Chapter 3

ARCHITECTURE

3.1 Introduction

The most important activity in building an expert system, is the development of a complete and consistent domain knowledge and an efficient reasoning strategy. A formal model which can represent various aspects of building an expert systems, like knowledge representation, its verification and reasoning, is a Petri Net. A formal theory helps in abstracting and representing about a multitude of tasks in a complex activity.

Petri nets are formal models that are simple, yet powerful to represent complex systems with concurrent and interacting components. A systems approach is possible [44]. With Petri nets, it is possible to integrate different aspects of building expert systems, viz. knowledge representation, knowledge base verification and reasoning in a single formal model. The static structure of the Petri Net can be used for modelling knowledge representation and verification aspects. The dynamic aspects of Petri Nets can be used to model the reasoning process in expert systems.

A Petri Net is a bipartite directed graph, with places and transitions as nodes and directed arcs connecting places to transitions and vice versa. A formal definition is in [72]. Petri nets provide the basic formalism to model rule bases and capture all crucial aspects of representation and verification [8, 15, 46, 73].
3.2 The Extended Petri Net Model

The traditional rule models in production systems do not capture real time abstractions like events and temporal relationships. The rule model has been extended to effectively represent knowledge in a real time environment. Four rule types have been defined. They are Autonomous rules, Clock Synchronised rules, Event Spanning rules and Time Spanning rules. The detailed description of the rule types is given in Section 4.3. The elementary Petri Net model cannot model these augmented rule types. An Extended Petri Net model has been defined to model these rule types.

The Extended Petri Net is defined as a nine tuple $\text{EPN} = <\text{VP}, \text{PP}, \text{HP}, \text{EP}, \text{PT}, \text{ET}, \text{RT}, \text{I}, \text{O}>$ where

- $\text{VP}$ is the set of value places. A value place($vp$) is a cumulative place [88] containing tokens representing the values of an attribute(parameter).

- $\text{PP}$ is the set of premise places. The presence of a 'T' token in a premise place($pp$) indicates truth of the premise, while a 'F' token indicates otherwise.

- $\text{HP}$ is the set of hold places. The hold place($hp$) represents hold slots in the rules.

- $\text{EP}$ is the set of event places. The presence of a token in an event place($ep$) indicates occurrence of the event.

- $\text{PT}$ is the set of premise transitions. A premise transition($pt$) represents the computation of a premise's truth value.

- $\text{ET}$ is the set of event transitions. An event transition ($et$) represents the computation of an event's occurrence.

- $\text{RT}$ is the set of rule transitions. A rule transition($rt$) represents a rule.

- $\text{I}$ is the input function which describes the input places of a transition and the input transitions of a place.
• 0 is the output function which describes the output places of a transition and the output transitions of a place.

The Extended Petri Net semantics are summarised in Table 3.1. For example, in the table, the first entry is about the Value Place. The graphic symbol for a *value place* is a circle with a shaded lower chord. The *value place* represents an attribute value. A *value place* can have only a *Rule transition* as its input. It can have either a *Premise Transition* or *Event Transition* as its output transitions. The other entries in the table represent the properties of the remaining places and transitions in an EPN. For a transition enabled by the absence of tokens in its input place, in the graphic representation the input arc is terminated at the transition end with a circle. An attribute can also be updated by an external input, then the corresponding value place will be a source place and has no input transitions. Only value places can be source places in the net. A hold place is a sink place. All transitions in the EPN have associated computational functions. The function associated with a premise transition computes the premise truth value. The function of an event transition determines if the event has occurred. The function associated with a rule transition performs the actions of the rule.

### 3.2.1 Representing Rules with Extended Petri Nets

The Extended Petri Net model can capture all the rule types denned in REX. For example, consider the following clock synchronised rule Cl. The knowledge represented by the rule Cl is: If the Yaw command of 10° *LEFT* is issued by the vehicle control system and the yaw feedback of 10° *LEFT* is obtained before the elapse of 2 seconds after the command is issued then it is concluded that the yaw rate in the vehicle is too high (Section 4.3 gives the detailed semantics of the different rule types). The EPN representation for the rule is given in Figure 3.1.

