1996

A Scheduling Expert Advisor for the Bond Environment

Mihai G. Sirbu

Dan C. Marinescu

Report Number:
96-045

A SCHEDULING EXPERT ADVISOR FOR THE BOND ENVIRONMENT

Mihai G. Sirbu
Dan C. Marinescu

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR 96-045
August 1996

A Scheduling Expert Advisor for the Bond Environment

Mihai G.Sirbu and Dan C. Marinescu
Computer Sciences Department
Purdue University, West Lafayette, In 47906, USA
Email: (sirbu, dcm)@cs.purdue.edu,

Abstract

In this paper we discuss intelligent agent support for parallel and distributed computing in a heterogenous environment. We provide an overview of the Bond environment and of services provided by a network on intelligent agents, then we discuss in depth the Scheduling Expert Advisor, SEA. The main function of the SEA is to process a high level description of a computational task provided by a user and ensure that all the objects needed for the task are available at the site selected for execution.

Intelligent Agent Support for Hetrogeneous Parallel and Distributed Computing

While intelligent agents [GAA95, FG95, EW94] are used extensively for information retrieval and data mining, there are virually no reports of their application in the area of parallel and distributed computing. In this paper we discuss an environment for parallel and distributed computing and present one of it major components, the Scheduling Expert Advisor. A feature distinguishing the Bond environment from other efforts in this area is the extensive use of knowledge processing. It seems natural that in a distributed environment based upon the client-server paradigm, at least some of the services be provided by intelligent agents.

The task of accommodating heterogeneity poses challenges difficult to carry out by less sophisticated means then knowledge processing. Take for example data and program migration, one of the activities needed in such an environment. Data migration can be accomplished by a script including commands to `tar/untar`, `compress/uncompress`, `encript/decript`, `ftp`, `login`, etc. But each of the steps mentioned above may fail and a script able to handle such errors is likely to be very complex. When one adds the requirement to move data

among systems with different operating systems e.g. Unix and NT, this solution becomes impractical. For example, the task of finding if enough space is available on the target system, one of the low level actions performed during data and program migration is considerably easier to implement as a set of facts and rules than as a script.

The intelligent agents in the Bond environment are specialized expert systems acting as servers able to perform tasks like program migration, data migration, scheduling, mapping, exporting objects, and so on. At the heart of the Bond systems are resource databases, which provide information about all the objects available to individual members of a group. Programs, data, and hardware objects are shared or used exclusively by the members of the group. The information about the services available in the system is provided by a server, the ``oracle'' running at a known port. All services including those provided by intelligent agents register themselves with the oracle.

The main function of the Scheduling Expert Advisor is to interpret a high level description of a computational task and to make all objects needed for computation available at the target system. The Scheduling Expert Advisor uses service provided by other intelligent agents to accomplish its task. It collaborates with the Mapping Expert Advisor to select a target system, with the Data Migration and possibly with the Program Migration Expert Advisors to make sure that program and data objects needed for the computation are available at the target system.

To exploit the benefits of knowledge processing we had to provide effective mechanism for the intelligent agents to collaborate with one another, and to adapt their behavior according to the feedback provided by the environment as a result of their action. The major contributions of this paper are such mechanisms. In the Bond environment, the facts and the rules used by an inference engine are modified dynamically as a result of user interactions, actions of other intelligent agents as well as feedback from the environment.

The Scheduling Expert Advisor, SEA, is developed in Clips [Cli93a-Cli93c]. Additional functions for socket communication are written in C. SEA interacts with clients through TCP sockets using text commands. We study a version using a KQML [DAR93, MLF96] interface. Clients and test programs are written in Tk [Ous94] and Expect [Lib95].

Rule-based Expert Systems

This section gives artificial intelligence background and provides some insight into the oper-

ation of expert systems. An expert system starts with information about an abstract universe model and then infers additional knowledge [GR94]. The new knowledge can be stored in the form of both facts and rules. The following discussion follows loosely the CLIPS language [Cli93a-Cli93c, Gia93], but the presented concepts are valid for any rule-based system. In an expert system, information is stored as *facts* (individual items) and *rules* (algorithmic knowledge). A fact stores knowledge about the problem universe, and is represented as an n-tuple ($n \geq 1$), in which the first element is a fact identifier and the other optional elements are fact arguments. A rule represents procedural information, and is a construct of the type: "`IF (antecedent) THEN (consequent)`". Alternative component names are *Left Hand Side,* LHS, for the antecedent, and *Right Hand Side*, RHS, for the consequent. If all the terms of the antecedent are true, the rule is activated. The system triggers one of the activated rules, and evaluates the expressions of the RHS in sequential order.

