Grupo de Computadores y Tiempo Real
Universidad de Cantabria

Modeling and Analysis Suite for Real Time Applications (MAST 1.2)

Description of the MAST Model

By:	José María Drake	drakej@unican.es
	Michael González Harbour	mgh@unican.es
	José Javier Gutiérrez	gutierjj@unican.es
	José Carlos Palencia	palencij@unican.es

1. Introduction

In this document we describe the basic characteristics of MAST, a Modeling and Analysis Suite for Real-Time Applications. MAST is still under development and tries to provide an open source set of tools that enable engineers developing real-time applications to perform schedulability analysis of their application.

The motivations for developing MAST are mainly that the schedulability analysis techniques have evolved a lot in the past decade, and in particular for fixed priority scheduled systems, such as those built with commercial operating systems or commercial languages. Today a full set of techniques exists for event-driven distributed real-time systems.

The new aspects that cannot be found in other tools that we know about are the following:

- A very rich model of the real time system is used. It is an event-driven model in which complex dependence patterns among the different tasks can be established. For example, tasks may be activated with the arrival of several events, or may generate several events at their output. This makes it ideal for analyzing real-time systems that have been designed using UML or similar design tools, which have event driven models of the system.
- The latest offset-based analysis techniques are used to enhance the results of the analysis. These techniques are much less pessimistic than previous schedulability analysis techniques.
- The toolset will be open source and fully extensible. That means that other teams may provide enhancements. The first version is intended for fixed priority systems, but dynamically scheduled systems may be added in the future.

2. Requirements

Develop a model to describe event-driven real time systems, with the following characteristics:

- Open model, that can include new characteristics or viewpoints of the system

- Should be able to handle most real-time systems built using commercial standard operating systems and languages (i.e., POSIX and Ada). This implies fixed priority scheduled systems, but the system will be extended in the future to other scheduling algorithms (EDF,...). Among fixed priorities, different scheduling strategies should be allowed:
  - preemptive and non preemptive
  - interrupt service routines
  - sporadic servers
  - polling
- Should be able to handle distributed systems.
- Emphasis is on event-driven systems in which each task may conditionally generate multiple events at its completion. A task may be activated by a conditional combination of one or more events. The external events arriving at the system should be of different kinds:
  - periodic
  - unbounded aperiodic
  - sporadic
  - bursty
  - singular (arriving only once)
- The system model should be rich enough to facilitate the independent description of overhead parameters such as:
  - Processor overheads.
  - Network Overheads
  - Network driver overheads
- Timing requirements should be allowed to be both hard and soft. Deadlines as well as maximum output jitter requirements should be allowed.
- The tool provides the user with capabilities to automatically calculate the following system parameters:
  - optimum priorities
  - possibility of deadlocks (not yet implemented)
  - priority ceilings for shared resources

The model is included in a toolset, with the following elements:

- The model is specified through an ASCII description that serves as the input of the analysis tools.
- Graphical editors and other tools generate the system using this ASCII description
- A parser converts the ASCII description of the system into an Ada data structure that is used by the tools

- A module exists to convert the Ada data structure back to the ASCII description

The MAST environment will integrate the following tools described in Figure 1:

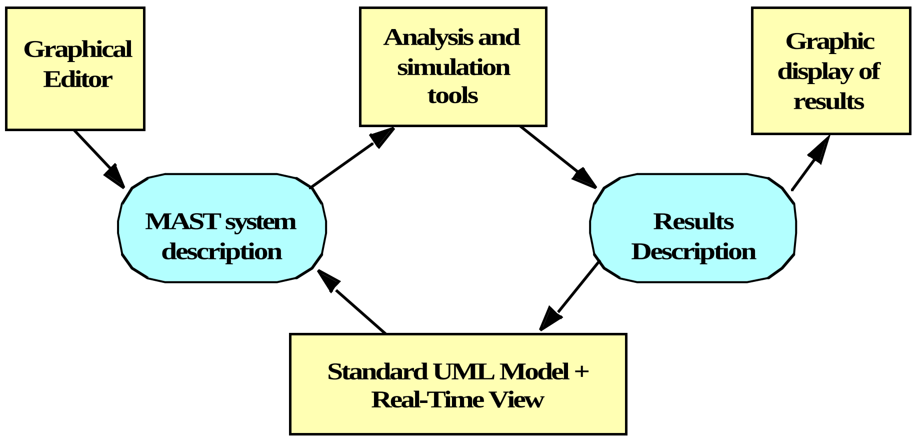

Graphical Editor
Analysis and simulation tools
Graphic display of results
MAST system description
Results Description
Standard UML Model + Real-Time View

Figure 1. MAST toolset environment

The analysis tools perform different kinds of worst-case analysis to determine the schedulability of the system. Blocking times relative to the use of shared resources are calculated automatically.

The simulation tools will be able to simulate the behavior of the system to check soft timing requirements

The graphical editor will allow the user describing the system and invoking the analysis tools. A graphical display of results will be available.

Using a (non real-time) UML tool, it is possible to describe a real-time view of the system by adding the appropriate classes and objects that are necessary to have the real-time behavior of the system described, and linking the system design with the real-time view as appropriate. Then, an automatic tool extracts the real-time description of the system from the UML description, generating the MAST description file. No special framework is needed with this approach, but the designer must incorporate the real-time view into the UML description.

Figure 2 represents the MAST toolset. The capabilities of the different tools are represented in the following table

Technique	**Single-Processor**	**Multi-Processor**	**Simple Transact.**	**Linear Transact.**	**Multiple Event T.**
Classic Rate Monotonic	✓		✓		
Varying Priorities	✓		✓	✓	
Holistic	✓	✓	✓	✓	
Offset Based Unoptimized	✓	✓	✓	✓	
Offset Based	✓	✓	✓	✓	
Multiple Event	✓	✓	✓	✓	✓

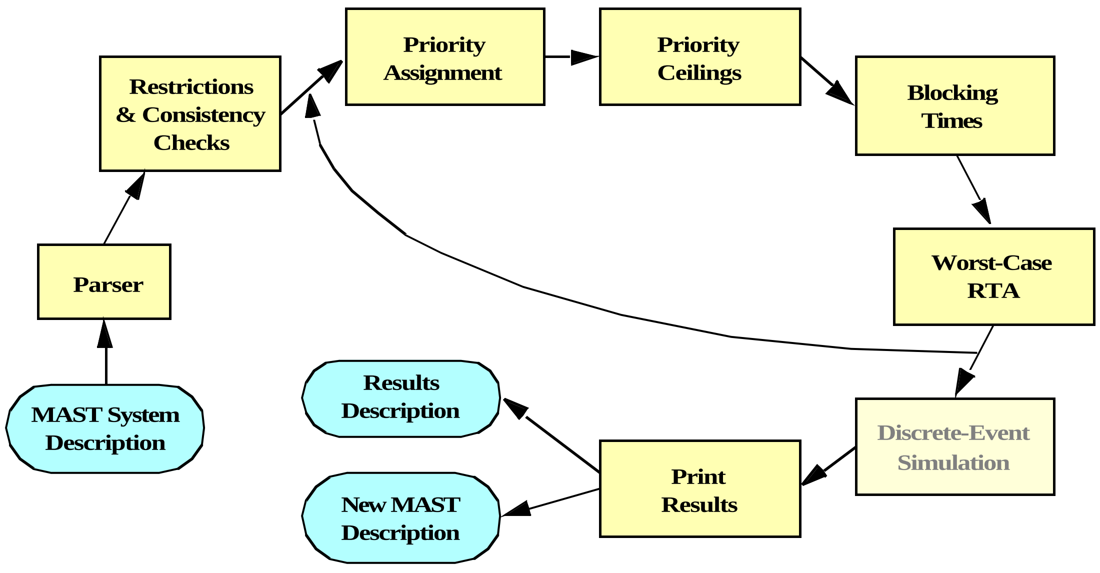


Figure 2. MAST Analysis tools

3. Real-Time System Model

A real-time system is modeled as a set of transactions. Each transaction is activated from one or more external events, and represents a set of activities that are executed in the system. Activities generate events that are internal to the transaction, and that may in turn activate other activities. Special event handling structures exist in the model to handle events in special ways. Internal events may have timing requirements associated with them.

Figure 3 shows an example of a system with one of its transactions highlighted. Transactions are represented through graphs showing the event flow. This particular transaction is activated by only one external event. After two activities have been executed, a multicast event handling object is used to generate two events that activate the last two activities in parallel.

We call the “boxes” that are included in the transaction *Event Handlers*. As we have mentioned, there are event handlers that just manipulate events, like the *Multicast* event handler in Figure 3. Another very important event handler is an *Activity*, which represents the execution of an operation, i.e., a procedure or function in a processor, or a message transmission in a network.

The elements that define an activity are described in Figure 4. We can see that each activity is activated by one *input event*, and generates an *output event* when completed. If intermediate events need to be generated, the activity would be partitioned into the appropriate parts. Each activity executes an *Operation*, which represents a piece of code (to be executed on a processor), or a message (to be sent through a network). An operation may have a list of *Shared Resources* that it needs to use in a mutually exclusive way.

The activity is executed by a *Scheduling Server*, which represents a schedulable entity in the *Processing Resource* to which it is assigned (a processor or a network). For example, the model for a scheduling server in a processor is a task. A task may be responsible of executing several activities (procedures). The scheduling server is assigned a *Scheduling Parameters* object that contains the information on the scheduling policy and parameters used.

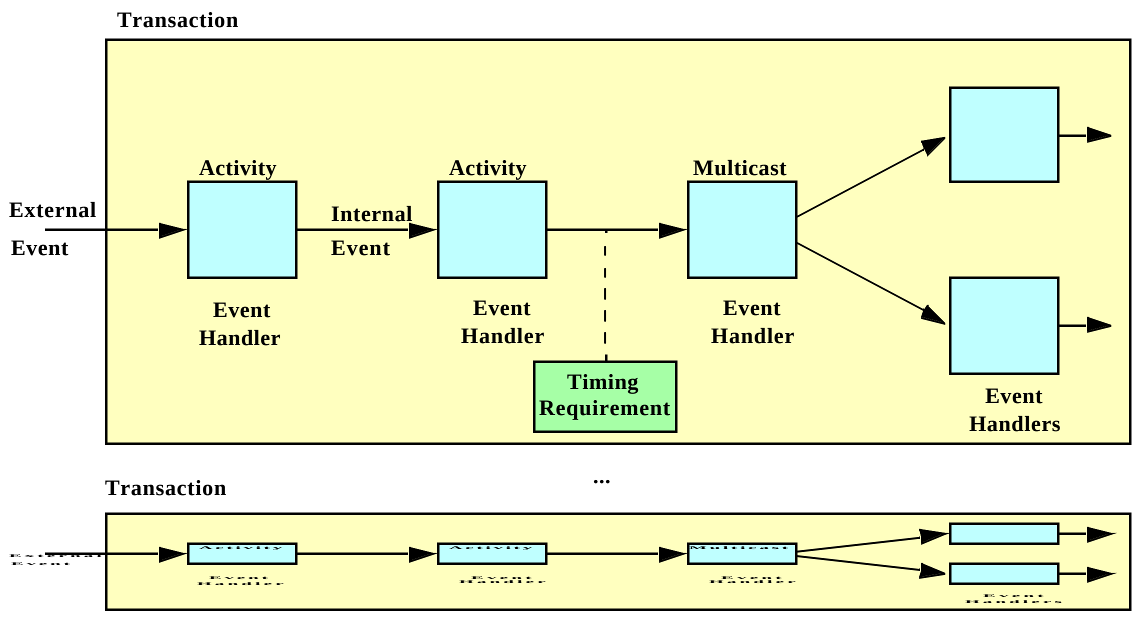


Figure 3. Real-Time System composed of transactions

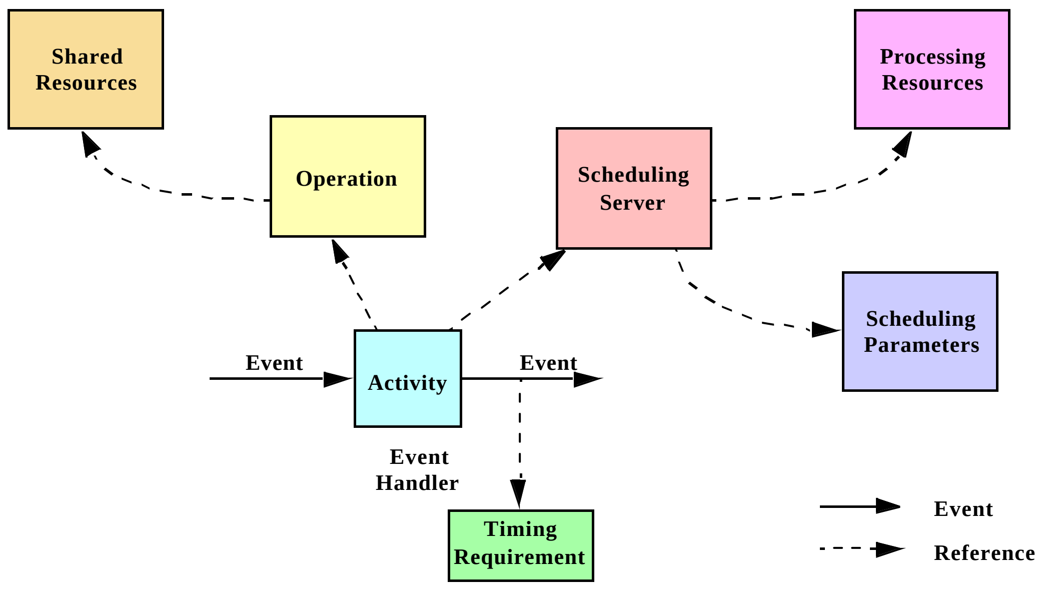


Figure 4. Elements that define an activity

4. MAST Output Files

The MAST tools produce several output files:

- *Console output*: Describes the work carried out by the tools, and any possible errors, in free format. If the verbose option is set, the tools provide a more detailed output. The last line in the file contains the string “`Final analysis status:` *`code`*”, where code is a single word that is either “DONE”, or some error indication.

