AN INDUSTRIAL APPLICATION USING BLACKBOARD ARCHITECTURE

A Thesis
Presented to the Institute of Science and Engineering
of IŞIK University
in Partial Fulfillment of the Requirements
For the Degree of
Master of Science
in
The Department of Computer Engineering

by
KEREM BURAK TÜNAY

IŞIK UNIVERSITY
2006

Approval of the Institude of Science and Engineering.

Prof. Dr. Hüsnü A. Erbay
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Selahattin Kuru

Head of the
Computer Engineering
Department

This is to certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Master of Science

Prof. Dr. Selahattin Kuru
Supervisor

Examining committee members

………………………………

………………………………

………………………………

ABSTRACT

AN INDUSTRIAL APPLICATION USING BLACKBOARD ARCHITECTURE

KEREM BURAK TÜNAY

This thesis implements control architecture for goal-driven blackboard systems. The architecture is based on searching a general goal tree by diminishing into sub-goal trees. The aim is to develop a problem solving architecture in the AI space via blackboard system. The basic elements of the architecture are goals, policies, strategies, facts, methods, and knowledge sources. The basic control loop employs a bidding mechanism to determine the knowledge source to be executed at the current cycle. A policy is a local scheduling criterion which guides to bidding process and it indicates which of the attributes of the knowledge sources are relevant in this process. A strategy is a global scheduling criteria such as depth-first, breadth-first etc. A method is a partially complete general goal tree structure representing high level knowledge on how to solve a problem. The architecture employs a control blackboard, and separate knowledge sources for the control problem and for representing the domain knowledge.

A production planning application is developed using this architecture. Both C++ and ABAP languages were used to implement this application.

Keywords: Blackboard systems, artificial intelligence, AI search algorithms, collaborating software, C++, SAP, ABAP, production planning.

ÖZET

KARATAHTA MİMARİSİ İÇİN ENDÜSTRİYEL UYGULAMA

KEREM BURAK TÜNAY

Bu tez amaç-güdümlü karatahta sistemleri için bir kontrol mimarisinin uygulamasını içermektedir. Mimari, genel amaç ağaçlarının alt-amaç ağaçlarına indirgenerek taranmasına dayanmaktadır. Tezin amacı, karatahta sistemini kullanarak yapay zeka alanında problem çözme mimarisi geliştirmektir. Amaçlar, genkurallar, stratejiler, yöntemler ve bilgi kaynakları mimarinin temel elemanlarını oluşturmaktadırlar. Ana kontrol döngüsü, o andaki çevrimde işlenecek bilgi kaynağını belirlemek için bir değerleme mekanizması kullanmaktadır. Burada genkurallar bilgi kaynaklarının hangi niteliklerinin kullanılacağını belirleyen lokal zamanlama kriterleridir. Öte yandan, strateji, önce-derine, önce-enine gibi global zamanlama kriteridir. Yöntemler ise, bir problemi nasıl çözmek gerektiği üzerine varolan yüksek düzeyde iki bilgiyi tanımlayan kısmen tamamlanmış genel amaç ağacı yapısıdır. Mimari, ayrı kontrol ve domen karatahtaları kullanır. Kontrol problemi ve domen ile ilgili bilgiler ayrı bilgi kaynakları ile temsil edilir.

Bu mimari kullanılarak bir üretim planlaması uygulaması geliştirilmiştir. Uygulamayı geliştirmek için, C++ ve ABAP dilleri birlikte kullanılmıştır.

Keywords : Karatahta sistemleri, yapay zeka, yapay zeka tarama algoritmaları, işbirliği yazılımları, C++, SAP, ABAP, üretim planlaması.

ACKNOWLEDGEMENTS

I would like to express my thanks to Prof. Dr. Selahattin Kuru for his comments, help and supervision on this topic and for his all supervision during our previous studies.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A blackboard system is a way of combining a set of diverse software modules to solve a problem using AI technique. In this thesis, an industrial production planning application will be proposed by using blackboard control architecture. The architecture is based on searching a general goal tree by diminishing into sub-goal trees. The objective is to improve the performance of this AI search of the blackboard systems by using real time experiences as high level knowledge sources.

The basic elements of the architecture are goals, policies, strategies, facts, methods, a basic control loop and knowledge sources.

The high level knowledge can be statistical data about a problem, some imitations of human experts and so on. We will define this high level knowledge in two leveled hierarchical way: methods and facts. While methods define a partially complete general goal tree structure how to solve the problem, facts have a higher priority that orders a way to execute in the search space. Both of them improve the performance of the search action in a blackboard system.

The basic control loop employs a bidding mechanism to determine the knowledge source to be executed at the current cycle. A policy is a local scheduling criterion which guides to bidding process and it indicates which of the attributes of the knowledge sources are relevant in this process. A strategy is a global scheduling criteria such as depth-first, breadth-first etc.

The next chapter discusses blackboard systems in detail. Then the proposed architecture is introduced in the terms of definition. After that, the implementation of the system in C++ will be explained in detail. The whole system flow chart and the control loop flowchart will be given to show how architecture works. Also, a flight ticketing example will be given to show how to use the system to solve problems and we will examine a trace of all the parts of the implemented program. As a second example we will give a routing planning model about a plastic injection machine in a workshop to show the power of the architecture. The last chapter gives an evaluation of the approach.

2. BLACKBOARD SYSTEMS

In attempting to solve a problem, a blackboard system employing an artificial intelligence (AI) technique typically performs a series of problem solving actions [1]. This process usually involves a search in the space of partial solutions, and each action extends the current partial solution. Then, the system collaborates these solutions to achieve a flexible, brainstorming style of problem solving exhibited by a group of diverse human experts working together to address problems that no single expert could solve alone.

Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialist finds sufficient information to make a contribution, she records the contribution on the blackboard, hopefully enabling other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved.

In this point we have a several issues into consideration. The first issue is the direction of the search, which is also called the search strategy. There are two basic search strategy, goal-driven and data-driven search strategies. In this thesis we are going to use goal-driven search strategy that begins with the goal to be solved, then finds rules or moves that could be used to generate this goal and determine what conditions must be true to use them. These conditions become the new goals, sub-goals, for the search. This process continues, working backward through successive sub-goals, hopefully until a path is generated that leads back to the facts of the problem. This is why; this strategy is also called backward chaining [2]. In data-driven search, sometimes called forward chaining, the problem solver begins with the given facts and a set of legal moves or rules for changing the state. Search proceeds by applying rules to facts to produce new facts. This process continues until it generates a path that satisfies the goal condition.

At each point in the problem solving process, more than one potential action may be possible. Then the second issue is to decide which of its potential actions the AI system should perform. This issue is called the control problem. To solve this problem, a control structure decides which action will be performed and the choice of this control structure is playing an urgent role on success and the efficiency of the system.

The third issue is related to the distribution of the problem solving capability among several problem-solving agents. In that sense, we have centralized systems on one hand - such as multi-agent systems - and distributed systems on the other.

In this thesis, blackboard architecture for the goal-driven strategy that uses the control architecture paradigm will be implemented. The basic elements of the architecture are goals, policies, strategies, methods, knowledge sources (KS's) and facts. Also the architecture employs a basic control loop that uses a bidding mechanism in choosing the knowledge source to be executed at the current cycle.

2.1. A BLACKBOARD SYSTEM IN DETAIL

A blackboard system is a way of combining a set of diverse software modules is to connect them according to their data-flow requirements [3]. In Figure 2-1, you can see a combination of five modules that shows the data-flow. Also this is called directly connected graph.

When appropriate, the modules can appear multiple times in the communication graph, but the connections are predetermined and direct. This approach can work well when both the module set and the appropriate communications among modules are static. When the specific modules are subject to change and/or when the ordering of modules cannot be determined until specific data values become known at execution time, the inflexibility of direct interaction becomes unwieldy. From a system-building perspective, direct interaction promotes the use of private communication protocols between modules.

Another approach is to use indirect and anonymous communication among modules via an intermediary, such as a blackboard data repository. In Figure 2-2 you can see how can a data repository can be controlled by a control intermediary. In this approach, all processing paths are possible, and the choice among paths can be made dynamically by a separate "moderator" mechanism that selects among the possible paths. The information placed on the blackboard is public, available to all modules, control mechanisms, newly added modules, and monitoring and debugging tools. Indirection reduces the number of communication interfaces that must be supported among highly collaborating modules.

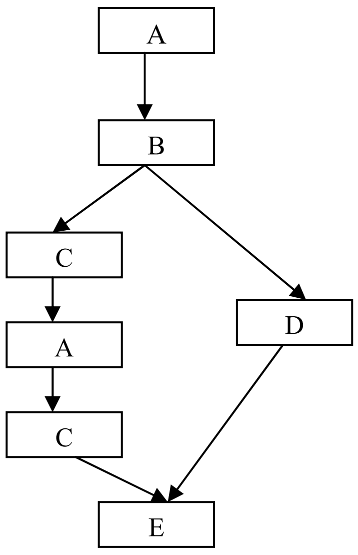


Figure 2-1 : Directly Connected Graph

.

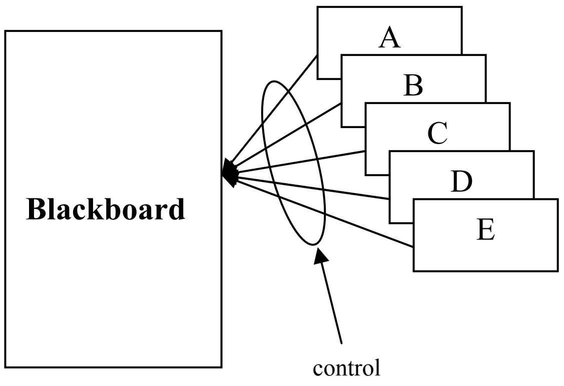


Figure 2-2 : Anonymously Interacting Modules

A blackboard system consists of three main components:

- **Knowledge sources** (KSs) are independent computational modules that together contain the expertise needed to solve the problem. KSs can be widely diverse in their internal representation and computational techniques and are *anonymous* in that they do not interact directly with one another or know what other specific KSs are present in the system.

- The **blackboard** is a global data repository containing input data, partial solutions, and other data that are in various problem-solving states. All KS interaction is via changes made on the blackboard.
- A **control component** that makes runtime decisions about the course of problem solving and the expenditure of problem-solving resources. The control component is separate from the individual KSs.

Figure 2-3 shows these three main components of the blackboard system:

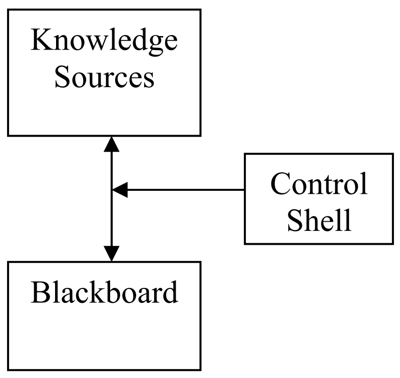


Figure 2-3 : Blackboard System Components

2.2. KNOWLEDGE SOURCES

Blackboard systems use a functional modularization of expertise. Each KS is a specialist at solving certain aspects of the overall application and is separate and independent of all other KSs. A KS does not require other KSs in making its contribution. Once it finds the information it needs on the blackboard, it can proceed without any assistance from other KSs. Furthermore, without making changes to any other KSs, additional KSs can be added to the blackboard system, poorer performing KSs can be enhanced, and inappropriate KSs can be removed. KSs perform relatively large computations, reflecting the processing required implementing their specialty.

A KS needs no knowledge of the expertise, or even the existence, of the others; however, it must be able to understand the state of the problem-solving process and the representation of relevant information on the blackboard. Each KS knows the conditions under which it can contribute to the solution and, at appropriate times, attempts to contribute information toward solving the problem. This knowledge that each KS has about when it might be able to contribute to the problem-solving process is known as a triggering condition.

At an abstract level, a blackboard system may appear to be very similar to a rule-based system: the blackboard system's blackboard and the rule-based system's working memory; the blackboard system's KSs and the rule-based system's production rules; event-based triggering of KSs and of rules; anonymous interaction of KSs and rules; and so on. Historically and operationally, however, blackboard-systems and rule-based systems are very different, especially in the size and scope of rules versus the size and complexity of KSs and in the relatively small number of large-grained control decisions that are made by a blackboard system versus the large number of fine-grained conflict-resolution decisions made by a rule-based system. With regard to knowledge granularity, KS s are substantially larger and more complex than each isomorphic rule in an expert system. While expert systems work by firing a rule in response to stimuli, a blackboard system works by executing an entire KS in response to an event. Each KS can be arbitrarily complex and internally different from one another. In particular, a single KS in a blackboard system could be implemented as a complete rule-based system [4].

KSs are not the active "agents" in a blackboard system. Instead, KS activations are the active entities competing for computational resources. A KS activation is the combination of the KS knowledge and a specific triggering context. The distinction between KSs and KS activations is important in applications where numerous events occur that trigger the same KS. In such cases, control decisions involve choosing among particular applications of the same KS knowledge, rather than among different KSs. Taking this distinction one step further, KSs are static repositories of knowledge while KS activations are the active "agents" that are created in response to each triggering context. These KS-activation "agents" remain alive only until the KS activation is executed or is canceled prior to execution.

2.3. THE BLACKBOARD

The blackboard component of a blackboard system serves as:

- a community memory of raw input data; partial solutions, alternatives, and final solutions; and control information
- a communication medium and buffer
- a KS trigger mechanism

Blackboard applications tend to have elaborate blackboard structures, with multiple levels of abstraction. Although this organization of blackboard data is often useful to the developer and user of the system, the principal reason is to make locating appropriate information on the blackboard more efficient. If the problem being solved is complex and the number of contributions placed on the blackboard becomes large, quickly locating pertinent information becomes a problem. A *KS execution* should not have to scan the entire blackboard to see if appropriate items have been placed on the blackboard by another KS execution.

One solution is to subdivide the blackboard into regions, each corresponding to a particular kind of information [5]. This approach is commonly used in blackboard systems, where different levels, planes, or multiple blackboards are used to group related objects. Similarly, ordering metrics can be used within each region, to organize information numerically, alphabetically, or by relevance. Advanced blackboard-system frameworks provide rich positional metrics for efficiently locating blackboard objects of interest .

Efficient retrieval is needed to support the use of the blackboard as a group memory for contributions generated by earlier KS executions. An important characteristic of the blackboard approach is the ability to integrate contributions for which relationships would be difficult to specify by the KS writer in advance. For example, a KS working on one aspect of the problem may put a contribution on the blackboard that does not initially seem relevant or immediately interesting to any other KS. Only until much later, when substantial work on other aspects of the problem has been performed, is there enough context to realize the value of the early contribution. By retaining these contributions on the blackboard, the system can save the results of these early problem-solving efforts, avoiding recomputing them later. Additionally, the blackboard control component can notice when promising contributions placed on the blackboard remain unused by other KSs and possibly choose to focus problem-solving activity on understanding why they did not fit with other contributions.

Typically, locating previously generated contributions of interest is dependent upon the context of other information being used by a KS. This makes a simple pattern-matching specification of the specific contributions difficult and computationally inefficient. Many contributions placed on the blackboard may never prove useful, and maintaining the state of numerous, partially completed patterns is expensive. Therefore, an important characteristic of blackboard systems is enabling a KS to efficiently inspect the blackboard to see if relevant information is present.

2.4. CONTROL COMPONENT

In a blackboard system, a separate control mechanism, sometimes called the control shell, directs the problem-solving process by allowing KSs to respond opportunistically to changes made to the blackboard. A blackboard system uses an incremental reasoning style: the solution to the problem is built one step at a time.

In a classic blackboard-system control approach, the currently executing KS activation generates events as it makes changes on the blackboard. Figure 2-4 shows this generation of changing events. These events are maintained until the executing KS activation is completed. At that point, the control shell uses the events to trigger and potentially activate KSs. The KS activations are ranked, and the most appropriate KS activation is selected for execution. For continuous applications, this KS-execution cycle continues indefinitely. For single solution based applications, this cycle continues until the problem is solved.

It is important that the control component in a blackboard system is able to make its selection among pending KS activations without possessing the detailed expertise of the individual KSs. Without such a separation, the modularity and independence of KSs would be lost. If specific knowledge of all the KSs had to be included within the control shell, it would have to be modified every time a KS was added or removed from the system. On the other hand, we do not want KSs to be making autonomous control decisions—in a blackboard system, control decisions are made by the control shell. In that point, facts and policies help us.

The solution is to separate control knowledge into generic, overall control knowledge contained in the control shell and detailed KS-specific control knowledge packaged with each KS. Then, whenever the control shell needs KS-specific control information, it asks the individual KSs for these estimates on how the KS will behave. This separation of control knowledge is shown in Figure 2-5. When a KS is triggered, the control shell passes the triggering context to the KS, which uses its KS-specific control knowledge to estimate factors such as the quality, importance, cost, and likelihood of successfully making potential contributions. This estimate is determined without actually performing the work to compute the contributions.

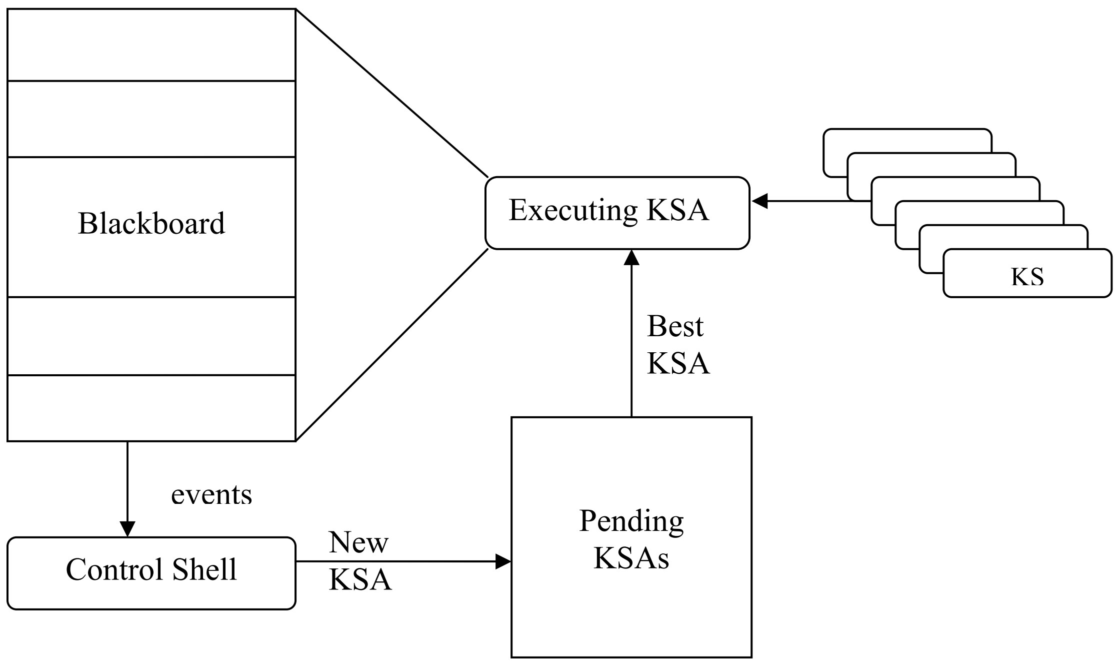


Figure 2-4 : Classic Blackboard System Cycle

Instead, each KS generates estimates of the contributions that would be generated by using fast, low-cost, approximations developed by the KS writer. These estimates are of the form, “If this activation is selected for execution, I estimate it will generate contributions of this type, with these qualities, while expending these resources.” The KS returns these estimates to the control shell which uses them in deciding how to proceed.

