then collect the specification of the data processing, generally in the form of Jackson diagrams. We apply Smith's graph reduction algorithm and heuristic design rules of software performance engineering. Remember that we have prepared these steps by the preliminary design of file distribution and the evaluation of distributed transactions, when necessary. On turning to the required system specification, we have to create one or more alternative hardware configurations and their queueing network models and at the same time

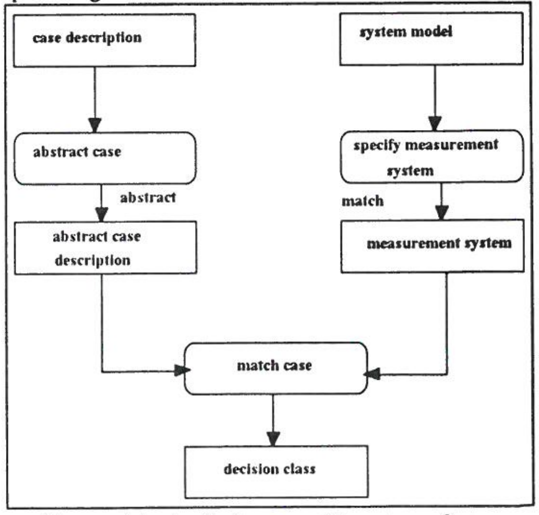


Figure 14. An Inference Structure for Assessment (Löckenhoff93)

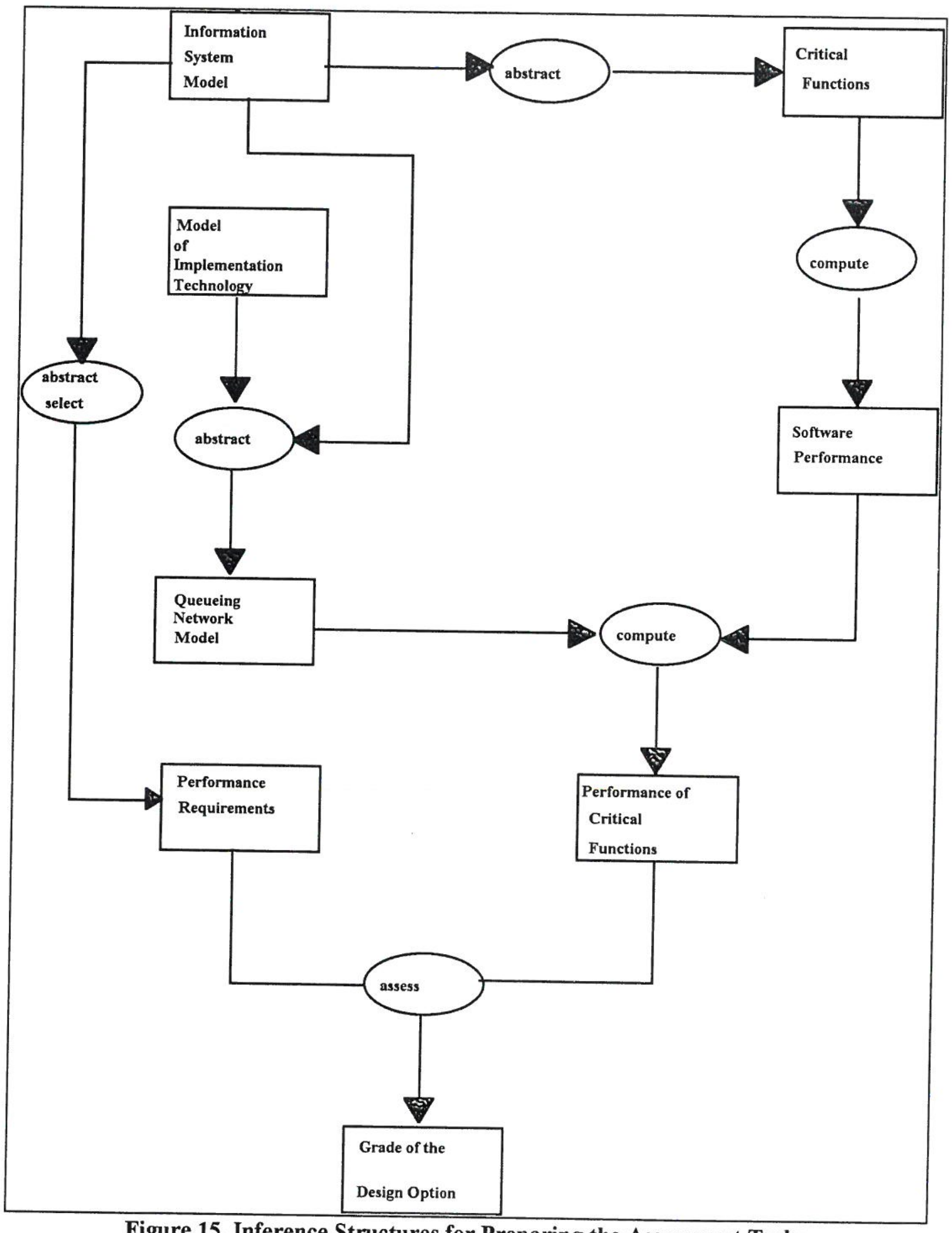


Figure 15. Inference Structures for Preparing the Assessment Tasks

we should extract the non-functional requirements from the Requirement Catalogue and from the Function Descriptions. Finally, we have to assess how close is a single Design Option to the Required System Non-Functional (performance) Specification.