- *Source destination file*: Describes the source of the MAST model of the analyzed system, including any elements introduced by the analysis tools into the system such as priorities, or priority ceilings. It follows the file format used for the MAST model. This file is only produced if the corresponding option is set.
- *Results file*: Describes the results of the analysis tools. If a filename is not provided for the results, they are written to the standard output, together with the Console Output. See Section 9 for a description of its format.

5. Type definitions

The following types are used in the definitions of the components of the MAST File and the MAST Results File:

- *Identifier*. String of characters following the rules described in the following section.
- *Priority*. Positive integer of implementation-defined range, defining the scheduling priority of tasks.
- *Interrupt_Priority*. Positive integer of implementation defined range, defining the scheduling priority of interrupt service routines.
- *Any_Priority*. Positive integer that is either in the *Priority* range or in the *Interrupt_Priority* range.
- *Normalized_Execution_Time*. Represents the execution time of an operation, as executed by a normalized processing resource of speed factor equal to one. It is obtained by multiplying the real execution time by the processing resource's speed factor.
- *Time*. Time interval in unspecified time units.
- *Absolute_Time*. Absolute time measured from and arbitrary time origin, in unspecified units.
- *Float*. It represents any float type.
- *Positive*. Integer positive number (excluding zero).
- *Natural*. Integer number that is greater than or equal to zero.
- *Percentage*. A floating point number representing a percentage, and followed by a "%" character. In some cases (slacks) the notation ">=num%" may be used to indicate that the actual result is greater than the specified number.
- *"Text"*: String of arbitrary characters, excluding the double quote character, and delimited within double quotes.
- *Date-Time*: String representing a date and time (hours, minutes and seconds) in the extended ISO 8601 format with no time zone:
  YYYY-MM-DDThh:mm:ss (e.g., 1997-07-16T19:20:30).
- *Pathname*: String representing a pathname of a file.

6. Writing the MAST File

The rules for writing the file with a real-time system according to the defined real-time system model are the following:

- Each object has the format:
  object_name (arguments);
- Most objects have a type and/or a name argument. In those cases, they are mandatory arguments, and they have to be defined as the first and second argument, respectively. All other arguments can go in any order, and are mostly optional.
- Blank spaces, tabs and new lines are ignored.
- Identifiers or names follow the Ada rules for composite identifiers: begin with a letter, followed by letters, digits, underscores ('_') or periods ('.').
- Identifiers or names can be expressed with or without quotes. A quoted name can be the same as one of the reserved words (appearing in bold face below).
- Each name that is referenced must have been defined earlier in the file.
- Float types without fractional part can be expressed without the decimal point.
- Comments are like in Ada: they begin with two dashes ("--"), anywhere in a line, and end at the end of the line.
- The description is not case-sensitive.

7. Elements of the MAST model

In this section we review in detail the particular classes and attributes of the different elements of the MAST model. The elements that we will review are:

- Processing Resources
- System Timers
- Network Drivers
- Scheduling parameters (policies, priorities...)
- Scheduling Servers (tasks, processes, threads,...)
- Shared resources (for mutually exclusive access)
- Operations (procedures, functions, messages,...)
- Events
- Timing Requirements
- Event Handlers
- Transactions
- Overall system model

7.1 Processing Resources

Common attributes:

- *Name*. A string.
- *Max Priority* and *Min Priority*. They define the range of priorities valid for normal operations on that processing resource. Special operations (such as interrupt service routines in processors) may have other priority ranges.
- *Speed factor*. All execution times will be expressed in normalized units. The real execution time is obtained by dividing the normalized execution time by the speed factor. The default value is 1.0.

Classes of Processing Resources:

- *Fixed Priority Processor*. It represents a processor scheduled with fixed priorities. It has the following additional attributes:
  - *Max Interrupt priority* and *Min Interrupt priority*. They define the range of priorities valid for activities scheduled by an interrupt service routine.
  - *Context Switch Overheads* (Worst, Average, Best).
  - *ISR Switch Overheads* (Worst, Average, Best).
  - *System Timer*. A reference to the system timer used (see below), that influences the overhead of the *System Timed Activities*.

```
Processing_Resource (
     Type                             => Fixed_Priority_Processor,
     Name                             => Identifier,
     Max_Priority                     => Priority,
     Min_Priority                     => Priority,
     Max_Interrupt_Priority           => Interrupt_Priority,
     Min_Interrupt_Priority           => Interrupt_Priority,
     Worst_Context_Switch             => Normalized_Execution_Time,
     Avg_Context_Switch               => Normalized_Execution_Time,
     Best_Context_Switch              => Normalized_Execution_Time,
     Worst_ISR_Switch                 => Normalized_Execution_Time,
     Avg_ISR_Switch                   => Normalized_Execution_Time,
     Best_ISR_Switch                  => Normalized_Execution_Time,
     System_Timer                     => System_Timer,
     Speed_Factor                     => Float);
```

- *Fixed Priority Network*. It represents a network that uses a priority based protocol for sending messages. There are networks that support priorities in their standard protocols (i.e., the CAN bus), and other networks that need an additional protocol that works on top of the standard ones (i.e., serial lines, ethernet). It has the following additional attributes:
  - *Packet Overhead* (Worst, Average, Best). This is the overhead associated to sending each packet, because of the protocol messages that need to be sent before or after each packet.
  - *Transmission kind*: *Simplex*, *Half Duplex*, of *Full Duplex*

- *Max Blocking*. The maximum blocking is caused by the non preemptability of message packets. It usually has the same value as the maximum packet transmission time, but its default value is zero, for the case in which the network overhead is negligible.

- *Max Packet Transmission Time* and *Min Packet Transmission Time.* The maximum time is used in the calculation of the overhead model of the network; the overhead is the packet overhead times the number of packets, which is calculated as the message transmission time divided by the maximum packet transmission time. The Minimum time represents the shortest period of the overheads associated to the transmission of each packet, and thus has a strong impact on the overhead caused by the network drivers in the processors using the network.

- *List of Drivers*. A list of references to network drivers, that contain the processor overhead model associated with the transmission of messages through the network. See the description of the drivers below.

```
Processing_Resource (
     Type                              => Fixed_Priority_Network,
     Name                              => Identifier,
     Max_Priority                      => Priority,
     Min_Priority                      => Priority,
     Packet_Worst_Overhead             => Normalized_Execution_Time,
     Packet_Avg_Overhead               => Normalized_Execution_Time,
     Packet_Best_Overhead              => Normalized_Execution_Time,
     Transmission                      => Simplex | Half_Duplex | Full_Duplex,
     Max_Blocking                      => Time,
     Max_Packet_Transmission_Time      => Time,
     Min_Packet_Transmission_Time      => Time,
     Speed_Factor                      => Float,
     List_of_Drivers                   => (
                                          Driver 1,
                                          Driver 2,
                                          ...));
```

7.2 System Timers

They represent the different overhead models associated with the way the system handles timed events. There are two classes:

- *Alarm Clock*. This represents systems in which timed events are activated by a hardware timer interrupt. The timer is programmed always to generate the interrupt at the time of the closest timed event. Consequently, each one can have its own interrupt. This represents an overhead. The attributes are:

  - *Overhead* (worst, average and best). This is the overhead of the timer interrupt, which is assumed to execute at the highest interrupt priority.

```
System_Timer = (
      Type                              => Alarm Clock
      Worst_Overhead                    => Normalized_Execution_Time,
      Avg_Overhead                      => Normalized_Execution_Time,
      Best_Overhead                     => Normalized_Execution_Time,
```

- *Ticker*. This represents a system that has a periodic ticker, i.e., a periodic interrupt that arrives at the system. When this interrupt arrives, all timed events whose expiration time has already passed, are activated. Other non timed events are handled at the time they are generated. In this model, the overhead by the timer interrupt is localized in a single periodic interrupt, but jitter is introduced in all timed events, because the best resolution is the ticker period. The attributes are:
    - *Overhead* (worst, average and best). This is the overhead of the timer interrupt, which is assumed to execute at the highest interrupt priority.
    - *Period*. Period of the ticker interrupt.

```
System_Timer = (
      Type                              => Ticker
      Worst_Overhead                    => Time,
      Avg_Overhead                      => Time,
      Best_Overhead                     => Time,
      Period                            => Time)
```

7.3 Network Drivers

They represent operations executed in a processor as a consequence of the transmission or reception of a message or a message packet through a network. We define two classes:

- *Packet Driver*. Represents a driver that is activated at each message transmission or reception. Its attributes are:
    - *Packet server*: The scheduling server that is executing the driver (which in turn has a reference to the processor, and the scheduling parameters)
    - *Packet Send Operation*. The operation that is executed each time a packet is sent.
    - *Packet Receive Operation*. The operation that is executed each time a packet is received.

```
Driver = (
      Type                              => Packet_Driver,
      Packet_Server                     => Scheduling_Server,
      Packet_Send_Operation             => Operation,
      Packet_Receive_Operation          => Operation)
```

- *Character Packet Driver*. It is a specialization of a packet driver in which there is an additional overhead associated to sending each character, as happens in some serial lines. Its attributes are those of a packet driver plus the following:
    - *Character server*: The scheduling server that is executing the part of the driver that is executed for each character sent or received (which in turn has a reference to the processor, and the scheduling parameters)

- *Character Send Operation*. The operation that is executed each time a character is sent.
- *Character Receive Operation*. The operation that is executed each time a character is received.
- *Character Transmission Time*. Time of character transmission.

```
Driver = (
     Type                               => Character_Packet_Driver,
     Packet_Server                      => Scheduling_Server,
     Packet_Send_Operation              => Operation,
     Packet_Receive_Operation           => Operation,
     Character_Server                   => Scheduling_Server,
     Character_Send_Operation           => Operation,
     Character_Receive_Operation        => Operation,
     Character_Transmission_Time        => Time)
```

7.4 Scheduling parameters

They represent the fixed priority scheduling policies and their associated parameters. The common attributes are:

- *Priority*. A natural number that represents the scheduling priority. It must be within the valid ranges for the scheduling parameters object.
- *Preassigned*. If this parameter is set to the value "*No*", the priority may be assigned by one of the priority assignment tools. Otherwise, the priority is fixed and cannot be changed by those tools. Its default value is "*No*" if no priority field appears, and "*Yes*" if a priority field appears.

The classes defined are:

- *Non Preemptible Fixed Priority Scheduler*. No additional attributes.

```
Fixed_Priority_Sched_Parameters = (
     Type                               => Non_Preemtible_FP_Policy,
     The_Priority                       => Priority,
     Preassigned                        => Yes | No)
```

- *Fixed Priority Scheduler*. Represents a fixed priority preemptive scheduler. No additional attributes.

```
Fixed_Priority_Sched_Parameters = (
     Type                               => Fixed_Priority_Policy,
     The_Priority                       => Priority,
     Preassigned                        => Yes | No)
```

- *Interrupt Fixed Priority Scheduler*. Represents an interrupt service routine. No additional attributes. The "*Preassigned*" field cannot be set to "*No*", because interrupt priorities are always preassigned.

```
Fixed_Priority_Sched_Parameters = (
     Type                               => Interrupt_FP_Policy,
     The_Priority                     => Interrupt_Priority,
     Preassigned                      => Yes | No)
```

- *Polling Scheduler*. Represents a periodic task that polls for the arrival of its input event. Thus, execution of the event may be delayed until the next period. Its additional attributes are:
    - *Polling Period*. Period of the polling task
    - *Polling Overhead* (Worst, Average, Best). Overhead of the polling task.

```
Fixed_Priority_Sched_Parameters = (
    Type                             => Polling_Policy,
    The_Priority                     => Priority,
    Preassigned                      => Yes | No,
    Polling_Period                   => Time,
    Polling_Worst_Overhead           => Normalized_Execution_Time,
    Polling_Avg_Overhead             => Normalized_Execution_Time,
    Polling_Best_Overhead            => Normalized_Execution_Time)
```

- *Sporadic Server Scheduler*. Represents a task scheduled under the sporadic server scheduling algorithm. Its additional attributes are:
    - *Background Priority*. Represents the priority at which the task executes when there is no available execution capacity
    - *Initial Capacity*. Its the initial value of the execution capacity.
    - *Replenishment Period*. It is the period after which a portion of consumed execution capacity is replenished.
    - *Max Pending replenishments*. It is the maximum number of simultaneously pending replenishment operations.

```
Fixed_Priority_Sched_Parameters = (
    Type                             => Sporadic_Server_Policy,
    Normal_Priority                  => Priority,
    Preassigned                      => Yes | No,
    Background_Priority              => Priority,
    Initial_Capacity                 => Time,
    Replenishment_Period             => Time,
    Max_Pending_Replenishments       => Positive)
```

The scheduling parameters may also be overridden on the operations definition.

```
Overridden_Sched_Parameters = (
    Type                             => Overridden_Fixed_Priority,
    The_Priority                     => Any_Priority)

Overridden_Sched_Parameters = (
    Type                             => Overridden_Permanent_FP,
    The_Priority                     => Any_Priority)
```

7.5 Scheduling Servers

They represent schedulable entities in a processing resource. There is only one class defined, named *Regular*. Its attributes are:

- *Name*
- *Scheduling Parameters*. Reference to the scheduling parameters
- *Processing Resource*. Reference to the scheduling resource

```
Scheduling_Server (
     Type                                  => Fixed_Priority,
     Name                                  => Identifier,
     Server_Sched_Parameters               => Fixed_Priority_Sched_Parameters,
     Server_Processing_Resource            => Identifier);
```

7.6 Shared Resources

They represent resources that are shared among different tasks, and that must be used in a mutually exclusive way. Only protocols that avoid unbounded priority inversion are allowed. There are two classes, depending on the protocol:

- *Immediate Ceiling Resource*. Uses the immediate priority ceiling resource protocol. This is equivalent to Ada's *Priority Ceiling*, or the POSIX *priority protect* protocol. Its attributes are:
  - *Name*.
  - *Ceiling*. Priority ceiling used for the resource. May be computed automatically by the tool, upon request.
  - *Preassigned*. If this parameter is set to the value "*No*", the priority ceiling may be assigned by the "Calculate Ceilings" tool. Otherwise, the priority ceiling is fixed and cannot be changed by those tools. Its default value is "*No*" if no ceiling field appears, and "*Yes*" if a ceiling field appears.

```
Shared_Resource (
     Type                                  => Immediate_Ceiling_Resource,
     Name                                  => Identifier,
     Ceiling                               => Any_Priority,
     Preassigned                           => Yes | No);
```

- *Priority Inheritance Resource*. Uses the basic priority inheritance protocol. Its attributes are:
  - *Name*.

```
Shared_Resource (
     Type                                  => Priority_Inheritance_Resource,
     Name                                  => Identifier);
```

7.7 Operations

They represent a piece of code, or a message. They all have the following common attributes:

- *Execution Time* (Worst, Average and Best). In normalized units. For messages, this represents the transmission time.
- *Overridden Scheduling Parameters*. Represents a priority level above the normal priority level that at which the operation would execute:
    - For a regular overridden priority (*Overridden_Fixed_Priority*), the change of priority is in effect only until the operation is completed.
    - For a permanent overridden priority (*Overridden_Permanent_FP*), the change of priority is in effect until another permanent overridden priority, or until the end of the segment of activities, i.e., a set of consecutive activities (consecutive in the transaction graph) executed by the same scheduling server.

