Modeling and Analysis Suite for Real Time Applications (MAST 1.3.7)

Description of the MAST Model

By:

José María Drake	drakej@unican.es
Michael González Harbour	mgh@unican.es
José Javier Gutiérrez	gutierjj@unican.es
Patricia López Martínez	lopezpa@unican.es
Julio Luis Medina	medinajl@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 analysing 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.
- Schedulers may be composed in a hierarchical way.
- The toolset is open source and fully extensible. That means that other teams may provide enhancements. The first version was intended for fixed priority systems, and the current version also supports dynamically scheduled systems.

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 (e.g., POSIX and Ada). This implies fixed priority scheduled systems, but the system should also support other scheduling algorithms (such as EDF) as well as hierarchical schedulers. 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 also be activated by a conditional combination of one or more events, through the following event handlers:
  - concentrator (merge)
  - barrier (join)
  - delivery_server (branch with decision made by sender)
  - query_server (branch with request from receiver)
  - multicast (fork)
- 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 and preemption levels for shared resources

  - system, transaction or processor slacks (percentage by which the system, processor or transaction operations may be increased while keeping the system schedulable)

- A simulator tool lets us obtain accurate data about average results and performance features. Likewise, the simulator lets us estimate worst and best case response parameters when the particular features of the model do not obey the restrictions of the worst-case schedulability analysis techniques.

The model is included in a toolset (Figure 1 and Figure 2), with the following elements:

- The model and the results are specified through an ASCII description that serves as the input and output of the analysis tools. Two ASCII formats have been defined: a text special-purpose format and an XML-based format.
- Graphical editors and other tools generate the system using one of these ASCII descriptions. They can then invoke the analysis tools.
- A parser converts any of the ASCII descriptions of the system into an Ada data structure that is used by the tools. A module is offered to convert the Ada data structure back into the chosen ASCII description.
- The XML format provides the designer with capabilities to use free standard XML tools to validate, parse, analyse, and display the model files.
- A results viewer is available to view the analysis results in a convenient way.

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

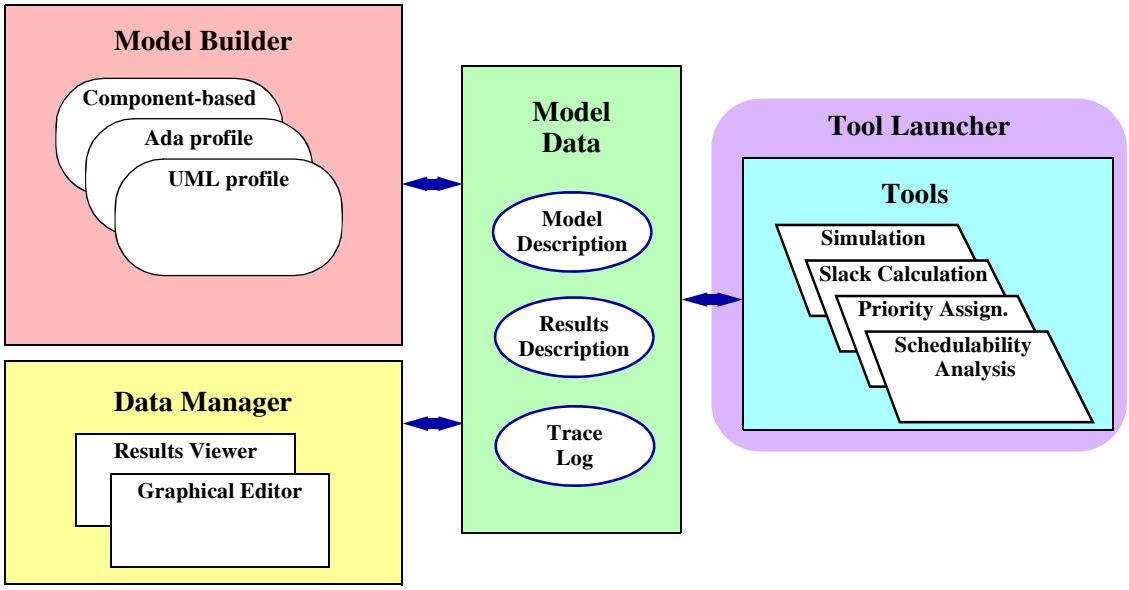


Figure 1. MAST toolset environment

- The schedulability 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 priority assignment tools are able to make an automatic assignment of priorities and priority ceilings, using optimum priority assignments when available, and heuristics or optimization techniques when the optimum assignment is not available.

- The slack calculation tools calculate the system, processing resource or transaction slacks by repeating the analysis in a binary search algorithm in which execution times are successively increased or decreased.
- The simulation tools are able to simulate the behaviour of the system to check soft timing requirements and generate temporal traces of the simulated execution.