**Cl:**

- Event: Yaw.and = 10° *LEFT*
- Time limit: 2 *seconds*
- Time operator: *BEFORE*
Table 3.1: Semantics of Places and Transitions in EPN

<table>
<thead>
<tr>
<th>Place/Transition</th>
<th>Symbol</th>
<th>Represents</th>
<th>Input type</th>
<th>Output type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP (value place)</td>
<td>Circle with shaded lower chord</td>
<td>Attribute value</td>
<td>RT</td>
<td>PT/ET</td>
</tr>
<tr>
<td>PP (premise place)</td>
<td>Circle</td>
<td>Premise truth value</td>
<td>VP</td>
<td>RT</td>
</tr>
<tr>
<td>HP (hold place)</td>
<td>Concentric circle</td>
<td>Hold slot</td>
<td>RT</td>
<td></td>
</tr>
<tr>
<td>EP (event place)</td>
<td>Circle with shaded upper chord</td>
<td>Event</td>
<td>ET</td>
<td>PT</td>
</tr>
<tr>
<td>PT (premise transition)</td>
<td>Vertical line</td>
<td>Computation of truth value</td>
<td>VP</td>
<td>PP</td>
</tr>
<tr>
<td>ET (event transition)</td>
<td>Pair of vertical lines</td>
<td>Computation of occurrence</td>
<td>VP</td>
<td>EP</td>
</tr>
<tr>
<td>RT (rule transition)</td>
<td>Box</td>
<td>Rule firing</td>
<td>PP</td>
<td>VP/HP</td>
</tr>
</tbody>
</table>

Premise: Yaw.fb = 10° LEFT

Hold: [message "yaw rate too high, check up"]

In the graphical representation, the event is represented by the event transition T1. The transition's input place is a value place representing the attribute Yaw.command. It's output place is a event place (P4). The premise is represented by the premise transition T3. The transition's input place is a value place (P3) and output place is a premise place (P6). The rule's time operator is BEFORE. The 'BEFORE' operator inhibits the firing of the rule after the lapse of 2 seconds from the occurrence of the event. This phenomenon is represented by the transition T2. The computational function associated with premise transition T2, counts a time of 2 seconds from the occurrence of the event (occurrence of a token in P4). After the lapse of 2 seconds a 'T' token is placed in P5. A 'F' token is in P5 before the lapse of two seconds. The rule transition 'T4' represents the complete rule. The transition is fired when the premise is satisfied within 2 seconds of the event occurrence. The computational function of T4 represents the rule's actions/hold.
Figure 3.1: EPN Representation for rule C1
Chapter 3. ARCHITECTURE

The graphical EPN representation has been transformed into two tables, suitable for manipulation by programs. The two tables are Place Property Table (PPT) and Transition Property Table (TPT). The PPT represents the places' properties. The TPT represents the transitions' properties. The dynamics of the Extended Petri Net (transition firings and markings) can be represented as updates in these two tables.

The structure of the Place Property Table is in Table 3.2. The Place-id represents the unique identification number of the place. The Place-type denotes the type of place (VP, PP, EP or HP). Input and output transition lists are the sets of input transitions and output transitions. Place semantics for a value place is the list of values, for a premise place it is a 'T' or 'F' token, for an event place and hold place it is the presence or absence of a token.