The following classes of operations are defined:

- *Simple*. Represents a simple piece of code or message. Additional attributes are:
    - *Shared resources to lock*. List of references to the shared resources that must be locked before executing the operation
    - *Shared resources to unlock*. List of references to the shared resources that must be unlocked after executing the operation
    - *Shared resources list*.

```
Operation (
     Type                              => Simple,
     Name                              => Identifier,
     Overridden_Sched_Parameters       => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time         => Normalized_Execution_Time,
     Avg_Case_Execution_Time           => Normalized_Execution_Time,
     Best_Case_Execution_Time          => Normalized_Execution_Time,
     Shared_Resources_To_Lock          => (
                                       Identifier,
                                       Identifier,
                                       ...),
     Shared_Resources_To_Unlock        => (
                                       Identifier,
                                       Identifier,
                                       ...));

Operation (
     Type                              => Simple,
     Name                              => Identifier,
     Overridden_Sched_Parameters       => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time         => Normalized_Execution_Time,
     Avg_Case_Execution_Time           => Normalized_Execution_Time,
     Best_Case_Execution_Time          => Normalized_Execution_Time,
     Shared_Resources_List             => (
                                       Identifier,
                                       Identifier,
                                       ...));
```

- *Composite*. Represents an operation composed of an ordered sequence of other operations, simple or composite. The execution time attribute of this class cannot be set, because it is the sum of the execution times of the comprised operations. Its additional attributes are:
    - *Operation List*: List of references to other operations

```
Operation (
     Type                            => Composite,
     Name                            => Identifier,
     Overridden_Sched_Parameters     => Overridden_Sched_Parameters,
     Composite_Operation_List        => (
                                     Identifier,
                                     Identifier,
                                     ...));
```

- *Enclosing*. Represents an operation that contains other operations as part of its execution. The execution time is not the sum of execution times of the comprised operations, because other pieces of code may be executed in addition. The enclosed operations need to be considered for the purpose of calculating the blocking times associated with their shared resource usage. Its additional attributes are:
    - *Operation List*: List of references to other operations

```
Operation (
     Type                            => Enclosing,
     Name                            => Identifier,
     Overridden_Sched_Parameters     => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time       => Normalized_Execution_Time,
     Avg_Case_Execution_Time         => Normalized_Execution_Time,
     Best_Case_Execution_Time        => Normalized_Execution_Time,
     Composite_Operation_List        => (
                                     Identifier,
                                     Identifier,
                                     ...));
```

7.8 Events

Events may be internal or external, and represent channels of event streams, through which individual event instances may be generated. An event instance activates an instance of an activity, or influences the behavior of the event handler to which it is directed.

- *Internal events*. They are generated by an event handler. Their attributes are:
    - *Name*.
    - *Timing Requirements*. Reference to the timing requirements imposed on the generation of the event. See the description of timing requirements below

```
Internal_Event = (
     Type                            => Regular,
     Event                              => Identifier)
     Timing_Requirements                => Timing_Requirement)
```

For the external events, the following classes are defined:

- *Periodic*. Represents a stream of events that are generated periodically. They have the following attributes:
  - *Name*.
  - *Period*. Event period.
  - *Max Jitter*. The event jitter is an amount of time that may be added to the activation time of each event instance, and is bounded by the maximum jitter attribute. It influences the schedulability of the system.
  - *Phase*. It is the instant of the first activation, if it had no jitter. After that time, the following events are periodic (possibly with jitter).

```
External_Event = (
     Type                               => Periodic,
     Name                               => Identifier,
     Period                             => Time,
     Max_Jitter                         => Maximum jitter of Periodic event,
     Phase                              => Absolute_Time);
```

- *Singular*. Represents an event that is generated only once. It has the following attributes:
  - *Name*.
  - *Phase*. It is the instant of the first activation.

```
External_Event = (
     Type                               => Singular,
     Name                               => Identifier,
     Phase                              => Absolute_Time);
```

- *Sporadic*. Represents a stream of aperiodic events that have a minimum interarrival time. They have the following attributes:
  - *Name*.
  - *Min Interarrival*. Minimum time between the generation of two events.
  - *Average Interarrival*. Average interarrival time
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* or *Poisson*.

```
External_Event = (
     Type                               => Sporadic,
     Name                               => Identifier,
     Avg_Interarrival                   => Time,
     Distribution                       => Uniform|Poisson,
     Min_Interarrival                   => Time);
```

- *Unbounded*. Represents a stream of aperiodic events for which it is not possible to establish an upper bound on the number of events that may arrive in a given interval. They have the following attributes:
  - *Name*.
  - *Average Interarrival*. Average interarrival time
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* or *Poisson*.

```
External_Event = (
     Type                                => Unbounded,
     Name                                => Identifier,
     Avg_Interarrival                    => Time,
     Distribution                        => Uniform|Poisson);
```

- *Bursty*. Represents a stream of aperiodic events that have an upper bound on the number of events that may arrive in a given interval. Within this interval, events may arrive with an arbitrarily low distance among them (perhaps as a burst of events). They have the following attributes:
  - *Name*.
  - *Bound_Interval*. Interval for which the amount of event arrivals is bounded
  - *Max_Arrivals*. Maximum number of events that may arrive in the *Bound_Interval*.
  - *Average Interarrival*. Average interarrival time.
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* or *Poisson*.

```
External_Event = (
     Type                                => Bursty,
     Name                                => Identifier,
     Avg_Interarrival                    => Time,
     Distribution                        => Uniform|Poisson,
     Bound_Interval                      => Time,
     Max_Arrivals                        => Positive);
```

7.9 Timing Requirements

They represent requirements imposed on the instant of generation of the associated internal event. There are different kinds of requirements:

- *Deadlines*. They represent a maximum time value allowed for the generation of the associated event. They are expressed as a relative time interval that is counted in two different ways:
  - *Local Deadlines*: they appear only associated with the output event of an activity; the deadline is relative to the arrival of the event that activated that activity.
  - *Global deadlines*: the deadline is relative to the arrival of a *Referenced Event*, that is an attribute of the deadline.

In addition, deadlines may be hard or soft:

- *Hard Deadlines*: they must be met in all cases, including the worst case
- *Soft Deadlines*: they must be met on average.

This gives way to four kinds of deadlines:

- *Hard Global Deadline*. Attributes are the value of the *Deadline*, and a reference to the *Referenced Event.*
- *Soft Global Deadline*. Attributes are the value of the *Deadline,* and a reference to the *Referenced Event.*
- *Hard Local Deadline*. The only attribute is the value of the *Deadline*.
- *Soft Local Deadline*. The only attribute is the value of the *Deadline.*

```
Timing_Requirement = (
     Type                              => Hard_Global_Deadline,
     Deadline                          => Time,
     Referenced_Event                  => Identifier)

Timing_Requirement = (
     Type                              => Hard_Local_Deadline,
     Deadline                          => Time)

Timing_Requirement = (
     Type                              => Soft_Global_Deadline,
     Deadline                          => Time,
     Referenced_Event                  => Identifier)

Timing_Requirement = (
     Type                              => Soft_Local_Deadline,
     Deadline                          => Time)
```

- *Max Output Jitter Requirement*: Represents a requirement for limiting the jitter with which a periodic internal event is generated. Output jitter is calculated as the difference between the worst-case response time and the best-case response time for the associated event, relative to a *Referenced Event* that is an attribute of this requirement. Consequently, the attributes are:
  - *Max Output Jitter*. Time value.
  - *Referenced Event*. Reference to an event.

```
Timing_Requirement = (
     Type                              => Max_Output_Jitter_Req,
     Max_Output_Jitter                 => Time,
     Referenced_Event                  => Identifier)
```

- *Max Miss Ratio*: Represents a kind of soft deadline in which the deadline cannot be missed more often than a specified ratio. Its attributes are
  - Deadline. Time Value
  - Ratio. Percentage representing the maximum ratio of missed deadlines

There are two kinds of Max Miss Ratio requirements: global or local:

- *Local Max Miss Ratio*. The deadline is relative to the activation of the activity to which the timing requirement is attached. It has no additional attributes.
- *Global Max Miss Ratio*. The deadline is relative to a *Referenced Event*, which is an additional attribute of this class.

```
Timing_Requirement = (
     Type                              => Global_Max_Miss_Ratio,
     Deadline                          => Time,
     Ratio                             => Percentage,
     Referenced_Event                  => Identifier)
```

```
Timing_Requirement = (
     Type                              => Local_Max_Miss_Ratio,
     Deadline                          => Time,
     Ratio                             => Percentage)
```

- *Composite*: An event may have several timing requirements imposed at the same time, which are expressed via a composite timing requirement. It is just a list of simple timing requirements.

```
Timing_Requirement = (
     Type                              => Composite,
     Requirements_List                 => (
                                       Timing_Requirement 1,
                                       Timing_Requirement 2,
                                       ...))
```

7.10 Event Handlers

Event handlers represent actions that are activated by the arrival of one or more events, and that in turn generate one or more events at their output. There are two fundamental classes of event handlers. The *Activities* represent the execution of an operation by a scheduling server, in a processing resource, and with some given scheduling parameters. The other operations are just a mechanism for handling events, with no runtime effects. Any overhead associated with their implementation is charged to the associated activities. Figure 5 shows the different classes of events.

- *Activity*. It represents an instance of an operation, to be executed by a scheduling server. Its attributes are:
    - *Input event*. Reference to the event
    - *Output event*. Reference to the event
    - *Activity Operation*. Reference to the operation
    - *Activity server*. Reference to the scheduling server (which in turn contains references to the scheduling parameters and the processing resource).

```
Event_Handler = (
     Type                              => Activity,
```

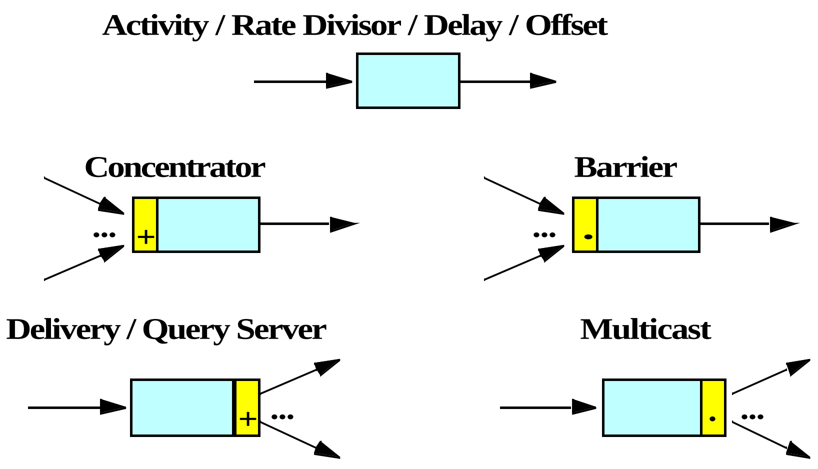


Figure 5. Classes of Event Handlers

```
        Input_Event                      => Identifier,
        Output_Event                     => Identifier,
        Activity_Operation               => Identifier,
        Activity_Server                  => Identifier)
```

- *System Timed Activity*. It represents an activity that is activated by the system timer, and thus is subject to the overheads associated with it. It only makes sense to have a *System Timed Activity* that is activated from an external event, or an event generated by the *Delay* or *Offset* event handlers (see below). It has the same attributes as the regular activity.

```
    Event_Handler = (
        Type                             => System_Timed_Activity,
        Input_Event                      => Identifier,
        Output_Event                     => Identifier,
        Activity_Operation               => Identifier,
        Activity_Server                  => Identifier)
```

- *Concentrator*. It is an event handler that generates its output event when any one of its input events arrives. Its attributes are:
    - *Input events*. References to the input events
    - *Output event*. Reference to the output event

```
    Event_Handler = (
        Type                             => Concentrator,
        Output_Event                     => Identifier,
        Input_Events_List                => (
                                         Identifier,
                                         Identifier,
                                         ...))
```

- *Barrier*. It is an event handler that generates its output event when all of its input events have arrived. For worst-case analysis to be possible it is necessary that all the input events are periodic with the same periods. This usually represents no problem if the concentrator is used to perform a "*join*" operation after a "*fork*" operation carried out with the *Multicast* event handler (see below). Its attributes are:
  - *Input events*. References to the input events
  - *Output event*. Reference to the output event

```
Event_Handler = (
     Type                                => Barrier,
     Output_Event                        => Identifier,
     Input_Events_List                   => (
                                         Identifier,
                                         Identifier,
                                         ...))
```

- *Delivery Server*. It is an event handler that generates one event in only one of its outputs each time an input event arrives. The output path is chosen at the time of the event generation. Its attributes are:
  - *Input event*. Reference to the input event
  - *Output events*. References to the output events
  - *Delivery Policy*. Is the policy used to determine the output path. It may be *Scan* (the output path is chosen in a cyclic fashion) or *Random.*

```
Event_Handler = (
     Type                                => Delivery_Server,
     Delivery_Policy                     => Scan|Random,
     Input_Event                         => Identifier,
     Output_Events_List                  => (
                                         Identifier,
                                         Identifier,
                                         ...))
```