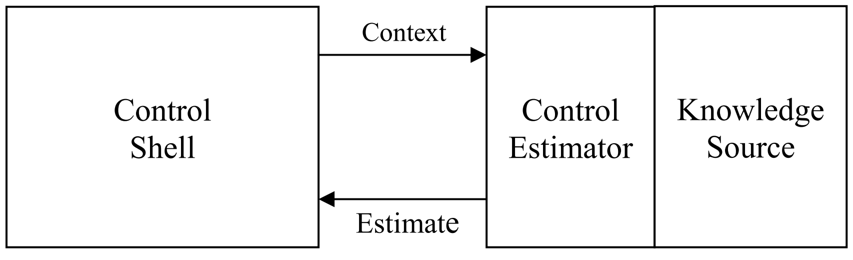


Figure 2-5 : Separation (Encapsulation) of Control Knowledge

2.5. BLACKBOARD SYSTEMS AS COLLABORATING SOFTWARE

As collaborating software, blackboard systems have six key challenges to be effective in problem solving [2]:

- **Representation:** getting software modules to understand one another
- **Awareness:** making modules aware when something relevant to them occurs
- **Investigation:** helping modules to quickly find information related to their current activities
- **Interaction:** creating modules that are able to use the concurrent work of others while working on a shared task
- **Integration:** combining results produced by other modules
- **Coordination:** getting modules to focus their activities on the right things at the right time.

Now let us discuss these challenges:

2.5.1. REPRESENTATION

The structure of information on the blackboard is at the center of the blackboard-system approach. In principle, the blackboard representation should not be based on any specific set of KSs. Instead, the design of the blackboard representation should stem directly from the characteristics of the application and the goal of allowing any potential KS to make contributions toward a solution. In practice, however, the design of the blackboard representations are not fully separated from a general sense of the kind of KSs that will be used in the application, and experience has demonstrated that choices made in the blackboard representation can have a major effect on system performance and complexity.

The KSs in a blackboard application must be able to correctly interpret the information recorded on the blackboard by other KSs. Additionally; the control shell may also need to understand aspects of blackboard data in order to make strategic focus-of-attention decisions. However, all aspects of blackboard data do not need to be understandable by all KSs. Many KSs only use data from one or two blackboard levels as input and only make modifications at a single blackboard level. Similarly, KSs may only operate on a few classes of blackboard objects. In a very practical sense, this characteristic means that portions of the blackboard may be relevant to only a few KSs and could be specialized to the interaction requirements among those KSs. Yet, private jargon shared by only a few KSs limits the flexibility of applying other KSs on that information in the future. In practice, there is a trade-off between the representational expressiveness of a specialized representation shared by only a few KSs and a fully general representation understood by all KSs. Determining the proper balance between a general and specialized representation is an important aspect of blackboard-application engineering.

In the basic blackboard-system control cycle, only a single KS activation is executing at any time. This KS execution runs to completion or termination by the control shell before another KS execution begins. To further simplify the architecture in this basic control cycle, only the executing KS is allowed to make changes to the blackboard while it is executing. This requirement eliminates the need to incorporate complex blackboard locking or transaction mechanisms that would slow down blackboard operations.

2.5.2. AWARENESS

In a blackboard application, KSs are triggered in response to specific types of blackboard events that indicate that the KS may be able to contribute to problem solving. Rather than having KSs continually poll the blackboard, the control shell is told about the kind of events in which each KS is interested. This is typically called registering the KS. The control shell maintains this triggering information and directly considers the KS for activation whenever that kind of event occurs. To be efficient, this triggering information is provided to the low-level blackboard repository accessor routines which only notify the control shell of events for which any KS is currently registered. KS triggering can be made highly efficient when the registration involves only simple, disjunctive trigger events.

2.5.3. INVESTIGATION

When a KS is triggered by one or more events, it must often look on the blackboard for other information that is related to these events. This search for associated data involves: 1) computing approximate attribute values for the kind of blackboard objects that are relevant to computations stemming from these triggering events, and then 2) finding those objects on the blackboard. For example, a KS that is triggered by the sudden movement of an unfriendly unit toward a friendly position might look on the blackboard for related movement of other unfriendly units that could indicate the initiation of an orchestrated threat. Units of interest would be unfriendly, within some radius of the friendly position, and may have recently changed their movements. The identity names of these units of interest are not known, nor are they linked to the unit whose change triggered the event or to units at the friendly position. The units of interest can only be determined by the approximate values of some of their attributes. The importance of such proximity-based associative retrieval to locate relevant objects that have been placed on the blackboard by other KSs is often overlooked in casual discussions of blackboard systems.

In Figure 2-6, most KS executions in a blackboard system involve the following steps:

1. The control shell is notified of an event of interest to the KS
2. This triggering context is used to activate the KS
3. The KS uses the triggering context to determine the ranges of attribute values that are relevant to the triggering context and looks on the blackboard to see what additional blackboard objects have attributes within those ranges
4. The KS uses the retrieved objects and the triggering context information to perform its computations
5. The results of this computation are written onto the blackboard

In this sequence, step 3 is the step associated with investigation.

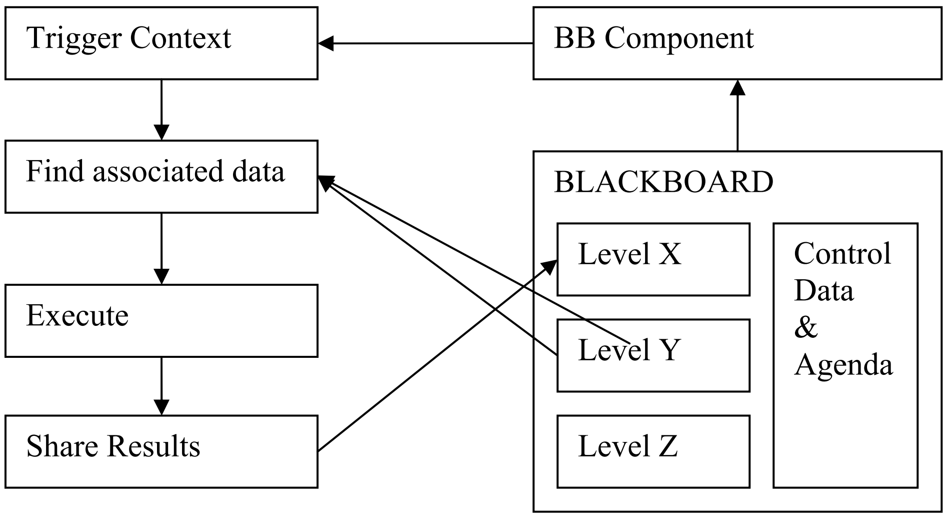


Figure 2-6 : KS Activities

2.5.4. INTERACTION

Blackboard systems prohibit direct interaction among modules, as all communication is done via the blackboard. Traditional blackboard systems have only a single control thread and execute only one KS activation at a time; once execution is started, the KS activation runs to completion or until it is aborted by the control shell. This means that all interaction among KS activations is serial, is unidirectional from earlier to later executions, can have unbounded latency, and is indirect via the blackboard. This severe restriction on interaction greatly simplifies the development of blackboard applications, but in certain situations this restriction can be a significant collaborating-software limitation [6].

Assume that KS A and B both are interested in the same event and can both do some initial work without interacting with one another. However, the initial work of A is needed for B to complete its work and vice versa. In this situation, the blackboard-application designer must artificially split A into two KSs, APRE and APOST, and similarly, B is split into BPRE and BPOST. Once APRE completes, BPOST can begin and, similarly, once BPRE completes, APOST can begin. If a lot of interaction is required, this KS-splitting approach can result in a large number of artificial KS fragments. Alternatively, the same iterative form of interaction can be achieved by creating KSs that are able to jump into later computations, based on the information present on the blackboard. In this case, multiple KS executions are still required to support the serial interaction, but the number of KSs present in the system does not need to be increased.

Parallel and distributed blackboard-system extensions of the classic, single-threaded blackboard architecture allow true concurrent KS executions, and this raises another important interaction issue. If the KSs are to remain anonymous and indirect in their interaction, then all interaction must still occur via changes to the blackboard. Executing KSs must be able to notice and respond to changes made to the blackboard during their execution to support such indirect interaction. We could also extend the KS model to allow for direct communication among co-executing KS activations. However, this is a major departure from the blackboard-system model, and it is problematic because of the uncertainty about which KS activations will be executing concurrently at any moment.

2.5.5. INTEGRATION

Integration and representation are closely linked in blackboard-system applications. The representation choices that are made not only affect the ability of KSs to use the results of others, but also how KS results are combined. In a blackboard application, integration of results involves three major activities [7]: relationship management, attribute merging, and value propagation.

The need for relationship management occurs when a KS execution wants to create a new object on the blackboard and the semantics of the blackboard representation requires that the relationship between the new object and some existing objects be represented. A simple example of this is the creation of a higher-level object as a result of identifying a set of lower-level supporting objects, such as creating a platoon object based on a set of individual unit objects. If this synthesis activity is performed by a single KS execution, the relationship between the new platoon object and the set of supporting unit objects can be easily represented by also creating support links that connect the objects. Such support links explicitly maintain the relationship between the objects on the blackboard.

What blackboard-system component should be responsible for maintaining these relationships? There are two approaches for this answer.

One approach makes each KS responsible for this. When an executing KS wants to create a new blackboard object, it must first check to see if a semantically equivalent object already exists which involves a blackboard retrieval. If one is found, the KS modifies the relationships of the existing object instead of creating a new object and relationships. This approach requires that each KS writer perform this check and that the semantics of equivalency are consistent across all KSs.

Another approach is to make equivalent-object checking and relationship management part of the unit-creation operation. In this case, the KS would ask to create a new platoon unit with links to the support units and the blackboard itself would perform the required bookkeeping. This latter approach begins to move the blackboard from a passive repository to a more active entity with application knowledge about what constitutes equivalency and how to handle duplicate creation requests.

The second integration activity is attribute merging. As with relationship management, we can have the KS execution determine the new belief value or we can have the blackboard object-creation routines do it automatically. The problem with the latter approach is that the knowledge required by the blackboard grows with the complexity of determining merged values. We certainly do not want to end up duplicating much of the knowledge used by KSs in computing new blackboard objects in automatic blackboard-integration routines. Clearly, as the number of object attributes that need to be appropriately merged grows, the complexity of work required by either every KS execution or the shared blackboard representation maintenance routines also grows.

The third integration activity is value propagation. Assume that a belief associated with a platoon object is a function of the beliefs of its supporting units and their spatial locations relative to one another. Assume a field report is received that contains a confirmed sighting of one of the supporting units of the platoon and that the KS execution that processes this information increases the belief value of the supporting unit. We would like this increased belief value to propagate to the platoon value, increasing our belief in it as well. Again, we could make this propagation be the responsibility of the executing KS or an activity of a more active blackboard repository. Similarly, suppose yet another KS execution, using different sensor data than was available to earlier KS executions, is able to compute a more accurate position for one of the support units and changes the position attribute of that support unit. We would like this new position value to be used to update the position attribute of the platoon unit and, potentially, the platoon unit's belief value if the new location of the supporting unit affects the belief calculation. Historically, blackboard systems have handled these integration activities in a very specific manner. Some applications placed the responsibility for these activities with the executing KSs. This required substantial discipline on the part of KS writers to maintain semantic consistency across KSs. Other applications placed this responsibility with the blackboard, risking duplication of KS knowledge and the potential for inconsistency if the way that the KS performed its activities was changed significantly. By careful modularization and sharing of code among KSs and the "active" blackboard, it is possible to reduce this duplication and risk. Finally, some applications dealt with value propagation by simply triggering and executing KSs again if important attributes used in their contributions changed. In this case, a re-executed KS simply replaced its original contributions with the latest version. Each of these approaches worked well enough in specific situations and, when used with care, allowed complex blackboard-system applications to be built.

Also, creating a principled integration model for a blackboard application requires close analysis of how the KSs in the application operate in conjunction with one another. From a practical standpoint, how we maintain the consistency of the result-integration model with the current KS set is an important issue. Just as KS-specific control expertise is developed and maintained with each KS, it is important to develop KS-specific models of result generation that can be incorporated into an overall result-integration model when the KS is added to the system. In Figure 2-7, you can see this separation of result integration. Such a capability remains to be developed, but it is an important research goal in enabling principled result integration in applications, that will have many KS changes throughout their lifetimes.

Finally, the degree that results are shared in a blackboard application has a direct relation to the complexity of the result-integration models. The integration model need only address results produced by KSs that are placed onto the blackboard, so there is a tension between limited sharing and aggressive sharing. Notice that, limited sharing is the small size of the integration model and aggressive sharing is a complex integration model. Principled result integration adds yet another consideration to degree of sharing design decisions.

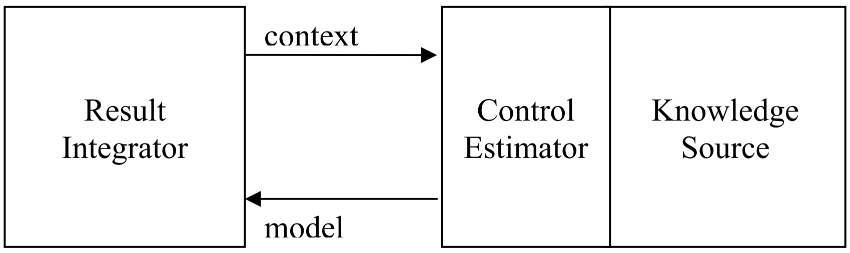


Figure 2-7 : Separation (Encapsulation) of Integration Knowledge

2.5.6. COORDINATION

The last collaborating-software challenge is running the right KSs on the right data at the right time. The opportunistic control that is the hallmark of blackboard systems is highly flexible, responsive, and generally efficient. During each control cycle, a traditional blackboard system makes a single, instantaneous choice of the best KS activation to execute and, if new conditions warrant, the system can focus its attention on a new area as early as the next cycle. As discussed earlier, executing only one KS activation at a time also greatly simplifies the architecture. Nevertheless, even achieving effective single-threaded control in a complex blackboard application can be challenging.

At any given moment, a blackboard application rarely lacks choices among a large number of potential KS activations to execute. These choices stem from multiple inputs arriving into the system, combinatory ways in which this data can be combined and used, and, in many applications, multiple KSs that can be applied to the same data. Figure 2-8 shows some models of these KSs. This results in a large and dynamic space of possible KS executions, of which only a small fraction can be pursued. Because blackboard systems operate incrementally, poor choices early on can result in triggering a large number of inappropriate downstream KS executions in response to the results generated by a single "inappropriate" KS execution. Agenda-based control uses a utility-based rating computed for each KS activation to select the best activation to execute in each cycle.

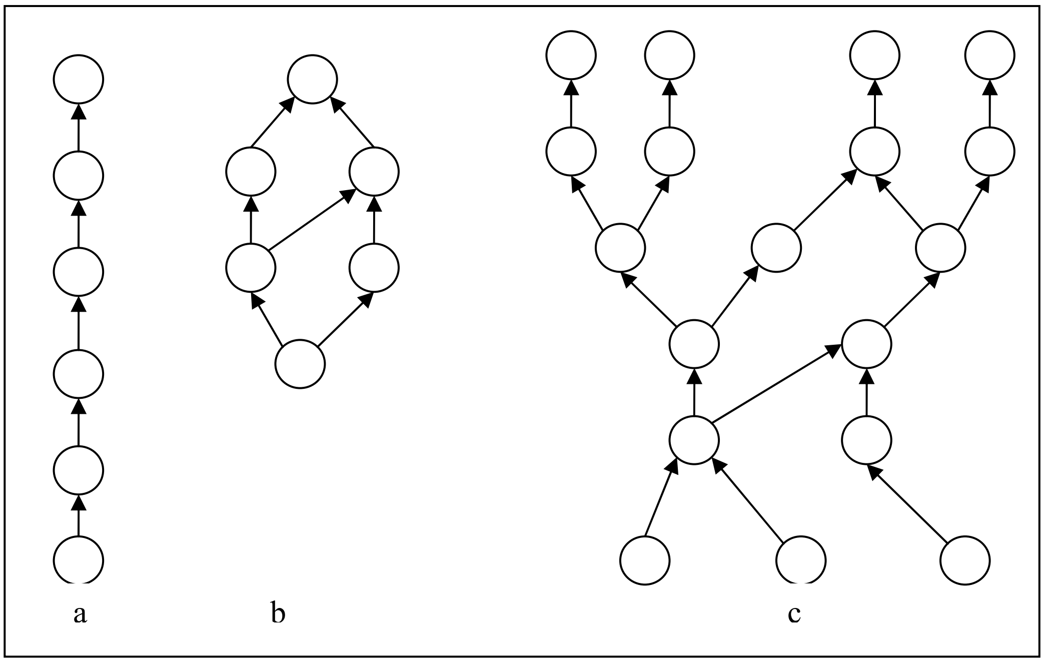


Figure 2-8 : Linearly and partially ordered KSs

This rating incorporates the estimates of what the activation will do if executed and more global requirements, such as parts of the solution that need attention. Unselected activations remain on the agenda, potentially to be executed in the future.

While the activations that are queued on the agenda await execution, the state of the blackboard and of overall problem solving is being changed by other KS executions. This results in a *queue-latency problem* [8] where the information associated with the KS activation becomes inconsistent with the current situation. In Figure 2-9, we can see a model of queue latency problem. One naive solution to this problem is to re-rate all KS activations on every cycle. However, since the number of pending KS activations can become large, this is not an efficient solution, particularly if the re-rating of each KS activiation involves searching the blackboard for changes relevant to the activation. Blackboard systems using this approach have needed to artificially limit the number of activations held pending or to re-rate only the topmost activations, under the assumption that the other ratings would not have changed too drastically. Other systems have organized the agenda in much the same way as the blackboard, so that the control shell could quickly identify pending KS activations that might be affected by changes on the blackboard. Event based re-triggering of pending KS activations is an example of this strategy.

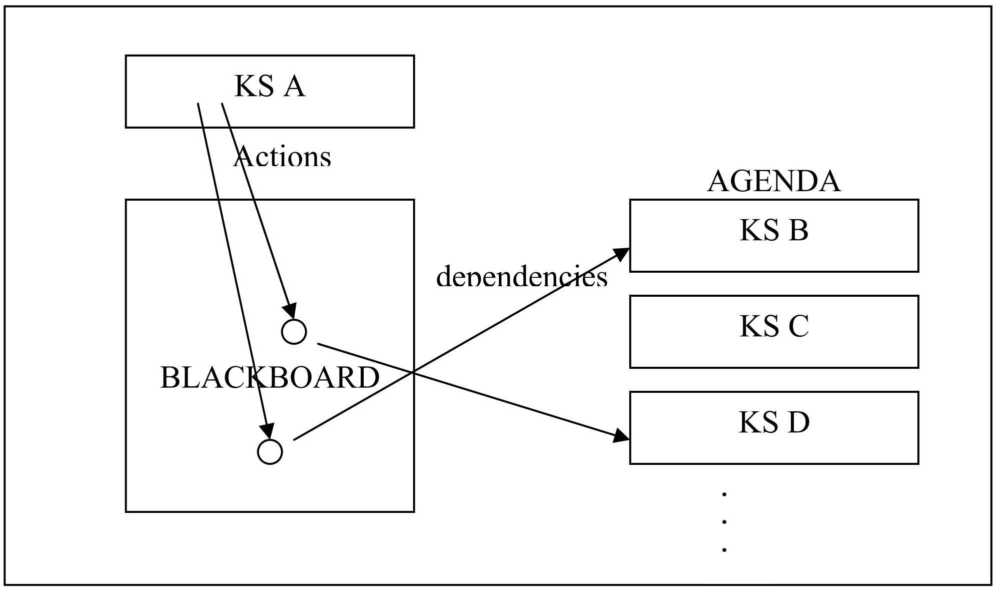


Figure 2-9 : The queue latency problem

In addition to the queue-latency problem, simple agenda based control techniques can introduce unwelcomed depth-first bias to opportunistic control. Consider the agenda shown in Figure 2-10. KS activations of A and B have the same rating with C close behind. From a control standpoint, these can be considered equally valid choices to be executed next. If A is selected and executed, its results may trigger a number of other KS activations, such as X, Y, and Z, potentially at higher ratings than B and C. If B had been selected instead of A, it might have triggered X0 and Y0, again with ratings much higher than A and C and potentially even higher than the ratings of X, Y, and Z. To be fair, and to make our control decisions as informed as possible, we should execute A, B, and C before executing any of the KS activations that are triggered by them.