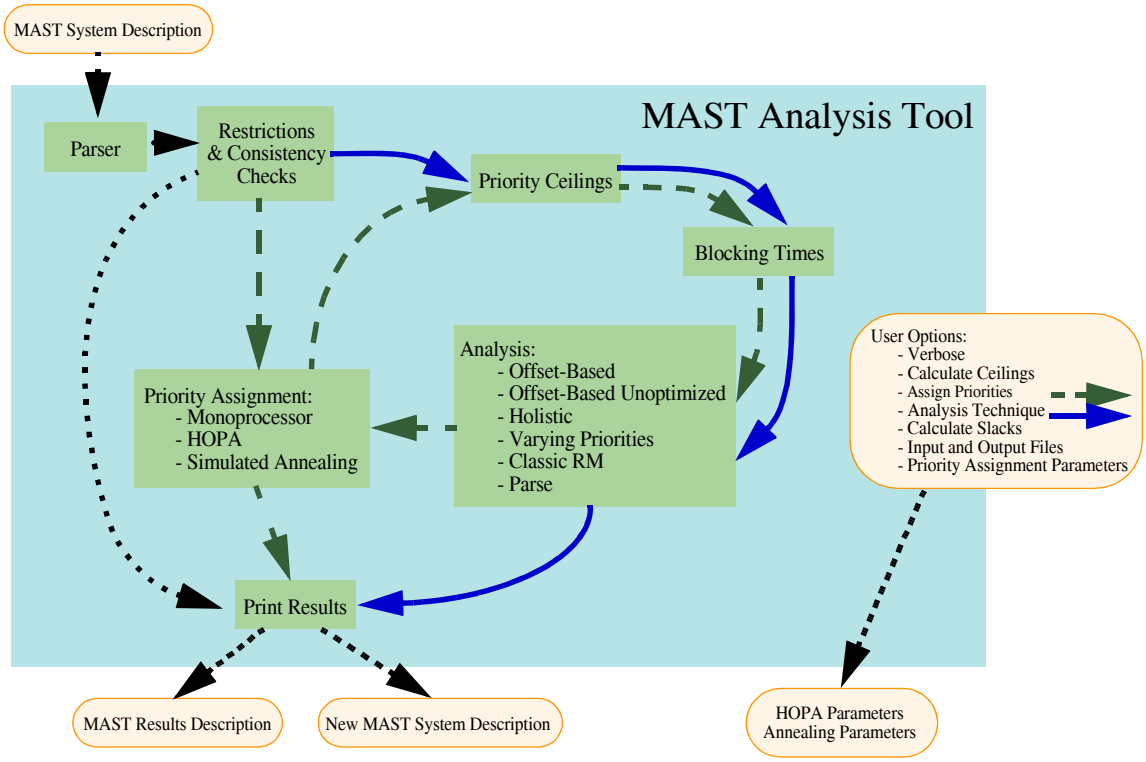


Figure 2. MAST Analysis tools

Using a standard CASE UML tool, it is possible to describe the real-time behaviour of the system by means of a set of appropriate UML modeling primitives that are defined in different profiles in accordance with the technology used to design the real-time system (object oriented, Ada language, component based, etc.). Then, an automatic tool is used to compile the UML real-time view and to build the MAST real-time model description. No special framework is needed with this approach, but the designer must incorporate the real-time view into the UML description. Refer to the UML-MAST project page[1] for more information of this issue.

Figure 2 represents the MAST toolset. The capabilities of the currently implemented tools are represented in the following tables. The tools with dark shading are still under development.

1.http://mast.unican.es/umlmast/

Table 1. Fixed-priority schedulability analysis tools

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	☑	☑	☑	☑	☑

Table 2. EDF schedulability analysis tools

Technique	Single-Processor	Multi-Processor	Simple Transact.	Linear Transact.	Multiple Event T.
Single Processor	☑		☑		
EDF_Within_Priorities	☑		☑		
Holistic	☑	☑	☑	☑	
Offset Based	☑	☑	☑	☑	

3. Real-Time System Model

A real-time situation is modelled as a set of concurrent transactions that compete for the resources offered by the platform. 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