- *Query Server*. It is an event handler that generates one event in only one of its outputs each time an input event arrives. The output path is chosen at the time of the event consumption by one of the activities connected to an output event. Its attributes are:
  - *Input event*. Reference to the input event
  - *Output events*. References to the output events
  - *Request Policy*. Is the policy used to determine the output path when there are several pending requests from the connected activities. It may be *Scan* (the output path is chosen in a cyclic fashion), *Priority* (the highest priority activity wins), *FIFO* or *LIFO.*

```
Event_Handler = (
     Type                                => Query_Server,
     Request_Policy                      => Priority|FIFO|LIFO|Scan,
     Input_Event                         => Identifier,
    Output_Events_List                 => (
                                       Identifier,
                                       Identifier,
                                       ...))
```

- *Multicast*. It is an event handler that generates one event in every one of its outputs each time an input event arrives. Its attributes are:
  - *Input event*. Reference to the input event
  - *Output events*. References to the output events

```
Event_Handler = (
    Type                               => Multicast,
    Input_Event                        => Identifier,
    Output_Events_List                 => (
                                       Identifier,
                                       Identifier,
                                       ...))
```

- *Rate Divisor*. It is an event handler that generates one output event when a number of input events equal to the *Rate Factor* have arrived. Its attributes are:
  - *Input event*. Reference to the input event
  - *Output event*. Reference to the output event
  - *Rate Factor*. Number of events that must arrive to generate an output event

```
Event_Handler = (
    Type                               => Rate_Divisor,
    Input_Event                        => Identifier,
    Output_Event                       => Identifier,
    Rate_Factor                        => Positive)
```

- *Delay*. It is an event handler that generates its output event after a time interval has elapsed from the arrival of the input event. Its attributes are:
  - *Input event*. Reference to the input event
  - *Output event*. Reference to the output event
  - *Delay Max Interval*. Longest time interval used to generate the output event
  - *Delay Min Interval*. Shortest time interval used to generate the output event

```
Event_Handler = (
    Type                               => Delay,
    Input_Event                        => Identifier,
    Output_Event                       => Identifier,
    Delay_Max_Interval                 => Time,
    Delay_Min_Interval                 => Time)
```

- *Offset*. It is similar to the *Delay* event handler, except that the time interval is counted relative to the arrival of some (previous) event. If the time interval has already passed when the input event arrives, the output event is generated immediately. Its attributes are the same as for the *Delay* event handler, plus the following:

- *Referenced Event*: Reference to the appropriate event.

```
Event_Handler = (
    Type                              => Offset,
    Input_Event                       => Identifier,
    Output_Event                      => Identifier,
    Delay_Max_Interval                => Time,
    Delay_Min_Interval                => Time,
    Referenced_Event                  => Identifier)
```

7.11 Transactions

The transaction is a graph of event handlers and events, that represents activities executed in the system which are interrelated. A transaction is defined with three different components that have already been described:

- A list of external events
- A list of internal events, with their timing requirements if any
- A list of Event handlers

In addition, each transaction has a *Name* attribute. There is only one class of transaction defined, called a *Regular* transaction.

```
Transaction (
    Type                              => Regular,
    Name                              => Identifier,
    External_Events                   => (
                                         External_Event 1,
                                         External_Event 2,
                                         ...),
    Internal_Events                   => (
                                         Internal_Event 1,
                                         Internal_Event 2,
                                         ...),
    Event_Handlers                    => (
                                         Event_Handler 1,
                                         Event_Handler 2,
                                         ...));
```

7.12 Overall Model

A Real-Time situation represents the overall MAST model of a real-time situation that a particular system may have, and that needs to be analyzed. Global information about the real-time situation is described in the Model object, which contains the following attributes:

- Model name: a string
- Model date: the date in which the real-time situation model was created.

```
Model (
    Model_Name                        => Identifier,
     Model_Date                        => YYYY-MM-DDThh:mm:ss);
```

8. Templates for the MAST File

```
-- Real-Time System Model

-- File format

-- This line is just an example of a comment

Model(
     Model_Name                        => Identifier,
     Model_Date                        => YYYY-MM-DDThh:mm:ss);

-- Resources

Processing_Resource (
     Type                              => Fixed_Priority_Processor,
     Name                              => Name of the processing resource,
     Max_Priority                      => Task Priority,
     Min_Priority                      => Task Priority,
     Max_Interrupt_Priority            => Interrupt Priority,
     Min_Interrupt_Priority            => Interrupt Priority,
     Worst_Context_Switch              => WCS Time for Processors,
     Avg_Context_Switch                => ACS Time for Processors,
     Best_Context_Switch               => BCS Time for Processors,
     Worst_ISR_Switch                  => WISR Time for Processors,
     Avg_ISR_Switch                    => AISR Time for Processors,
     Best_ISR_Switch                   => BISR Time for Processors,
     System_Timer                      => System_Timer,
     Speed_Factor                      => Float);
         -- real execution times = normalized execution times/Speed_Factor;
         -- Ticker Overhead is real execution time

Processing_Resource (
     Type                              => Fixed_Priority_Network,
     Name                              => Name of the processing resource,
     Max_Priority                      => Message Priority,
     Min_Priority                      => Message Priority,
     Packet_Worst_Overhead             => PWO for Networks,
     Packet_Avg_Overhead               => PAO for Networks,
     Packet_Best_Overhead              => PBO for Networks,
     Transmission                      => Simplex | Half_Duplex | Full_Duplex,
     Max_Packet_Transmission_Time      => Max Packet transmission time,
     Min_Packet_Transmission_Time      => Min Packet transmission time,
     Speed_Factor                      => Float,
     List_of_Drivers                   => (
                                       Driver 1,
                                       Driver 2,
                                       ...));
     -- Overheads are normalized execution times.
     -- Real execution times = normalized_execution_time/processor speed
     -- Packet_Transmission_Time is the real transmission time

-- System Timers

System_Timer = (
     Type                             => Ticker
     Worst_Overhead                   => Worst Overhead of ticker,
     Avg_Overhead                     => Avg Overhead of ticker,
     Best_Overhead                    => Best Overhead of ticker,
     Period                           => Period of ticker for Processors)

System_Timer = (
     Type                             => Alarm Clock
     Worst_Overhead                   => Worst Overhead of timer,
     Avg_Overhead                     => Avg Overhead of timer,
     Best_Overhead                    => Best Overhead of timer,

-- Drivers

Driver = (
     Type                             => Packet_Driver,
     Packet_Server                    => Scheduling_Server,
     Packet_Send_Operation            => Simple Operation,
     Packet_Receive_Operation         => Simple Operation)
     -- The scheduling server and the operations are embedded in the
     -- description, with all their attributes, but without the keywords
     -- "Scheduling_Server" or "Operation"

Driver = (
     Type                             => Character_Packet_Driver,
     Packet_Server                    => Scheduling_Server,
     Packet_Send_Operation            => Simple Operation,
     Packet_Receive_Operation         => Simple Operation,
     Character_Server                 => Scheduling_Server,
     Character_Send_Operation         => Simple Operation,
     Character_Receive_Operation      => Simple Operation,
     Character_Transmission_Time      => Transmission Time)
     -- The scheduling server and the operations are embedded in the
     -- description, with all their attributes, but without the keywords
     -- "Scheduling_Server" or "Operation"

-- Shared Resources

Shared_Resource (
     Type                             => Immediate_Ceiling_Resource,
     Name                             => Name of the data resource,
     Ceiling                          => Ceiling of resource, any priority,
     Preassigned                      => No);

Shared_Resource (
     Type                             => Priority_Inheritance_Resource,
     Name                               => Name of the data resource);

-- Operations

Operation (
     Type                               => Simple,
     Name                               => Name of the operation,
     Overridden_Sched_Parameters        => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time          => WCET,
     Avg_Case_Execution_Time            => ACET,
     Best_Case_Execution_Time           => BCET,
     Shared_Resources_To_Lock           => (
                                        Shared Resource Name 1,
                                        Shared Resource Name 2,
                                        ...),
     Shared_Resources_To_Unlock         => (
                                        Shared Resource Name 1,
                                        Shared Resource Name 2,
                                        ...));
     -- The resources specified under Shared_Resources_To_Lock are locked
     -- before the operation starts, in the order defined.
     -- The resources specified under Shared_Resources_To_Unlock are unlocked
     -- after the operation completes, in the order defined.
     -- WCET, ACET and BCET are normalized execution times.
     -- Real execution times = normalized_execution_time/speed factor

Operation (
     Type                               => Simple,
     Name                               => Name of the operation,
     Overridden_Sched_Parameters        => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time          => WCET,
     Avg_Case_Execution_Time            => ACET,
     Best_Case_Execution_Time           => BCET,
     Shared_Resources_List              => (
                                        Shared Resource Name 1,
                                        Shared Resource Name 2,
                                        ...));
     -- This is an alternative way to declare a simple object. The resources
     -- specified under Shared_Resources_List are locked before the operation
     -- starts, in the order defined, and are unlocked when the operation
     -- finishes, in the reverse order.
     -- WCET, ACET and BCET are normalized execution times.
     -- Real execution times = normalized_execution_time/speed factor

Operation (
     Type                               => Composite,
     Name                               => Name of the operation,
     Overridden_Sched_Parameters        => Overridden_Sched_Parameters,
     Composite_Operation_List           => (
                                        Operation Name 1,
                                        Operation Name 2,
                                        ...));

Operation (
     Type                               => Enclosing,
     Name                               => Name of the operation,
     Overridden_Sched_Parameters        => Overridden_Sched_Parameters,
     Worst_Case_Execution_Time          => WCET,
     Avg_Case_Execution_Time            => ACET,
     Best_Case_Execution_Time           => BCET,
     Composite_Operation_List           => (
                                        Operation Name 1,
                                        Operation Name 2,
                                        ...));
     -- WCET, ACET and BCET are normalized execution times.
     -- Real execution times = normalized_execution_time/speed factor

-- Scheduling Servers

Scheduling_Server (
     Type                               => Fixed_Priority,
     Name                               => Name of the server,
     Server_Sched_Parameters            => Fixed_Priority_Sched_Parameters,
     Server_Processing_Resource         => Name of the Processing Resource);

-- Transactions

Transaction (
     Type                               => Regular,
     Name                               => Name of the transaction,
     External_Events                    => (
                                        External_Event 1,
                                        External_Event 2,
                                        ...),
     Internal_Events                    => (
                                        Internal_Event 1,
                                        Internal_Event 2,
                                        ...),
     Event_Handlers                     => (
                                        Event_Handler 1,
                                        Event_Handler 2,
                                        ...));

-- External Events

External_Event = (
     Type                               => Periodic,
     Name                               => Name of the event,
     Period                             => Period of the Periodic event,
     Max_Jitter                         => Maximum jitter of Periodic event,
     Phase                              => Phase of Periodic event);
        -- The Phase represents the absolute start time of the first period,
        --  i.e., the first event generation time if Max_Jitter=0
```

```
External_Event = (
     Type                              => Singular,
     Name                              => Name of the event,
     Phase                             => Phase of Periodic event);
         -- The Phase represents the absolute time at which the event
         -- is generated

External_Event = (
     Type                              => Sporadic,
     Name                              => Name of the event,
     Avg_Interarrival                  => Average interarrival time,
     Distribution                      => Uniform|Poisson,
     Min_Interarrival                  => Minimum interarrival time);

External_Event = (
     Type                              => Unbounded,
     Name                              => Name of the event,
     Avg_Interarrival                  => Average interarrival time,
     Distribution                      => Uniform|Poisson);

External_Event = (
     Type                              => Bursty,
     Name                              => Name of the event,
     Avg_Interarrival                  => Average interarrival time,
     Distribution                      => Uniform|Poisson,
     Bound_Interval                    => Interval of Bursty events,
     Max_Arrivals                      => Maximum number of arrivals);

-- Timing requirements

Timing_Requirement = (
     Type                              => Hard_Global_Deadline,
     Deadline                          => Deadline,
     Referenced_Event                  => Name of Event)

Timing_Requirement = (
     Type                              => Hard_Local_Deadline,
     Deadline                          => Deadline)

Timing_Requirement = (
     Type                              => Soft_Global_Deadline,
     Deadline                          => Deadline,
     Referenced_Event                  => Name of Event)

Timing_Requirement = (
     Type                              => Soft_Local_Deadline,
     Deadline                          => Deadline)

Timing_Requirement = (
     Type                              => Global_Max_Miss_Ratio,
     Deadline                          => Deadline,
     Ratio                           => Percentage,
     Referenced_Event                => Name of Event)

Timing_Requirement = (
     Type                            => Local_Max_Miss_Ratio,
     Deadline                        => Deadline,
     Ratio                           => Percentage)

Timing_Requirement = (
     Type                            => Max_Output_Jitter_Req,
     Max_Output_Jitter               => Maximum output jitter,
     Referenced_Event                => Name of Event)

Timing_Requirement = (
     Type                            => Composite,
     Requirements_List               => (
                                     Timing_Requirement 1,
                                     Timing_Requirement 2,
                                     ...))

-- Scheduling Parameters

Fixed_Priority_Sched_Parameters = (
     Type                            => Non_Preemtible_FP_Policy,
     The_Priority                    => Priority,
     Preassigned                     => Yes | No)

Fixed_Priority_Sched_Parameters = (
     Type                            => Fixed_Priority_Policy,
     The_Priority                    => Priority,
     Preassigned                     => Yes | No)

Fixed_Priority_Sched_Parameters = (
     Type                            => Interrupt_FP_Policy,
     The_Priority                    => Interrupt Priority,
     Preassigned                     => Yes)

Fixed_Priority_Sched_Parameters = (
     Type                            => Polling_Policy,
     The_Priority                    => Priority,
     Preassigned                     => Yes | No,
     Polling_Period                  => Period of polling
     Polling_Worst_Overhead          => Worst overhead of polling
     Polling_Avg_Overhead            => Average overhead of polling
     Polling_Best_Overhead           => Best overhead of polling)
       -- Polling overheads are relative execution times

Fixed_Priority_Sched_Parameters = (
     Type                            => Sporadic_Server_Policy,
     Normal_Priority                 => Priority,
     Preassigned                     => Yes | No,
     Background_Priority               => Background priority,
     Initial_Capacity                  => Initial Capacity,
     Replenishment_Period              => Replenishment period,
     Max_Pending_Replenishments        => Maximum of pending replenishment)

Overridden_Sched_Parameters = (
     Type                              => Overridden_Fixed_Priority,
     The_Priority                      => Priority)

Overridden_Sched_Parameters = (
     Type                              => Overridden_Permanent_FP,
     The_Priority                      => Priority)

-- Internal Events

Internal_Event = (
     Type                              => Regular,
     Event                             => Name of the event)
     Timing_Requirements               => Timing_Requirement)

   -- Note: Events can be internal or external. External events are declared
   --       as described before.
   --       Internal events are declared as part of the transaction.
   --      Each event can only be referenced by one event handler as an input
   --       event, and by one event handler as an output event

-- Event Handlers

Event_Handler = (
     Type                              => Activity,
     Input_Event                       => Name of the Event,
     Output_Event                      => Name of the Event,
     Activity_Operation                => Name of the operation,
     Activity_Server                   => Name of the scheduling server)

Event_Handler = (
     Type                              => System_Timed_Activity,
     Input_Event                       => Name of the Event,
     Output_Event                      => Name of the Event,
     Activity_Operation                => Name of the operation,
     Activity_Server                   => Name of the scheduling server)

Event_Handler = (
     Type                              => Concentrator,
     Output_Event                      => Name of the Event,
     Input_Events_List                 => (
                                       Name of the Event 1,
                                       Name of the Event 2,
                                       ...))

Event_Handler = (
     Type                              => Barrier,
      Output_Event                     => Name of the Event,
      Input_Events_List                => (
                                       Name of the Event 1,
                                       Name of the Event 2,
                                       ...))

Event_Handler = (
      Type                             => Delivery_Server,
      Delivery_Policy                  => Scan|Random,
      Input_Event                      => Name of the Event,
      Output_Events_List               => (
                                       Name of the Event 1,
                                       Name of the Event 2,
                                       ...))

Event_Handler = (
      Type                             => Query_Server,
      Request_Policy                   => Priority|FIFO|LIFO|Scan,
      Input_Event                      => Name of the Event,
      Output_Events_List               => (
                                       Name of the Event 1,
                                       Name of the Event 2,
                                       ...))

Event_Handler = (
      Type                             => Multicast,
      Input_Event                      => Name of the Event,
      Output_Events_List               => (
                                       Name of the Event 1,
                                       Name of the Event 2,
                                       ...))

Event_Handler = (
      Type                             => Rate_Divisor,
      Input_Event                      => Name of the Event,
      Output_Event                     => Name of the Event,
      Rate_Factor                      => Factor of Rate Divisor)

Event_Handler = (
      Type                             => Delay,
      Input_Event                      => Name of the Event,
      Output_Event                     => Name of the Event,
      Delay_Max_Interval               => Maximum delay interval,
      Delay_Min_Interval               => Minimum delay interval)

Event_Handler = (
      Type                             => Offset,
      Input_Event                      => Name of the Event,
      Output_Event                     => Name of the Event,
      Delay_Max_Interval               => Maximum delay interval,
      Delay_Min_Interval               => Minimum delay interval,
    Referenced_Event                 => Name of referenced event)
```