The structure of the Transition Property Table is in Table 3.3. The Trans. id is the unique transition identifier. The Trans. type is the transition type. I(t) represents the set of input places. O(t) represents the set of output places. The transition semantics for a premise and event transition is the associated computational function. The semantics are represented by a <function name-[time]>. We assume that the function denoted by <function name> embodies the computation of the truth value of the premise or event. For transitions representing time counters like T2 in Figure 3.1, the function name is T. The transition semantics of a rule transition is also represented by a computational function which performs the actions of the rule. The semantics of a rule with an output hold place are represented by a generic Hold function. Priority is for rule transitions and indicates rule priority. The rule status indicates whether the rule is not enabled, enabled or fired. An inhibitor arc is represented by a tilde preceding the input place in the TPT. The PPT and TPT
3.2.2 Forward Reasoning in Extended Petri Nets

The forward reasoning algorithm in production systems is the repeated execution of the following cycle.

- **Match**: For each rule, determine if the premises are satisfied.
- **Select**: Choose a rule whose premises are satisfied.
- **Act**: Execute the actions of the selected rule.

This reasoning algorithm can be modelled as a traversal of the net by firing enabled transitions along the path. An update to a value place enables a set of premise transitions. When the premise transition is fired, the associated computational function calculates the truth value of the premise and places a 'T' or 'F' token in its output premise place. A rule transition with appropriate tokens in its input premise places is enabled and can be fired. Firing a rule transition invokes the
associated function which performs the actions of the rule by updating the output value or hold places of the rule transition. The updated value place in turn enables some premise transitions and the cycle continues. Thus forward chaining can be easily modelled with Extended Petri Nets.

Consider the rule Cl. In a scenario, let us assume that the Yaw command of 10° LEFT is issued. In this case, the event of rule Cl is satisfied. Now, the transition T1 is fired, and a token is placed in event place P4. Let us assume that immediately, the Yaw feedback becomes 10° LEFT. Now, premise transition T3 is enabled and fired. A 'T' token is placed in premise place P6. Since the feedback became 10° LEFT almost immediately, let us assume that the 2 second limit has not elapsed. So, a T' token is present in premise place P5. Since a tilde is present on the arc to transition T4, this is an enabling condition for transition T4 (the rule premises and temporal relationships are satisfied). Since the transition is enabled, it is fired and a token is placed in place P7. Real time expert systems need to fire multiple rules concurrently, in order to handle simultaneously occurring events in the external world. The asynchronous dynamic nature of the net model allows us to model concurrent multiple rule firing. In Petri nets, transition firing is instantaneous. However, in actual implementations, the process takes finite time, and uninhibited concurrent firing can lead to race conditions. This results in loss of integrity of the places (working memory). So, a mechanism to protect the integrity of the working memory is required. Such a mechanism is termed as interference analysis in multiple rule firing systems. The reasoning cycle with multiple rule firing will be

- Match :For each rule, determine if the premises are satisfied.
- Select :Choose a set of non-conflicting rules whose premises are satisfied.
- Act :Execute the actions of the selected rules concurrently.

3.2.3 Reasoning Algorithm

The reasoning algorithm for the multiple rule firing forward reasoning system is given below.
reason()

Execute the following steps repeatedly.

1. Determine the set of Attributes (value places) that are updated.

2. For each of the updated attributes determine the set of premises and events that have to be evaluated (i.e. find the set of output premise and event transitions for each updated value place).

3. For each of the premises determined in the earlier step, evaluate the premises and determine their truth value (i.e. for each premise transition determined in step 2, evaluate the premise function and place either a "1" or 'F' token in the corresponding output premise places).

4. (a) For each of the events determined in step 2, evaluate the events and determine if they have occurred (i.e. for each event transition determined in step 2, evaluate the event and place a token in the corresponding output event place, if the event has occurred).

(b) For each of the events determined to have occurred in step 4(a), find the list of premises to be evaluated (i.e. for each event place in which a token is placed in step 4(a), find the set of output premise transitions).

(c) For each of the premises determined in step 4(b), evaluate the premises and determine their truth value (i.e. for each premise transition determined in step 4(b), evaluate the transition function and place either a 'T' or 'F' token in the corresponding output premise place).