This is one simple example of some of the problems that result from making instantaneous, history and purpose free, control decisions, and this problem was observed in the original Hearsay-II blackboard system [4].

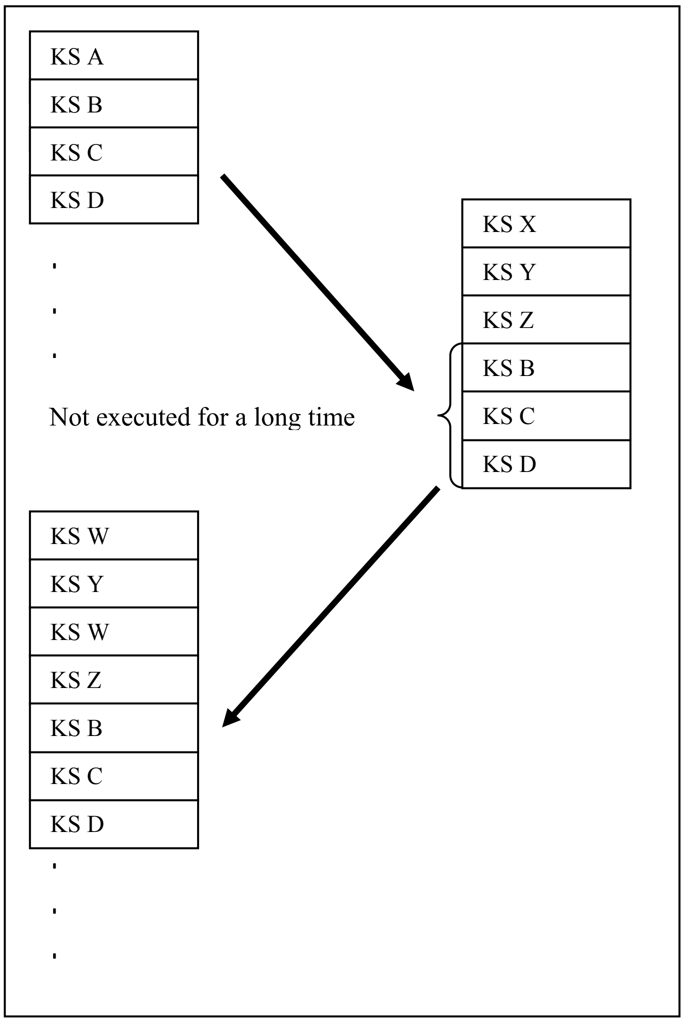


Figure 2-10 : Depth-First Search Bias

3. THE PROPOSED ARCHITECTURE

The proposed architecture extends and elaborates the standard architecture and has the following characteristics:

- The blackboard control architecture defines an explicit control blackboard.
- The blackboard control architecture defines explicit control knowledge source
- The blackboard control architecture defines a simple, adaptive scheduling mechanism.

The basic control loop of the proposed control architecture employs the following three steps:

- Update the set of pending goals
- Select a pending goal
- Execute the owner KS of the goal selected

The basic control loop is expressed in terms of goals rather than KSs. The set of all goals to relevant to a problem form a general goal tree. Let's discuss the proposed control architecture.

3.1. ELEMENTS OF THE CONTROL LOOP

Before to go in deep of the architecture, let us summarize the elements of the architecture:

a. **Knowledge Sources**: Our computational modules that together contain the expertise needed to solve the problem.
b. **Policies:** Our local scheduling criterion which guides to bidding process and it indicates which of the attributes of the knowledge sources are relevant in the process.
c. **Bids**: Our mechanism to determine the knowledge source to be executed at the current cycle by evaluation of each parameters in the knowledge source. And if we want to see all the solutions for a problem, then, we can add the bidding mechanism at the end of the architecture.
d. **Strategies**: Our global scheduling criteria such as depth-first, breadth-first etc.
e. **Methods**: A method is a partially complete general goal tree structure representing high level knowledge on how to solve a problem. The method is used to reduce the number of children of the nodes. Hence, the size of the problem is reducing in the search space. Also, we can give the execution order of the children nodes.

f. **Facts**: Facts are like knowledge sources. We can think the facts as nodes, however, domain facts have no local evaluation parameters and always higher priority than knowledge sources.

3.2. THE CONTROL LOOP IN DETAIL

The pseudo code of control loop is given in Figure 3-1:

```
take the goal that is in front of the queue
if there is a KS that solves the problem
        put KS into solution list
if there are pending domain facts then
        if goal matches a pending domain fact
                put domain fact into solution list
        elseif goal unifies some pending domain facts
                select one of the domain facts
                unify it
                put domain fact into solution list
        endif.
if there are no pending domain facts
        identify the policy for the goal and the owner KSs
        set the goal tree
        reduce pending KSs according to the method
        get the bids of the KSs
        select the winning KS according to the current policy
        Execute the KS to generate the goal to be posted
```

Figure 3-1 : The Control Loop of the proposed Architecture

After the system gets the global data (KSs, strategy, facts, policies and methods), it enters to the control loop. The control loop's purpose is to find an optimal solution set about the problem defined.

At first, the control architecture defines the starting point of the problem. Then, it generates a new stack and appends all the adjacent nodes to the starting point. The nodes are selected according to the strategy. After that, the control loop searches the facts for these adjacent nodes. If any found, then the system puts this fact into the solution queue. If there is no fact about the starting point, then the system eliminates the adjacent nodes according to the method. After eliminating nodes, the system calculates the bids of the remaining nodes, and appends the higher valued node to the solution list. Then, the system takes the selected node as the starting point and regenerates the stack that includes all the adjacent nodes to the starting point. This process goes on until the problem is solved or there are no more nodes. In the next chapter, we will discuss the implementation of the structure.

3.3. A SIMPLE EXAMPLE OF ARCHITECTURE

For better understandability of the control architecture, let us give a simple but effective example.

3.3.1. FLIGHT TICKETING PLAN

In this example, we will see how can we apply this structure to an application. The example application finds the optimal flight plan for a customer.

What does this simple application do is:

- Takes all the flight info into memory
- Takes additional information for the flights (facts, methods, policies)
- Then takes the origin and the final destination from the user.
- Finally, finds a solution according to information given

3.3.2. THE SCENARIO

If we write a scenario for this application, it would be like this:

There is a small airline company that makes charter flights accros USA. The company has eleven different flight routes to the most popular eight cities. The cities are :

- New York
- Chicago
- Denver
- Toronto
- Calgary
- Los angeles
- Dallas
- Houston

Each flight service have a distance between origin and the destination of the flight and each route have its own price. The distance and the price are affecting to the selection of service at the bidding mechanism in the program. As we said before, all knowledge sources and policies act like human experts. So, each flight services act like a server, and the bidding mechanism will act like a customer.

The flight routes are shown in Table 3.1:

Table 3.1 : Flight Services List of the Company

From	To	Distance(Miles)	Price(USD)
New York	Chicago	900	350
New York	Toronto	500	350
New York	Denver	1800	550
Chicago	Denver	1000	400
Denver	Dallas	1000	400
Denver	Houston	1000	400
Denver	Los Angeles	1000	400
Toronto	Calgary	1700	500
Toronto	Los Angeles	2500	850
Toronto	Chicago	500	350
Houston	Los Angeles	1500	475

If we draw a map of the flights, it would be seen like in Figure 3-2

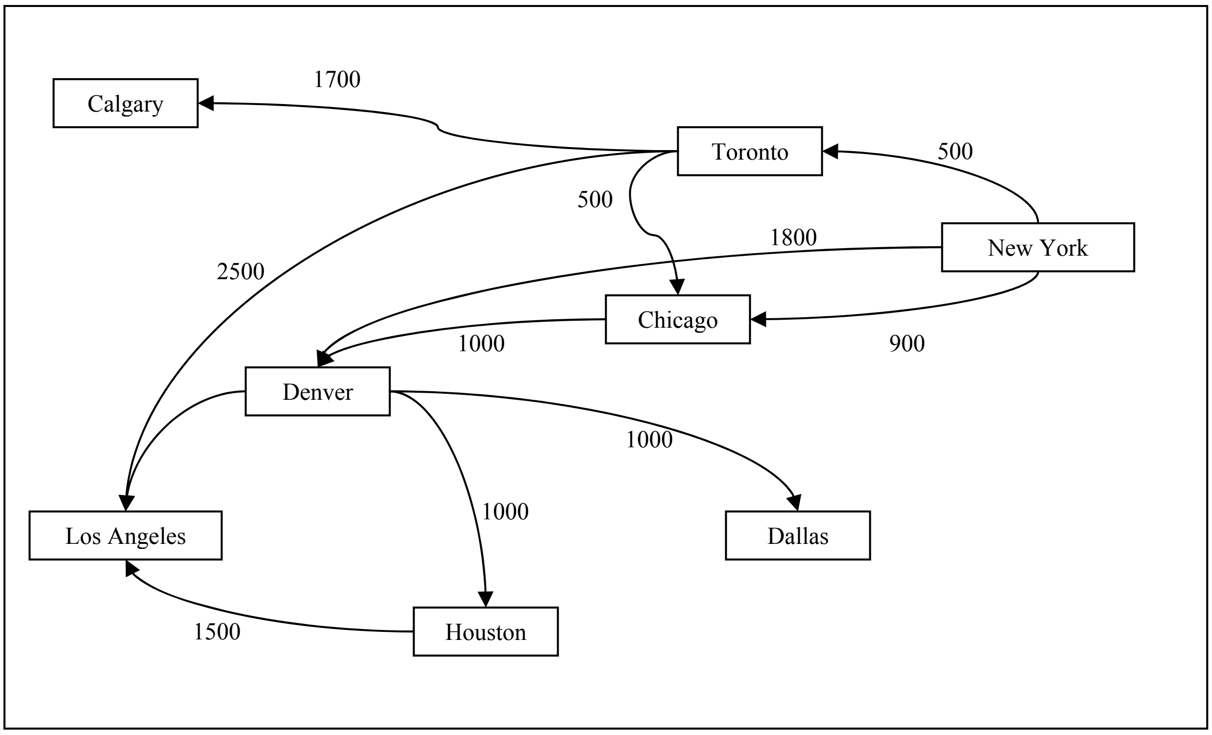


Figure 3-2 : Map of the Flights

And a customer want to go from New York to Los Angeles. Then, lets see what will happen!

3.3.3. ELEMENTS, KNOWLEDGE SOURCES AND FACTS

▪ **The elements of the program:** Actually we have mentioned about the elements of the architecture, however, there are some special things that we have to explain. First of all, our class name is **ControlBB** again. However, we have changed the name of **Goals** structure to **FlightInfo** since the scenario is about a flight planning program. Also, we have changed its parameters name. Hence, it is easier to follow the program steps in terms of program parameters. The new structure can be seen in the Figure 3-3.

```
struct FlightInfo {
  string from;    // departure city
  string to;      // destination city
  int   distance; // distance between from and to
  int   price;    // price
  bool  skip;     // used in backtracking
  short key;      // key field to select the best solution
```

Figure 3-3 : The FlightInfo Structure for KS in the blackboard

As is seen in Figure 3-3, the '**source_node**' was named as '**from**', the child node was named '**to**'. '**distance**' and '**price**' are our '**cost_parameters'.** Remember that we can add as many cost_parameters as we want into the proposed structure. **skip** and **key** parameteres are the same. Then, all other parameter names were adjusted in this way.

▪ **Knowledge Sources :** In the example, each of the flight services are one knowledge source. So we have eleven KSs in this scenario. Although we have six parameters in FlightInfo structure, at the beginning of the program, we just need four of them. The origin of the flight service, the destination, the distance between origin and destination, and its cost to the customer. In the source code, we have used **addflight** function to add the KSs into the memory. The code part can be seen in Figure 3-4.

▪ **Facts :** A fact is a data structure that includes two objects. And 'facts' is a list that consists of one or more facts. For this example, a fact's first object is the origin of the flight and the second one is the flight's destination. In the source code, **SetFacts()** function is used to add some facts into memory.

```
// Add flight connections to database.
  bb.addflight("New York", "Chicago", 900, 350);
  bb.addflight("Chicago", "Denver", 1000, 400);
  bb.addflight("New York", "Toronto", 500, 350);
  bb.addflight("New York", "Denver", 1800, 550);
  bb.addflight("Toronto", "Calgary", 1700, 500);
  bb.addflight("Toronto", "Los Angeles", 2500, 850);
  bb.addflight("Toronto", "Chicago", 500, 350);
  bb.addflight("Denver", "Dallas", 1000, 400);
  bb.addflight("Denver", "Houston", 1000, 400);
  bb.addflight("Houston", "Los Angeles", 1500, 475);
  bb.addflight("Denver", "Los Angeles", 1000, 400);
```

Figure 3-4: Adding flight Information into the system

Like addflight function it is enough to write "**bb.SetFacts("New York", "Denver");**" to add a fact into the facts list. For our scenario, just one fact(from New York to Denver) was added into memory to show how it affect to the solution. Later, we will see how does this fact affect to the solution.

▪ **Policies :** There is one policy for all the flight connections in the system, however we can add different policies for all the connections. The system will check if there is a special policy for the current connection. If not, it will use the general policy to evaluate the bids. For this scenario, three policy parameters were defined. These parameters have all negative effects in the bidding mechanism, however, it may not to be always negative. The policy parameters are :

1. The distance between origin and the destination,
2. The price between origin and the destination,
3. The number of transit flights of the plan.

As you see, these three parameters have negative effects on the customer. Because, when one of these parameters increase, then the satisfaction of the customer will decrease.

▪ **Methods:** For our scenario we will give just one method. By using the method, we will exclude New York – Chicago connection. Then we will see how this would affect to the performance of the system.

3.3.4. EXECUTION OF THE SYSTEM

To test the proposed architecture entirely, we will execute the program three times.

At the first time, we will not add any facts or methods to see how system searches the general goal tree. Then we will add the method and rerun the program to see the effects of methods. And thirdly, we will add the fact to the program (the method is also will be included) to reach full performance control of the program. Also, we will analyze the traces of these three iterations of the program to compare their performances.

Before go any further, it is good to know that, in all three iterations, the program inputs will be the same. It means that, in all three cases, the origin will be New York and the destination will be Los Angeles, and all the KSs will be the same. In this case the general goal tree for New York will be like Figure 3-5.

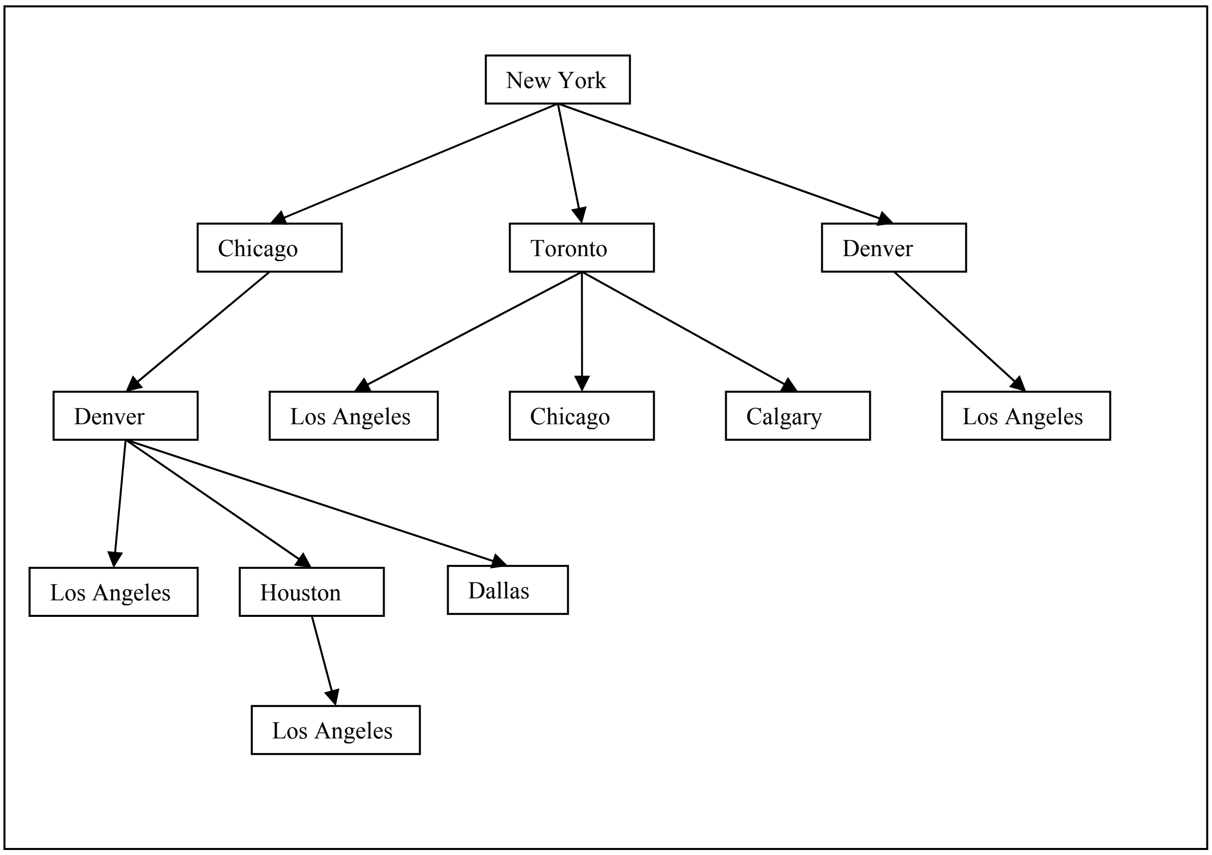


Figure 3-5 : General Goal Tree for NewYork

- **First Run**

In the first run, we do not add nor any facts neither any methods to the system. Then the system defines the general goal tree as the searching area. The control architecture makes a search in the whole goal tree. The type of the search is depend on the strategy that we define in the program. In this example a breadth-first search is made and all the solutions were written in the solutions.txt file. Then the system makes the bids of the three solutions according to policy. And selects the best solution between them. The output of the file is shown in Figure 3-6. In solutions.txt file, as we can see there are three solutions found, and the second was selected as the best solution. After the system finds three solutions, a bidding mechanism also helps us to select the best.

```
New York to Toronto to Los Angeles
Distance is 3000
Price is 2850
New York to Denver to Los Angeles
Distance is 2800
Price is 1550
New York to Chicago to Denver to Houston to Los Angeles
Distance is 4400
Price is 2650

THE Selected Route is:
New York -> Denver -> Los Angeles
Distance :2800
Price :1550
Steps :2
BID :-10260
KEY :2
```

Figure 3-6 : The output of Solutions.txt file

- **Second Run**

In the second run, we just add the method to exclude the flight from New York to Los Angeles. Hence the searching area was diminished very effectively. After exclusion of destination Chicago the new goal tree will be like Figure 3-7. In this new goal tree, we are seeing that the leftmost node in the first level of the search tree and all of its children were deleted in the search area. Figure 3-8 shows the output file after the second run of the program. Here also, we can see that the third solution of the first run was gone. And this output proves that using a method can be an effective way to improve the performance and quality to find solutions.