Some of the inferences and the assessment task have already been implemented with the objects in ***SPP-OL.***

4.3 Future Work

By extending the expertise model and the related tasks to cover the assessment, the configuration and technical design tasks could lead to an intelligent design assistant and to improved quality and better decision making in the Business System and Technical System Options.

- Alternative user interfaces to each objects, i.e. enhancing the existing object library with interactive easy to use data-gathering windows or screens.
- Further automating the data exchange between a repository and the object library, creating a set of objects that would read the relevant data from different sources , e. g. ASCII files with predefined syntactical rules, directly from the data dictionary, etc.
- In a (hypothetical) system the intelligent assistant would be tightly coupled to the data dictionary of a CASE tool and this system would actively support the designers' search for alternatives.

The data- or object base of the system should consist of :

- The technical description and classification of the available technology, performance parameters of hardware devices, associate prices.
- The software performance classification (DBMS, File Handler, other tools for implementing certain tasks or functions).
- The expected and estimated workload data.

The hypothetical knowledge base will contain the following:

- The application system specific data:
  - performance goals and objectives,
  - functional requirements (which the performance predictions can be derived from).
- The knowledge for making use of performance engineering tools (e. g. the ***SPP-OL*** object library, etc.).
- The modelling knowledge of the Information System, the way of describing the functional requirements.

REFERENCES & BIBLIOGRAPHY

[Brodie82]BRODIE, M. L. and SILVA, E., **Active and Passive Component Modelling: ACM/PCM**, in T. W.Olle, H.G. Sol and A.A. Verrijn-Stuart (Eds.) Information System Design Methodologies: A Comparative View, ELSEVIER Science Publishers B. V. (North-Holland), 1982, pp. 41-91.

[Calzarossa86]CALZAROSSA, M. and TRIVEDI, K. S., **Performance Analysis Using Behavior Graphs**, *Proc. CMG '86,* Conference on Management and Performance Evaluation, Las Vegas, NE, December 9-12, 1986, pp. 395-397, Computer Measurement Group Inc., 1986.

[Calzarossa90]CALZAROSSA, M., MARIE, R. A. and TRIVEDI, K. S., **System Performance with User Behavior Graphs**, Performance Evaluation, Vol. 11, 1990, pp. 155-164.

[Cameron83]CAMERON, J.R., **JSP and JSD: The Jackson Approach to Software Development**, IEEE COMPUTER SOC., 1983.

[CCTA90]CCTA, **SSADM Version 3 and Capacity Planning**, Information Systems Engineering Division, CCTA, Norwich, 1990.

[CCTA90A]**CCTA (Central Computer and Telecommunication Agency),** SSADM Version 4 Reference Manuals, Vols 1,2,3,4 NCC Blackwell Ltd, Manchester, Oxford, 1990.

[CCTA91]**CCTA (Central Computer and Telecommunication Agency),** PRINCE, Structured Project Management, NCC Blackwell Ltd , Manchester, Oxford, 1991.

[David92]DAVID, A., **SSADM and Capacity Planning, Information Systems Engineering Library**, CCTA, London: HMSO, 1992.

[Digitalk86]Digitalk Inc., **Smalltalk/V, Object-Oriented Programming System (OOPS),** Los Angeles, CA, 1986.

[Hedges94]HEDGES, M. and TURNER, P., **SSADM V4+'** Hungarian SSADM Forum Meeting, John von Neumann Society for Computer Sciences, Budapest, December 1994.

[Jackson82] JACKSON, M. A., **System Development,** PRENTICE -HALL, Englewood Cliffs, NJ, 1982.

[Lazowska84] LAZOWSKA, E. D., ZAHORJAN, J., GRAHAM, G. S. and SEVCIK, K. C, **Quantitative System Performance: Computer System Analysis Using Queueing Network Models,** PRENTICE-HALL International Inc., Englewood Cliffs, NJ, 1984.

[Longworth86]LONGWORTH, G. and NICHOLS, D., **SSADM Manual,** Vols. 1-2, NCC Blackwell, 1986.

[Longworth88]LONGWORTH, G., NICHOLS, D. and ABBOT, J., **SSADM Developer's Handbook,** NCC Publications, Blackwell, Manchester, 1988.

[Löckenhoff93]LÖCKENHOFF, C. and VALENTE, A., **A Library of Assessment Modelling Components**, in Löckenhoff, Fensel and Studer (Eds.) 3rd KADS Meeting, Munich, 1993.