5. For the premises determined to be true in steps 3 and 4, find the set of rules that are likely to have matched (i.e. for each premise place updated in steps 3 and 4, find the set of output rule transitions).

6. For the rules determined in step 5, find the set of rules, all of whose premises are true (i.e. find the set of enabled rule transitions in the net).
7. From the set of matched rules determined in step 6, find the set of rules that can be fired while maintaining working memory integrity (i.e. perform interference analysis).

8. Perform the actions of the rules determined in step 7 (i.e. fire the enabled rule transitions and update the corresponding output value and hold places).

3.3 REX's Architecture

REX is an expert system shell architecture for building real time expert systems reasoning with continuous streams of input data [67, 70]. REX has been designed to meet the requirements discussed in Section 1.2. Data in real time domains is multi faceted and has many properties. Simple data structures cannot capture the semantics of real time data. Further, the expert systems have to reason with historic data. This requires a structured and efficient data management scheme, which are readily available in an object oriented model with its facilities for data abstraction, encapsulation and inheritance. Further, the augmented rules in REX can also be modelled in an object oriented fashion, leading to a uniform treatment of both data and knowledge. Hence, we have used an object oriented data and knowledge model in REX. The REX architecture is designed based on the Extended Petri Net model and the reasoning algorithm presented in the earlier section. The REX architecture is depicted in Figure 3.2. It consists of

- an external data interface
- an object manager which manages the associated data and knowledge store
- a common work area consisting of
- object instances
- an Attribute Event Premise (AEP) Index Table
- an Attribute Table
- a Premise Table
- an Event Table and
- a Rule Table

• a reasoning subsystem consisting of
  - an acquisition module
  - an evaluation modules
  - and a scheduling module

### 3.4 External Data Interface

The *External Data Interface* consists of two tasks. They are

1. *External Data Acquisition Task (EDAT)*
2. *Attribute Update Task (AUT)*

**External Data Acquisition Task**

The *External Data Acquisition Task (EDAT)* accepts preprocessed sensor data from the external world in continuous streams of data packets. The data packet format though varying from application to application has a generic structure shown in Figure 3.3.

The data packet has a fixed format. Each data packet has a time reference, which indicates the time at which the data values in the packet are sensed in the external world. The position of the datum identifies the corresponding attribute. The EDAT will obtain these data packets and store them in the buffer. The *Attribute Update Task* would do further processing on this buffer.
Figure 3.2: ARCHITECTURE OF REX

EDAT  External Data Acquisition Task
AUT   Attribute Update Task
KAT   Knowledge Acquisition Task
SAT   State Acquisition Task
PET   Premise Evaluation Task
EET   Event Evaluation Task
RET   Rule Evaluation Task
IAT   Interference Analysis Task
RFS   Rule Firing Scheduler
RFT   Rule Firing Task
Chapter 3. ARCHITECTURE

<table>
<thead>
<tr>
<th>time reference</th>
<th>attribute value</th>
<th>attribute value</th>
<th>attribute value</th>
</tr>
</thead>
</table>

Figure 3.3: Generic structure of a data packet

Attribute Update Task

The Attribute Update Task (AUT) processes the data placed in the buffer by EDAT. It extracts the data values from the packet and posts them in the Attribute Table of the Work Area. The AUT will then invoke the Object Manager. The Object Manager will form object instances by using the updated Attribute Table. The newly created object instances will be placed in the Object Instances space of the Work Area. The Reasoning Subsystem is then triggered. It uses the Attribute Table and Object Instances space in the reasoning process. Thus external world data is available for the Reasoning Subsystem through the External Data Interface.

The objective of dividing the job of external data interface into EDAT and AUT is to avoid loss of external data. The Attribute Table is a common data area used by the external data interface, object manager and the reasoning subsystem. Hence, synchronisation of access to the Attribute table is necessary. If the external data interface is implemented as a single task, there is a possibility that while waiting for access to the Attribute table, external data arrives and is lost. The EDAT and AUT are organised using the producer-consumer paradigm. The EDAT produces data from the external world and the AUT consumes the same by updating the Attribute Table.