Figure 1 illustrates the block architecture of a rule-based expert system. The facts are stored

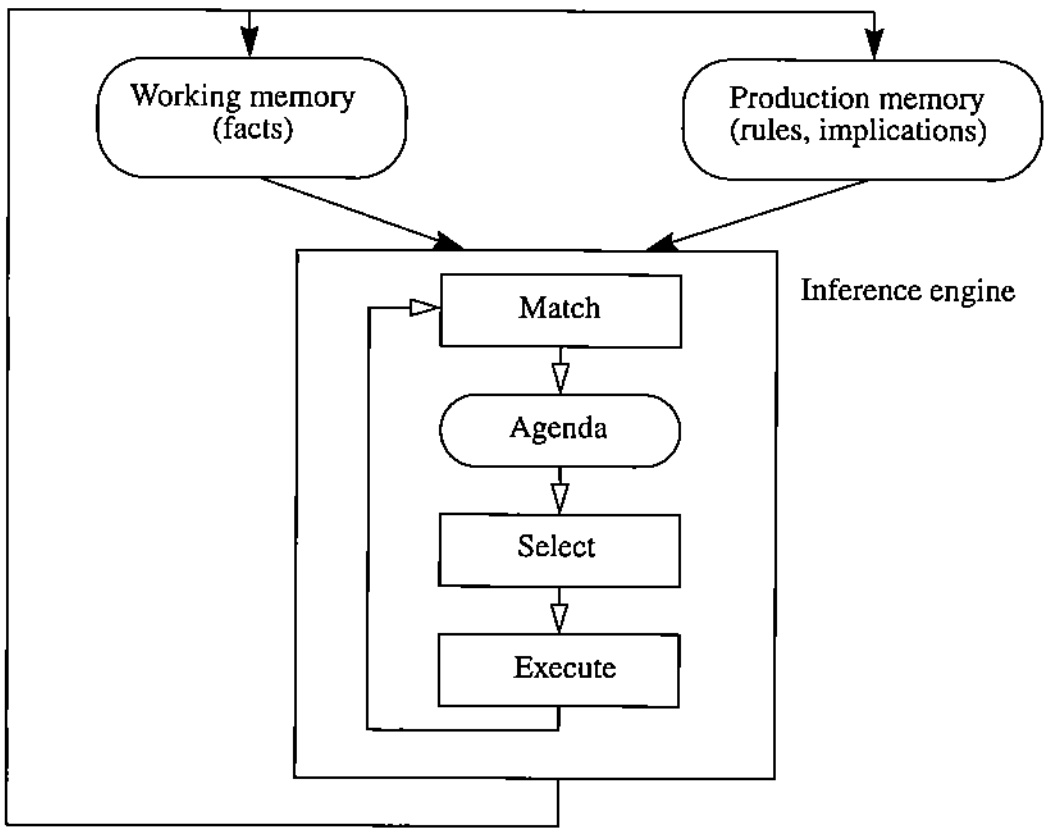


Figure 1. Architecture of a Rule-based Expert System

in the *working memory* and the rules are stored in the *production memory.* The *Inference*

Engine (IE) runs a three-step infinite cycle of matching, selecting and execution. The first step matches the available facts against all rules. This is done by special algorithms to improve efficiency and avoid combinatorial complexity. The activated rules are placed on a list called the system's *agenda*. Next, the IE sorts the agenda and selects the top rule for execution. The sorting criteria is central to the operation of the expert system. In the third step, the RHS actions of the triggered rule are executed. As side effects, changes in the working and production memory can occur. Usually only facts are changed, although the mechanism for dynamic rule changes is present. The original rule is then removed from the agenda to prevent repeated activation by the same facts. The IE cycle continues with a new matching step, and stops if no rule is activated.

The Bond Environment

The Bond environment [SM96] currently under development at Purdue University is designed to support concurrent execution of parallel and/or sequential programs on computing platforms with different architecture and system software, interconnected by a high speed network. We consider a model of parallel and distributed computing which allows an individual working in a group to provide a high level description of the problem to be solved and let an intelligent environment determine a sequence of actions optimal in some sense leading to the desired result. To accomplish this goal the environment has several inference engines and maintains a set of resource databases containing the description of the computing platforms and networks, information about the programs, the services, and the data available to the group, and to each individual within the group.

Bond is a groupware system which supports batch as well as interactive execution. It is designed to run on top of different operating systems, makes no assumptions concerning the communication libraries used by the parallel programs, and supports the management of hardware and software objects. It consists of a kernel, resource databases, remote services including Expert Advisors, and a user interface. The user interface provides access to a set of computing engines interconnected by a high speed network. The environment allows a user to provide a high level description of the problem to be solved, including execution and data dependencies. The Scheduling Expert Advisor converts this description into a set of complex tasks and returns a task schedule to the kernel. The Bond kernel uses other agents e.g. Program and Data Replication Advisors, the Mapping Expert Advisor, etc. to execute simple tasks. Each simple task implies running a pro-

gram with a particular data set on a target system under the supervision of a Bond process. This supervisory process informs the environment about the outcome of the execution and allows the Scheduling Advisor to proceed with the scheduling of the next task or to attempt an error recovery procedure.