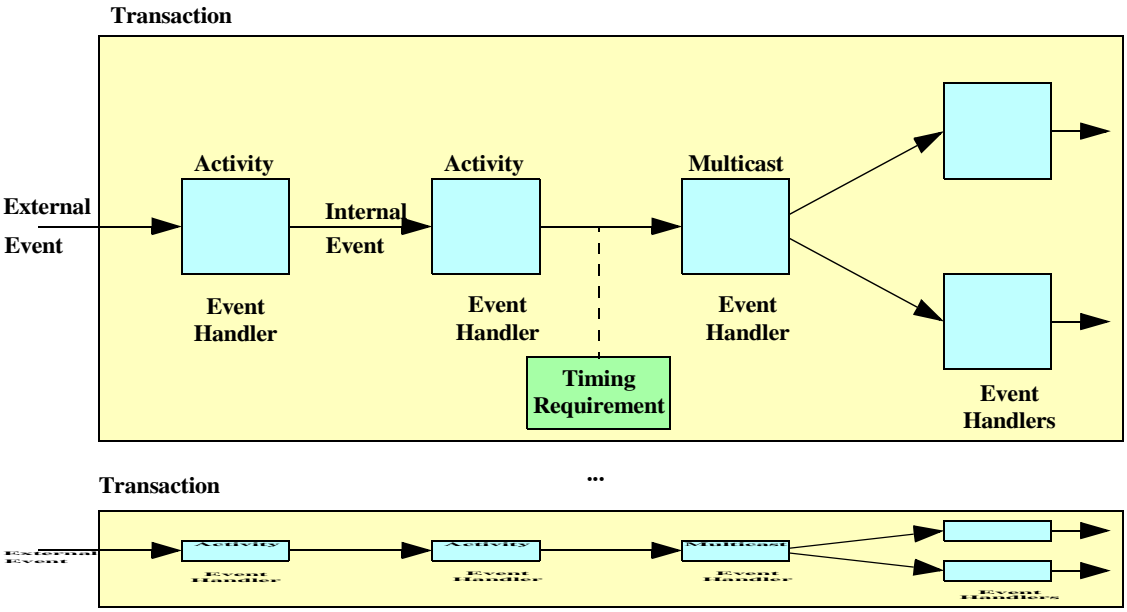


Figure 3. Real-Time System composed of transactions

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 *Scheduler* to which it is assigned. This scheduler belongs to a *Processor* or a *Network*, although we will see that when hierarchical scheduling is modelled the situation is somehow more complex. For example, the model for a scheduling server in a processor is a task or thread. A thread 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.

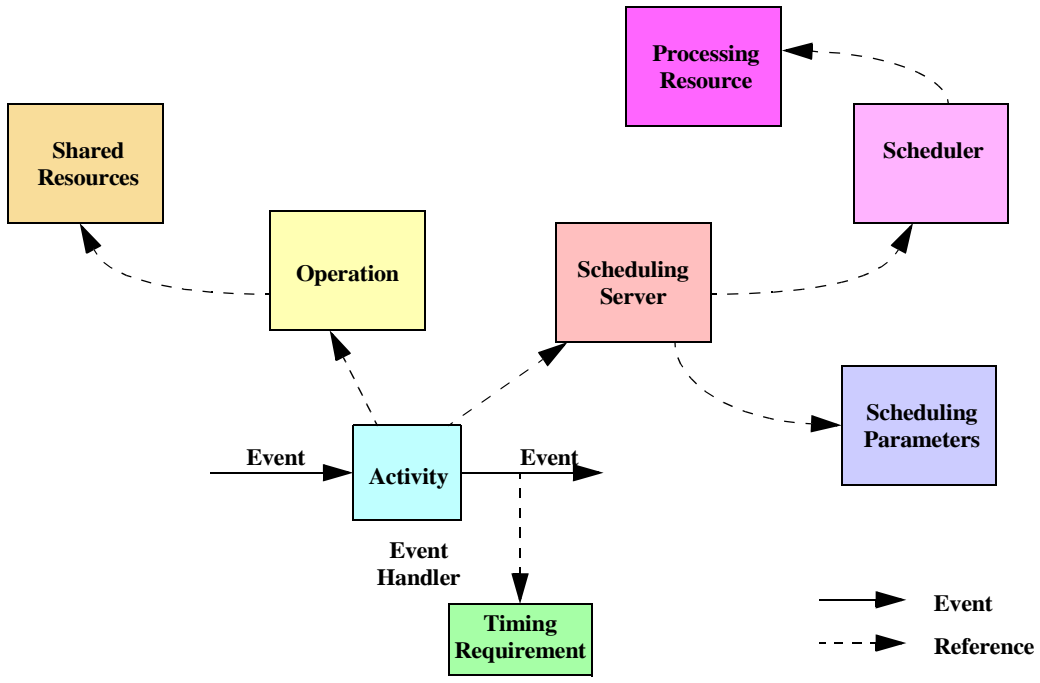


Figure 4. Elements that define an activity

Some of the attributes of MAST objects are not part of the real-time model but are information that is relevant to some of the analysis tools. In that case the attribute name is identified with a tool-specific prefix. The only prefix that is currently defined is:

- *RTA_*: This prefix identifies a tool-specific attribute that is applicable only to the worst-case response time analysis tools.

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 analysed 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.
- *Preemption_Level*. Natural integer of implementation-defined range, defining the preemption level of scheduling servers (threads) and shared resources, used in the SRP protocol for mutually exclusive access to shared resources.
- *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*. Floating point number that represents the amount of processing resource capacity that is required for the execution of an operation. It is expressed as the execution time of an operation, when it is 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.
- *Bit_Count*. Floating point number that represents the length of a message in bits of information. It is converted to normalized execution time (i.e., transmission time) by dividing it by the *throughput*, measured in bits per time unit.

- *Throughput*. Floating point number that represents the transmission bandwidth of a communication network in bits per time unit.
- *Time*. Floating point number that represents a time interval in unspecified time units.
- *Absolute_Time*. Floating point number that represents an 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 with the special-purpose format

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 optional, except when marked.
- 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).
- 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.

There is now need to define an identifier before it is used, as opposed to what happened in previous versions.

In Appendix A, we describe the syntax and rules for writing the Mast files with the XML-Mast format.