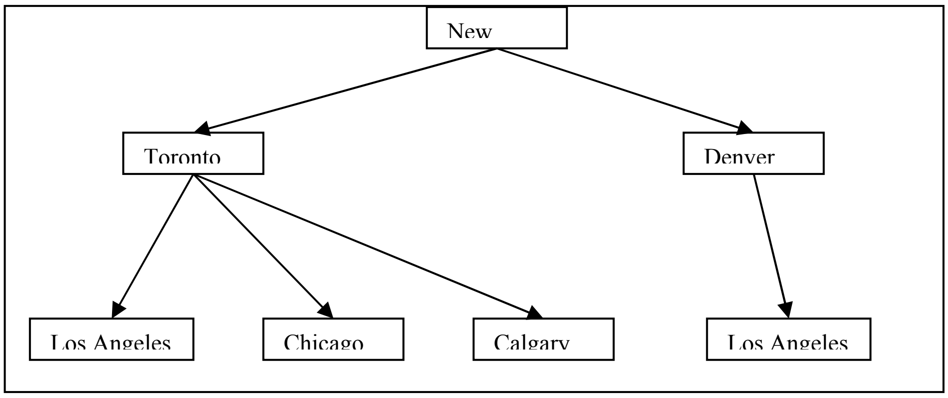


Figure 3-7 : The new sub-goal tree after adding the methods

```
New York to Toronto to Los Angeles
Distance is 3000
Price is 2850
New York to Denver to Los Angeles
Distance is 2800
Price is 1550
THE Selected Route is:
New York -> Denver -> Los Angeles
Distance :2800
Price :1550
Steps :2
BID :-13356
KEY :2
```

Figure 3-8 : The output of solutions.txt file after second run

- **Third Run**

In the third run, we have added the fact New York -> Denver. By this way, we exclude all other possible flights from the search tree. And we just guarantee that New York -> Denver is the first leg for a flight plan to go from New York to Los Angeles. Then, there is no more doubt that, it will search just two nodes to find a solution. In Figure 3-9 you can see the sub-goal tree after the fact is found. And the system will just search the children of Denver node after executing this KS. The output of the program is seen in Figure 3-10.

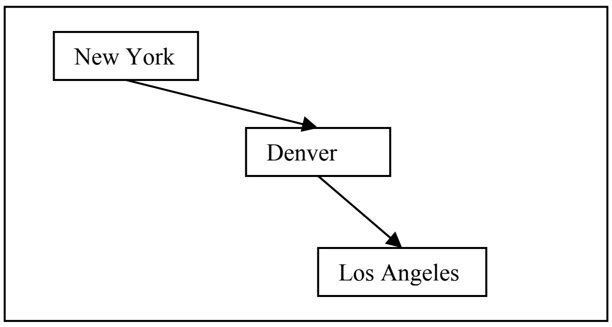


Figure 3-9 : The new sub-goal tree after adding fact

```
New York to Denver to Los Angeles
Distance is 2800
Price is 1550
THE Selected Route is:
New York -> Denver -> Los Angeles
Distance :2800
Price :1550
Steps :2
BID :-13356
KEY :1
```

Figure 3-10 : The output of the solutions.txt after third run of the program

4. IMPLEMENTATION

The proposed architecture is implemented in C++ language. C++ is a language that supports the concept of an "object", and provides a uniform means for referring to the objects in its universe. C++ provides an object-oriented model. [9]

The reason why the control architecture is implemented in an object-oriented language is that the possibility to orientate programming to objects allows us to design applications from a point of view more like a communication between objects rather than on a structured sequence of code. Hence, we can use the same architecture in many different decision making procedures.

4.1. GENERAL FLOW OF THE IMPLEMENTATION

The general flow of the proposed architecture is simple. As you can see in the Figure 4-1:

- Firstly, the system gets all the knowledge sources into memory from an outer system or from a user.
- Secondly, the system gets the problem definition.
- In the third step, the system gets all the facts, policies and methods into memory.
- Then, the system executes the control loop to find a solution set about the problem.
- Finally, the system returns the solution set if it can find any

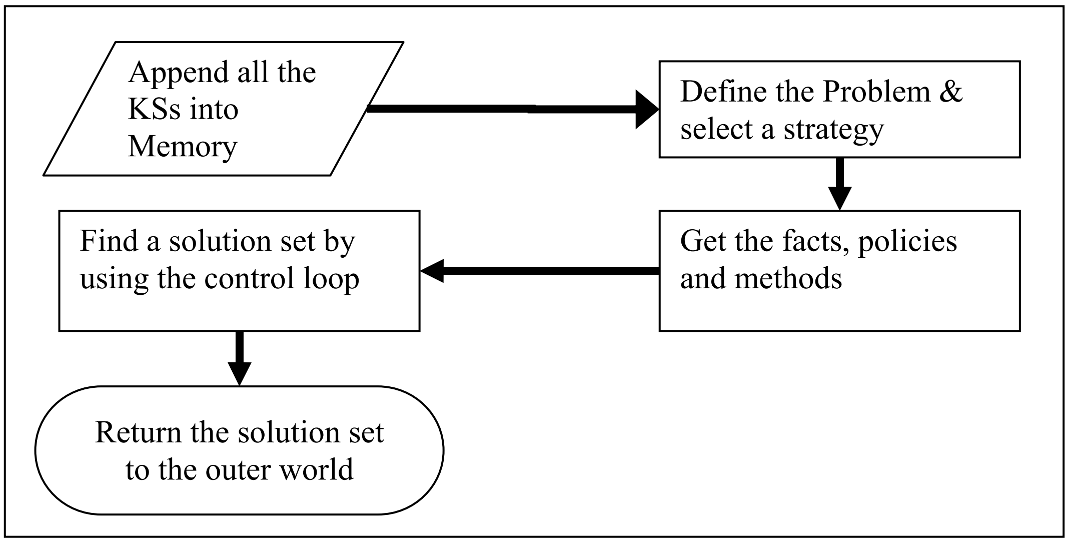

Append all the KSs into Memory
Define the Problem & select a strategy
Get the facts, policies and methods
Find a solution set by using the control loop
Return the solution set to the outer world

Figure 4-1 : Flowchart of the proposed architecture

4.2. ARCHITECTURE ELEMENTS

The elements shown below are general elements for the architecture. And these can be changed according to specified implementations.

First of all, lets define the data structures in the architecture:

- **Struct Goals :** This is the structure that defines a KS in the system. Its elements are:
    - **string source_node :** this is a string that defines a goal in the blackboard
    - **string child_node :** this is a string that defines a child of the goal in the blackboard.
    - **int _cost_parameter1 :** this is an integer parameter that you can define any property of the service between source_node and child_node. For example, if you are planning a daily touristic trip, it can define a quality of a restaurant. Or it can define the cost of the restaurant. Also we can define as many _cost_parameters as we want. It will affect the bidding mechanism.
    - **bool skip :** this flag is used for backtracking information
    - **int key :** this field provides a relation between KSs and solutions. It is like a key field in a database table.

- **Struct facts :** This structure defines a fact in the system. Its elements are:
    - **string source_node :** this is the string that defines a goal in the blackboard
    - **string child_node :** this is the string that defines the child node of the goal node in the blackboard.

- **Struct Methods :** This is the structure that defines a method in the blackboard system.
    - **string source_node :** this is the string that defines a goal in the blackboard.
    - **string child_node :** this is the string that defines tha child node of the curent goal node in the blackboard.
    - **char option :** this character can take just two values. ‘I’ or ‘E’. ‘I’ means include the KS and ‘E’ means exclude the KS in the goal tree. During search mechanism, this option is used to eliminate or to make an execution order the KSs. By this way, we can improve the perfromance of the search mechanism.
    - **int key :** this integer is useful only if the option is ‘I’. As we said above, the **option** ‘I’ means include the KS. And via using **key** integer, we can define an execution order of the sub-goals.

- **Struct Solutions :** This structure defines a complete solution in the blackboard.
  - **string source_node :** this is the string that defines a goal in the blackboard.
  - **string end_node :** this is the string that defines the last node of the solution list.
  - **int cost_parameter1 :** this parameter defines the sum of the cost_parameter1 in the solution list of the blackboard. We have to define one cost_parameter for each cost_parameters in the **goals** structure.
  - **int steps :** this parameter defines the number of steps in the solution list. This variable, is used to calculate the bid of the solution.
  - **int bid :** this parameter defines a total bid of the found solution. It is calculated by using the policies and the cost parameters.
  - **int key :** this is the key to make a relation of the found solutions. This is the second leg of the relation between KSs and solutions.

- **Struct Policy :** this structure defines a policy for the blackboard system. The policy can be defined for all KSs and can be specialized for any of the KSs. What parameters have this structure are :
  - **string source_node :** this is the string that defines a goal. Its value can be a goal value or an asterisk(*) . It its value is star, then it means it is valid for all KSs in the blackboard.
  - **string child_node :** this string defines a child of the goal node. Like source_node this parameter also can take an asterisk(*) value. If source_node contains a normal goal and child_node contains an asterisk then, it would mean the policy is valid for all child nodes for the goal defined in the KS.
  - **int rule1 :** this parameter defines a coefficient to calculate the bid by multiplying the first cost_parameter of the KS. For each **cost_parameter**, there must be a **rule** parameter.

- **Struct key :** this is the structure that defines a counter to make a relationship between solutions and KSs.
  - **int key :** this integer defines a unique key identifying the number of the solution, and its KSs.

Now lets define the Blackboard of the system. The blackboard is a class that defines all the KS information, policies, facts, methods, search functions, and so on.

Here are the blackboard elements:

- **Class ControlBB :** The base class of the blackboard system is ControlBB.
    - **vector goals** : this vector stores all KSs that were given to the system. It is constructed from the structure Goals.
    - **vector sub_goals** : this vector stores the goals that are adjacent to the starting point(i.e. children of the goal node). Like vector **goals** it is also constructed from the structure Goals.
    - **stack btStack** : this stack includes backtracking information. It is constructed from structure Goals. It includes executed KSs in the search space. We can call it (b)ack(t)racting Stack.
    - **stack slStack** : this stack contains the solution queue with key information that is used to sort and read at any point. The solution queue consists of all steps of the problem solution. That means that, if a solution consists of five KSs then, slStack includes five KSs. We can call it (s)o(l)ution stack.
    - **stack solutions** : this stack is used for bidding mechanism. It includes the problem definition(the first and the last goals), and the total of cost_parameters. And also, if there is more than one solutions, then it includes all solutions for the defined problem.
    - **stack bids** : this stack consists of all information in stack **solutions**. In addition to this information, the bid parameter of the structure is also calculated. It has the same structure as the stack **solutions**.
    - **stack policy** : this stack stores the policies. It is constructed from Policy structure.
    - **vector methods** : stores the methods for all the goals
    - **vector facts** : stores the fact about a starting point
    - **bool match**() : this function returns true if there is a direct connection between the starting point and the final goal
    - **Bool find_depth(string from, &sub_goals)** :This function makes the depth_first search to find a solution. And then, returns true if the problem was solved. During this function is executed, the solution queue is filled.

- **Bool find_breadth(string from, &sub_goals)** : This function makes the breadth_first search to find a solution. And then, returns true if the problem was solved. During this function is executed, the solution queue is filled.
- **Bool find_least(string from, &sub_goals)** : This function makes the depth_first search to find a solution. And then, returns true if the problem was solved. During this function is executed, the solution queue is filled.
- **void add_goal()** : This function adds KSs to the memory.
- **void show_solution()**: If there is a solution about the problem, then this function shows the solution queue.
- **void select_best()**: After evaluating the bids of KSs, this function selects the highest valued solution.
- **void set_policy()**: This function sets the policies of the blackboard.
- **void get_policy()**: this function gets the policy of the current service.
- **void evaluate_bid()**: This function calculates the bid of the service.
- **void set_facts()**: This function sets the fact vector for all the goals.
- **bool is_a_fact**: This function returns true if there is a fact for the current node.
- **void define_strategy()**: This function defines the searching strategy of the architecture such as depth_first, breadth_first etc.
- **void solve_depth()**: Solves the problem using depth_first strategy
- **void solve_breadth()**: Solves the problem using breadth_first strategy
- **void solve_least()**: solves the problem using least_cost strategy
- **bool problem_solved()**: returns true if there is a solution set in the memory.

4.3. THE CONTROL LOOP FLOWCHART

If we write the loop in terms of the architecture functions it seems like in figure(4-2).

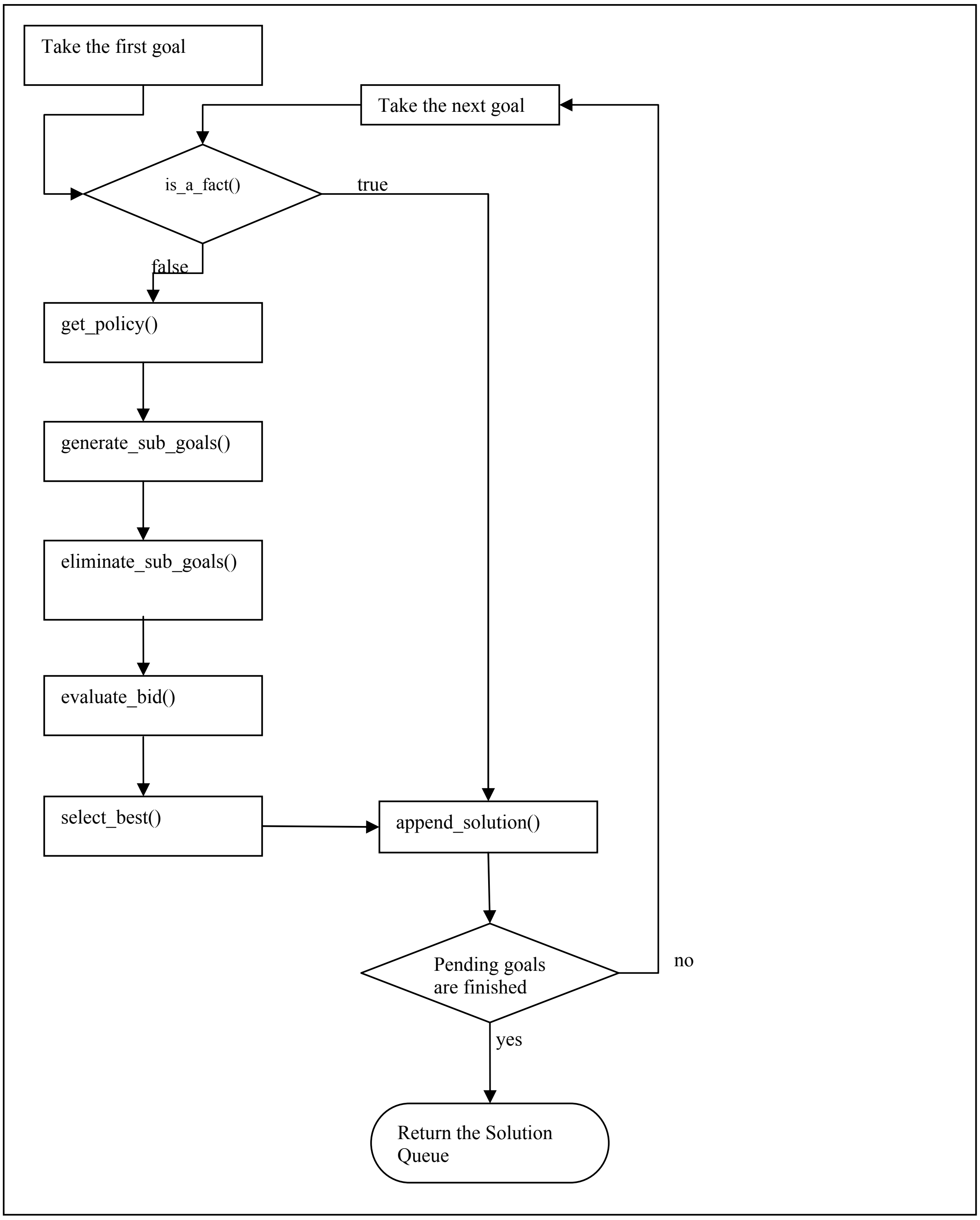


Figure 4-2 : Flow chart of control loop

5. EXAMPLES

In this chapter, two examples will be given. The first example is about ticketing problem on a flight schedule. And the second chapter is about routing planning in a factory on plastic injection machines.

5.1. APPLICATION: ROUTING PLANNER IN A FACTORY

In the previous example, we saw that how can architecture handle the AI search basically. But we can give another example to express its power. Consider, a factory produces several types of vacuum cleaners and one of its workshops produces all the plastic parts of the vacuum cleaners. The company uses an ERP program. Then, since the ERP program does not have an efficient planning tool, according to the scenario, our program has to make a plan for a plastic injection machine. There are several types of moulds in several colors. Setting-up the injection machine takes a time, and we want to reduce this setup time. Our program will take all information from a formatted file automatically (that would come from the ERP program), and then will make a plan that the machine will produce the parts in optimal time.

5.1.1. THE SCENARIO

As the scenario, we will analyze Elektropak's production process and then we will solve the production planning issue by our architecture.

Let us give some information about Elektropak: Elektropak is a company that produces vacuum cleaners, flat-irons and other small home appliances. Their products take Arzum, Conti and Rowenta trademarks in the market. Elektropak uses SAP system [10] to manage all its information in an integrated environment. All departments of Elektropak - such as accounting, sales and distribution, production etc. - are connected each other in SAP system.

In Elektropak, we can define a production process in four main steps [11]:

- **Design of a product**: in this step, technical designs of the product are drawn in R&D department. Then bills of materials (BOM) [12] are created and loaded into SAP. A BOM includes all parts of a product. The elements of a BOM are raw materials, semi-products, accessories and moulds. Also, semi-products have their own BOMs. Other elements do not have their own BOM and generally they are obtained from outer sources.

- **Design of routing plans in SAP**: in this step, for all new designed products, a production scheme is constructed. This scheme shows all the following information:
    - In which order, the elements of product in the BOM will be produced.
    - How much time would take the production of a unit product. (Also semi-product)
    - What will be the quantity of a minimum party of the product?
    - How the capacity of machines will be affected.
    - After all, how much will be the cost of the product

  Hence, Elektropak adds a new type of product to its product spectrum.

- **Production Orders:** According to customers' purchase orders and material resource planning (MRP) data, the system generates planned production orders for the products/semi products.

- **Production:** The last step is production of products. After producing all semi-products in the BOM, all the parts of the product is assembled in another workshop in the company.

The biggest workshop of Elektropak is injection workshop. In this workshop, all the plastic parts in the whole product spectrum of the company are produced. In SAP, there are 5 different types of injection machines. And the moulds of these machines differ from each other. In Table 5-1 you see a list of injection machine types, their names and the numbers of these machines in Elektropak. The number of machines is important because we want to reduce the number of working machines to reduce the cost.