When activated, Bond creates a user environment, reflecting information from shared and private resource databases. The services and the Expert Advisors invoked in behalf of a user share the same view of the environment. The set of services and Expert Advisors are distributed and they can be accessed via an *oracle*. The system is open-ended, as new services are added they are registered with the oracle. Some of the services are replicated and the oracle directs a request for service to the server capable of providing the service in an optimal way.

Other Expert Advisors use facts stored in shared knowledge bases to determine if similar task have been carried out previously, and based upon the size of the current problem suggest alternative ways to carry out the computations, provide estimates of the execution time on different configurations. The Data Replication Advisor determines if the data needed for the computation is available at the execution site and performs a variety of operations related to data staging. For example it determines if enough storage space is available at the execution site, then establishes if data conversion is necessary, if so decides where it should take place, compresses and eventually encrypts the data and finally makes a copy of the data at the execution site. The Program Movement Advisor provides similar functionality for program staging. When the remote execution completes, the EA extracts the relevant facts and stores them into shared knowledge bases.

Overview of the Scheduling Expert Advisor

This paper shows the use of an expert advisor for the scheduling of complex program execution sequences. The data and execution dependencies of the component programs are encoded in a set of facts and rules which control the expert system. Rule activation models the scheduling of programs which have all their dependencies satisfied. The process is simple and has significant advantages over the static approach using scripts.

The Scheduling Expert Advisor, SEA, is a layer positioned between the problem description provided by the user and the Bond execution environment, as shown in Figure 2. The SEA processes the High Level Description, HLD, and generates a knowledge-based representation of the

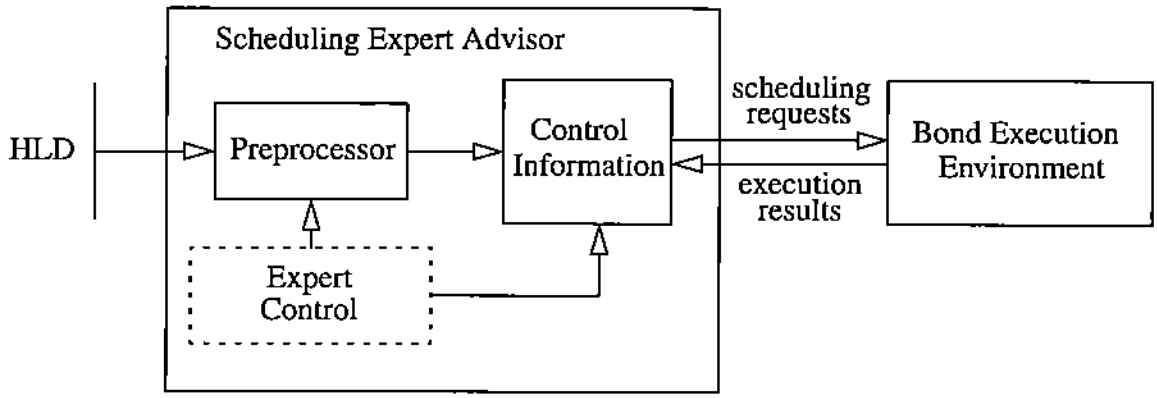


Figure 2. The Scheduling Expert Advisor processes a high level description of the problem and converts it into Scheduling Control Information (facts and rules). Then it sends scheduling requests to the environment.

problem. Next, the SEA submits program scheduling requests to the Bond execution environment. The exit status of the executed programs is returned to the SEA, which updates its internal state. Successful executions validate conditions for the other programs, which are then scheduled. The user can program actions to be taken in case of program failure, for example start an expert advisor specific for error recovery or take direct control of the execution.

The High Level Description, is processed by the SEA and converted into a set of rules and facts called Scheduler Control Information, SCI. The process is similar to compiling a source program into intermediate code. The SCI is in fact an independent expert system which automatically schedules programs in the Bond environment according to the input HLD. The design principle follows the inference engine algorithm described in Section . In a rule-based expert system, the antecedents of each rule are matched with all the available facts. If all the conditions of a rule are satisfied, the rule is activated. One active rule is executed based on the selection mechanism. The scheduling of a program in a complex processing follows the same principle. Some conditions have to be satisfied before the program can be started. The most common conditions are data and execution dependencies. Data dependencies appear when a previous program has to terminate successfully to provide input data for the next step. An example of execution dependency arises when the programs in a group have to be co-scheduled to exchange intermediate results. A group is scheduled only when all the component programs are ready for execution. The basic idea behind the Scheduling Expert Advisor is to associate rules with the scheduling of programs, and

antecedent conditions with execution and data dependencies. When all the dependencies are satisfied, the rule is activated and the program is scheduled.