9. Results File Format

The results of the analysis are stored in the *results file* and are attached to different elements of the MAST model:

- the overall system:
  - slacks
  - traces
- transactions:
  - timing results: for each output event global response times (worst, best average) and maximum output jitter
  - transaction-specific slack
- processing resources:
  - slack
  - utilization
  - scheduler queue size
- operations:
  - slack
- scheduling servers:
  - priorities
- shared resources:
  - priority ceilings
  - queue size

The format of the results file is described next. The *results file* is in text format and follows the same rules as the MAST model file (see Section 6, “Writing the MAST file”). The *results file* contains objects of the following types, without any particular ordering imposed:

9.1 Real-Time Situation

The overall system results are relative to a real-time situation that has been analyzed, and contain a set of results (described below) and the following attributes:

- *Model_Name*: Name of the analyzed real-time situation model.
- *Model_Date*: Date of last modification of the analyses real-time situation model, in the ISO 8601 format `YYYY-MM-DDThh:mm:ss`.
- *Generation_Tool*: Quoted text representing the name of the tool that generated the results.

- *Generation_Profile*: Quoted text representing the command and options used to invoke the tool for the generation of the results.
- *Generation_Date*: Date of generation of results, in the ISO 8601 format `YYYY-MM-DDThh:mm:ss`.

```
Real_Time_Situation (
     Model_Name                              => Identifier,
     Model_Date                              => YYYY-MM-DDThh:mm:ss,
     Generator_Tool                          => "Text",
     Generation_Profile                      => "Text",
     Generation_Date                         => YYYY-MM-DDThh:mm:ss,
     Results                                 => (
                                             Result 1,
                                             Result 2,
                                             ...));
```

The specific results that may refer to a real-time situation are:

- *Slack*: If positive, it is the percentage by which all the execution times of all the operations in the real-time situation may be increased while still keeping the system schedulable. If negative, it is the percentage by which all the execution times of all the operations in the real-time situation have to be decreased to make the system schedulable. If zero, it means that the system is just schedulable.

```
Result = (
     Type                                    => Slack,
     Value                                   => Percentage)
```

- *Trace*: It describes the name of a file where trace information on the simulation of a MAST real-time situation can be found.

```
Result = (
     Type                                    => Trace,
     Pathname                                => Pathname)
```

9.2 Transaction

The transaction results are relative to a transaction in the system that has been analyzed, and contain the name of the transaction and a set of results (described below), using the following format:

```
Transaction (
     Name                                    => Identifier,
     Results                                 => (
                                             Result 1,
                                             Result 2,
                                             ...));
```

The specific results that may refer to a real-time situation are:

- *Slack*: If positive, it is the percentage by which all the execution times of all the operations used by the transaction may be increased while still keeping the system schedulable. If negative, it is the percentage by which all the execution times of all the operations used by the transaction have to be decreased to make the system schedulable. If zero, it means that the transaction is just schedulable.

```
Result = (
     Type                              => Slack,
     Value                             => Percentage)
```

- *Timing_Result*: Represents the timing results of a relevant event of the transaction and obtainable by a schedulability analysis tool. Its attributes are:
  - *Event_Name*: Name of event. The timing results always corresponds to the activity or activities that generated the event represented by this name.
  - *Worst_Local_Response_Time*: Worst local response time, measured as the worst difference between the activation and completion times of the activity that generated the event with this result.
  - *Best_Local_Response_Time*: Best local response time, measured as the best difference between the activation and completion times of the activity that generated the event with this result.
  - *Worst_Blocking_Time*: Worst-case delay caused by the used of shared resources. It represents the blocking time for the segment of activities preceding the referenced event. A segment of activities is a set of consecutive activities (consecutive in the transaction graph) that are run by the same scheduling server.
  - *Num_Of_Suspensions*: Maximum number of suspensions caused by shared resources, for the segment of activities preceding the referenced event.
  - *Worst_Global_Response_Times*: List of global response times each representing the worst-case response time relative to a particular input event.
  - *Best_Global_Response_Times*: List of global response times each representing the best-case response time relative to a particular input event.
  - *Jitters*: List of maximum output jitter values, each representing the maximum jitter relative to a particular input event.

```
Result = (
     Type                              => Timing_Result,
     Event_Name                        => Identifier,
     Worst_Local_Response_Time         => Time,
     Best_Local_Response_Time          => Time,
     Worst_Blocking_Time               => Time,
     Num_Of_Suspensions                => Natural,
     Worst_Global_Response_Times       => (
                                       Global_Response_Time 1,
                                       Global_Response_Time 2,
                                       ...),
     Best_Global_Response_Times        => (
                                       Global_Response_Time 1,
                                       Global_Response_Time 2,
                                       ...),
     Jitters                      => (
                                       Global_Response_Time 1,
                                       Global_Response_Time 2,
                                       ...));
```

- *Simulation_Timing_Result*: Represents the timing results of a relevant event of the transaction and obtained by a simulation tool. Its attributes are those of a *Timing_Result* plus the following:
  - *Avg_Local_Response_Time*: Average local response time, measured as the average difference between the activation and completion times of the activity that generated the event with this result.
  - *Avg_Blocking_Time*: Average-case delay caused by the used of shared resources. It represents the average blocking time for the segment of activities preceding the referenced event. A segment of activities is a set of consecutive activities (consecutive in the transaction graph) that are run by the same scheduling server.
  - *Max_Preemption_Time*: Maximum time spent by the activity preceding the event in the scheduler ready queue, while having been activated by a specific event instance. This is equivalent to the time the activity is being preempted by higher priority activities.
  - *Suspension_Time*: Maximum time spent in the activity input queue by the event that triggered the activity preceding the event to which this result is attached. This time is larger than zero only if the triggering event arrives while the activity is still busy processing a previous event.
  - *Num_Of_Queued_Activations*: Maximum number of pending activations in the input queue of the activity preceding the referenced event.
  - *Avg_Global_Response_Times*: List of global response times each representing the average-case response time relative to a particular input event.
  - *Local_Miss_Ratios*: List of local miss ratios, each representing the ratio of events that have missed a specific soft local deadline.
  - *Global_Miss_Ratios*: List of global miss ratios, each representing the ratio of events generated at a specific input event channel, that have missed a specific soft global deadline.

```
Result = (
     Type                         => Simulation_Timing_Result,
     Event_Name                   => Identifier,
     Worst_Local_Response_Time    => Time,
     Avg_Local_Response_Time      => Time,
     Best_Local_Response_Time     => Time,
     Worst_Blocking_Time          => Time,
     Avg_Blocking_Time            => Time,
     Max_Preemption_Time          => Time,
     Suspension_Time              => Time,
     Num_Of_Suspensions           => Natural,
     Num_Of_Queued_Activations    => Natural,
     Worst_Global_Response_Times       => (
                                      Global_Response_Time 1,
                                      Global_Response_Time 2,
                                      ...),
     Avg_Global_Response_Times         => (
                                      Global_Response_Time 1,
                                      Global_Response_Time 2,
                                      ...),
     Best_Global_Response_Times        => (
                                      Global_Response_Time 1,
                                      Global_Response_Time 2,
                                      ...),
     Jitters                           => (
                                      Global_Response_Time 1,
                                      Global_Response_Time 2,
                                      ...),
     Local_Miss_Ratios                 => (
                                      Miss_Ratio 1,
                                      Miss_Ratio 2,
                                      ...),
     Global_Miss_Ratios                => (
                                      Global_Miss_Ratio 1,
                                      Global_Miss_Ratio 2,
                                      ...));
```

A *Global_Response_Time* contains the following attributes:

- *Referenced_Event*: Name of referenced input event, used for calculating the response time.
- *Time_Value*: Global response time, calculated as the difference between the arrival of the input referenced event and the generation of the event to which the result is attached, and adding the input jitter.

```
Global_Response_Time = (
     Referenced_Event                  => Identifier,
     Time_Value                        => Time),
```

A *Miss_Ratio* contains the following attributes:

- *Deadline*: Soft deadline against which the response time is compared to determine the ration of missed deadlines.
- *Ratio*: Percentage of events that have missed the soft deadline, relative to the total number of events.

```
Miss_Ratio = (
     Deadline                          => Time,
     Ratio                             => Percentage),
```

A *Global_Miss_Ratio* contains the following attributes:

- *Referenced_Event*: Name of referenced input event, used for calculating the response time.

- *Miss_Ratios*: List of miss ratios.

```
Global_Miss_Ratio = (
    Referenced_Event                => Identifier,
    Miss_Ratios                     => (
                                       Miss_Ratio 1,
                                       Miss_Ratio 2,
                                       ...)),
```

9.3 Processing_Resource

The processing resource results are relative to a processing resource in the system that has been analyzed, and contain the name of the resource and a set of results (described below), using the following format:

```
Processing_Resource(
    Name                            => Identifier,
    Results                         => (
                                       Result 1,
                                       Result 2,
                                       ...));
```

The specific results that may refer to a processing resource are:

- *Slack*: If positive, it is the percentage by which all the execution times of all the operations executed in the processing resource may be increased while still keeping the system schedulable. If negative, it is the percentage by which all the execution times of all the operations executed in the processing resource have to be decreased to make the system schedulable. If zero, it means that the processing resource is just schedulable.

```
Result = (
    Type                            => Slack,
    Value                           => Processing resource slack)
```

- *Utilization*: This result measures the relation, in percentage, between the time that the processing resource is being used to execute activities, and the total elapsed time. It may contain the following attributes:
    - *Total*: overall utilization in the processing result.
    - *Application*: utilization of the processing resource by the application code, i.e., without the overhead elements included in the MAST model: context and interrupt switches, network drivers, and system timers.
    - *Context_Switch*: utilization of the processing resource by context and interrupt switch activities.
    - *Timer*: utilization of the processing resource by the system timer overhead.
    - *Driver*: utilization of the processing resource by the network drivers overhead.

```
Result = (
    Type                            => Detailed_Utilization,
    Total                              => percentage,
    Application                        => percentage,
    Context_Switch                     => percentage,
    Timer                              => percentage,
    Driver                             => percentage)
```

- *Ready_Queue_Size*: It contains the following attributes:
  - *Max_Num*: Maximum number of scheduling servers that are simultaneously ready in the processing resource.

```
Result = (
     Type                               => Ready_Queue_Size,
     Max_Num                            => Positive)
```

9.4 Operation

The operation results are relative to an operation in the system that has been analyzed, and contain the name of the operation and a set of results (described below), using the following format:

```
Operation (
     Name                               => Name of the operation,
     Results                            => (
                                        Result 1,
                                        Result 2,
                                        ...));
```

The specific results that may refer to an operation are:

- *Slack*: If positive, it is the percentage by which the execution times of the operation may be increased while still keeping the system schedulable. If negative, it is the percentage by which the execution times of the operation have to be decreased to make the system schedulable. If zero, it means that the system is just schedulable with regard to this operation.

```
Result = (
     Type                               => Slack,
     Value                              => Percentage)
```

9.5 Scheduling Server

The scheduling server results are relative to a scheduling server in the system that has been analyzed, and contain the name of the scheduling server and a set of results (described below), using the following format:

```
Scheduling_Server (
     Name                               => Name of the scheduling server,
     Results                            => (
                                        Result 1,
                                        Result 2,
                                        ...));
```

The specific results that may refer to a scheduling server are:

- *Scheduling_Parameters*: The scheduling parameters that were used in the analyzed system. Usually they are only written to the file if they were automatically calculated by the priority assignment tools. See section on "Scheduling Parameters" for a description of their format.

```
Result = (
    Type                             => Scheduling_Parameters,
    Server_Sched_Parameters          => Fixed_Priority_Sched_Parameters)
```

9.6 Shared Resource

The shared resource results are relative to a shared resource in the system that has been analyzed, and contain the name of the shared resource and a set of results (described below), using the following format:

```
Shared_Resource (
    Name                             => Name of the shared resource,
    Results                          => (
                                     Result 1,
                                     Result 2,
                                     ...));
```

The specific results that may refer to a shared resource are:

- *Ceiling*: The priority ceiling automatically calculated by the MAST tool. Only shared resources of the type *Immediate_Ceiling_Resource* may have this type of result.

```
Result = (
    Type                             => Priority_Ceiling,
    Ceiling                          => Any_Priority)
```

- *Queue_Size*: Size of the waiting queue of the shared resource. It contains the following attributes:
    - *Max_Num*: Maximum number of threads that were queued in the shared resource, waiting to lock it.

```
Result = (
    Type                             => Queue_Size,
    Max_Num                          => Maximum number)
```

- *Utilization*: It measures the total time that the shared resource has been locked during a simulation, relative to the total elapsed time

```
Result = (
    Type                             => Utilization,
    Total                            => percentage)
```

10. Example of a Single-Processor System: CASEVA

CASEVA is a robot designed for automatic welding of junctions between pieces that don't have axial symmetry. It has an embedded controller that uses a VME-bus based computer (an HP 743rt) running HP-RT as its real-time operating system. The application software is concurrent, and written in Ada. The basic characteristics of its tasks are shown in Figure 6.