3.5 Object Manager

An unified object oriented paradigm is used to store data and knowledge in REX. The conventional rule structure is augmented to represent knowledge in REX. A taxonomy of rule classes(types) with inheritance relationships has been defined in REX (detailed description is in Section 4.3). All rules in the rule base are instances of one of the rule classes. The Object Manager in addition to it data management tasks, is responsible for the rule base management. The Object Manager performs
the following tasks

- Schema Evolution
- Creation and Management of Object Instances
- Storage of Object Instances
- Support for predefined methods on Objects
- Work Area Initialisation

The Object Manager has to perform the above tasks both for data and rule objects. However, there is no schema evolution for rule objects as its schema is fixed. The Object Manager has to provide the necessary functions to create, edit and store rule objects. The Manager in addition will provide facilities for rule compilation and integration of the generated code with the reasoning subsystem. The Object Manager will also retrieve the rule objects during work area initialisation and build the Premise, Event and Rule tables.

### 3.5.1 Schema Evolution

A schema in an object oriented data model consists of a set of classes bound by a certain hierarchical relationships. The evolution of the schema is through the definition of classes and the hierarchical relationships among the classes. The Object Manager provides functions for defining a class, its attributes, its methods, default values, legal values, super classes and subclasses. The entire object schema is defined using these functions.

### 3.5.2 Creation and Management of Object Instances

The Object Manager a unique identifier called the Object Identifier (OID) for each object, at the time of creation. All references to the object are made using the object identifier. The necessary function for creation of new instances is provided in
the Object Manager. Further, all objects belonging to a particular class are stored in a doubly linked list in the Object Instances space. The Object Manager handles this list and provides the instances needed to the Reasoning Subsystem.

3.5.3 Retrieval of Object Instances

Object instances are usually available in the Object Instances space of the Work Area. However, due to space constraints and low frequency of use, some instances may be shifted to the secondary store. The Object Manager decides when an object should be shifted to secondary store. So, when required it has to retrieve these instances from the secondary store to the Work Area. Objects are stored in two different formats, one for the main memory and another for the secondary storage. Hence retrieval of objects involves format conversion. The Object Manager provides the necessary functions to implement this task.

3.5.4 Secondary Storage Management

The object instances are shifted to the secondary store when they exceed the space constraints of the Work Area. The secondary store also provides the persistency of object instances. It has been built using a B+ indexed file structure. The index is maintained on the Object Identifiers. The Object Manager provides the functions for maintaining the secondary store and index. A similar secondary storage for rule objects is also to be managed by the Object Manager

3.5.5 Support for Predefined Methods on Objects

A set of methods have been defined for all classes in the taxonomy. These methods perform tasks like attribute update, attribute value retrieval and object display. The Object Manager supports these functions.
3.5.6 Work Area Initialisation

The Work Area will consist of all state information and knowledge necessary for the Reasoning Subsystem. The Object Manager has to initialise this area during system startup phase. The Object Manager will build the Attribute Table from the class taxonomy. Similarly, from the rule base, the Object Manager will build the Premise Table, Event Table and Rule Table in the Work Area. The Attribute - Event - Premise Index Table is built next. After the Work Area has been initialised by the Object Manager, the reasoning process can be initiated.

3.6 Work Area

The Work Area is more than the traditional working memory. It maintains all state information and knowledge indices. It is designed to help build an efficient reasoning subsystem. The Work Area consists of

- **Object Instances Space**
- **Attribute Table**
- **Attribute Event Premise Index Table**
- **Premise Table**
- **Event Table and**
- **Rule Table**

The Place Property Table (PPT) and Transition Property Tables (TPT) of the Extended Petri Net are divided into four different tables viz. Attribute Table, Premise Table, Event Table and Rule Table to design an efficient reasoning system.
3.6.1 Object Instances Space

The Object Instances Space of the Work Area consists of recent object instances in a temporal order. Instances of all classes defined in the object taxonomy of the expert system are stored in this space and are used in evaluating Spanning premises (refer Section 4.3 for details about Spanning Premises).