An important difference between the SEA and an usual expert system is the asynchronous nature of the SEA program scheduling, presented in Figure 3. The scheduling information is sent

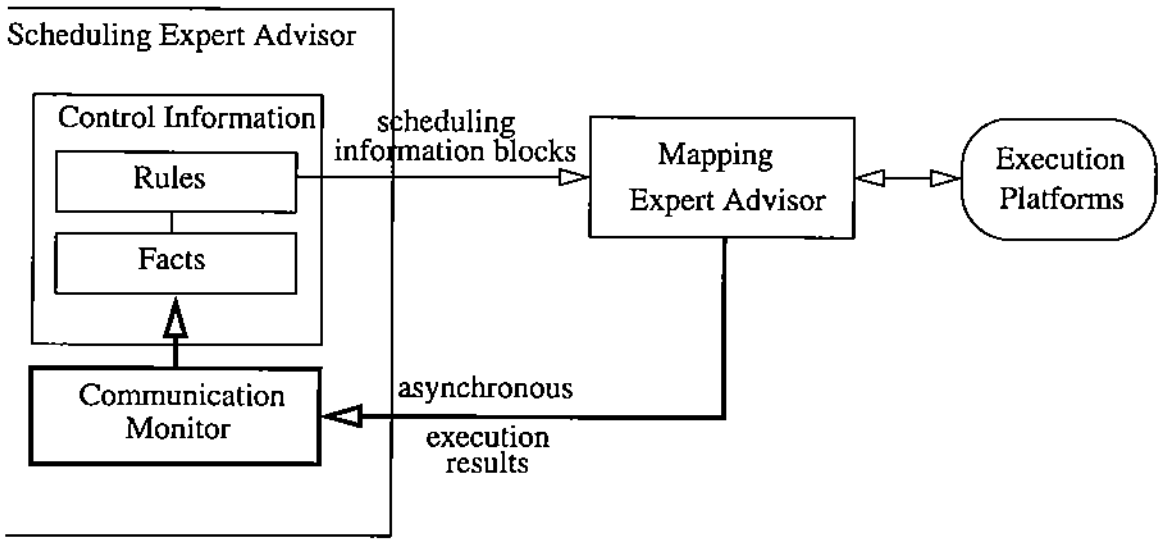


Figure 3. Asynchronous Communication with the SEA. The SEA sends requests to the MEA which in turn generates mappings. The Bond execution environment responds when a mapped activity completes. The MEA informs the SEA that the request was satisfied. The result is entered as a new fact of the SCI.

to the execution environment, but it is unknown how long the execution will take. Some programs might take hours or even days to complete. As such, scheduling of a program is a complete rule by itself. While an expert system normally runs only one active rule at a time, the SEA can schedule all ready programs at the same time, providing an added superconcurency bonus. The downside is that a connection has to be open to receive asynchronous return codes when an individual program terminates. The return codes are converted into facts which are placed in the working memory, activating in turn processing rules.

When a program is scheduled by the SEA, the necessary information is passed to the Mapping Expert Advisor, MEA, which selects an execution target, starts the program in the control environment, and monitors its execution status. The binary execution result (OK or Error) is reported to the SEA. Additional error information can be reported to the SEA for diagnostic and

recovery purposes. A successful execution validates the output files, which are used by other programs as input files. This is also an implied execution sequence dependency. The output files validate the next programs, and the process continues until all available programs are executed.

The SEA is not tightly coupled with the user interface or with the Bond Execution environment. Any interface which can generate a HLD of a problem workflow is acceptable. At the execution level, SEA generates a scheduling information block which is used as input for the Mapping Expert Advisor. Any program able to parse such a block can be used for execution scheduling, if it returns valid diagnostic information about the outcome of the program execution.

Task Execution Specification

A graphical interface allows the user to specify the workflow of a given problem. The sequence of programs, with associated input and output files is described. Parallel execution of program groups are specified. Links are established between output and input files of various programs. The GUI translates this description into an intermediate representation which is parsed by the Scheduling Expert Advisor. We have named the graphical presentation of the workflow High Level Description, or HLD. This description is similar to different module interconnection languages [RS94].

The central HLD concept is the *Execution Block*, or *Block* for short. The basic execution block is an executable program. The internal components of a basic HLD program block are presented in Figure 4. *Input Files* 1 to n are associated with the program. The *Control Input File* guides execution with specific options, so it is processed separately from the other input files. Command line arguments for the program are provided, although in most cases the information is in the control input file. The user can select a *Target System* for the program execution, or this decision can be left to the Mapping EA. A *Supervisor* program monitors the execution of the program, and determines if the execution was successful or not. The supervisor returns the *Execution Report* to the MEA. The program generates a number of *Output Data Files*, and the output results are validate by the *Verify Filters* (VF_1-VF_k). The VFs are started by the supervisor in case of successful execution, and report if the output complies with their requirements or not. The verify filters act similar to assertions in a general programming language.