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
- Schedulers (primary scheduler, secondary schedulers,....)
- Scheduling Policies (fixed priorities, EDF,...)
- Scheduling parameters (priorities, deadlines,...)
- Synchronization parameters (preemption levels,...)
- 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

They model the processing capacity of a hardware component that executes some of the modelled system activities, which are generally pieces of code or messages to be transferred.

Common attributes:

- *Name*. A string.
- *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: there is an abstract class, called *Processor*, that models a device capable of executing pieces of application code; another abstract class, called *Network*, models a communication system specialized in the transmission of messages among processors. One concrete class is defined for each of these abstract resources:

- *Regular Processor*[1]. It represents a physical processor with the basic overheads associated with its hardware interrupt services. 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. Their default values are the maximum and minimum values of the *Any_Priority* type, respectively.
- *ISR Switch Overheads* (Worst, Average, Best).
- *System Timer*. A reference to the hardware system timer used (see below), that influences the overhead of the *System Timed Activities* and specifies the processor overhead implicitly introduced by the scheduler or the operating system to implement its own time management services. The default value is no system timer.

```
Processing_Resource (
    Type                          => Regular_Processor,
    Name                          => Identifier,
    Speed_Factor                  => Float,
    Worst_ISR_Switch              => Normalized_Execution_Time,
    Avg_ISR_Switch                => Normalized_Execution_Time,
    Best_ISR_Switch               => Normalized_Execution_Time,
    Max_Interrupt_Priority        => Interrupt_Priority,
    Min_Interrupt_Priority        => Interrupt_Priority,
    System_Timer                  => System_Timer);
```

- *Packet Based Network*[1]. It represents a network that uses some kind of real time protocol based on non-preemptible packets 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). A network has the following additional attributes:
  - *Transmission kind*: *Simplex*, *Half Duplex*, of *Full Duplex*. The default value is *Half_Duplex*.
  - *Throughput*: Normalized network bandwidth in bits per time unit. The actual network throughput is affected by the *Speed Factor*. The default value is 0 bits per time unit.
  - *Max Blocking*. The maximum blocking that may be experienced before a high priority message can be transmitted, caused by the non preemptability of message packets, including both the application message and the protocol information sent with it. It usually has a value equal to the maximum packet transmission time plus the transmission time of the protocol information. Its default value is zero, indicating a negligible network blocking.
  - *Max Packet Size* and *Min Packet Size*. They describe the amount of data included in a packet, excluding any protocol information. The maximum size is used in the calculation of the number of packets into which a large message is split, calculated as the ceiling of the message size divided by the maximum packet size. This number is multiplied by the packet overhead time of the scheduler to calculate the

1. Another class of processor called the “Fixed_Priority_Processor” is supported. It includes a scheduler with a fixed priority scheduling policy and is defined only for backwards compatibility with MAST 1.2.

1. Another class of network called the “Fixed_Priority_Network” is supported. It includes a scheduler with a fixed priority scheduling policy and is defined only for backwards compatibility with MAST 1.2.

network overhead. The Minimum size is used to calculate the shortest period of the overheads associated with 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. Their default values are the maximum value of the *Bit_Count* type.

- *Max Packet Transmission Time* and *Min Packet Transmission Time*. These parameters represent another option for specifying the *Max Packet Size* and the *Min Packet Size*, but using time units instead of a bit count. The size is obtained by multiplying the time by the *Throughput*. If these parameters appear in the description, the associated size parameters should not be present. Their default values are a very large time value.
- *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. The default is an empty list.

```
Processing_Resource (
     Type                                => Packet_Based_Network,
     Name                                => Identifier,
     Speed_Factor                        => Float,
     Throughput                          => Float,
     Transmission                        => Simplex | Half_Duplex | Full_Duplex,
     Max_Blocking                        => Normalized_Execution_Time,
     Max_Packet_Size                     => Bit_Count,
     Min_Packet_Size                     => Bit_Count,
     Max_Packet_Transmission_Time        => Normalized_Execution_Time,
     Min_Packet_Transmission_Time        => Normalized_Execution_Time,
     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. Their default values are zero.

```
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 a priority level higher than the highest interrupt priority. Their default values are zero.
  - *Period*. Period of the ticker interrupt. Its default value is a very large time.

```
System_Timer = (
    Type                          => Ticker
    Worst_Overhead                => Normalized_Execution_Time,
    Avg_Overhead                  => Normalized_Execution_Time,
    Best_Overhead                 => Normalized_Execution_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 three 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), or a reference to it.
  - *Packet Send Operation*. The operation that is executed by the *packet server* each time a packet is sent, or a reference to it.
  - *Packet Receive Operation*. The operation that is executed by the *packet server* each time a packet is received, or a reference to it.
  - *Message Partitioning*. A "Yes|No" value that determines whether or not the driver is capable of partitioning long messages into several packets and rebuilding the message at the other end. This attribute influences the overhead model of the driver. The default value is "Yes".
  - *RTA Overhead Model*. This is a tool-specific attribute that applies only to the worst-case response time analysis tools. A value that determines the overhead model that should be used for the driver. Currently two values are supported: coupled or decoupled. In the coupled overhead model a message send and a message receive operation are attached to the transaction that causes the transmission. In the decoupled overhead model the send operation and receive operation are executed periodically, with a period equal to the minimum packet transmission time (or some other driver-dependent time). Both models are pessimistic, but the coupled model is more appropriate for systems with low network utilization, while the decoupled model is more appropriate for systems with high network utilization and few messages being partitioned (i.e., mostly short messages). The default value is "Decoupled".