The plastic semi-products that were ordered to produce, are produced by these machines. In a technical plan of a plastic semi-product, there are the following characteristics:

- The color code of the semi-product
- The plastic type of the semi-product
- The mould type that would be attached to the machine

- The production time for a unit semi-product

Table 5.1 : The list of injection machines

Machine Group	Machine Names	# of Machines
E1000G	1000 Gr. Injection Machines	9
E1600G	2000 & 1600 Gr. Injection Machines	8
E200G	600-100 Gr. Injection Machines	21
E2400G	2400 Gr. Injection Machines	1
E750G	750 Gr. Injection Machines	7

An ABAP report was written in Elektropak's SAP system to get a list of production orders for the machines that we have seen Table 5-1. In this list, we can see the following elements:

- Finishing date of the planned product
- Machine code for the semi-product
- Mould (setup) code for the semi-product
- Setup-time of the mould
- Color code of the semi-product
- Production quantity
- Current stock quantity of the semi-product
- Unit production time of the semi-product

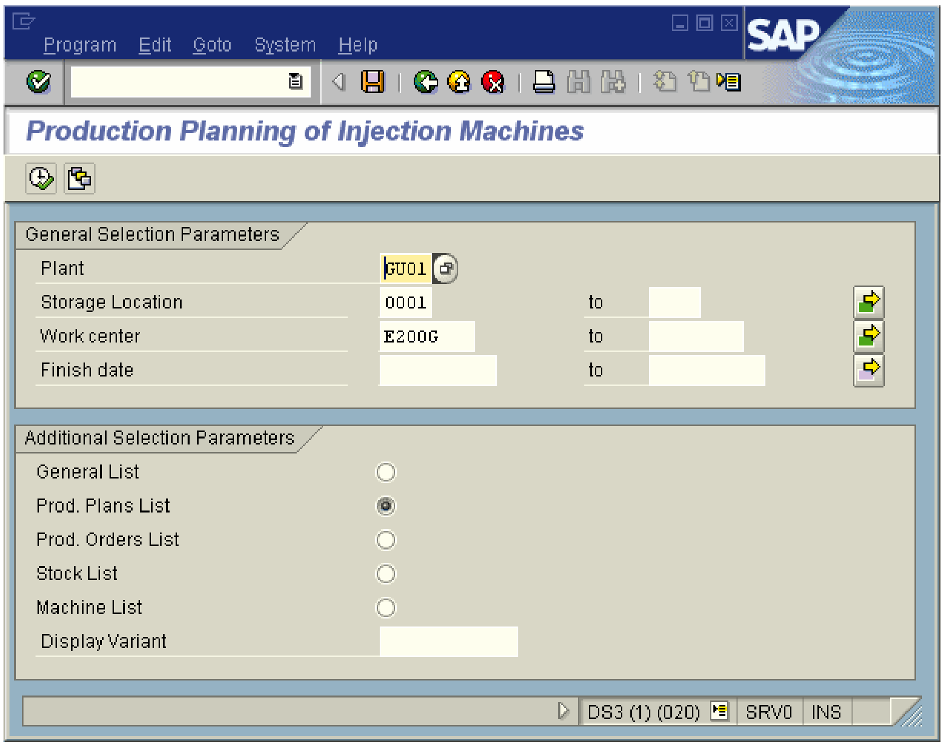
The purpose in production planning is producing maximum quantity of products with a minimum cost. Though we can see the semi-products list, within their mould types, setup times, dead-lines for the production etc, it becomes more complicated to handle all the production in the workshop manually. And at this point, our architecture will help us.

5.1.2. ABAP PROGRAM : ZPLANTEST

For SAP part of the example, a program was written in ABAP [13, 14] in the test system of Elektropak system. The program's name is ZPLANTEST. The general inputs of this program are plant name, storage location, machine codes and planned finishing dates. Also there are additional input parameters that define the type of the list. In Figure 5-1, we are seeing all the input parameters of the ABAP program. In this program, we can take several different lists either to see or to download into architecture. ALV tool (ABAP List Viewer) of

SAP was used to get the list. ALV is a special reporting tool in SAP [15] that will help us to analyze the problem more specifically:

- **Routing plans of semi-products**: in this list, we are seeing routing plan information of semi-products. The columns are material number, material explanation, routing plan number, machine code(this is also calles work center in SAP system), setup (mould) code, unit production speed and color code of the product. In Figure 5-2, we can see the routing plan list.
- **Production orders of semi-products**: In this list, there are five columns: production order number, material number, material description, order quantity and finishing date. In Figure 5-3, we can see this list.
- **Stock List**: If there are some stocks in the warehouse, then we can see this information in this list Figure 5-4.
- **Machines List**: Figure 5-5 shows the machines list. This is the list that we can see all the machines and all of their compatible mould codes. Also, we can see setup times of moulds in seconds.
- **General List**: This is the list that we all merged into one list. We also would download this list to the local computer to use further in the architecture. In Figure 5-6 we can see the screenshot of this list.



Production Planning of Injection Machines

General Selection Parameters

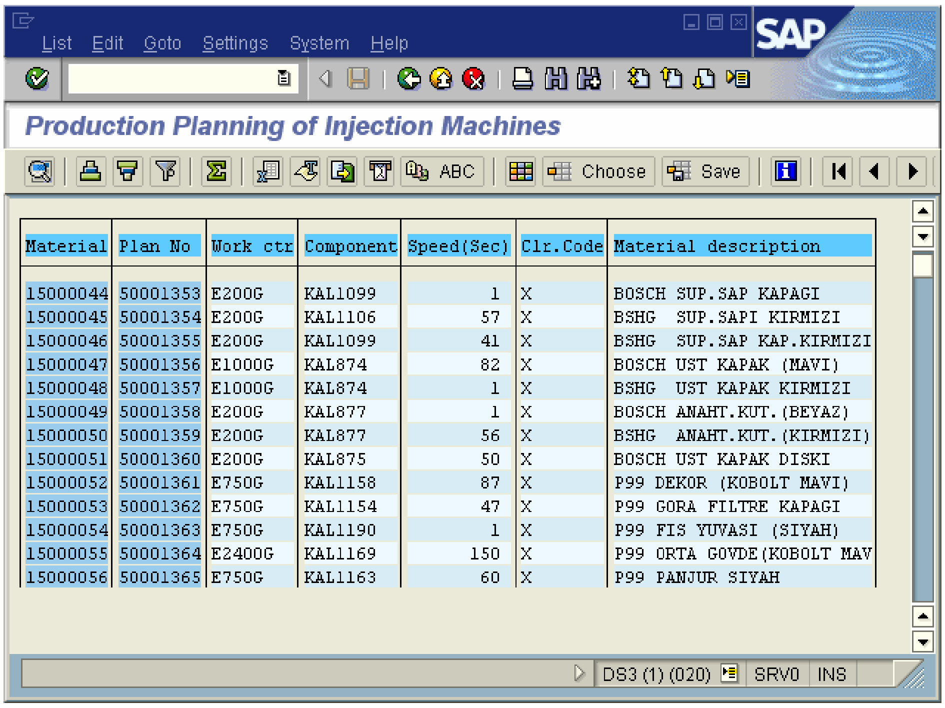
Plant	GU01		
Storage Location	0001	to	
Work center	E200G	to	
Finish date		to	

Additional Selection Parameters

General List	○
Prod. Plans List	●
Prod. Orders List	○
Stock List	○
Machine List	○
Display Variant	

DS3 (1) (020) SRV0 INS

Figure 5-1 : Selection parameters screen of ZTESTPLAN program

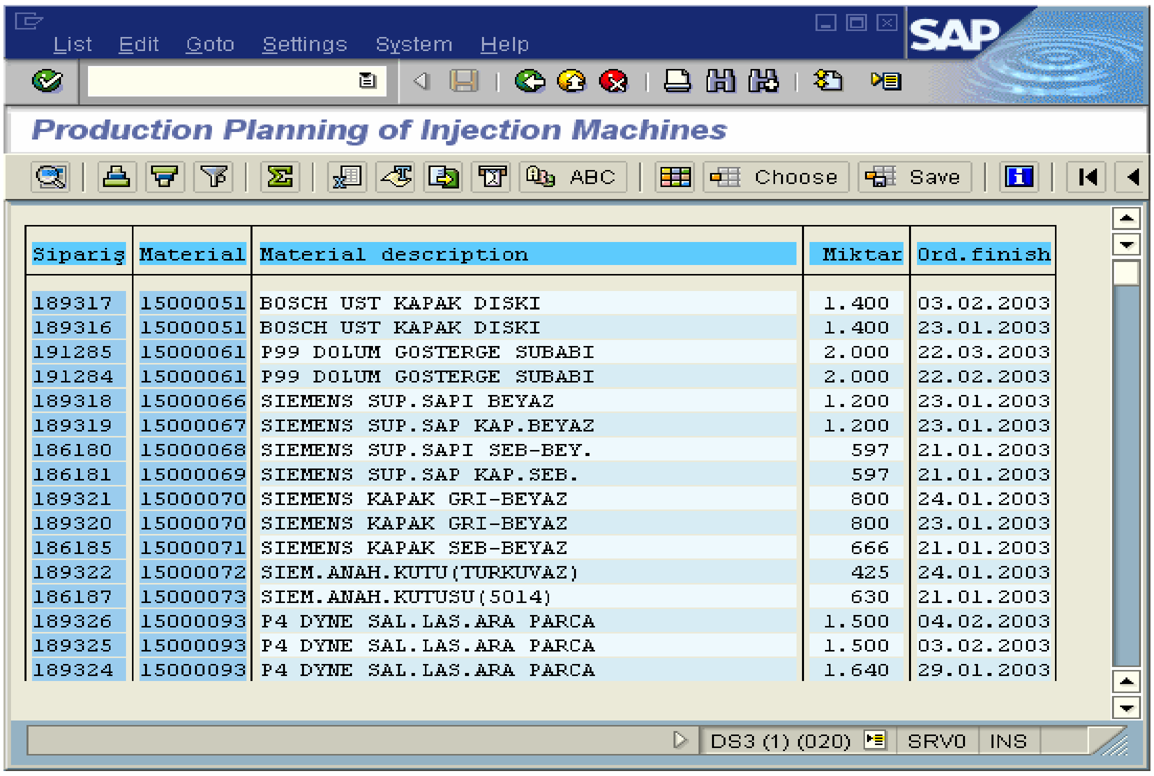


Production Planning of Injection Machines

Material	Plan No	Work ctr	Component	Speed(Sec)	Clr.Code	Material description
15000044	50001353	E200G	KAL1099	1	X	BOSCH SUP.SAP KAPAGI
15000045	50001354	E200G	KAL1106	57	X	BSHG SUP.SAPI KIRMIZI
15000046	50001355	E200G	KAL1099	41	X	BSHG SUP.SAP KAP.KIRMIZI
15000047	50001356	E1000G	KAL874	82	X	BOSCH UST KAPAK (MAVI)
15000048	50001357	E1000G	KAL874	1	X	BSHG UST KAPAK KIRMIZI
15000049	50001358	E200G	KAL877	1	X	BOSCH ANAHT.KUT.(BEYAZ)
15000050	50001359	E200G	KAL877	56	X	BSHG ANAHT.KUT.(KIRMIZI)
15000051	50001360	E200G	KAL875	50	X	BOSCH UST KAPAK DISKI
15000052	50001361	E750G	KAL1158	87	X	P99 DEKOR (KOBOLT MAVI)
15000053	50001362	E750G	KAL1154	47	X	P99 GORA FILTRE KAPAGI
15000054	50001363	E750G	KAL1190	1	X	P99 FIS YUVASI (SIYAH)
15000055	50001364	E2400G	KAL1169	150	X	P99 ORTA GOVDE(KOBOLT MAV
15000056	50001365	E750G	KAL1163	60	X	P99 PANJUR SIYAH

DS3 (1) (020) SRV0 INS

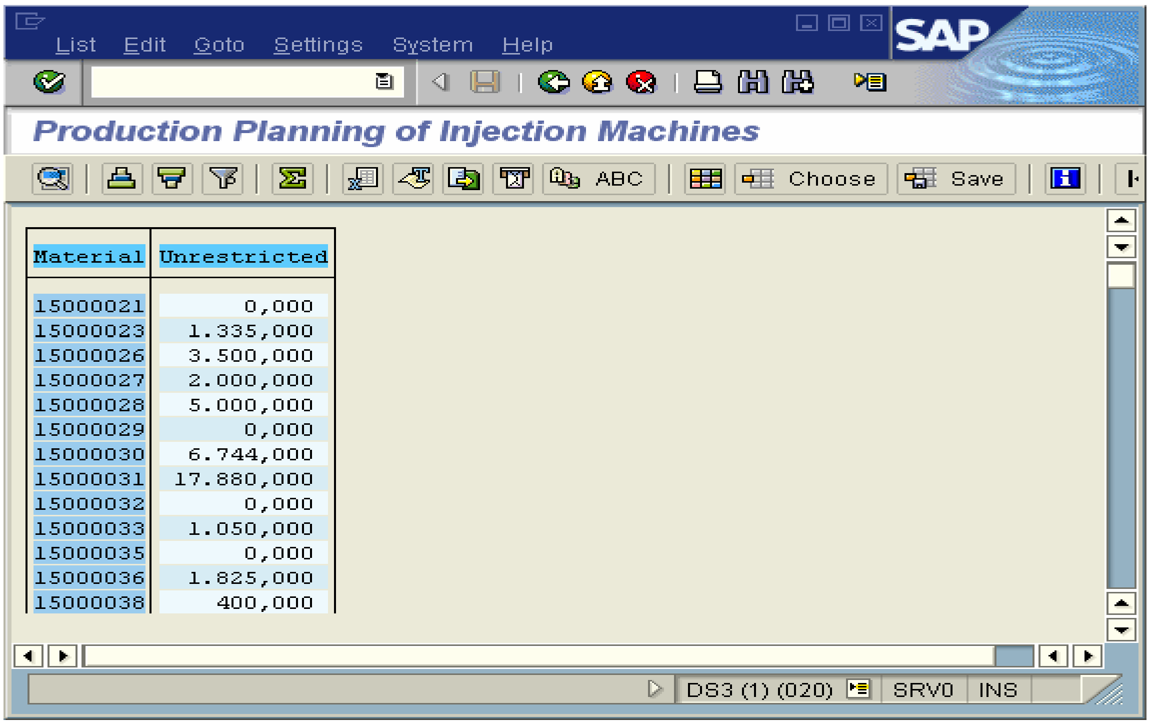
Figure 5-2 : Output of production plans list



Production Planning of Injection Machines

Sipariş	Material	Material description	Miktar	Ord.finish
189317	15000051	BOSCH UST KAPAK DISKI	1.400	03.02.2003
189316	15000051	BOSCH UST KAPAK DISKI	1.400	23.01.2003
191285	15000061	P99 DOLUM GOSTERGE SUBABI	2.000	22.03.2003
191284	15000061	P99 DOLUM GOSTERGE SUBABI	2.000	22.02.2003
189318	15000066	SIEMENS SUP.SAPI BEYAZ	1.200	23.01.2003
189319	15000067	SIEMENS SUP.SAP KAP.BEYAZ	1.200	23.01.2003
186180	15000068	SIEMENS SUP.SAPI SEB-BEY.	597	21.01.2003
186181	15000069	SIEMENS SUP.SAP KAP.SEB.	597	21.01.2003
189321	15000070	SIEMENS KAPAK GRI-BEYAZ	800	24.01.2003
189320	15000070	SIEMENS KAPAK GRI-BEYAZ	800	23.01.2003
186185	15000071	SIEMENS KAPAK SEB-BEYAZ	666	21.01.2003
189322	15000072	SIEM.ANAH.KUTU(TURKUVAZ)	425	24.01.2003
186187	15000073	SIEM.ANAH.KUTUSU(5014)	630	21.01.2003
189326	15000093	P4 DYNE SAL.LAS.ARA PARCA	1.500	04.02.2003
189325	15000093	P4 DYNE SAL.LAS.ARA PARCA	1.500	03.02.2003
189324	15000093	P4 DYNE SAL.LAS.ARA PARCA	1.640	29.01.2003

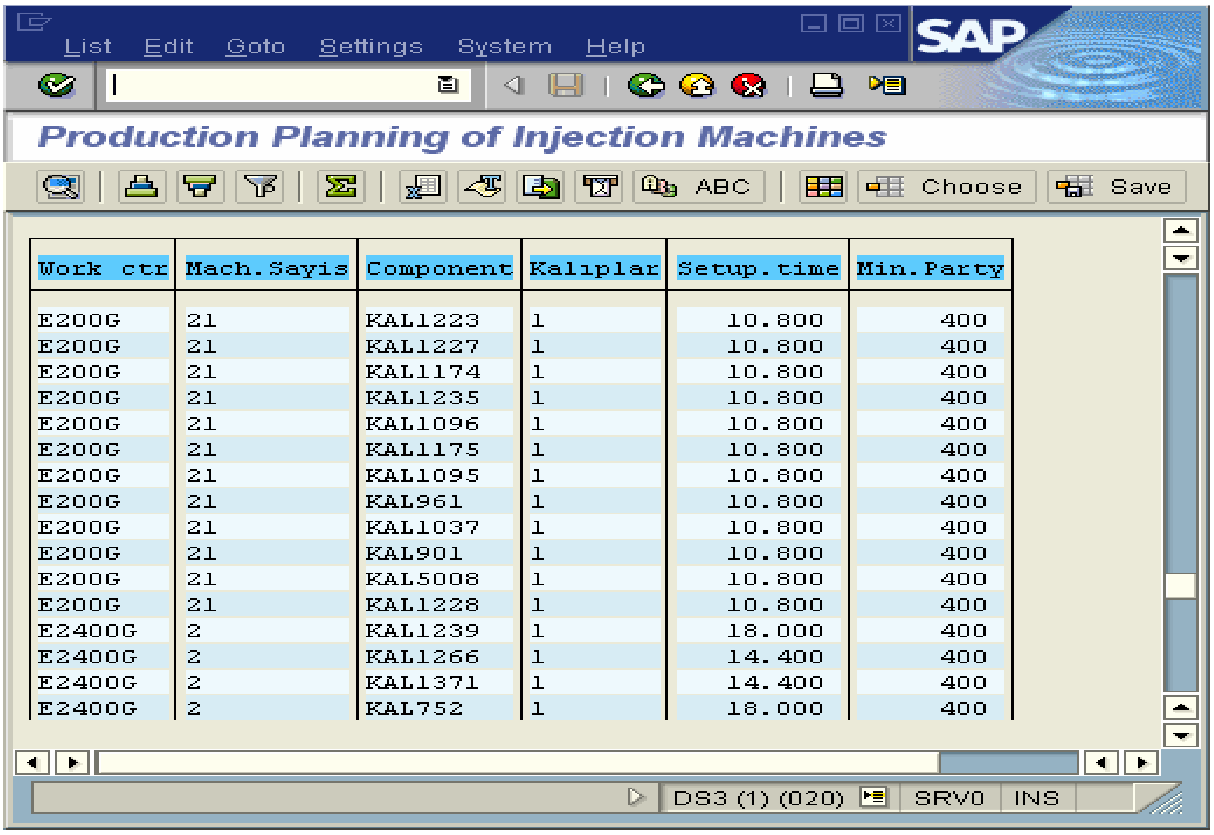
Figure 5-3 : Production Orders List



Production Planning of Injection Machines

Material	Unrestricted
15000021	0,000
15000023	1.335,000
15000026	3.500,000
15000027	2.000,000
15000028	5.000,000
15000029	0,000
15000030	6.744,000
15000031	17.880,000
15000032	0,000
15000033	1.050,000
15000035	0,000
15000036	1.825,000
15000038	400,000