Composite execution blocks are created by recursive application of composition rules on basic blocks. The list of control composition rules contains: (1) block sequence, (2) loop, (3)

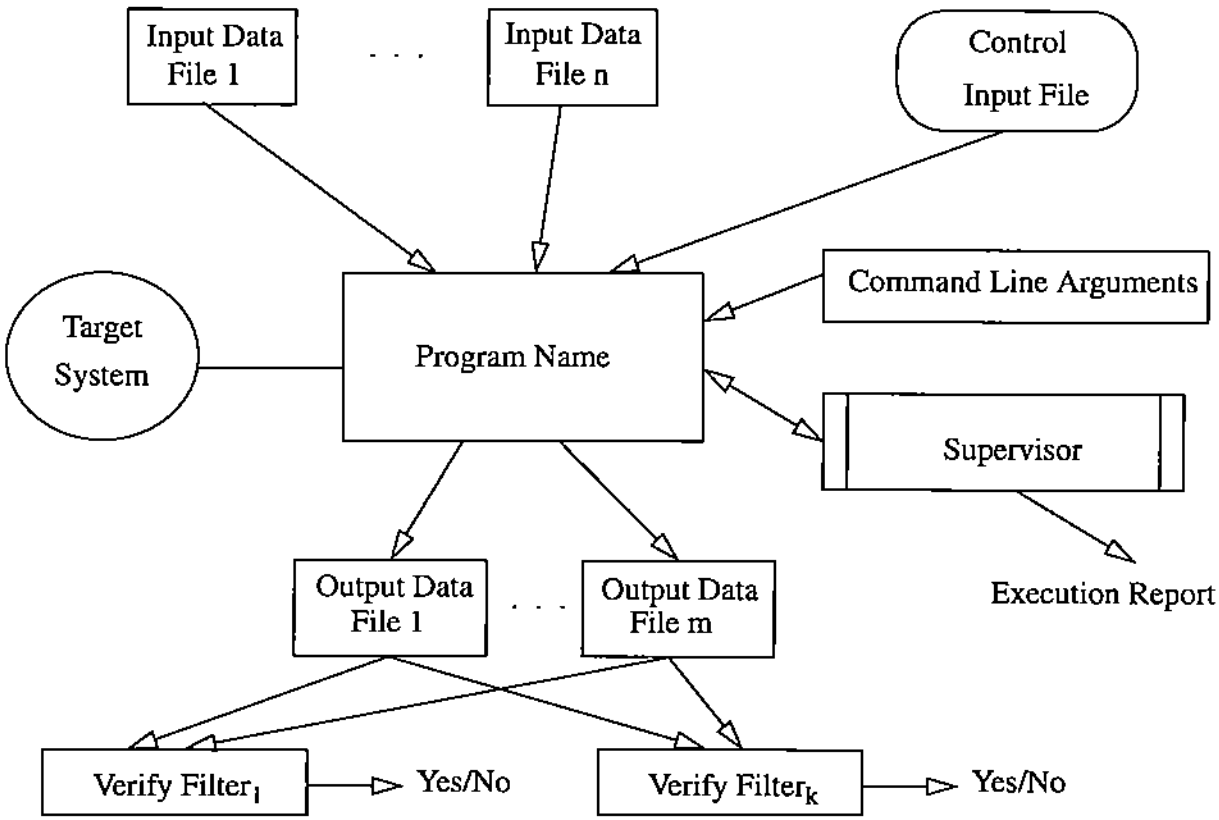
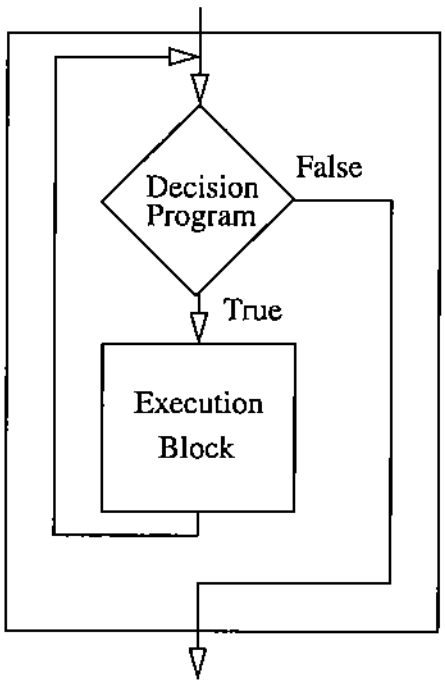