```
Driver = (
     Type                              => Packet_Driver,
     Packet_Server                     => Scheduling_Server | Identifier,
     Packet_Send_Operation             => Operation | Identifier,
     Packet_Receive_Operation          => Operation | Identifier,
     Message_Partitioning              => Yes | No,
     RTA_Overhead_Model                => Coupled | Decoupled)
```

- *Character Packet Driver*. It is a specialization of a packet driver in which there is an additional overhead associated with 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), or a reference to it.
  - *Character Send Operation*. The operation that is executed by the *character server* each time a character is sent, or a reference to it.
  - *Character Receive Operation*. The operation that is executed by the *character server* each time a character is received, or a reference to it.
  - *Character Transmission Time*. Time of character transmission. It determines the period used for the overhead models of the character send and character receive operations.

```
Driver = (
     Type                              => Character_Packet_Driver,
     Packet_Server                     => Scheduling_Server | Identifier,
     Packet_Send_Operation             => Operation | Identifier,
     Packet_Receive_Operation          => Operation | Identifier,
     Message_Partitioning              => Yes | No,
     RTA_Overhead_Model                => Coupled | Decoupled,
     Character_Server                  => Scheduling_Server | Identifier,
     Character_Send_Operation          => Operation | Identifier,
     Character_Receive_Operation       => Operation | Identifier,
     Character_Transmission_Time       => Time)
```

- *RT-EP Packet Driver*. It is a specialization of a packet driver that characterizes the Real-Time Ethernet Protocol, RTEP[1]. This is a token-passing protocol with prioritized messages, that uses a two-phase mechanism to send each information packet: in first place there is a priority arbitration phase in which a token is rotated through each station or processing node to determine which is the node with the highest priority message; in the second phase that node is granted permission to transmit a packet with information. Compared to a regular packet driver, there are additional overheads associated with the transmission and reception of the tokens, as well as with the error recovery mechanisms. In the *Decoupled RTA Overhead Model*, the period of the overhead transactions used to model the message send and receive operations is not the minimum packet transmission

1. See: J.M. Martínez, M. González Harbour, and J.J. Gutiérrez. “RT-EP: Real-Time Ethernet Protocol for Analyzable Distributed Applications on a Minimum Real-Time POSIX Kernel”. Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age, RTLIA 2003, Porto (Portugal), July 2003.

time, as with the *Packet_Driver*, but rather a combination of several attributes. See the RT-EP documentation for a description of the overhead model. The attributes of the *RT-EP Packet Driver* are those of a *Packet Driver* plus the following:

- *Number of stations*: The number of stations or processors connected with the RTEP network. This attribute is used to determine the token rotation time and several overhead values. The default value is the largest integer (Integer'Last).
- *Token delay*: The configurable delay introduced during the handling of a token to slow the token transmission time in order to bound the overhead of each of the processors connected to the RT-EP network. The default value is zero, which implies maximum overhead and minimum latency.
- *Failure Timeout*: This is the configurable timeout used to determine that there has been a packet loss due to a failure in the network. The default value is a large time, which would imply no error recovery.
- *Token transmission retries*: Maximum number of retransmissions that we allow for each packet containing a protocol token. The default value is zero.
- *Packet transmission retries*: Maximum number of retransmissions that we allow for each packet with user information. The default value is zero.
- *Packet interrupt server*: The scheduling server that is executing the interrupt service routine that handles each incoming packet, independently of whether it is a packet with information or a protocol token. It may be the scheduler server itself, or a reference to an external scheduling server.
- *Packet ISR operation*: The operation executed by the packet interrupt server for each incoming packet, independently of whether it is a packet with information or a protocol token.
- *Token check operation*: The operation executed by the *packet server* to receive and check a token packet, or a reference to it.
- *Token manage operation*: The operation executed by the *packet server* to send a token, or a reference to it.
- *Packet discard operation*: The operation executed by the *packet server* when a packet is received that is addressed to another processing node, or a reference to it.
- *Token retransmission operation*: The operation executed by the *packet server* when a lost token is retransmitted, or a reference to it.
- *Packet retransmission operation*: The operation executed by the *packet server* when a packet with information that was lost is retransmitted, or a reference to it.