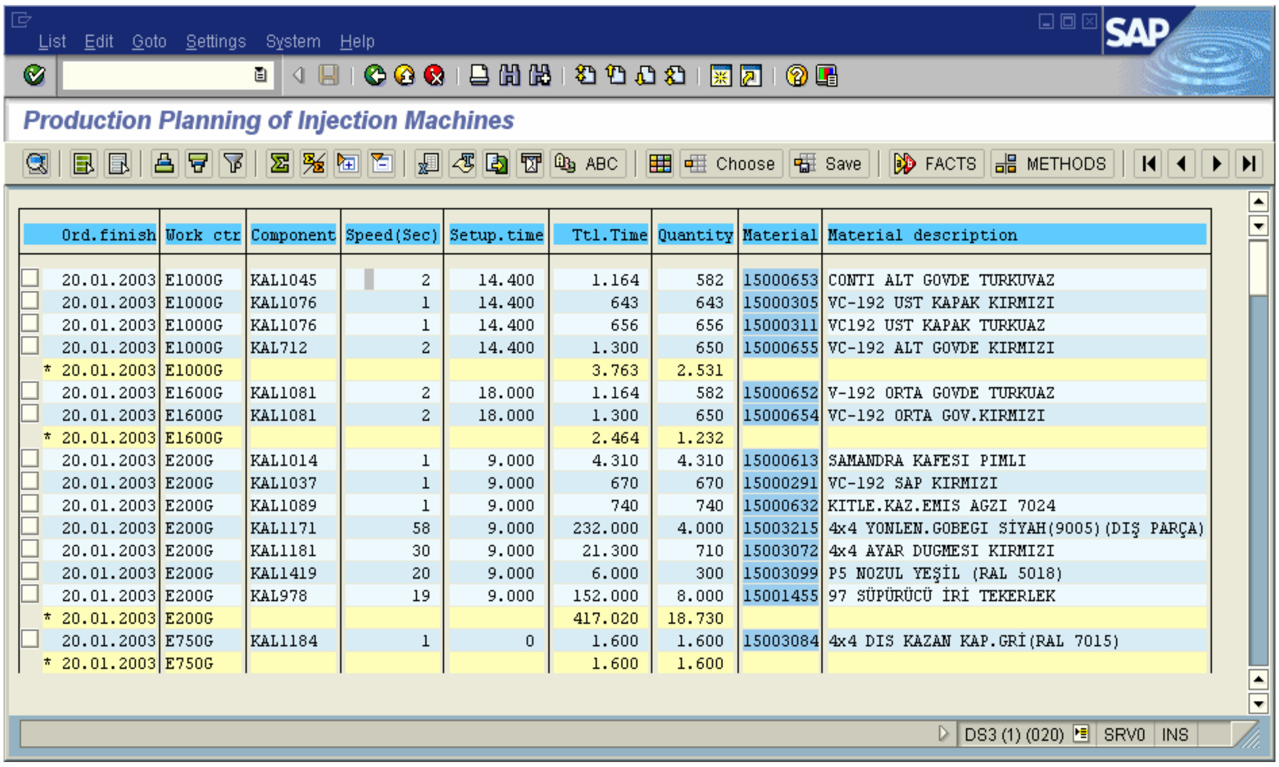
Figure 5-4 : Stock list of products


Production Planning of Injection Machines

Work ctr	Mach.Sayis	Component	Kaliplar	Setup.time	Min.Party
E200G	21	KAL1223	1	10.800	400
E200G	21	KAL1227	1	10.800	400
E200G	21	KAL1174	1	10.800	400
E200G	21	KAL1235	1	10.800	400
E200G	21	KAL1096	1	10.800	400
E200G	21	KAL1175	1	10.800	400
E200G	21	KAL1095	1	10.800	400
E200G	21	KAL961	1	10.800	400
E200G	21	KAL1037	1	10.800	400
E200G	21	KAL901	1	10.800	400
E200G	21	KAL5008	1	10.800	400
E200G	21	KAL1228	1	10.800	400
E2400G	2	KAL1239	1	18.000	400
E2400G	2	KAL1266	1	14.400	400
E2400G	2	KAL1371	1	14.400	400
E2400G	2	KAL752	1	18.000	400

DS3 (1) (020) SRV0 INS

Figure 5-5 : Machines List


Production Planning of Injection Machines

Ord.finish	Work ctr	Component	Speed(Sec)	Setup.time	Ttl.Time	Quantity	Material	Material description
20.01.2003	E1000G	KAL1045	2	14.400	1.164	582	15000653	CONTI ALT GOVDE TURKUVAZ
20.01.2003	E1000G	KAL1076	1	14.400	643	643	15000305	VC-192 UST KAPAK KIRMIZI
20.01.2003	E1000G	KAL1076	1	14.400	656	656	15000311	VC192 UST KAPAK TURKUAZ
20.01.2003	E1000G	KAL712	2	14.400	1.300	650	15000655	VC-192 ALT GOVDE KIRMIZI
* 20.01.2003	E1000G				3.763	2.531		
20.01.2003	E1600G	KAL1081	2	18.000	1.164	582	15000652	V-192 ORTA GOVDE TURKUAZ
20.01.2003	E1600G	KAL1081	2	18.000	1.300	650	15000654	VC-192 ORTA GOV.KIRMIZI
* 20.01.2003	E1600G				2.464	1.232		
20.01.2003	E200G	KAL1014	1	9.000	4.310	4.310	15000613	SAMANDRA KAFESI PIMLI
20.01.2003	E200G	KAL1037	1	9.000	670	670	15000291	VC-192 SAP KIRMIZI
20.01.2003	E200G	KAL1089	1	9.000	740	740	15000632	KITLE.KAZ.EMIS AGZI 7024
20.01.2003	E200G	KAL1171	58	9.000	232.000	4.000	15003215	4x4 YONLEN.GOBEGI SİYAH(9005)(DIŞ PARÇA)
20.01.2003	E200G	KAL1181	30	9.000	21.300	710	15003072	4x4 AYAR DUGMESI KIRMIZI
20.01.2003	E200G	KAL1419	20	9.000	6.000	300	15003099	P5 NOZUL YEŞİL (RAL 5018)
20.01.2003	E200G	KAL978	19	9.000	152.000	8.000	15001455	97 SÜPÜRÜCÜ İRİ TEKERLEK
* 20.01.2003	E200G				417.020	18.730		
20.01.2003	E750G	KAL1184	1	0	1.600	1.600	15003084	4x4 DIS KAZAN KAP.GRİ(RAL 7015)
* 20.01.2003	E750G				1.600	1.600		

DS3 (1) (020) SRV0 INS

Figure 5-6 : General list

5.1.3. ELEMENTS, KNOWLEDGE SOURCES AND FACTS

Above, we have introduced an ABAP program to feed a complex list of production orders to our composed architecture. Now, we will analyze our control architecture and see how it would make a plan for a specific machine for a specific finishing day.

A general goal tree actually includes the combination of production orders of the product. For example, if we have three products (A,B,C) to be produced in the same date and in the same machine, then the general goal tree for this machine would be seen like in the Figure 5-7. In this figure, it is guaranteed that all these products will be produced in the same date and in the same machine. However, their mould code and color code can differ from each other. For example, A and B can have the same mould code with different colors while C has the same color with A and different mould code from A and B.

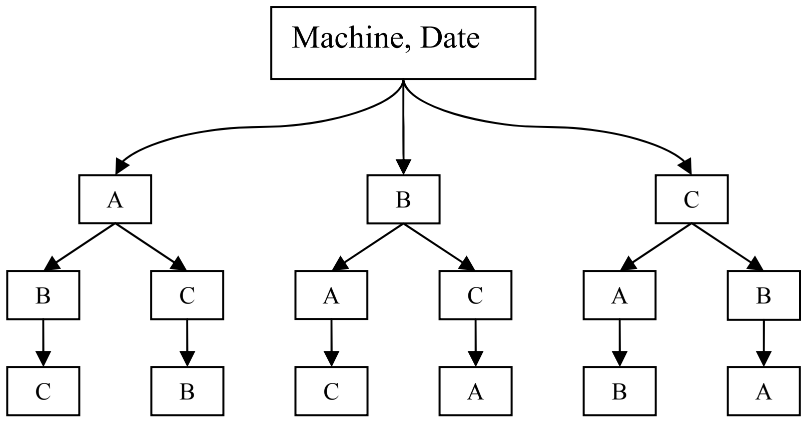


Figure 5-7 : Example for production combination tree for a machine

- **Knowledge Sources:** All KSs come from SAP system in a tabular formatted text file. And they are read from the text file. A KS of this program includes these parameters :
  - Production date
  - Machine Code
  - Mould Code (Setup Code)
  - Setup Time (in seconds)
  - Color Code
  - Product code
  - Unit production time (in seconds)

        - Number of products to be produced

All we have to do in the architecture is to generate a production order for these machines to reduce the production cost and increase efficiency.

- **Facts:** For this example, a fact can be defined as a production priority of a product. For example, consider we have two types of production on the same machine with the same mould int the same day. Their planned order quantities also are same. The first product will be yellow and the second one will be red. Then, if you need red products first, you can give it as a fact. A fact has the following parameters in the system :
    - Production date
    - Machine Code
    - Mould Code
    - Color Code
    - Previous production code (source_node)
    - Product to be produced (child_node)

  Hence, we can specify a production order with these parameters. In figure(24), if we want to force the production of B after C, it will be enough to express with the parameters written above.

- **Methods:** Within a method, we would exclude or include KSs in the same level of the search. A method has the following elements :
    - Production date
    - Machine code
    - Mould code
    - Color code
    - Previous production code (source_node)
    - Product to be produced (child_node)
    - Option to exclude or include the KS

- **Policies:** The policy parameters must be set to reduce the cost of the productions. To reduce the cost, we have to reduce the number of machines, the working time of the machines, the number of the mould changes and the number of the color changes.

While these parameters are valid, also, we hae to produce as many products as we can do.So let us define the parameters that effect to the cost of productions :

- **Machine setup times :** the most important time taking parameter is machine mould setups. Because, changing a mould in a machine takes from 2 to 5 hours. If we consider this time interval as seconds, changing a mould of a machine can vary from 7200 seconds to 18000 seconds. Stopping a machine means increasing the cost. So we must maximize the number of productions after changing a mould.
- **Color setups :** This is also an important parameter to reduce cost. It does not take so much time as mould setup, however changing colors many times will reduce effectiveness of the production. And it means to waste the raw-material of the product. And be sure that it will have an extra cost for a unit production. Also, we must maximize the number of productions after changing the color.
- **The number of products to be produced :** We have to select the bigger production order first in terms of quantity.

According to these parameters we can define our policy with the following parameters and their coefficients:

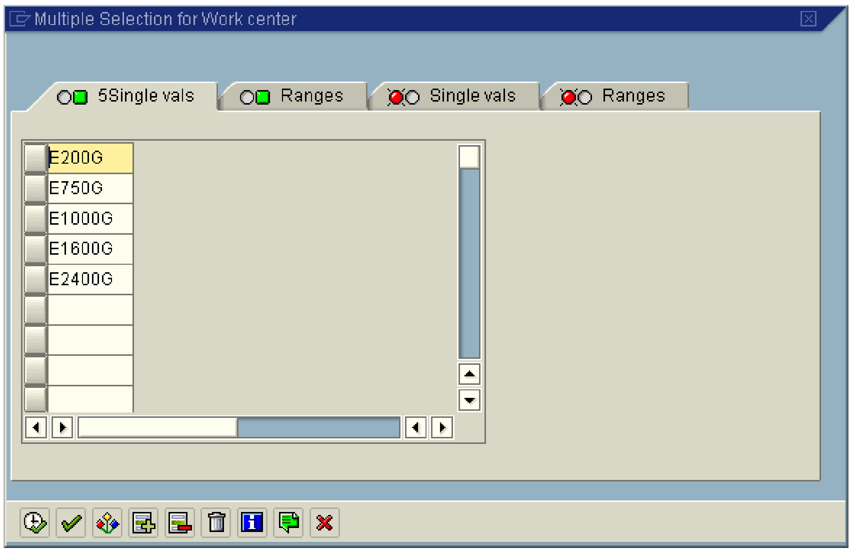
1. **Date** : This parameter is used to select policy by the program. It can take a specific date value or take (*) asterisk character to validate the policy for all entries.
2. **Machine Code** : This parameter is also used to select appropriate policy for bidding mechanism. Also it can take a specific machine name or can take (*) asterisk character.
3. **Mould Code** : This parameter also helps to identify the appropriate policy for bidding mechanism.
4. **Product to be Produced** : It mostly takes (*) to define its validty to all products to be produced in a specific day, machine and mould code. But if you want to define an exact policy for a specific production, it may take the value of production code also.
5. **Setup Time** : This is the first and most important coefficient. Because, pluggin-in of a mould in a machine takes a lot of time and this is the most

effective wasting time parameter. We will give the biggest negative coefficient in the program.

6. **Total Production Time** : This value has the multiplication of unit production time and total number of products for a specific finishing date. The importance of this coefficient is not so much like setup time. Generally, its effect is negative and considered with the number of products.
7. **Number of products** : As we said before, we want to maximize the nuber of products in the minimum of time. So this coefficient will take a positive value..

5.1.4. EXECUTION OF THE SYSTEM

In ABAP program, we can give more than one value for storage location, machine code and finishing date as selection parameters. While you see the selection screen for ZPLANTEST program in Figure 5-1, you can also see the multiple selection screen in Figure 5-8. In this screen, we can give a number range of single values for the selected variable in the program. Also, we can choose include or exclude these values.



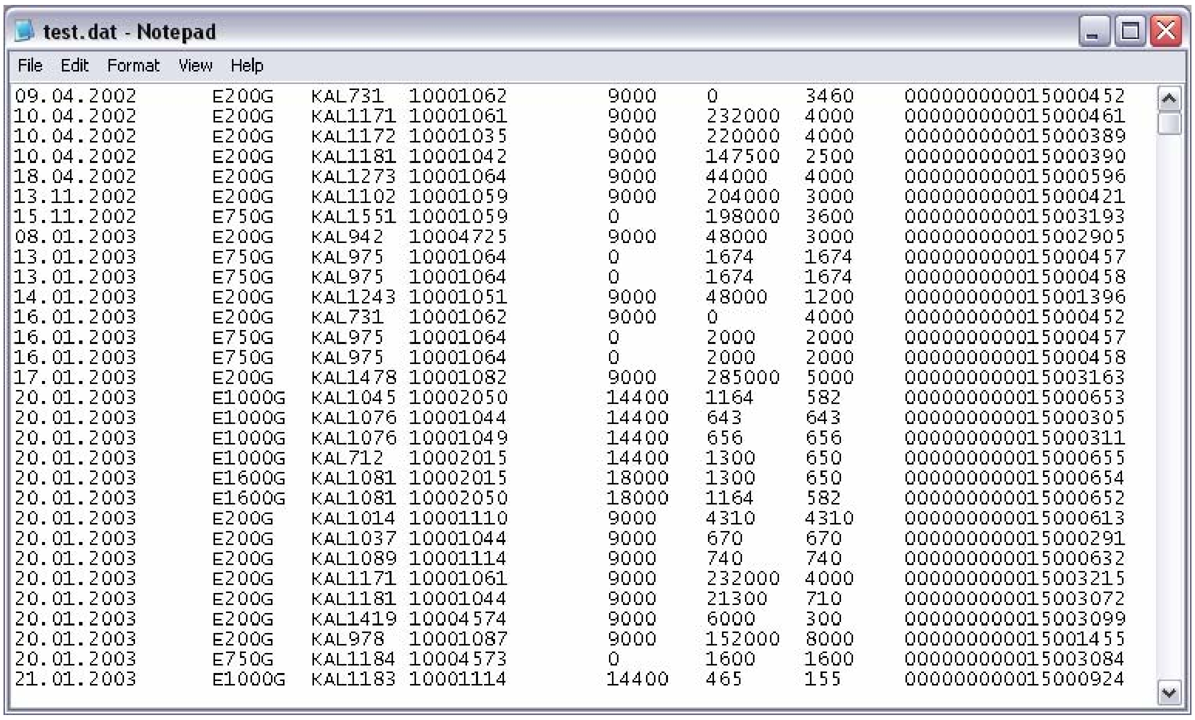
E200G
E750G
E1000G
E1600G
E2400G

Figure 5-8 : Multiple Selection Screen for Machine Codes

For our example, we have chosen two storage locations and five machine codes to get the production list. After execution of the ABAP program, a 711 lined list was generated. And then we have downloaded to the computer where we run the architecture program.

After downloading to the local computer, the tabular formatted list is seen like in Figure 5-9. There are eight columns : the production finishing date, machine code, mould code, color code, setup time of the machine, total production time, production amount and finally product code. As we have seen before, every line of this list will be our KSs.

We will download the policies, facts and methods from SAP. However, I will try to complete the policy to the presentation. This is why, the control mechanism was given manually for right now.



```
09.04.2002      E200G    KAL731  10001062        9000    0       3460    000000000015000452
10.04.2002      E200G    KAL1171 10001061        9000    232000  4000    000000000015000461
10.04.2002      E200G    KAL1172 10001035        9000    220000  4000    000000000015000389
10.04.2002      E200G    KAL1181 10001042        9000    147500  2500    000000000015000390
18.04.2002      E200G    KAL1273 10001064        9000    44000   4000    000000000015000596
13.11.2002      E200G    KAL1102 10001059        9000    204000  3000    000000000015000421
15.11.2002      E750G    KAL1551 10001059        0       198000  3600    000000000015003193
08.01.2003      E200G    KAL942  10004725        9000    48000   3000    000000000015002905
13.01.2003      E750G    KAL975  10001064        0       1674    1674    000000000015000457
13.01.2003      E750G    KAL975  10001064        0       1674    1674    000000000015000458
14.01.2003      E200G    KAL1243 10001051        9000    48000   1200    000000000015001396
16.01.2003      E200G    KAL731  10001062        9000    0       4000    000000000015000452
16.01.2003      E750G    KAL975  10001064        0       2000    2000    000000000015000457
16.01.2003      E750G    KAL975  10001064        0       2000    2000    000000000015000458
17.01.2003      E200G    KAL1478 10001082        9000    285000  5000    000000000015003163
20.01.2003      E1000G   KAL1045 10002050        14400   1164    582     000000000015000653
20.01.2003      E1000G   KAL1076 10001044        14400   643     643     000000000015000305
20.01.2003      E1000G   KAL1076 10001049        14400   656     656     000000000015000311
20.01.2003      E1000G   KAL712  10002015        14400   1300    650     000000000015000655
20.01.2003      E1600G   KAL1081 10002015        18000   1300    650     000000000015000654
20.01.2003      E1600G   KAL1081 10002050        18000   1164    582     000000000015000652
20.01.2003      E200G    KAL1014 10001110        9000    4310    4310    000000000015000613
20.01.2003      E200G    KAL1037 10001044        9000    670     670     000000000015000291
20.01.2003      E200G    KAL1089 10001114        9000    740     740     000000000015000632
20.01.2003      E200G    KAL1171 10001061        9000    232000  4000    000000000015003215
20.01.2003      E200G    KAL1181 10001044        9000    21300   710     000000000015003072
20.01.2003      E200G    KAL1419 10004574        9000    6000    300     000000000015003099
20.01.2003      E200G    KAL978  10001087        9000    152000  8000    000000000015001455
20.01.2003      E750G    KAL1184 10004573        0       1600    1600    000000000015003084
21.01.2003      E1000G   KAL1183 10001114        14400   465     155     000000000015000924
```

Figure 5-9 : Downloaded production orders list

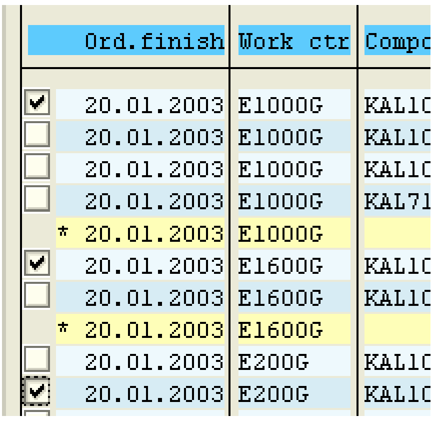
For policy, it is optimum to give a different policy to each machines on the list, because since the energy needs, setup times of the machines vary, we may want to change the policy coefficients for each machine. If we have given different policies for each machine, then the policies would seen like this :

- **Policy(***,E200G,*,*,-1,0.1,3**)**
- **Policy(***,E750G,*,*,-1,0.1,3**)**
- **Policy(***,E1000G,*,*,-1,0.1,3**)**
- **Policy(***,E1600G,*,*,-1,0.1,3**)**
- **Policy(***,E2400G,*,*,-1,0.1,3**)**

These policies mean, for all KSs, the setup time coefficient will be taken as ‘-1’, total production time coefficient will be taken as ‘0.1’ and we will have to multiply production order quantities by ‘3’ for the bidding mechanism. But now we just give a general policy for the whole list. It is seen like this :

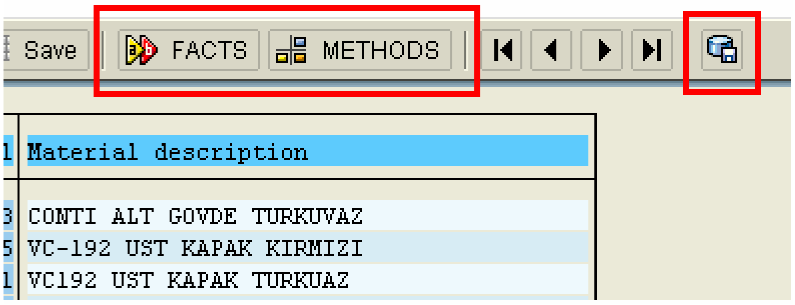
Policy(*,*,*,*,-1,0.1,3)

In figure 5-10, we are seeing the lines of a list with boxes. To specify the facts, we just click these leftmost boxes of the lines in the general list and then we click to FACT button like in the Figure 5-11. By this way, SAP system will download the facts list to the architecture. With the same way, if we want to specify methods, we select the lines and then, download the list to the local computer. On the other side, we have to be aware that there must be only one fact in a production level. If we want to set high priority for more than one production orders, them we must select methods.