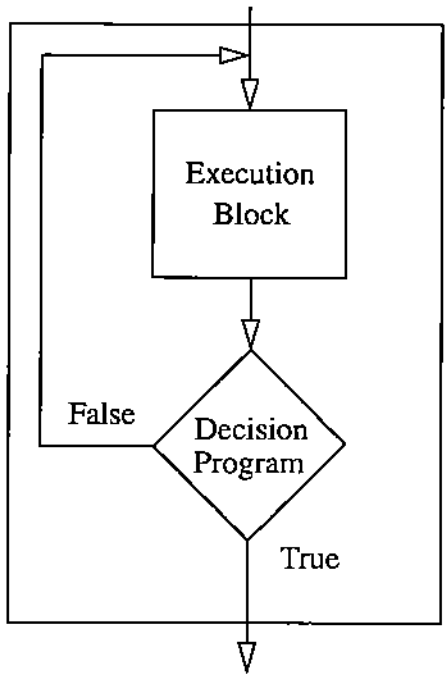
Figure 4. The Internal Components of a Basic Program Block

forced alternative, (4) computed alternative, and (5) group with parallel execution of component blocks. In a *block sequence*, individual blocks are executed only after the previous block in the sequence has successfully terminated. It is similar to sequential execution of the statements in a programming language. The loops are composed of an execution block and a decision program, as shown in Figure 5. The structures are similar to traditional programming language. The role of a logical expression is taken by a *Decision Program*, which controls the execution flow. A *forced alternative* arises when there are a number of equivalent programs that have the same processing effect, and the user manually selects one before execution. The *computed alternative* involves a Decision Program which selects one of the available paths in the description. A *parallel group* contains programs that must be scheduled at the same time due to communication dependencies.

By combining multiple blocks we can use combination techniques to group any number of subblocks into a higher level block. Usually each program has a number of additional elements associated with it, such as names of input and output files.



a) Initial Test Loop: While-Do



b) Final Test Loop: Repeat-Until

Figure 5. HLD Loop Structures

The Scheduling Expert Advisor

The Scheduling Expert Advisor has the following functions:

- Parse a new HLD and convert it into rules and facts (pre-processing).
- Determine the programs available for execution at any time.
- Schedule the program that can be run in the current step. A scheduling request is passed to the Mapping Expert Advisor for each individual program or group of programs.
- Run in asynchronous (each program is scheduled when all the conditions are fulfilled) or synchronous mode (all programs available for execution in one step are co-scheduled; whenever a program terminates, a new scheduling cycle is started).
- Report the programs that have not been executed, and might never be due to a possible HLD programming error.
- Clear the working and production memory of the current HLD, and prepare for a new script.

Generation of the new rules and facts for a HLD and the scheduling of programs are the most important functions of the SEA.

The program scheduling results from the interaction of the new generated rules. The state

transition diagram of a generic program is presented in Figure 6. Each arrow in the transition dia-