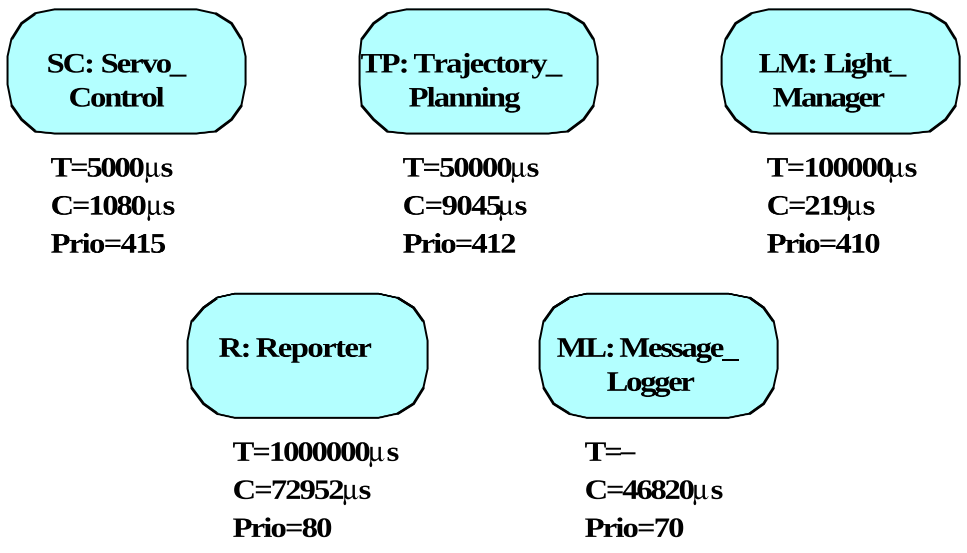


Figure 6. Basic Characteristics of the tasks of the CASEVA controller

Communication and synchronization between the different tasks is asynchronous, and based on shared resources implemented using Ada's protected objects. In this document we present a simplified view of the shared resources and associated protected operations, to make the description shorter. The following table shows the characteristics of the simplified protected objects and operations.

Shared Resource	**Operation**	**WCET (µs)**	**Used by**
Servo_Data	Read_New_Point New_Point	87 54	SC TP
Arm	Read_Axis_Positions Control_Servos	135 99	SC, R SC
Lights	Turn_On Turn_Off Time_Lights	74 71 119	TP TP LM
Alarms	Read_All Set	78 59	SC, TP, R SC, TP
Error_Log	Notify_Error Get_Error_From_Queue	85 79	TP ML

The MAST description of this system is shown next:

```
-- Real_time Situation

Model(
     Model_Name=> Caseva,
     Model_Date=> 2000-01-01);

-- Processing Resources

Processing_Resource (
     Type                   => Fixed_Priority_Processor,
     Name                   => Processor_1,
     Worst_Context_Switch => 102.5,
     System_Timer =>
             (Type           => Alarm_Clock,
              Worst_Overhead=> 50));

-- Scheduling Servers

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Servo_Control,
     Server_Sched_Parameters => (
             Type           => Fixed_Priority_policy,
             The_Priority   => 415),
     Server_Processing_Resource => Processor_1);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Trajectory_Planning,
     Server_Sched_Parameters => (
             Type           => Fixed_Priority_policy,
             The_Priority   => 412),
     Server_Processing_Resource=> Processor_1);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Light_Manager,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 410),
     Server_Processing_Resource=> Processor_1);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Reporter,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 80),
     Server_Processing_Resource=> Processor_1);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Message_Logger,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority  => 70),
     Server_Processing_Resource=> Processor_1);

-- Resources

Shared_Resource (
     Type    => Immediate_Ceiling_Resource,
     Name    => Servo_Data);

Shared_Resource (
     Type    => Immediate_Ceiling_Resource,
     Name    => Arm);

Shared_Resource (
     Type    => Immediate_Ceiling_Resource,
     Name    => Lights);

Shared_Resource (
     Type    => Immediate_Ceiling_Resource,
     Name    => Alarms);

Shared_Resource (
     Type    => Immediate_Ceiling_Resource,
     Name    => Error_Log);


-- Operations

-- Critical Sections

Operation (
     Type                     => Simple,
     Name                     => Read_New_Point,
     Worst_Case_Execution_Time => 87,
       Shared_Resources_List=> (Servo_Data));

Operation (
     Type                     => Simple,
     Name                     => New_Point,
     Worst_Case_Execution_Time => 54,
       Shared_Resources_List=> (Servo_Data));

Operation (
     Type                     => Simple,
     Name                     => Read_Axis_Positions,
     Worst_Case_Execution_Time => 135,
       Shared_Resources_List=> (Arm));

Operation (
     Type                     => Simple,
     Name                     => Control_Servos,
     Worst_Case_Execution_Time => 99,
       Shared_Resources_List=> (Arm));

Operation (
     Type                     => Simple,
     Name                     => Turn_On,
     Worst_Case_Execution_Time => 74,
       Shared_Resources_List=> (Lights));

Operation (
     Type                     => Simple,
     Name                     => Turn_Off,
     Worst_Case_Execution_Time => 71,
       Shared_Resources_List=> (Lights));

Operation (
     Type                     => Simple,
     Name                     => Time_Lights,
     Worst_Case_Execution_Time => 119,
       Shared_Resources_List=> (Lights));

Operation (
     Type                     => Simple,
     Name                     => Read_All_Alarms,
     Worst_Case_Execution_Time => 78,
       Shared_Resources_List=> (Alarms));

Operation (
     Type                     => Simple,
     Name                     => Set,
     Worst_Case_Execution_Time => 59,
       Shared_Resources_List=> (Alarms));

Operation (
     Type                     => Simple,
     Name                     => Notify_Error,
     Worst_Case_Execution_Time => 85,
       Shared_Resources_List=> (Error_Log));

Operation (
     Type                     => Simple,
     Name                     => Get_Error_From_Queue,
     Worst_Case_Execution_Time => 79,
       Shared_Resources_List=> (Error_Log));

-- Enclosing operations

Operation (
     Type   => Enclosing,
     Name   => Servo_Control,
     Worst_Case_Execution_Time => 1080,
     Composite_Operation_List =>
            (Read_New_Point,Read_Axis_Positions,Control_Servos,
                 Read_All_Alarms,Set));

Operation (
     Type   => Enclosing,
     Name   => Trajectory_Planning,
     Worst_Case_Execution_Time => 9045,
     Composite_Operation_List =>
            (New_Point, Turn_On, Turn_Off,
                 Read_All_Alarms,Set,Notify_Error));

Operation (
     Type   => Enclosing,
     Name   => Light_Manager,
     Worst_Case_Execution_Time => 119,
     Composite_Operation_List =>
            (Time_Lights));

Operation (
     Type   => Enclosing,
     Name   => Reporter,
     Worst_Case_Execution_Time => 72952,
     Composite_Operation_List =>
            (Read_Axis_Positions,Read_All_Alarms));

Operation (
     Type   => Enclosing,
     Name   => Message_Logger,
     Worst_Case_Execution_Time => 46820,
     Composite_Operation_List =>
            (Get_Error_From_Queue));


-- Transactions

Transaction (
     Type   => Regular,
     Name   => Servo_Control,
     External_Events => (
            (Type  => Periodic,
             Name  => E1,
             Period => 5000)),
     Internal_Events => (
            (Type  => regular,
             name  => O1,
                 Timing_Requirements => (
                    Type   => Hard_Global_Deadline,
                    Deadline  => 5000,
                    Referenced_Event => E1))),
     Event_Handlers => (
                (Type          => System_Timed_Activity,
                 Input_Event   => E1,
                 Output_Event  => O1,
                 Activity_Operation => Servo_Control,
                 Activity_Server=> Servo_Control)));

Transaction (
     Type    => Regular,
     Name    => Trajectory_Planning,
     External_Events => (
             (Type   => Periodic,
              Name   => E2,
              Period => 50000)),
     Internal_Events => (
             (Type   => regular,
              name   => O2,
                   Timing_Requirements => (
                      Type    => Hard_Global_Deadline,
                      Deadline   => 50000,
                      Referenced_Event => E2))),
     Event_Handlers => (
             (Type          => System_Timed_Activity,
              Input_Event   => E2,
              Output_Event  => O2,
              Activity_Operation => Trajectory_Planning,
              Activity_Server=> Trajectory_Planning)));

Transaction (
     Type    => Regular,
     Name    => Light_Manager,
     External_Events => (
             (Type   => Periodic,
              Name   => E3,
              Period => 100000)),
     Internal_Events => (
             (Type   => regular,
              name   => O3,
                   Timing_Requirements => (
                      Type    => Hard_Global_Deadline,
                      Deadline   => 100000,
                      referenced_event => E3))),
     Event_Handlers => (
             (Type          => System_Timed_Activity,
              Input_Event   => E3,
              Output_Event  => O3,
              Activity_Operation => Light_Manager,
              Activity_Server=> Light_Manager)));

Transaction (
     Type    => Regular,
     Name    => Reporter,
     External_Events => (
             (Type   => Periodic,
              Name   => E4,
              Period => 1000000)),
     Internal_Events => (
             (Type   => regular,
              name   => O4,
                   Timing_Requirements => (
                      Type    => Hard_Global_Deadline,
                      Deadline   => 1000000,
                      referenced_event => E4))),
     Event_Handlers => (
             (Type          => System_Timed_Activity,
              Input_Event   => E4,
              Output_Event  => O4,
              Activity_Operation => Reporter,
              Activity_Server=> Reporter)));

Transaction (
     Type    => Regular,
     Name    => Message_Logger,
     External_Events => (
             (Type          => Unbounded,
              Name          => E5,
               Avg_Interarrival=> 1000000)),
     Internal_Events => (
             (Type   => regular,
              name   => O5)),
     Event_Handlers => (
             (Type          => Activity,
              Input_Event   => E5,
              Output_Event  => O5,
              Activity_Operation => Message_Logger,
              Activity_Server=> Message_Logger)));
```

11. Example of Linear_Transactions: RMT

The following example will show the aspect of the MAST file format that has been chosen to represent the timing behavior of real-time applications. The example is a simplification of the control system of a teleoperated robot. This is a distributed system with two specialized nodes: a local robot controller, and a remote teleoperation station, where the operator manipulates the controls, and gets information about the system status. Figure 7 shows a diagram of the software architecture. The system has three transactions; one of them, the main control loop, implies execution in different processing resources, and has a global end-to-end deadline. Communication is through an ethernet network used in master-slave mode to achieve hard real-time behavior.