[Martin81]MARTIN, J., **Design and Strategy for Distributed Data Processing,** PRENTICE-HALL, Englewood Cliffs, NJ, 1981.

[Matheron90]MATHERON, J. P., **Comprendre Merise, Outils Conceptuels et Organisationnels,** Editions EYROLLES, 1990.

[Molnár93a]MOLNÁR, B. and SIMON, E., **Creating Responsive Information Systems with the Help of SSADM**, Austrian-Hungarian Joint Seminar on Software Engineering, Technical University of Budapest, Budapest, 4-5 February, 1993.

[Molnár93b]MOLNÁR, B. and SIMON, E., **Responsive Information Systems and Application of AI**, Proceedings of the Third Hungarian Conference on Artificial Intelligence, Budapest, April 6-8, 1993, Publ. John von Neumann Society for Computer Sciences, 1993.

[Molnár93c]MOLNÁR, B., **A Proposal for Linking a Structured Methodology and the Techniques of Performance Engineering,** Research Report, School of Computer Science and Information Technology, University of Wolverhampton, December, 1993.(CEC, Community Action in Science and Technology with Central and Eastern Europe, Individual Fellowship N° 7955).

[Molnár95]MOLNÁR, B., **A Methodology for Designing Responsive Information Systems, Integrating the Pragmatic and Theoretical Approaches within SSADM Environment,** Ph.D Thesis, Department of Mathematics and Computer Science, the Faculty of Electrical Engineering and Informatics, Technical University of Budapest, 1995.

[Motzkin90] MOTZKIN, D., **Distributed Database Design - Optimization vs Feasibility,** INFORMATION SYSTEMS, Vol. 15, No. 6, 1990, pp. 615-625.

[Motzkin90b] MOTZKIN, D., **An Adaptive Data Distribution Model for Distributed Databases Based on Empirical Measurement of Local and Global Database Transactions,** in P. Zunde and D. Hocking (Eds.) Empirical Foundations of Information and Software Systems, PLENUM PRESS, New York, 1990, pp. 329-339.

[Mukkamala90]MUKKAMALA, R. and BRUELL, S. C., **Efficient Schemes To Evaluate Transaction Performance in Distributed Systems,** THE COMPUTER JOURNAL, Vol. 33, No. 1, 1990, pp. 79-89 .

[Rumbaugh91]RUMBAUGH. J., BLAHA, M., PREMERLANI, W., EDDY, F. and LORENSEN, W., **Object-Oriented Modeling and Design,** PRENTICE-HALL, Englewood Cliffs, NJ, 1991.

[Schreiber93]SCHREIBER, G., WIELINGA, B. and BREUKER, J., **KADS, A Principled Approach to Knowledge-Based System Development**, ACADEMIC PRESS, London, 1993.

[Slater94]SLATER, C., **'SSADM V4.2'**, ISUG Autumn Conference, UK, 1994.

[Smith86]SMITH, C. U., **Independent General Principles for Constructing Responsive Software Systems,** ACM TRANSACTIONS

ON COMPUTER SYSTEMS, Vol. 4, No. 1, February 1986, pp. 1-31.

[Smith88]SMITH, C. U., **Applying Synthesis Principles To Create Responsive Software Systems**, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,, Vol. 14, No. 10, 1988, pp. 1394-1405.

[Smith90] SMITH, C. U., **Performance Engineering of Software Systems**, ADDISON-WESLEY, Reading, MA, 1990.

[Smith92]SMITH, C. U., **Integrating New and "Used" Modeling Tools for Performance Engineering,** in G. Balbo and G. Serazzi (Eds.) Computer Performance Evaluation, NORTH-HOLLAND/ELSEVIER Science Publishers, B.V., 1992, pp. 153-163.

[Smith93]SMITH, C. U., **Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives**, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. 19, No. 7, July 1993.

[Smith93a] SMITH, C. U., **Software Performance Engineering**, in L. Donatiello and R. Nelson (Eds.) Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance '93 and Sigmetrics '93, pp. 509-536, Lecture Notes in Computer Science 729, SPRINGER-VERLAG, Berlin-Heidelberg, 1993.

[Turner90]TURNER, W. S., LANGENHORST, R. P., HICE, G. F., EILERS, H. B. and UIJTTENBROEK, A. A., **SDM System Development Methodology,** ELSEVIER Science Publishers B.V. (North-Holland)/Pandata, 1990.

[Wielinga92]WIELINGA, B. J., SCHREIBER, A. TH. and BREUKER, J. A., **KADS: A Modelling Approach to Knowledge Engineering**, KNOWLEDGE ACQUISITION, Vol. 4, No. 1, 1992, pp. 5-53.

[Yourdon75] YOURDON, E. and CONSTANTINE, L. L., **Structured Design**, YOURDON PRESS, 1975.