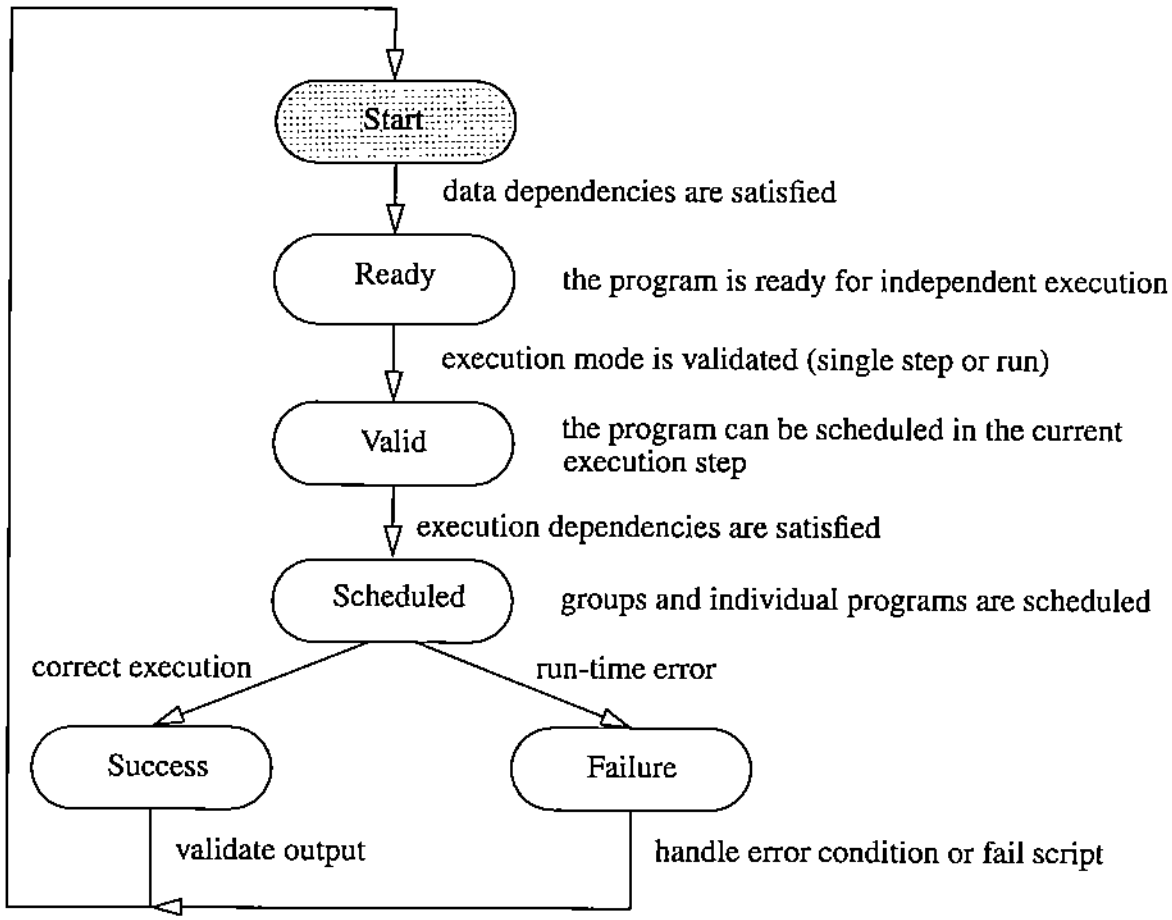


Figure 6. State Transition Diagram of a Program in the Scheduling Expert Advisor

gram represents a single rule or a set of rules in the Scheduler Control Information. The rules are named after the destination state, for example the rules leading to the Valid State are called Valid Rules. Most states are represented by SEA facts, named as in the transition diagram. Initially, the programs are in the *Start State*, when partial dependencies are satisfied. There is no specific fact associated with a program in the Start State. When all data dependencies are satisfied, the *Ready Rule* executes, and the program enters the *Ready State,* which is marked by a corresponding fact. The Ready Rule triggers asynchronously, whenever all the data dependencies of a program are satisfied. Each target program has its unique Ready Rule in the SCI rule base.

The *Valid Rules* control the HLD execution flow. In *Run Mode*, a program can be scheduled at any time, so the transition from the Ready state to the Valid state is immediate and asynchro-

nous. In *Step Mode*, the programs can be scheduled only at predetermined events (*steps*). All the programs holding in the Ready State during the previous cycle are validated synchronously. Programs entering Ready State after the validation transition have to wait there until the next validation cycle. There is a Valid Rule for the Run Mode, and a Valid Rule for the Step Mode. These rules are common for all programs. If only the Run Mode is desired, the Valid Rules and the Valid state are eliminated from the transition diagram, and the Scheduling Rules respond to the Ready facts and not to the Valid facts. The execution mode is selected by the initial execution request and can be changed at any time. The same SCI rule base is used in all execution modes.

The *Scheduling Rules* differ for groups of programs that must be co-scheduled and for individual programs which are executed alone. An independent program is scheduled as soon as it enters the Valid state. Groups of programs are scheduled when all their execution dependencies are satisfied, in most cases when all the programs in the group are in the Valid state. Each Scheduling Rule creates a *Scheduling Block* of information which is passed to the Mapping Expert Advisor and an internal hook for the execution result. The Scheduling Rules are triggered asynchronously. Once a program has executed, the results are entered as specific *Success* or *Failure* facts in the knowledge base. In case of success, the output files are marked as valid, and can satisfy data dependencies of other programs. In case of failure, the SEA tries to recover from the error or lets the user handle the error.

Example

The preprocessing step of the Scheduling Expert Advisor converts the text or graphic HLD description into a set of facts and rules used to control the SEA. In this section we present a possible structure of the SCI facts and rules, the structure actually used by the Bond SEA.

The facts represent the status of various input files or control conditions. The preprocessor determines the set of *Absolute Input Files,* AIF, not generated by HLD programs and used as input for one of the HLD blocks. Bond checks if AIFs exist in the system. For the available AIF, the preprocessor adds a "`file file-name valid`" fact to the SCI. A fact "`file file-name invalid`" marks the missing AIF files.

The following rule types are generated by the preprocessor, as seen in Figure 6: (1) ready rule, (2) program or group scheduling rule, (3) successful execution rule, and (4) failed execution rule. In the following examples we ignore the Valid state which is controlled by shared rules.

The ready rules detect when a program has all the execution conditions satisfied. For example, program_N depends only on its input files:

```
(defrule ready_program_N
    (file input_data_file_1 valid)
    . . .
    (file input_data_file_k valid)
=>
    (assert (ready program_N input_data_file_1 ...
                input_data_file_k))
)
```

The step control changes the program state from Ready to Valid. If the program has no other execution constrains, the generated scheduling rule is:

```
(defrule schedule_program_N
    factx <- (valid program_N args)
=>
    (retract factx)
    (schedulef program_N args)
)
```

A group of programs (program_A to program_J) which execute concurrently has a single scheduling rule:

```
(defrule schedule_group_M
    fact_a <- (Valid program_A args_a)
    . . .
    fact_j <- (valid program_J args_j)
=>
    (retract fact_a)
    . . .
    (retract fact_j)
    (schedulef program_A args_a)
    . . .
    (schedulef program_J args_j)
)
```

The schedulef function passes the execution request to the Mapping Expert Advisor, and prepares a hook to insert the execution results in the working memory. The generated facts are:

```
(result program_N OK)
(result program_N error)
```

The rule for a successful execution validates the output files of the current program:

```
(defrule success_program_N
    factq <- (result program_N OK)
```

```
=>
     (retract factq)
     (validate output_data_file_1)
     . . .
     (validate output_data_file_m)
)
```

The validation procedure removes a possible invalid fact for the file name and generates a valid fact for the file (`file file_name valid`). The scheduling process continues with the new facts (valid files), which can trigger execution of the next programs.