```
Driver = (
     Type                              => RTEP_Packet_Driver,
     Packet_Server                     => Scheduling_Server | Identifier,
     Packet_Send_Operation             => Operation | Identifier,
     Packet_Receive_Operation          => Operation | Identifier,
     Message_Partitioning              => Yes | No,
     RTA_Overhead_Model                => Coupled | Decoupled,
     Number_Of_Stations                => Integer,
     Token_Delay                       => Time,
```

```
    Failure_Timeout                 => Time,
    Token_Transmission_Retries      => Integer,
    Packet_Transmission_Retries     => Integer,
    Packet_Interrupt_Server         => Scheduling_Server | Identifier,
    Packet_ISR_Operation            => Operation | Identifier,
    Token_Check_Operation           => Operation | Identifier,
    Token_Manage_Operation          => Operation | Identifier,
    Packet_Discard_Operation        => Operation | Identifier,
    Token_Retransmission_Operation  => Operation | Identifier,
    Packet_Retransmission_Operation => Operation | Identifier)
```

7.4 Schedulers

They represent the operating system objects that implement the appropriate scheduling strategies to manage the amount of processing capacity that has been assigned to them.

Schedulers can have a hierarchical structure, like the one shown in Figure 5. A *Primary Scheduler* operates by offering the whole processing capacity of its associated base processor to its associated scheduling servers. A *Secondary Scheduler* is allowed to deliver to its scheduling servers just the processing capacity that it receives from its associated scheduling server, scheduled in turn by some other scheduler.

Common attributes:

- *Name*. A string.
- *Policy*. It defines the scheduling policy that is implemented by the scheduler to deliver the underlying processing capacity among the scheduling servers that are associated with it. It is a mandatory attribute.

There are two classes of schedulers, according to the source of processing capacity:

- *Primary Scheduler*. It represents the base system scheduler for its associated processing resource. It has the following additional attribute:
  - *Host*. A reference to the processing resource.

```
Scheduler (
    Type                            => Primary_Scheduler,
    Name                            => Identifier,
    Policy                          => Scheduling_Policy,
    Host                            => Identifier); --Processing_Resource
```

- *Secondary Scheduler*. It represents a scheduler that is able to handle only a certain fraction of processing capacity, which in turn is served by a particular scheduling server through another scheduler associated to it. It has the following additional attribute:
  - *Server*. A reference to the scheduling server.

```
Scheduler (
    Type                            => Secondary_Scheduler,
    Name                            => Identifier,
    Policy                          => Scheduling_Policy,
    Server                          => Identifier); --Scheduling_Server
```

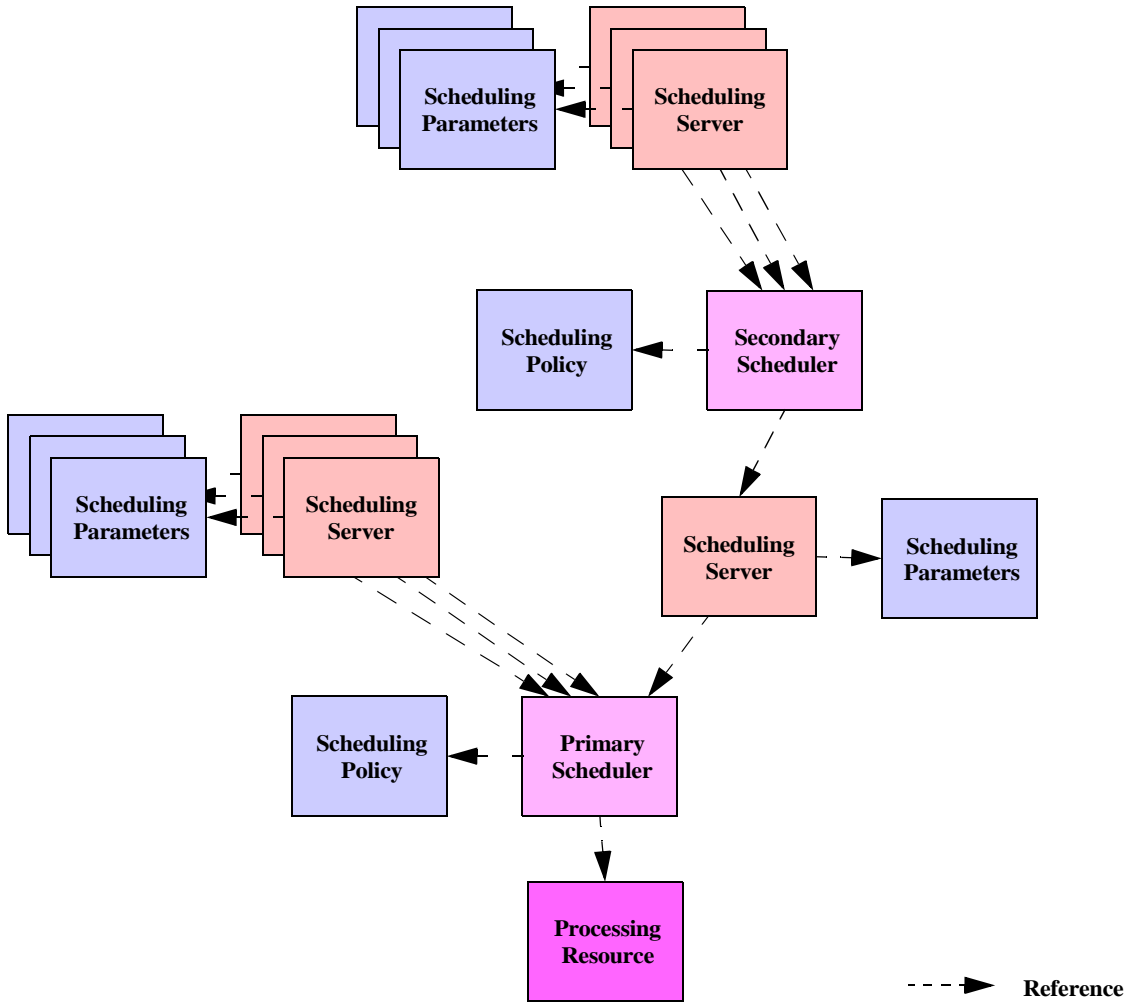


Figure 5. Hierarchical scheduler structure

7.5 Scheduling policies

A scheduling policy represents the basic strategy that is implemented in a scheduler to deliver the assigned processing capacity to its associated scheduling servers. Each of these servers has a *Scheduling_Parameters* object that describes the parameters used by the scheduler.

Classes of Scheduling Policies:

- *Fixed Priority*. It represents a fixed-priority policy. It has the following attributes:
  - *Context Switch Overheads* (Worst, Average, Best). Their default values are zero.
  - *Max Priority* and *Min Priority*. They define the range of priorities that are valid for normal operations on scheduling servers scheduled with this policy. Special operations (such as interrupt service routines in processors) may have other priority ranges. The default values are respectively the maximum and minimum values of the *Any_Priority* type.