	Ord.finish	Work ctr	Compo
✓	20.01.2003	E1000G	KAL1C
☐	20.01.2003	E1000G	KAL1C
☐	20.01.2003	E1000G	KAL1C
☐	20.01.2003	E1000G	KAL71
	* 20.01.2003	E1000G	
✓	20.01.2003	E1600G	KAL1C
☐	20.01.2003	E1600G	KAL1C
	* 20.01.2003	E1600G	
☐	20.01.2003	E200G	KAL1C
✓	20.01.2003	E200G	KAL1C

Figure 5-10 : Selection boxes on the general list

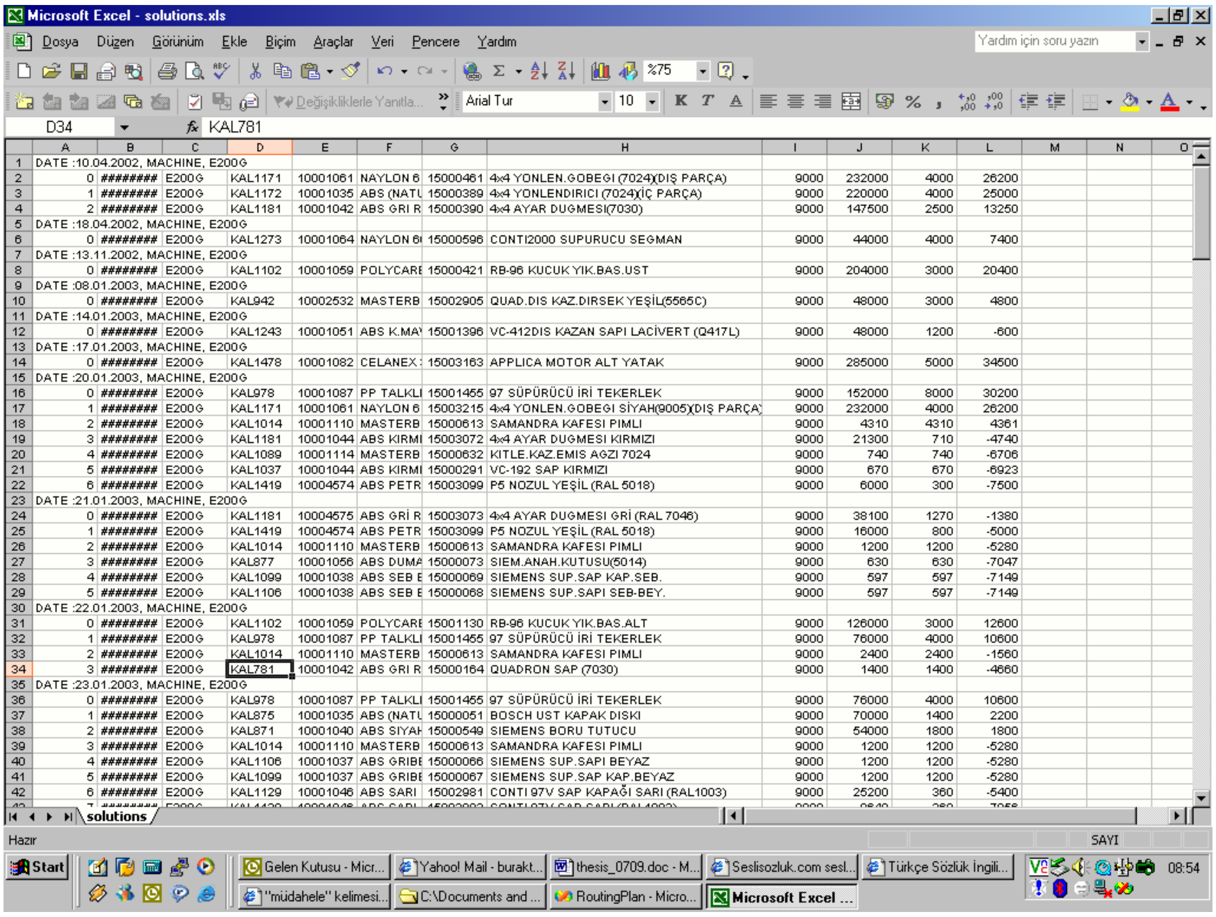


Material description
CONTI ALT GOVDE TURKUVAZ
VC-192 UST KAPAK KIRMIZI
VC192 UST KAPAK TURKUAZ

Figure 5-11 : Download Buttons

After setting up the control parameters, we run the program that includes our control architecture. The program first takes, methods, facts, knowledge sources into memory. Then it starts to loop in the goals. For each pairs of the date and the machine codes, the program generates new sub-goal trees. As a strategy, a breadth-first search would be enough. Because, depth-first search does not have a meaning for this example.

After executing the application we get the output file our local computer. Since it is tabular formatted text file, we can any program that supports this format. Also we can upload to SAP system again if needed. In Figure 5-12 you can see MS Excel screenshot of the output file.



Microsoft Excel - solutions.xls

D34 KAL781

	A	B	C	D	E	F	G	H	I	J	K	L
1	DATE :10.04.2002, MACHINE, E200G											
2	0	########	E200G	KAL1171	10001061	NAYLON 6	15000461	4x4 YONLEN.GOBEGI (7024)(DIŞ PARÇA)	9000	232000	4000	26200
3	1	########	E200G	KAL1172	10001035	ABS (NATU	15000389	4x4 YONLENDIRICI (7024)(İÇ PARÇA)	9000	220000	4000	25000
4	2	########	E200G	KAL1181	10001042	ABS GRI R	15000390	4x4 AYAR DUGMESI(7030)	9000	147500	2500	13250
5	DATE :18.04.2002, MACHINE, E200G											
6	0	########	E200G	KAL1273	10001064	NAYLON 6	15000596	CONTI2000 SUPURUCU SEGMAN	9000	44000	4000	7400
7	DATE :13.11.2002, MACHINE, E200G											
8	0	########	E200G	KAL1102	10001059	POLYCARE	15000421	RB-96 KUCUK YIK.BAS.UST	9000	204000	3000	20400
9	DATE :08.01.2003, MACHINE, E200G											
10	0	########	E200G	KAL942	10002532	MASTERB	15002905	QUAD.DIS KAZ.DIRSEK YEŞİL(5565C)	9000	48000	3000	4800
11	DATE :14.01.2003, MACHINE, E200G											
12	0	########	E200G	KAL1243	10001051	ABS K.MAY	15001396	VC-412DIS KAZAN SAPI LACİVERT (Q417L)	9000	48000	1200	-600
13	DATE :17.01.2003, MACHINE, E200G											
14	0	########	E200G	KAL1478	10001082	CELANEX	15003163	APPLICA MOTOR ALT YATAK	9000	285000	5000	34500
15	DATE :20.01.2003, MACHINE, E200G											
16	0	########	E200G	KAL978	10001087	PP TALKLI	15001455	97 SÜPÜRÜCÜ İRİ TEKERLEK	9000	152000	8000	30200
17	1	########	E200G	KAL1171	10001061	NAYLON 6	15003215	4x4 YONLEN.GOBEGI SİYAH(9005)(DIŞ PARÇA)	9000	232000	4000	26200
18	2	########	E200G	KAL1014	10001110	MASTERB	15000613	SAMANDRA KAFESI PIMLI	9000	4310	4310	4361
19	3	########	E200G	KAL1181	10001044	ABS KIRMI	15003072	4x4 AYAR DUGMESI KIRMIZI	9000	21300	710	-4740
20	4	########	E200G	KAL1089	10001114	MASTERB	15000632	KITLE.KAZ.EMIS AGZI 7024	9000	740	740	-6706
21	5	########	E200G	KAL1037	10001044	ABS KIRMI	15000291	VC-192 SAP KIRMIZI	9000	670	670	-6923
22	6	########	E200G	KAL1419	10004574	ABS PETR	15003099	P5 NOZUL YEŞİL (RAL 5018)	9000	6000	300	-7500
23	DATE :21.01.2003, MACHINE, E200G											
24	0	########	E200G	KAL1181	10004575	ABS GRİ R	15003073	4x4 AYAR DUGMESI GRİ (RAL 7046)	9000	38100	1270	-1380
25	1	########	E200G	KAL1419	10004574	ABS PETR	15003099	P5 NOZUL YEŞİL (RAL 5018)	9000	16000	800	-5000
26	2	########	E200G	KAL1014	10001110	MASTERB	15000613	SAMANDRA KAFESI PIMLI	9000	1200	1200	-5280
27	3	########	E200G	KAL877	10001056	ABS DUMA	15000073	SIEM.ANAH.KUTUSU(5014)	9000	630	630	-7047
28	4	########	E200G	KAL1099	10001038	ABS SEB E	15000069	SIEMENS SUP.SAP KAP.SEB.	9000	597	597	-7149
29	5	########	E200G	KAL1106	10001038	ABS SEB E	15000068	SIEMENS SUP.SAPI SEB-BEY.	9000	597	597	-7149
30	DATE :22.01.2003, MACHINE, E200G											
31	0	########	E200G	KAL1102	10001059	POLYCARE	15001130	RB-96 KUCUK YIK.BAS.ALT	9000	126000	3000	12600
32	1	########	E200G	KAL978	10001087	PP TALKLI	15001455	97 SÜPÜRÜCÜ İRİ TEKERLEK	9000	76000	4000	10600
33	2	########	E200G	KAL1014	10001110	MASTERB	15000613	SAMANDRA KAFESI PIMLI	9000	2400	2400	-1560
34	3	########	E200G	KAL781	10001042	ABS GRI R	15000164	QUADRON SAP (7030)	9000	1400	1400	-4660
35	DATE :23.01.2003, MACHINE, E200G											
36	0	########	E200G	KAL978	10001087	PP TALKLI	15001455	97 SÜPÜRÜCÜ İRİ TEKERLEK	9000	76000	4000	10600
37	1	########	E200G	KAL875	10001035	ABS (NATU	15000051	BOSCH UST KAPAK DISKI	9000	70000	1400	2200
38	2	########	E200G	KAL871	10001040	ABS SIYAH	15000549	SIEMENS BORU TUTUCU	9000	54000	1800	1800
39	3	########	E200G	KAL1014	10001110	MASTERB	15000613	SAMANDRA KAFESI PIMLI	9000	1200	1200	-5280
40	4	########	E200G	KAL1106	10001037	ABS GRIBE	15000066	SIEMENS SUP.SAPI BEYAZ	9000	1200	1200	-5280
41	5	########	E200G	KAL1099	10001037	ABS GRIBE	15000067	SIEMENS SUP.SAP KAP.BEYAZ	9000	1200	1200	-5280
42	6	########	E200G	KAL1129	10001046	ABS SARI	15002981	CONTI 97V SAP KAPAĞI SARI (RAL1003)	9000	25200	360	-5400

Figure 5-12 : Excel screenshot of the output file after executing application

So these are the advantages of this application after using blackboard control architecture:

- By using collaboration specialties of a blackboard system, you can easily interact with this application as a module of a bigger system.

- You can arrange the output type what type ever you want to get. For example, you can automatically feed the interfaces of PLC machines by the output of this application.
- This is a platform independent application, so you can compile this algorithm in different OS'es.
- Two different planning examples about this architecture prove that the range of the application area is wide. This is to say we can manage all scheduling problems in the industry by using this architecture.
- Two phased high level knowledge provides an exact control over artificial intelligence.

The only disadvantage is, there is always a risk about facts or methods that they may not have the right values as you want. So, they are to be implemented wisely.

5.2. OTHER APPLICATIONS IN THE MARKET

These kind of scheduling applications are commonly used in the world. However in Turkey, this is a new solution area and this is a good chance to get a better place on this topic. There are two examples that would be explained in this thesis. The first one is Trigger and the second one is Preactor. Trigger is a Turkish application and not released yet. Preactor is the most popular scheduling program in the world. They are very different from each other, and our application carries all advantages of these applications.

5.2.1. TRIGGER

Trigger is a scheduling program for plastic injection machines. It collects all the information about operation processes, capacities of machines, production orders and other data of the company. Then, it generates a schedule of production orders. This is new software in the market, and it is specially being developed for BEKO. However, it will be delivered to the market soon.

Trigger uses Simulated Annealing algorithm in the decision making process. Its purpose is to find optimum solution according to characteristics of a plastic. Simulated annealing is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space [16]. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

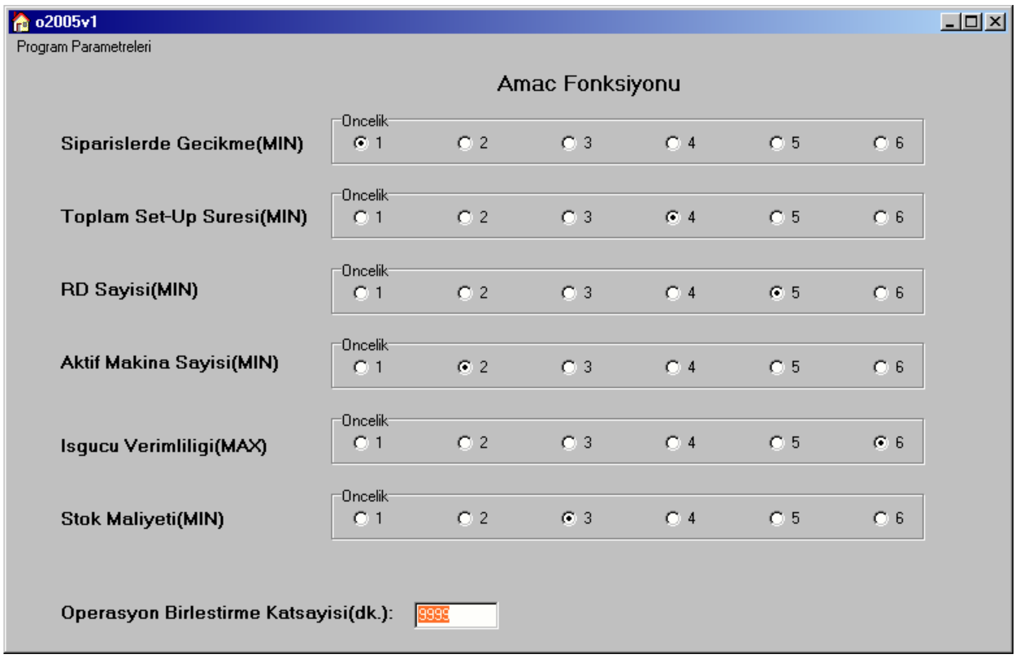
Hence, more plastic products would be developed with less energy. The pseudo code of the simulated annealing algorithm can be seen in Figure 5-13 :

```
s := s0; e := E(s)                    // Initial state, energy.
sb := s; eb := e                      // Initial "best" solution
k := 0                                // Energy evaluation count.
while k < kmax and e > emax           // While time remains & not good enough:
  sn := neighbour(s)                  //Pick some neighbor.
  en := E(sn)                         //Compute its energy.
  if en < eb then                     //Is this a new best?
    sb := sn; eb := en                //Yes, save it.
  if random() < P(e, en, temp(k/kmax)) then  //Should we move to it?
    s := sn; e := en                  //Yes, change state.
  k := k + 1                          //One more evaluation done
return sb                             //Return the best solution found.
```

Figure 5-13 : Pseudo Code of simulated annealing algorithm

As an advantage, we can say that Trigger would work very good for plastic and metal products. However, this will not work for other scheduling needs of the industry. And this is a big disadvantage in the software market. Because, this disadvantage restricts the application range of the program. Another disadvantage is, incompatibility with any operating system other than Microsoft Windows.

In Figure 5-14, Figure 5-15 and Figure 5-16, we are seeing some simple screenshots from the first beta of Trigger.

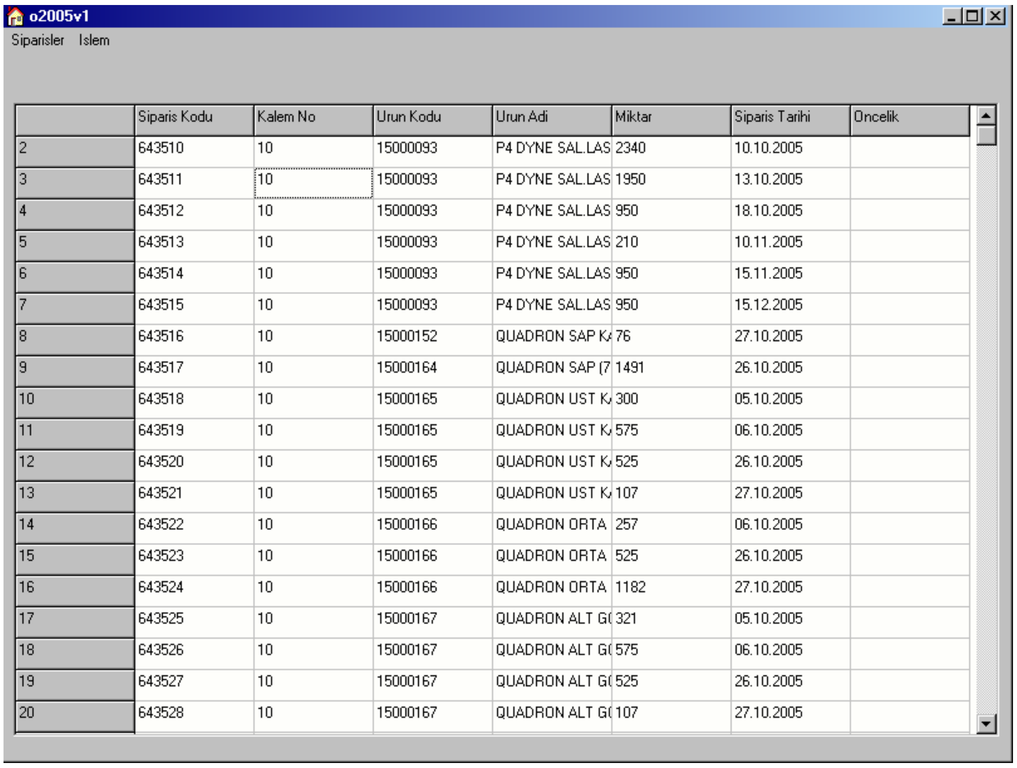


Amac Fonksiyonu

	Oncelik
Siparislerde Gecikme(MIN)	[✓ 1] 2 3 4 5 6
Toplam Set-Up Suresi(MIN)	1 2 3 [✓ 4] 5 6
RD Sayisi(MIN)	1 2 3 4 [✓ 5] 6
Aktif Makina Sayisi(MIN)	1 [✓ 2] 3 4 5 6
Isgucu Verimliligi(MAX)	1 2 3 4 5 [✓ 6]
Stok Maliyeti(MIN)	1 2 [✓ 3] 4 5 6