The execution failure rule triggers an error recovery procedure or fails the entire process:

```
(defrule error_program_N
     factq <- (result program_N error)
=>
     (... report and process error)
     (... revalidate program_N | ignore | STOP )
)
```

Conclusions

There are major differences between an execution controlled by a program or a script and one controlled by an expert advisor. (a) A program or a script has limited adaptability properties, it needs to invoke error recovery routines to handle error conditions. An expert advisor can provide recovery rules invoked automatically when an error condition occurs, without additional overhead. (b) In an expert system environment individual operations can be enhanced without changing the entire system. Rules in an expert system are loosely coupled. Changes to one rule generator will not propagate to other generators or control rules. (c) In an expert advisor, the dependencies can be specified in any order. Each program has a number of conditions must

be valid before the program is started. There is no order in which these conditions have to be entered or fulfilled. The program will be scheduled for execution only when all conditions are satisfied. Multiple programs that can be started at the same time can be in concurrent execution. If the dependencies have to be satisfied in a specific order, we generate a chain of ready rules. The first rule checks the first dependency and when executed activates the second ready rule. The second rule checks the next dependency and so on, until all conditions have been satisfied in order. (d) Scheduler Control Information can be saved to a file and loaded in a separate expert system. This expert system can work decoupled from the original SEA to create a specialized Scheduling

Expert Advisor, used to control execution of a particular workflow. (e) A simple deadlock detection mechanism is available. If programs are placed into a circular dependency list, none of them can be executed. SEA recognizes the condition and informs the user of the fact. Dead sequences of program which cannot be executed are also reported by the SEA.

Acknowledgments

This work is supported in part by a grant from the Scalable I/O Initiative, by Intel Corporation, by National Science Foundation through grants BIR-9301210 and MCR-9527131, and by a grant from Purdue Research Foundation.

References

[Cli93a] *CLIPS Reference Manual. Volume I: Basic Programming Guide*. Clips version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, June 2nd 1993.

[Cli93b] *CLIPS Reference Manual. Volume II: Advanced Programming Guide*. Clips version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, June 2nd 1993.

[Cli93c] *CLIPS Reference Manual. Volume III: Interfaces Guide*. Clips version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, June 2nd 1993.

[DAR93] DARPA Knowledge Sharing Initiative. Specification of the KQML Agent Communication Language. DARPA Knowledge Sharing Initiative, External Interfaces Working Group Draft. June 15, 1993. WWW URL:
`http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps`

[EW94] Oren Etzioni and Daniel Weld, A Softbot-Based Interface to the Internet. *Communications of the ACM*, 37(7):72-76, July 1994.

[FG95] Stan Franklin and Art Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. WWW URL:
`http://www.msci.memphis.edu/~franklin/AgentProg.html`

[GAA95] Don Gilbert, Manny Aparicio, Betty Atkinson, Steve Brady, Joe Ciccarino, Benjamin Grosof, Pat O'Connor, Damian Osisek, Steve Pritko, Rich Spagana, Les Wilson. IBM Intelligent Agents White Paper. WWW URL:
`http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm`

[Gia93] Joseph Giarratano. *CLIPS User's Guide*. Clips version 6.0. Software Technology Branch, Lyndon B. Johnson Space Center. May 28th, 1993.

[GR94] Joseph Giarratano and Gary Riley. *Expert Systems, Principles and Programming*. PWS publishing Company, 1994. ISBN 0-534-93744-6.

[Lib95] Don Libes. *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*. O'Reilly and Associates, Inc., 1995. ISBN 1-56592-090-2.

[MLF96] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of KQML as an Agent Communication Language. In [WMT96], pages 347-360.

[Ous94] John K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley Publishing Company, 1994. ISBN 0-201-63337-X.

[RS94] M. D. Rice and S. B. Seidman. A Formal Model for Module Interconnection Languages. *IEEE Transactions on Software Engeneering* 20(1):88-101, January 1994.

[SM96] Mihai G. Sirbu and Dan C. Marinescu. Bond - A Parallel Virtual Environment. In *Proceedings of HPCN Europe '96*, pages 722-728. Volume 1067 of Lecture Notes in Computer Science, Springer Verlag, 1996. ISBN 3-540-61142-8.

[WMT96] Michael Wooldridge, Jorg P. Muller, and Milind Tambe. *Intelligent Agents II - Agent Theories, Architectures, and Languages*. Volume 1037 of Lecture Notes in Artificial Intelligence, Springer Verlag, 1996. ISBN 3-540-60805-2.