```
Scheduling_Policy (
    Type                                => Fixed_Priority,
```

```
    Worst_Context_Switch          => Normalized_Execution_Time,
    Avg_Context_Switch            => Normalized_Execution_Time,
    Best_Context_Switch           => Normalized_Execution_Time,
    Max_Priority                  => Priority,
    Min_Priority                  => Priority);
```

- *EDF*. It represents an Earliest Deadline First policy. It has the following attributes:
  - *Context Switch Overheads* (Worst, Average, Best). Their default values are zero.

```
Scheduling_Policy (
    Type                          => EDF,
    Worst_Context_Switch          => Normalized_Execution_Time,
    Avg_Context_Switch            => Normalized_Execution_Time,
    Best_Context_Switch           => Normalized_Execution_Time);
```

- *Fixed Priority Packet Based*. It represents a fixed priority policy used in a packet oriented communication network. Network packets are assumed to be non preemptible. It has the following attributes:
  - *Packet Overhead* (Worst, Average, Best). This is the overhead associated with sending each packet, because of the protocol messages or headers that need to be sent before or after each packet. There are two ways of specifying the overheads, one using transmission time, and another one using a bit count (which is then translated to time using the throughput attribute of the network associated to the scheduler that uses this policy). The default values for these attributes are zero.
  - *Max Priority* and *Min Priority*. They define the range of priorities valid for messages to be sent using this policy. The default values are respectively the maximum and minimum values of the *Any_Priority* type.

```
Scheduling_Policy (
    Type                          => FP_Packet_Based,
    Packet_Worst_Overhead         => Normalized_Execution_Time,
    Packet_Avg_Overhead           => Normalized_Execution_Time,
    Packet_Best_Overhead          => Normalized_Execution_Time,
    Packet_Overhead_Max_Size      => Bit_Count,
    Packet_Overhead_Avg_Size      => Bit_Count,
    Packet_Overhead_Min_Size      => Bit_Count,
    Max_Priority                  => Priority,
    Min_Priority                  => Priority);
```

7.6 Scheduling parameters

These parameters are attached to a scheduling server and are used by its scheduler to make its scheduling decisions when the server is competing with other servers. Some scheduling policies allow several compatible scheduling behaviours to coexist in the system. For example, the fixed priority scheduling policy allows both preemptive and non preemptive activities. These different scheduling behaviours are determined by the scheduling parameters type and are also called the per-server policy parameters.

There are several classes of scheduling parameters. There are some restrictions on the compatibility between scheduling parameters and the scheduling policy of the associated

scheduler. The scheduling parameters described in Section 7.6.1 are only applicable to schedulers with the fixed priority policy. Those described in Section 7.6.2 are only applicable to schedulers with the EDF policy.

7.6.1 Fixed priority scheduling parameters

All the fixed priority scheduling parameters have the following set of common attributes:

- *Priority*. A natural number that represents the scheduling priority. It must be within the valid ranges for the associated scheduler. Its default value is the minimum value of the *Any_Priority* type.
- *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 in the MAST description.

The classes defined are:

- *Non Preemptible Fixed Priority Policy*. Activities scheduled with these parameters are non preemptible. No additional attributes.

```
Sched_Parameters = (
     Type                              => Non_Preemptible_FP_Policy,
     The_Priority                      => Priority,
     Preassigned                       => Yes | No)
```

- *Fixed Priority Policy*. Represents a regular preemptive fixed priority parameters object. No additional attributes.

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

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

```
Sched_Parameters = (
     Type                              => Interrupt_FP_Policy,
     The_Priority                      => Interrupt_Priority,
     Preassigned                       => Yes)
```

- *Polling Policy*. Represents the scheduling mechanism by which there is 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. Its default value is zero.
  - *Polling Overhead* (Worst, Average, Best). Overhead of the polling task. Their default values are zero.