3.6.2 Attribute Table

The Attribute Table is a repository of current data about all object attributes defined in the object taxonomy. In addition to storing the attribute's current value, it stores other information like valid life span and update flag which is useful for the Reasoning Subsystem. Chapter 5 presents the structure of the Attribute Table and its use. The Attribute Table can be viewed as a sort of cache storage for the Match Process. The Acquisition Module can retrieve all current object attribute values from the Attribute Table without accessing the Object Instances Space through the Object Manager.

3.6.3 Attribute Event Premise Index Table

The Attribute Event Premise (AEP) Index Table is a static table used by the Acquisition Module. The Match Process in REX is an incremental process. In every inference cycle, it is sufficient to evaluate only those premises/events, whose truth values/occurrence flags could have changed due to attribute updates. The AEP Index Table maintains, for every attribute the list of premises and events in which the attribute participates. This table is used by KAT of the Acquisition Module to determine the premises and events to be evaluated.

3.6.4 Premise Table

The Premise Table contains information about all the premises present in the rule base. It stores information such as truth value and any event dependency(like in a
clock synchronised rule). The truth values are updated by the PET and used by the RET to form the MRS. Details about the table structure and use are in Section 5.3.

### 3.6.5 Event Table

The Event Table contains information about the events defined in the rule base. Information such as time limit, time operator (both defined for clock synchronised rules in Chapter 4), event occurrence flag are stored in this table. It also stores the list of premises to be evaluated if the event occurs. This table is used by the Event Evaluation Task of the Evaluation Module. Details about the table structure and use are in Section 5.3.

### 3.6.6 Rule Table

The Rule Table contains information about the type of rule (refer Section 4.3), its premises, actions and priority. The information in Rule Table is used by the Rule Evaluation Task to form the Matched Rule Set. The Rule Firing Scheduler uses this table to retrieve the action codes of the rules and schedule their execution. This table like the AEP Index Table is a static table and is not updated during run time. Details about the table structure and use are in Section 5.3.

### 3.7 Reasoning Subsystem

Real time expert systems monitoring continuous streams of input data are reactive in nature. So, the reasoning process is data driven. Hence a forward chaining problem solving paradigm is ideal for such applications. The REX's Reasoning Subsystem uses the forward chaining with multiple rule firing model to handle simultaneously occurring events in the external world. The Reasoning Subsystem executes the following cycle repeatedly.

- **Match**: For each rule, determine if the premises are satisfied.
• Select : Choose a set of non-conflicting rules whose premises are satisfied.

• Act : Execute the actions of the selected rules concurrently.

The Match phase of the cycle is performed by the Acquisition and Evaluation Modules. The Scheduling Module implements the select and act phases.

3.7.1 Acquisition Module

The Acquisition Module is responsible for determining changes in the current system state and the relevant knowledge to be applied. The module consists of two tasks,

• State Acquisition Task (SAT) and

• Knowledge Acquisition Task (KAT).

State Acquisition Task

The Match process in REX is an incremental process. The process is shown in Figure 3.4. The set of satisfied rules is determined by using the updates to the Attribute Table as the seeds (i.e. initiation points for determining the rules). The State Acquisition Task (SAT) has to determine the changes made to the Attribute Table either by the Attribute Update Task or Rule Firing Tasks.