Operasyon Birlestirme Katsayisi(dk.): 9999

Figure 5-14 : Trigger – min/max function parameters menu



	Siparis Kodu	Kalem No	Urun Kodu	Urun Adi	Miktar	Siparis Tarihi	Oncelik
2	643510	10	15000093	P4 DYNE SAL.LAS	2340	10.10.2005	
3	643511	10	15000093	P4 DYNE SAL.LAS	1950	13.10.2005	
4	643512	10	15000093	P4 DYNE SAL.LAS	950	18.10.2005	
5	643513	10	15000093	P4 DYNE SAL.LAS	210	10.11.2005	
6	643514	10	15000093	P4 DYNE SAL.LAS	950	15.11.2005	
7	643515	10	15000093	P4 DYNE SAL.LAS	950	15.12.2005	
8	643516	10	15000152	QUADRON SAP K/	76	27.10.2005	
9	643517	10	15000164	QUADRON SAP (7	1491	26.10.2005	
10	643518	10	15000165	QUADRON UST K/	300	05.10.2005	
11	643519	10	15000165	QUADRON UST K/	575	06.10.2005	
12	643520	10	15000165	QUADRON UST K/	525	26.10.2005	
13	643521	10	15000165	QUADRON UST K/	107	27.10.2005	
14	643522	10	15000166	QUADRON ORTA	257	06.10.2005	
15	643523	10	15000166	QUADRON ORTA	525	26.10.2005	
16	643524	10	15000166	QUADRON ORTA	1182	27.10.2005	
17	643525	10	15000167	QUADRON ALT G(	321	05.10.2005	
18	643526	10	15000167	QUADRON ALT G(	575	06.10.2005	
19	643527	10	15000167	QUADRON ALT G(	525	26.10.2005	
20	643528	10	15000167	QUADRON ALT G(	107	27.10.2005	

Figure 5-15 : Trigger – Production Orders Menu

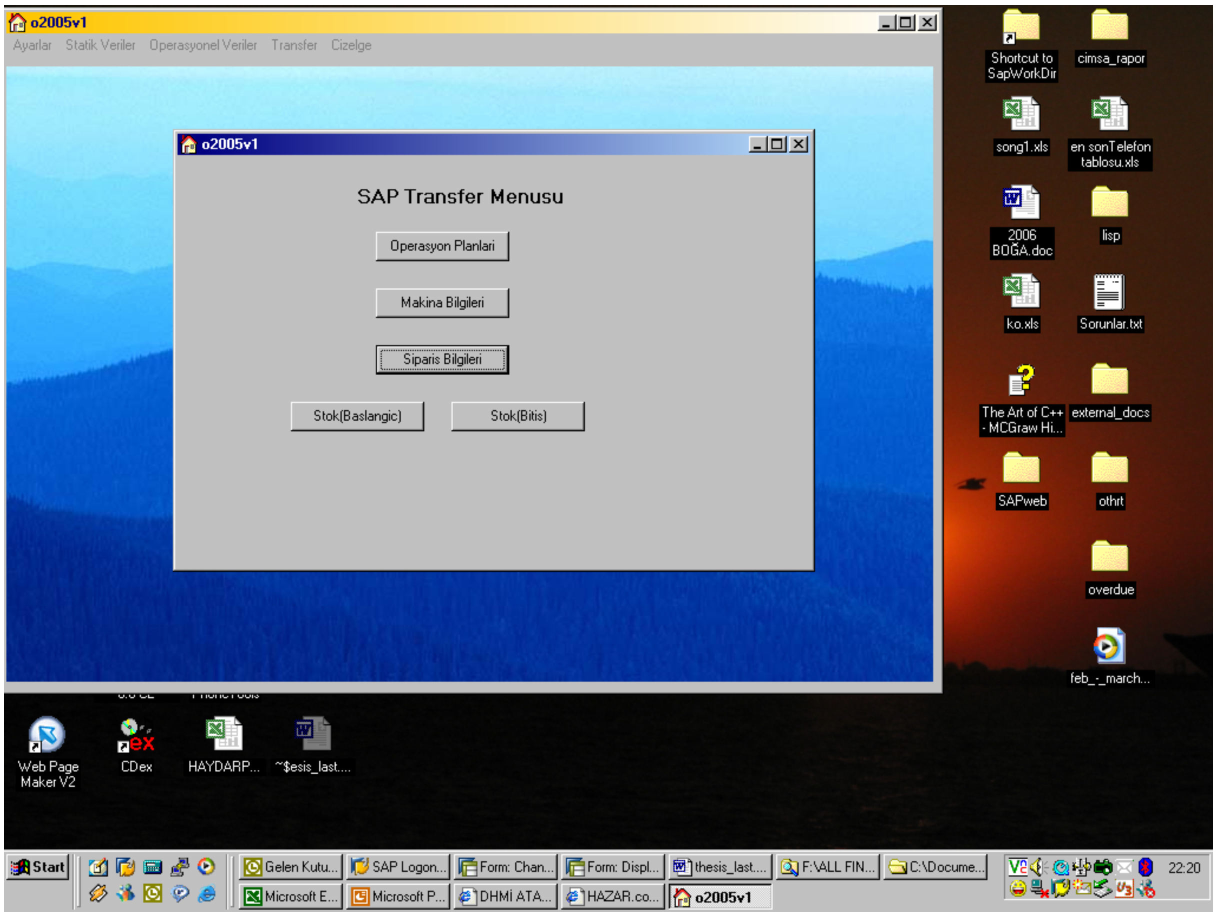


Figure 5-16 : Trigger - SAP Transfer Menu

5.2.2. PREACTOR

Preactor is the most popular scheduling solution in the world. Preactor is a software package that provides a planner with an interactive decision support tool that balances demand and capacity [17]. With Preactor, you can make production planning, production scheduling and supply chain management. Preactor is also running PC based applications and it is compatible with only Microsoft Operating systems. Before using Preactor, you have to configure whole production system of the company into it. This configuration setup time can take up to three months of a year.

Preactor have three main products: Preactor 200, Preactor 300 and Preactor APS [18]. Preactor 200 and Preactor 300 are ‘Finite Capacity Scheduling (FCS)’ software and Preactor APS is ‘Advanced Planning and Scheduling’ software. While in FCS, you just can schedule your production orders; in APS you can schedule your operations also. In Preactor, you use

Gannt chart to generate schedules. That means, user generates his own schedule via Preactor manually. Not like Trigger or our application.

There is more than 5000 companies use Preactor to plan their production. Some of these companies are, Cosworth Racing, Delphi, Imperial Tobacco, Pfizer, Philips and Vienna Airport.

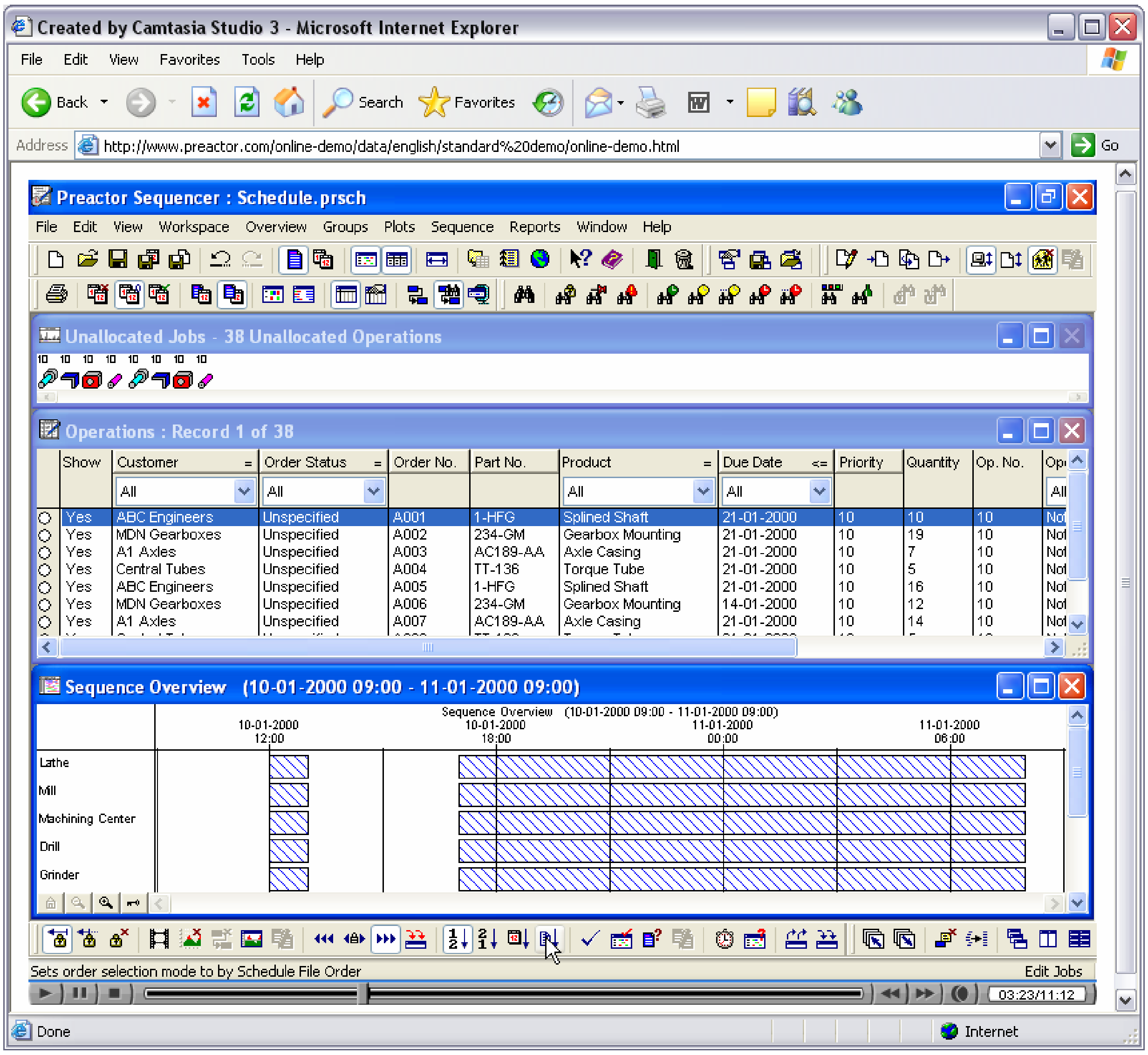
So what is the advantage and disadvantage of this software package? First of all, let us mention about advantages :

- Because the software is used widely, its technical support is very good.
- User interactive menus are very good; you can easily drag and drop the elements of a production order.
- Supports very wide range of industrial applications.

Then let us see disadvantages of Preactor:

- There is no automated decision tool in the package. The user must decide which production would be produced first. By this way, user can make mistakes during decision.
- Manual decision increases working hour of a worker.
- You have to enter all the raw data from your MRP or ERP system and configure them wisely. And it takes a lot of time before using it.
- Just Microsoft Windows compatible. You cannot use under Linux or Unix based systems.

You can see two screenshots from Figure 5-17 and Figure 5-18.


Created by Camtasia Studio 3 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.preactor.com/online-demo/data/english/standard%20demo/online-demo.html Go
Preactor Sequencer : Schedule.prsch
File Edit View Workspace Overview Groups Plots Sequence Reports Window Help
Unallocated Jobs - 38 Unallocated Operations
Operations : Record 1 of 38
Sequence Overview (10-01-2000 09:00 - 11-01-2000 09:00)
10-01-2000 12:00
10-01-2000 18:00
11-01-2000 00:00
11-01-2000 06:00
Lathe
Mill
Machining Center
Drill
Grinder
Sets order selection mode to by Schedule File Order
Edit Jobs
03:23/11:12
Done
Internet

Show	Customer	Order Status	Order No.	Part No.	Product	Due Date	Priority	Quantity	Op. No.	Op
	All	All			All	All				All
Yes	ABC Engineers	Unspecified	A001	1-HFG	Splined Shaft	21-01-2000	10	10	10	Not
Yes	MDN Gearboxes	Unspecified	A002	234-GM	Gearbox Mounting	21-01-2000	10	19	10	Not
Yes	A1 Axles	Unspecified	A003	AC189-AA	Axle Casing	21-01-2000	10	7	10	Not
Yes	Central Tubes	Unspecified	A004	TT-136	Torque Tube	21-01-2000	10	5	10	Not
Yes	ABC Engineers	Unspecified	A005	1-HFG	Splined Shaft	21-01-2000	10	16	10	Not
Yes	MDN Gearboxes	Unspecified	A006	234-GM	Gearbox Mounting	14-01-2000	10	12	10	Not
Yes	A1 Axles	Unspecified	A007	AC189-AA	Axle Casing	21-01-2000	10	14	10	Not

Figure 5-17 : Preactor – Main window screenshot

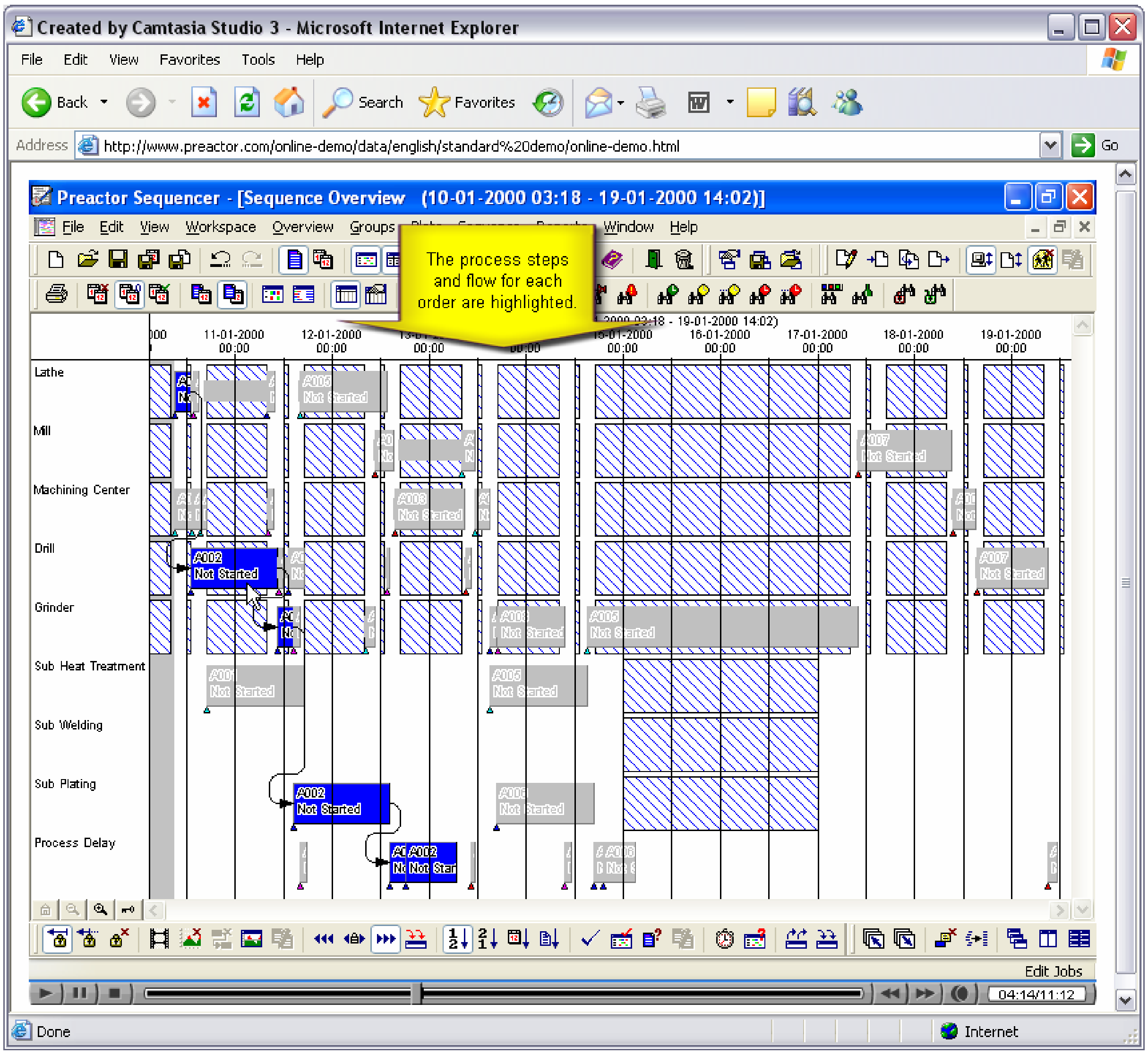

Created by Camtasia Studio 3 - Microsoft Internet Explorer
http://www.preactor.com/online-demo/data/english/standard%20demo/online-demo.html
Preactor Sequencer - [Sequence Overview (10-01-2000 03:18 - 19-01-2000 14:02)]
The process steps and flow for each order are highlighted.
Lathe
Mill
Machining Center
Drill
Grinder
Sub Heat Treatment
Sub Welding
Sub Plating
Process Delay
A002 Not Started
Edit Jobs
Done
Internet

Figure 5-18 : Preactor – A Sequence overview window

6. CONCLUSION AND EVALUATION

The control architecture introduced for goal-driven blackboard systems is based on searching a general goal tree. The basic elements of the architecture are goals, policies, strategies, KSs, methods, and facts. It employs a basic control loop that uses a bidding mechanism in choosing the knowledge source to be executed at the current cycle. The bidding mechanism is guided by a policy. The policy can be called local scheduling criteria for this control architecture. A strategy on the other hand, is a global scheduling criterion such as depth-first etc. Strategies and policies together determine how a partial solution is to be extended in the control loop. Then the search space can be diminished by applying methods and facts. The methods and facts are high level knowledge on how to solve a problem. And they have to be well known before applying. Because they force to change the direction of the solution.

We have used this control architecture in an industrial production planning application and then, we compared our architecture with two applications. As a conclusion, we saw that the control architecture can achieve other applications' issues. Also our simple example flight scheduling application proves that this control architecture can be used in most of all kind of planning and scheduling applications.

Furthermore, high level knowledge over artificial intelligence provides us an exact and flexible intervention over scheduling and planning. None of the present applications of the market provide this special feature. And this is a big advantage of our application.

As a disadvantage, we can say that there is always a risk about facts or methods that they may not have right values. This is why, when expressing these high level knowledge sources we have to be careful.

In the future, some other search strategies may be implemented. Some standardized sort algorithms can be applied to handle search in more effective way. Also, at each run of the program, a system can record a list of solutions and can make some statistical work to use in the architecture itself as methods and facts.

REFERENCES

1. Ferda Bek, A Goal-Driven Control Architecture for Blackboard Systems, Boğaziçi University, 1986
2. Daniel D. Corkill, Blackboard and Multi-Agent Systems & the future, Dept. of Computer Science, University of Massachusetts, 2003
3. Daniel D. Corkill, Blackboard Systems, AI Expert, 6(9): 40-47, September, 1991
4. L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.
5. Alan H. Bond and Les Gasser, Readings in Distributed Artificial Intelligence, Morgan Kaufmann, 1988.
6. V. R. Lesser, R. C. Whitehair, D. D. Corkill, and J. A. Hernandez. Goal relationships and their use in a blackboard architecture. Academic Press, 1989.
7. V. R. Lesser and L. D. Erman., Distributed interpretation: A model and experiment. IEEE Transactions on Computers, Dec. 1980.
8. V. Jagannathan, R. Dodhiawala, and L. S. Baum. Blackboard Architectures and Applications. Academic Press, 1989.
9. Herbert Schildt, The Art of C++, McGraw Hill/Osborne © 2004
10. http://www.sap.com/turkey/index.epx
11. http://help.sap.com/saphelp_46c/helpdata/en/ba/df293581dc1f79e10000009b38f889/frameset.htm, Production and Planning Control in SAP
12. MM Materials Management in SAP, Release 46C, SapPress, 2000
13. BC ABAP User's Guide, Release 40B, SapPress, 1999
14. BC ABAP Dictionnary, Release 40B, SapPress, 1999
15. Serdar Şimşekler, An Easy Reference for ALV Grid Control, SapPress, 2004
16. http://en.wikipedia.org/wiki/Simulated_annealing
17. http://www.preactor.com/default.asp
18. http://www.uytes.com.tr/cizelgeleme/preactor.html

APPENDIX A

All the program codes and the soft document of this thesis can be found in the attached CD.