In the MAST description we can see that we declare, in this order, the processing resources, the scheduling servers, the shared resources, the operations, and finally, the transactions. The timing requirements are embedded in the events described in the transactions. The timers (and

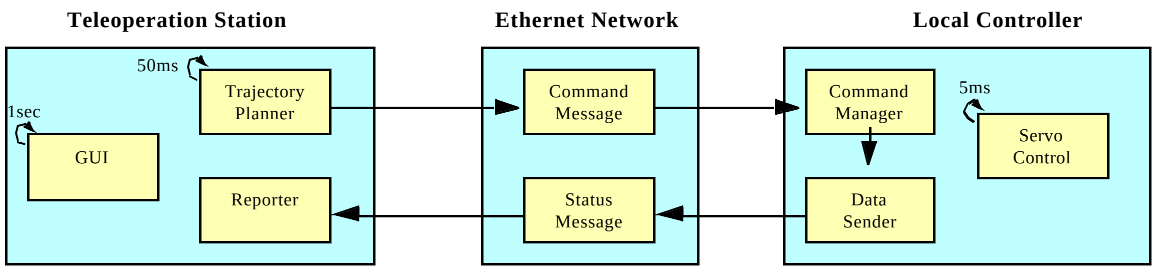


Figure 7. Architecture of the teleoperated robot controller

also the network drivers) are embedded in the description of the processing resources. The scheduling parameters are embedded in the description of the scheduling servers. Finally, the events and event handlers are embedded in the description of the transactions. The description is shown next:

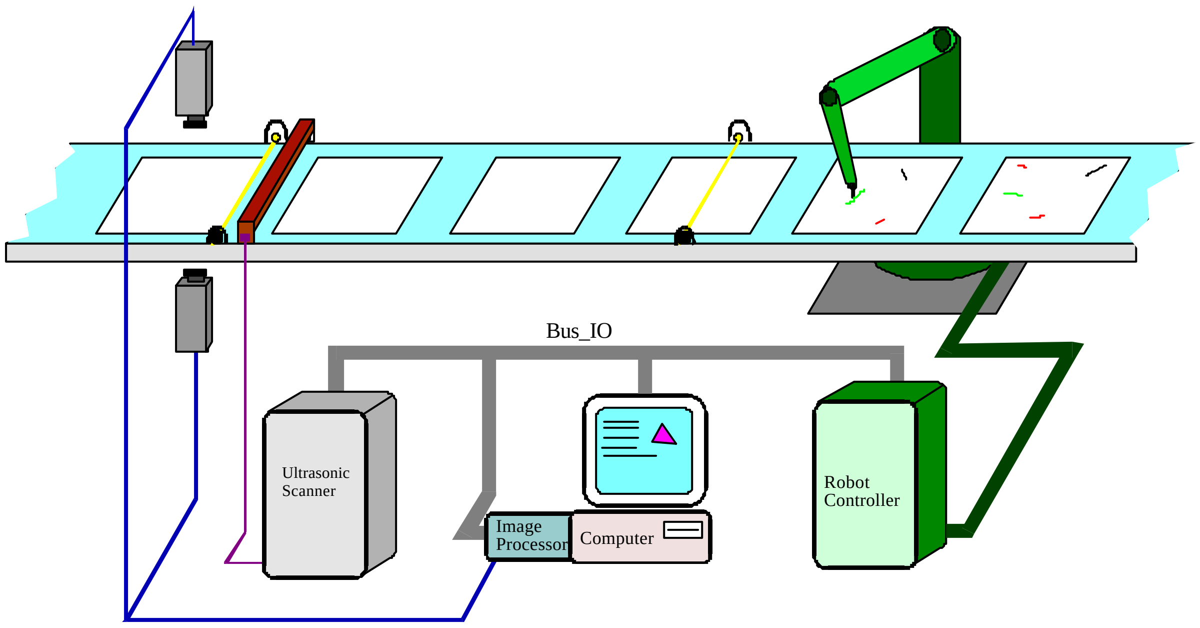
```
-- Real-Time Situation

Model(
     Model_Name=> RMT,
     Model_Date=> 2002-11-23T10:22:33);

-- Processing Resources

Processing_Resource (
     Type                     => Fixed_Priority_Processor,
     Name                     => Teleoperation_Station,
     Worst_Context_Switch => 102.5,
     System_Timer             =>
              (Type           => Alarm_Clock,
               Worst_Overhead=> 50));

Processing_Resource (
     Type                     => Fixed_Priority_Processor,
     Name                     => Local_Controller,
     Worst_Context_Switch => 15,
     System_Timer                =>
              (Type           => Alarm_Clock,
               Worst_Overhead=> 10));

Processing_Resource (
     Type                     => Fixed_Priority_Network,
     Name                     => Ethernet,
     Transmission                => Half_Duplex);

-- Scheduling Servers

Scheduling_Server (
     Type                     => Fixed_Priority,
     Name                     => Servo_Control,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 415),
     Server_Processing_Resource=> Local_Controller);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Command_Manager,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 412),
     Server_Processing_Resource=> Local_Controller);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Data_Sender,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 410),
     Server_Processing_Resource=> Local_Controller);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Trajectory_Planner,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 80),
     Server_Processing_Resource=> Teleoperation_Station);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Reporter,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 79),
     Server_Processing_Resource=> Teleoperation_Station);

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => GUI,
     Server_Sched_Parameters=> (
             Type           => Fixed_Priority_policy,
             The_Priority   => 60),
     Server_Processing_Resource=> Teleoperation_Station);

-- Message scheduler

Scheduling_Server (
     Type                   => Fixed_Priority,
     Name                   => Message_Scheduler,
     Server_Sched_Parameters=> (
              Type           => Fixed_Priority_policy,
              The_Priority   => 1),
         Server_Processing_Resource=> Ethernet);

-- Resources

Shared_Resource (
         Type    => Immediate_Ceiling_Resource,
         Name    => Status);

Shared_Resource (
         Type    => Immediate_Ceiling_Resource,
         Name    => Commands);

Shared_Resource (
         Type    => Immediate_Ceiling_Resource,
         Name    => Servo_Data);

-- Operations

-- Critical Sections

Operation (
         Type                     => Simple,
         Name                     => Read_Status,
         Worst_Case_Execution_Time => 87,
           Shared_Resources_List=> (Status));

Operation (
         Type                     => Simple,
         Name                     => Write_Status,
         Worst_Case_Execution_Time => 54,
           Shared_Resources_List=> (Status));

Operation (
         Type                     => Simple,
         Name                     => Set_Command,
         Worst_Case_Execution_Time => 135,
           Shared_Resources_List=> (Commands));

Operation (
         Type                     => Simple,
         Name                     => Get_Command,
         Worst_Case_Execution_Time => 99,
           Shared_Resources_List=> (Commands));

Operation (
         Type                     => Simple,
         Name                     => Read_Servos,
         Worst_Case_Execution_Time => 74,
           Shared_Resources_List=> (Servo_Data));
```

```
Operation (
     Type                  => Simple,
     Name                  => Write_Servos,
     Worst_Case_Execution_Time => 71,
       Shared_Resources_List=> (Servo_Data));

-- Enclosing operations

Operation (
     Type    => Enclosing,
     Name    => Command_Manager,
     Worst_Case_Execution_Time => 9045,
     Composite_Operation_List =>
            (Write_Servos));

Operation (
     Type    => Enclosing,
     Name    => Data_Sender,
     Worst_Case_Execution_Time => 1220,
     Composite_Operation_List =>
            (Read_Servos));

Operation (
     Type    => Enclosing,
     Name    => Servo_Control,
     Worst_Case_Execution_Time => 1019,
     Composite_Operation_List =>
            (Read_Servos,Write_Servos));

Operation (
     Type    => Enclosing,
     Name    => Trajectory_Planner,
     Worst_Case_Execution_Time => 7952,
     Composite_Operation_List =>
            (Get_Command));

Operation (
     Type    => Enclosing,
     Name    => Reporter,
     Worst_Case_Execution_Time => 2086,
     Composite_Operation_List =>
            (Write_Status));

Operation (
     Type    => Enclosing,
     Name    => GUI,
     Worst_Case_Execution_Time => 146820,
     Composite_Operation_List =>
            (Read_Status,Set_Command));
```

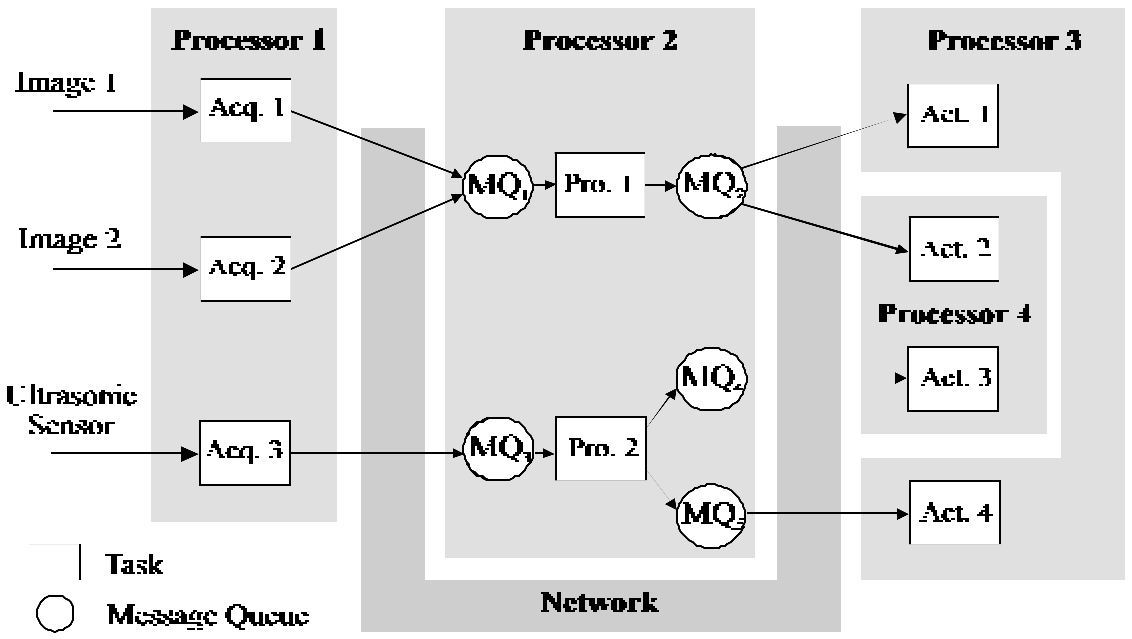
```
-- Network operations

Operation (
     Type    => Simple,
     Name    => Command_Message,
     Worst_Case_Execution_Time => 4850);

Operation (
     Type    => Simple,
     Name    => Status_Message,
     Worst_Case_Execution_Time => 5080);


-- Transactions

Transaction (
     Type    => Regular,
     Name    => Servo_Control,
     External_Events => (
            (Type          => Periodic,
             Name          => E1,
             Period => 5000)),
     Internal_Events => (
            (Type   => regular,
             name   => O1,
                 Timing_Requirements => (
                    Type    => Hard_Global_Deadline,
                    Deadline   => 5000,
                    referenced_event => E1))),
     Event_Handlers => (
            (Type          => System_Timed_Activity,
             Input_Event   => E1,
             Output_Event  => O1,
             Activity_Operation => Servo_Control,
             Activity_Server=> Servo_Control)));

Transaction (
     Type    => Regular,
     Name    => Main_Control_Loop,
     External_Events => (
            (Type   => Periodic,
             Name   => E2,
             Period => 50000)),
     Internal_Events => (
            (Type   => regular,
             name   => O2),
            (Type   => regular,
             name   => O3),
            (Type   => regular,
             name   => O4),
            (Type  => regular,
             name  => O5),
            (Type  => regular,
             name  => O6),
            (Type  => regular,
             name  => O7,
                 Timing_Requirements => (
                    Type   => Hard_Global_Deadline,
                    Deadline  => 50000,
                    referenced_event => E2))),
     Event_Handlers => (
            (Type          => System_Timed_Activity,
             Input_Event   => E2,
             Output_Event  => O2,
             Activity_Operation => Trajectory_Planner,
             Activity_Server=> Trajectory_Planner),
            (Type          => Activity,
             Input_Event   => O2,
             Output_Event  => O3,
             Activity_Operation => Command_Message,
             Activity_Server=> Message_Scheduler),
            (Type          => Activity,
             Input_Event   => O3,
             Output_Event  => O4,
             Activity_Operation => Command_Manager,
             Activity_Server=> Command_Manager),
            (Type          => Activity,
             Input_Event   => O4,
             Output_Event  => O5,
             Activity_Operation => Data_Sender,
             Activity_Server=> Data_Sender),
            (Type          => Activity,
             Input_Event   => O5,
             Output_Event  => O6,
             Activity_Operation => Status_Message,
             Activity_Server=> Message_Scheduler),
            (Type          => Activity,
             Input_Event   => O6,
             Output_Event  => O7,
             Activity_Operation => Reporter,
             Activity_Server=> Reporter)));

Transaction (
     Type    => Regular,
     Name    => GUI,
     External_Events => (
            (Type  => Periodic,
             Name  => E3,
             Period => 1000000)),
     Internal_Events => (
            (Type  => regular,
             name   => O8,
                 Timing_Requirements => (
                    Type    => Hard_Global_Deadline,
                    Deadline   => 1000000,
                    referenced_event => E3))),
     Event_Handlers => (
            (Type          => System_Timed_Activity,
             Input_Event   => E3,
             Output_Event  => O8,
             Activity_Operation => GUI,
             Activity_Server=> GUI)));
```

12. Example of Multiple_Event_Transactions

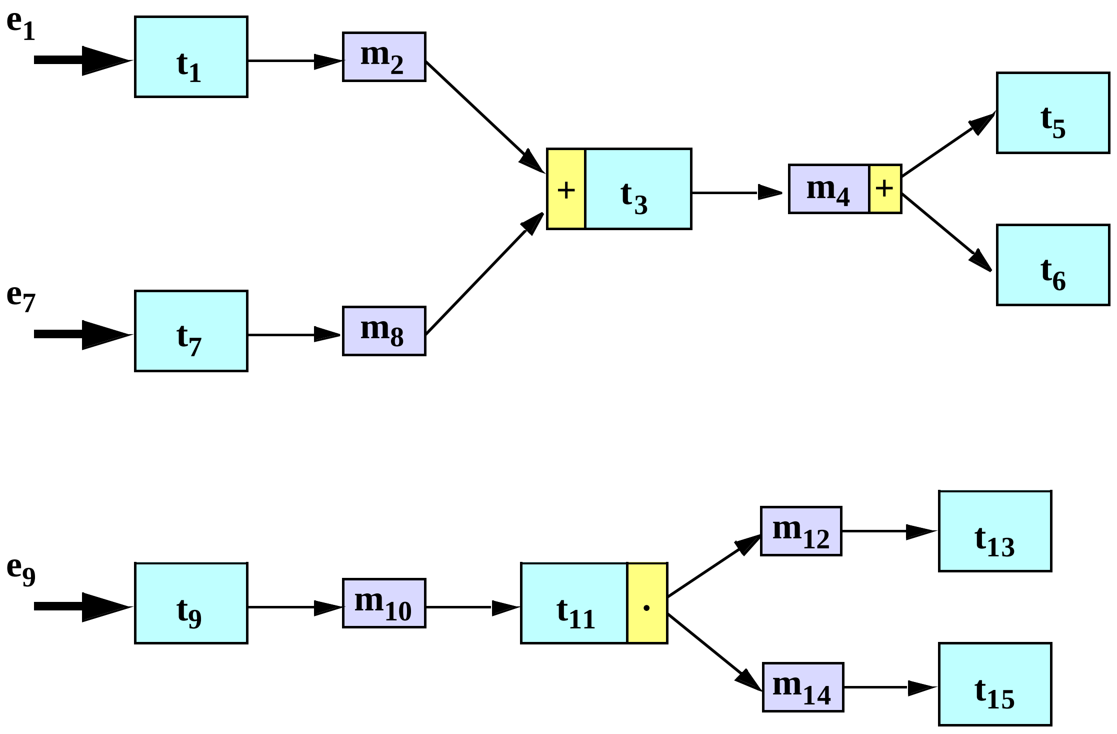
Example of steel bars inspection:



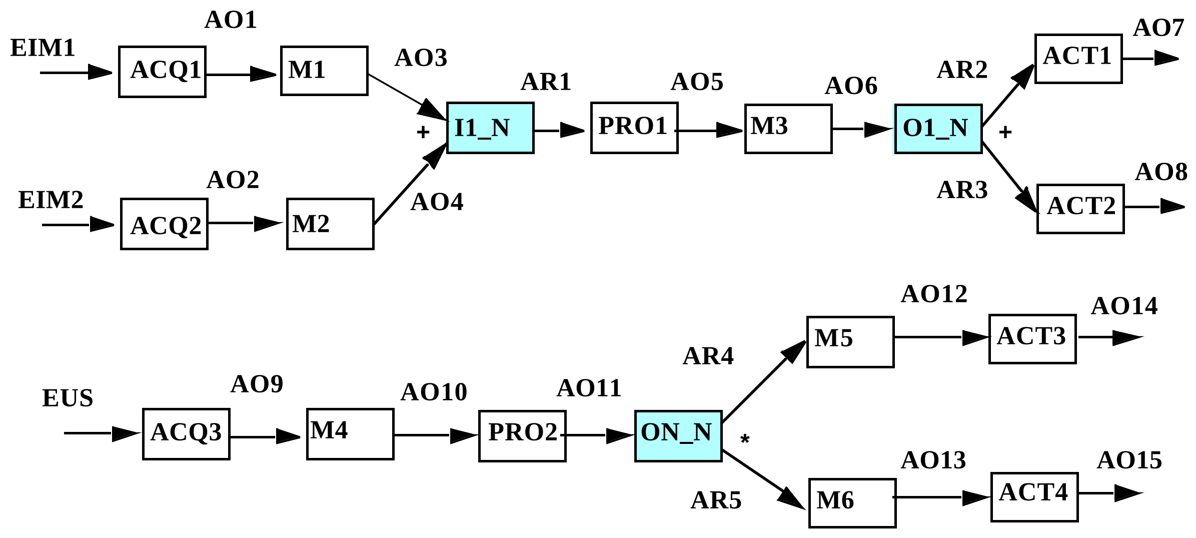
Software Architecture for this example:



Multiple event synchronization model for this example:



Graph for the example:



Input File for the Multiple-Event Example

```
-- Real-Time System Model for the Example
-- All the timing requirements are global deadlines
-- 5 Processing resources
-- 0 Data resources
-- 15 Operations
-- 15 Scheduling Servers
-- 2 Transactions
--       1 --> 2 External Events
--            11 Internal_Events
--             10 Event Handlers (8 Activities, 2 others)
--       2 --> 1 External Event
--            9 Internal Events
--             8 Event Handlers (7 Activities, 1 other)

-- Real-Time Situation

Model(
     Model_Name=> Steel_Bars_Inspection,
     Model_Date=> 2001-09-12T20:21:45);

-- Resources

Processing_Resource (
     Type                     => Fixed_Priority_Processor,
     Name                     => Processor_1,
     Worst_Context_Switch => 50,
     Avg_Context_Switch   => 15,
     Best_Context_Switch  => 10);

Processing_Resource (
     Type                     => Fixed_Priority_Processor,
     Name                     => Processor_2,
     Worst_Context_Switch => 50,
     Avg_Context_Switch   => 150,
     Best_Context_Switch  => 10);

Processing_Resource (
     Type                 => Fixed_Priority_Processor,
     Name                 => Processor_3,
     Worst_Context_Switch => 50,
     Avg_Context_Switch   => 150,
     Best_Context_Switch  => 10);

Processing_Resource (
     Type                 => Fixed_Priority_Processor,
     Name                 => Processor_4,
     Worst_Context_Switch => 50,
     Avg_Context_Switch   => 150,
     Best_Context_Switch  => 10);

Processing_Resource (
     Type                              => Fixed_Priority_Network,
     Name                              => Network,
     Max_Packet_Transmission_Time      => 100);

-- Operations

Operation (
     Type                       => Simple,
     Name                       => ACQ1,
     Worst_Case_Execution_Time  => 50,
     Avg_Case_Execution_Time    => 50,
     Best_Case_Execution_Time   => 50);

Operation (
     Type                       => Simple,
     Name                       => ACQ2,
     Worst_Case_Execution_Time  => 50,
     Avg_Case_Execution_Time    => 50,
     Best_Case_Execution_Time   => 50);

Operation (
     Type                       => Simple,
     Name                       => ACQ3,
     Worst_Case_Execution_Time  => 820,
     Avg_Case_Execution_Time    => 820,
     Best_Case_Execution_Time   => 820);

Operation (
     Type                       => Simple,
     Name                       => PRO1,
     Worst_Case_Execution_Time  => 100,
     Avg_Case_Execution_Time    => 100,
     Best_Case_Execution_Time     => 100);

Operation (
     Type                         => Simple,
     Name                         => PRO2,
     Worst_Case_Execution_Time    => 750,
     Avg_Case_Execution_Time      => 750,
     Best_Case_Execution_Time     => 750);

Operation (
     Type                         => Simple,
     Name                         => ACT1,
     Worst_Case_Execution_Time    => 100,
     Avg_Case_Execution_Time      => 100,
     Best_Case_Execution_Time     => 100);

Operation (
     Type                         => Simple,
     Name                         => ACT2,
     Worst_Case_Execution_Time    => 100,
     Avg_Case_Execution_Time      => 100,
     Best_Case_Execution_Time     => 100);

Operation (
     Type                         => Simple,
     Name                         => ACT3,
     Worst_Case_Execution_Time    => 725,
     Avg_Case_Execution_Time      => 725,
     Best_Case_Execution_Time     => 725);

Operation (
     Type                         => Simple,
     Name                         => ACT4,
     Worst_Case_Execution_Time    => 740,
     Avg_Case_Execution_Time      => 740,
     Best_Case_Execution_Time     => 740);

Operation (
     Type                         => Simple,
     Name                         => M1,
     Worst_Case_Execution_Time    => 100,
     Avg_Case_Execution_Time      => 100,
     Best_Case_Execution_Time     => 100);

Operation (
     Type                         => Simple,
     Name                         => M2,
     Worst_Case_Execution_Time    => 100,
     Avg_Case_Execution_Time      => 100,
     Best_Case_Execution_Time     => 100);
```

```
Operation (
     Type                          => Simple,
     Name                          => M3,
     Worst_Case_Execution_Time     => 50,
     Avg_Case_Execution_Time       => 50,
     Best_Case_Execution_Time      => 50);

Operation (
     Type                          => Simple,
     Name                          => M4,
     Worst_Case_Execution_Time     => 150,
     Avg_Case_Execution_Time       => 150,
     Best_Case_Execution_Time      => 150);

Operation (
     Type                          => Simple,
     Name                          => M5,
     Worst_Case_Execution_Time     => 230,
     Avg_Case_Execution_Time       => 230,
     Best_Case_Execution_Time      => 230);

Operation (
     Type                          => Simple,
     Name                          => M6,
     Worst_Case_Execution_Time     => 250,
     Avg_Case_Execution_Time       => 250,
     Best_Case_Execution_Time      => 250);

-- Scheduling Servers

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SACQ1,
     Server_Sched_Parameters       => (
            Type           => Fixed_Priority_Policy,
            The_Priority   => 1),
     Server_Processing_Resource    => Processor_1);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SACQ2,
     Server_Sched_Parameters       => (
            Type           => Fixed_Priority_Policy,
            The_Priority   => 2),
     Server_Processing_Resource    => Processor_1);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SACQ3,
     Server_Sched_Parameters       => (
            Type           => Fixed_Priority_Policy,
                The_Priority  => 3),
         Server_Processing_Resource  => Processor_1);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SPRO1,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 1),
         Server_Processing_Resource  => Processor_2);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SPRO2,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 2),
         Server_Processing_Resource  => Processor_2);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SACT1,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 1),
         Server_Processing_Resource  => Processor_3);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SACT2,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 1),
         Server_Processing_Resource  => Processor_4);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SACT3,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 2),
         Server_Processing_Resource  => Processor_4);

Scheduling_Server (
         Type                        => Fixed_Priority,
         Name                        => SACT4,
         Server_Sched_Parameters     => (
                Type           => Fixed_Priority_Policy,
                The_Priority   => 2),
         Server_Processing_Resource  => Processor_3);
```

```
Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM1,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 1),
     Server_Processing_Resource   => Network);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM2,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 3),
     Server_Processing_Resource   => Network);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM3,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 2),
     Server_Processing_Resource   => Network);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM4,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 4),
     Server_Processing_Resource   => Network);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM5,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 5),
     Server_Processing_Resource   => Network);

Scheduling_Server (
     Type                          => Fixed_Priority,
     Name                          => SM6,
     Server_Sched_Parameters       => (
             Type           => Fixed_Priority_Policy,
             The_Priority   => 6),
     Server_Processing_Resource   => Network);

-- Transactions

Transaction (
Type    => Regular,
Name    => Trans1,
External_Events => (
        (Type           => Periodic,
         Name           => EIM1,
         Period         => 1000,
         Max_Jitter     => 0,
         Phase          => 0),
        (Type           => Periodic,
         Name           => EIM2,
         Period         => 1000,
         Max_Jitter     => 0,
         Phase          => 0)),
Internal_Events => (
        (Type   => regular,
         name   => AO1),
        (Type   => regular,
         name   => AO2),
        (Type   => regular,
         name   => AO3),
        (Type   => regular,
         name   => AO4),
        (Type   => regular,
         name   => AO5),
        (Type   => regular,
         name   => AO6),
        (Type                           => regular,
         name                           => AO7,
         Timing_Requirements            => (
                Type                    => Composite,
                Requirements_List       => (
                      (Type             => Hard_Global_Deadline,
                       Deadline         => 1000,
                       referenced_event => EIM1),
                      (Type             => Hard_Global_Deadline,
                       Deadline         => 1000,
                       referenced_event => EIM2)))),
        (Type                           => regular,
         name                           => AO8,
         Timing_Requirements            => (
                Type                    => Composite,
                Requirements_List       => (
                      (Type             => Hard_Global_Deadline,
                       Deadline         => 1000,
                       referenced_event => EIM1),
                      (Type             => Hard_Global_Deadline,
                       Deadline         => 1000,
                       referenced_event => EIM2)))),
        (Type   => regular,
         name   => AR1),
        (Type   => regular,
         name   => AR2),
        (Type   => regular,
         name   => AR3)),
 Event_Handlers => (
        (Type                 => Activity,
         Input_Event          => EIM1,
         Output_Event         => AO1,
         Activity_Operation   => ACQ1,
         Activity_Server      => SACQ1),
        (Type                 => Activity,
         Input_Event          => EIM2,
         Output_Event         => AO2,
         Activity_Operation   => ACQ2,
         Activity_Server      => SACQ2),
        (Type                 => Activity,
         Input_Event          => AO1,
         Output_Event         => AO3,
         Activity_Operation   => M1,
         Activity_Server      => SM1),
        (Type                 => Activity,
         Input_Event          => AO2,
         Output_Event         => AO4,
         Activity_Operation   => M2,
         Activity_Server      => SM2),
        (Type                 => Activity,
         Input_Event          => AR1,
         Output_Event         => AO5,
         Activity_Operation   => PRO1,
         Activity_Server      => SPRO1),
        (Type                 => Activity,
         Input_Event          => AO5,
         Output_Event         => AO6,
         Activity_Operation   => M3,
         Activity_Server      => SM3),
        (Type                 => Activity,
         Input_Event          => AR2,
         Output_Event         => AO7,
         Activity_Operation   => ACT1,
         Activity_Server      => SACT1),
        (Type                 => Activity,
         Input_Event          => AR3,
         Output_Event         => AO8,
         Activity_Operation   => ACT2,
         Activity_Server      => SACT2),
        (Type                 => Concentrator,
         Output_Event         => AR1,
         Input_Events_List    => (
                             AO3,
                             AO4)),
        (Type                 => Delivery_Server,
         Input_Event          => AO6,
         Output_Events_List  => (
                                AR2,
                                AR3))));

Transaction (
Type    => Regular,
Name    => Trans2,
External_Events =>(
     (Type          => Periodic,
      Name          => EUS,
      Period        => 1000,
      Max_Jitter    => 0,
      Phase         => 0)),
Internal_Events => (
     (Type   => regular,
      name   => AO9),
     (Type   => regular,
      name   => AO10),
     (Type   => regular,
      name   => AO11),
     (Type   => regular,
      name   => AO12),
     (Type   => regular,
      name   => AO13),
     (Type                          => regular,
      name                          => AO14,
      Timing_Requirements           => (
             Type          => Hard_Global_Deadline,
             Deadline      => 10000,
             referenced_event => EUS)),
     (Type                          => regular,
      name                          => AO15,
      Timing_Requirements           => (
             Type          => Hard_Global_Deadline,
             Deadline      => 10000,
             referenced_event => EUS)),
     (Type   => regular,
      name   => AR4),
     (Type   => regular,
      name   => AR5)),
Event_Handlers => (
     (Type                 => Activity,
      Input_Event          => EUS,
      Output_Event         => AO9,
      Activity_Operation   => ACQ3,
      Activity_Server      => SACQ3),
     (Type                 => Activity,
      Input_Event          => AO9,
      Output_Event         => AO10,
      Activity_Operation   => M4,
      Activity_Server      => SM4),
(Type                 => Activity,
Input_Event          => AO10,
Output_Event         => AO11,
Activity_Operation   => PRO2,
Activity_Server      => SPRO2),
(Type                 => Activity,
Input_Event          => AR4,
Output_Event         => AO12,
Activity_Operation   => M5,
Activity_Server      => SM5),
(Type                 => Activity,
Input_Event          => AR5,
Output_Event         => AO13,
Activity_Operation   => M6,
Activity_Server      => SM6),
(Type                 => Activity,
Input_Event          => AO12,
Output_Event         => AO14,
Activity_Operation   => ACT3,
Activity_Server      => SACT3),
(Type                 => Activity,
Input_Event          => AO13,
Output_Event         => AO15,
Activity_Operation   => ACT4,
Activity_Server      => SACT4),
(Type                 => Multicast,
Input_Event          => AO11,
Output_Events_List   => (
               AR4,
               AR5))));
```