Knowledge Acquisition Task

Since, the REX reasoning system is data driven, the current input data external world determines the relevant knowledge to be applied. The Knowledge Acquisition Task (KAT) determines this knowledge by using the state information captured by the SAT and the Attribute Event Premise (AEP) Index Table.
Figure 3.4: Incremental Match
3.7.2 Evaluation Module

The primary function of the Evaluation Module is to determine the set of matched rules in the current context. The module interacts with the Acquisition Module during the course of state evaluation. It is composed of three tasks. They are

- **Premise Evaluation Task (PET)**
- **Event Evaluation Task (EET)** and
- **Rule Evaluation Task (RET).**

Premise Evaluation Task

Knowledge in REX is modelled using augmented Premise-Action rules (refer Section 4.3 for details about the knowledge representation scheme). The premises are evaluated by the Premise Evaluation Task using the contents of the Attribute Table to determine the applicability of rules. The KAT of the Acquisition Module determines the list of premises to be evaluated. The PET executes the corresponding premise evaluation functions to determine the set of satisfied premises (i.e. the set of premises which values are true). REX employs the strategy of executing compiled code rather than interpreting premises. This is done to speed up the evaluation process.

A real time reasoning system, should possess adequate facilities for representing knowledge about system trends, representing events and their temporal relationships. A simple and generic Premise-Action knowledge representation facility cannot capture complex abstractions in real time environments. In REX, two augmented rule representation facilities termed as Clock Synchronised rules and Spanning rules and are provided (details are presented in Section 4.3) for this purpose.

Spanning rules are event based, and they can system trends. The evaluation of spanning premises in these rules is based on the calculation of historical trends in the external environment. The Attribute Table contains only values representing the current state. System state at earlier points of time (i.e. historical data) are
available as object instances in the Object Instances space. The State Acquisition Task interacts with the Object Manager to retrieve the necessary instances and passes the same to the PET.

Event Evaluation Task

Clock synchronised rules represent event based knowledge. The evaluation of premises in these rules is triggered by the occurrence of a predefined event. The Event Evaluation Task determines event occurrences by evaluating events in a manner similar to premise evaluation by PET. For all events, that have occurred, the EET finds out from the Event Table, the set of premises that must be evaluated (these premises are in those clock synchronised rules and spanning rules that are to be triggered in the current context). This list of premises is passed to the PET, which determines the truth value of these premises.

Rule Evaluation Task

The Rule Evaluation Task determines the set of matched rules (rule instantiations) and forms the Matched Rule Set. It uses the Rule Table and Premise Table coupled with the set of satisfied premises to form the Matched Rule Set (conflict set in OPS5 terminology) which is then passed to the Scheduling Module for further processing.

3.7.3 Scheduling Module

The Scheduling Module is responsible for the creation of rule firing tasks and scheduling their execution. As explained earlier, REX adopts an asynchronous multiple rule firing model to enable reasoning about multiple events concurrently. The Scheduling Module consists of two tasks,

- Interference Analysis Task (IAT) and
- Rule Firing Scheduler (RFS).
Interference Analysis Task

Taking a simplistic view, the *Scheduling Module* just needs to create tasks for each of the rule firings and put them on the schedule queue for execution. (There is no need to interpret the rule actions (consequents), as they are also compiled into execution code similar to the premises). However, the fundamental issue involved in multiple rule firing models, is work area consistency. For example, assume that the following two rules are present in the Matched Rule Set. If the *Scheduling Module* simply creates two tasks for the execution of the rule actions concurrently, it leads to inconsistency in the Attribute Table and consequently in further reasoning.

R1:
- Premise: \( A.a_1 < 30; B.b_2 > 40 \)
- Action: \( D.d_1 = 100 \)

R2:
- Premise: \( D.d_1 < 50 \)
- Action: \( A.a_1 = 100 \)

This inconsistency arises because the actions of one rule invalidate the premises of another rule. If both the rules are fired (action execution) concurrently then there is no equivalent serial execution path. This is termed as *interference* [31] in multiple rule firing models. In order to ensure consistency of the Work Area, the *Scheduling Module* must detect such rules and inhibit them from firing concurrently. This task is known as *interference analysis*. The *Interference Analysis Task (IAT)* performs this function. IAT performs interference analysis on the Matched Rule Set generated by the *Evaluation Module*, and it generates as output a set of rules called the *Eligible Rule Set (ERS)*. The rules in ERS possess the property of maintaining Work Area consistency even if fired concurrently.

The task of interference analysis requires comparing the actions of every rule with the premises of every other rule. This results in combinatorial explosion, unless the number of combinations to be tested is pruned using suitable techniques. Different
techniques and approaches to perform interference analysis are proposed [31, 33, 40, 56, 57, 81], some of which are surveyed in Chapter 2. These techniques require both off-line (compile time) and run time analysis of the rules and are space and compute intense. The trade off usually applied in these techniques is between their execution time and the degree of concurrency. In REX, a new approach to interference analysis is designed and implemented. This technique requires little compile time analysis and is less compute intense than most other techniques. The IAT implements this algorithm and forms the ERS. The design details are presented in Chapter 6.

Rule Firing Scheduler

The Rule Firing Scheduler (RFS) creates separate tasks for each of the rules in ERS. The tasks execute asynchronously and make updates to the Attribute Table. The successful completion of all the rule firing tasks signals the completion of a single inference cycle and starts a new cycle. Since, the different rule firing tasks execute asynchronously, without waiting for another rule firing task, we terms our REX as an asynchronous production system.

3.8 Rule Firing Tasks

The Rule Firing Tasks (RFT’s) execute the actions of rules present in the ERS. The RFT’s update the Attribute Table. These tasks are shown as being outside the Reasoning Subsystem in Figure 3.2. This is because they are a consequence of the Reasoning Subsystem performing its function, rather than a part of the subsystem itself.

The Rule Firing Tasks on completion should trigger the Acquisition Module of the Reasoning Subsystem to start the next inference cycle. Similarly, the Object Manager should be asked to form new object instances based on the attribute value updates made by the rule firings. If every firing task, triggers the Acquisition Module and the Object Manager, there would be a plethora of task copies of Object Manager and Acquisition Module vying to run. Multiple match processes lead to
race conditions and inconsistent results. The Match Process should start after all rule firings are complete. In order to ensure this, all RFT's execute a Set_and_Test function before triggering the Acquisition Module. The Rule Firing Scheduler of the Scheduling Module while creating the rule firing tasks set the variable RFT_number to the number of tasks being created. The Set_and_Test function would use this variable. The pseudo code of this function is

\[
\text{Set\_and\_Test()} \\
\quad \text{RFT\_number} -= -;
\]

\[
\quad \text{if(RFT\_number} == 0) \\
\quad \quad \text{trigger Acquisition Module and Object Manager}
\]

The execution of Set_and_Test function by all the RFT's will ensure that the Match Process is triggered only after all rule firings are complete.

3.9 Other Factors

The Attribute Table is a common memory area which is referenced by most process. The AUT and the rule firing tasks update the Attribute Table, while all processes reference it. The correctness of the MRS formed by the Match Process depends on the integrity of the Attribute Table. Since, multiple processes can update the Attribute Table concurrently, measures to ensure its integrity are necessary. The Attribute Table should not be updated while the Reasoning Sub-system is referencing it. A locking scheme is employed to synchronise access by the External Data Interface and the Reasoning Subsystem to the Attribute Table. This synchronisation is a primary reason for bifurcating the External Data Interface into two separate tasks, one for obtaining external world data (EDAT) and another for update the Attribute Table (AUT). The External Data Interface is allowed to updating the Attribute Table while rule firings take place. This is because the set of attributes updated by the External Data Interface and Rule Firing Tasks will be disjoint (rule firings cannot update sensor data).
3.10 Summary

In this chapter, an Extended Petri Net model has been defined. This EPN model can capture the representation, reasoning and verification aspects in real-time systems. The REX architecture based on this EPN model is presented. The different tasks and memory structures in REX were explained using a schematic diagram of the architecture. The data and knowledge representation scheme is the subject of the next chapter.