```
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 Policy*. 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. Its default value is the minimum value of the *Priority* type.
  - *Initial Capacity*. Its the initial value of the execution capacity. Its default value is zero.
  - *Replenishment Period*. It is the period after which a portion of consumed execution capacity is replenished. Its default value is zero.
  - *Max Pending replenishments*. It is the maximum number of simultaneously pending replenishment operations. Its default value is one.

```
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)
```

Part of the fixed priority scheduling parameters may also be overridden for a particular operation, by including an overridden scheduling parameters object in its definition.

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

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

7.6.2 EDF Scheduling Parameters

All the EDF scheduling parameters have the following set of common attributes:

- *Deadline*. Relative deadline of the associated scheduling server. Its default value is a Large Time.

- *Preassigned*. If this parameter is set to the value "*No*", the deadline may be assigned by one of the deadline assignment tools. Otherwise, it is fixed and cannot be changed by those tools. Its default value is "*No*" if no deadline field appears in the MAST description, and "*Yes*" if any of them is present.

There is only one class of EDF scheduling parameters defined at the moment:

- *Earliest Deadline First Policy*. Represents a task scheduled according to the "earliest deadline first" policy. It has no additional attributes.

```
Sched_Parameters = (
    Type                                => EDF_Policy,
    Deadline                            => Time,
    Preassigned                         => Yes | No)
```

7.7 Synchronization Parameters

These parameters are attached to a scheduling server to specify the parameters used by that server when performing a mutually exclusive access to shared resources.

The synchronization parameters should be specified whenever the scheduling policy is such that the given *Scheduling Parameters* do not have enough information for the synchronization protocols used. For example, no synchronization parameters are needed for the priority inheritance or immediate priority ceiling protocols, because the only information they require from the server is its priority.

The only class defined for the synchronization parameters is named *Stack Resource Protocol Parameters* because it is associated with that synchronization protocol. Its attributes are:

- *Preemption Level*. A natural number that represents the level of preemptability of the scheduling server in relation to the shared resource. It must be within the valid ranges for the implementation. Its default value is the minimum value of the *Preemption_Level* type.
- *Preassigned*. If this parameter is set to the value "*No*", the value of the preemption level may be assigned by any of the specific design tools. Otherwise, it is fixed and cannot be changed by those tools. Its default value is "*No*" if no preemption level field appears in the MAST description, and "*Yes*" if it is present.

```
Synch_Parameters = (
    Type                                => SRP_Parameters,
    Preemption_Level                    => Preemption_Level,
    Preassigned                         => Yes | No)
```

7.8 Scheduling Servers

They represent schedulable entities in a processing resource. There is only one class defined, named *Regular*[1]. Its attributes are:

1. For backwards compatibility with MAST 1.2 and previous versions, there is a different way to describe the scheduling server using the *Fixed_Priority* type.

- *Name*
- *Scheduling Parameters*. Object with the parameters that are needed by the scheduler to apply the corresponding scheduling policy.
- *Synchronization Parameters*. Object with the parameters that are needed by the scheduler to apply the corresponding synchronization protocol in the mutually exclusive access to shared resources. It should be present whenever the synchronization protocols used by the server need more information than the one available in the scheduling parameters.
- *Scheduler*. Reference to the scheduler that serves it.

```
Scheduling_Server (
    Type                                => Regular,
    Name                                => Identifier,
    Server_Sched_Parameters             => Sched_Parameters,
    Synchronization_Parameters          => Synch_Parameters,
    Scheduler                           => Identifier);
```

7.9 Shared Resources

They represent resources that are shared among different threads or tasks, and that must be used in a mutually exclusive way. Only protocols that avoid unbounded priority inversion are allowed. There are three 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. Its default value is the maximum value of the *Any_Priority* type.
  - *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);
```

- *Stack Based Resource*. Uses the Stack Resource Protocol (SRP). This is similar to the immediate ceiling protocol but works for non-priority-based policies. Its attributes are:
  - *Name.*
  - *Preemption Level*. Level of preemptability used for the resource. May be computed automatically by the MAST tools, upon request. Its default value is the maximum value of the *Preemption_Level* type.
  - *Preassigned*. If this parameter is set to the value "*No*", the preemption level may be assigned by a tool. Otherwise, the priority ceiling is fixed and cannot be changed by any tool. Its default value is "*No*" if no preemption level field appears in the MAST description, and "*Yes*" if a preemption level field appears.

```
Shared_Resource (
     Type                              => SRP_Resource,
     Name                              => Identifier,
     Preemption_Level                  => Preemption_Level,
     Preassigned                       => Yes | No);
```

7.10 Operations

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

- *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:
  - *Execution Time* (Worst, Average and Best). In normalized units. For messages, this represents the transmission time. The default values are very large time values for the worst and average, and zero for the best execution time.
  - *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:
  - *Execution Time* (Worst, Average and Best). In normalized units. For messages, this represents the transmission time. The default values are very large time values for the worst and average, and zero for the best execution time.
  - *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,
                                      ...));
```

- *Message_Transmission*. Represents a message to be transmitted through a network. Its additional attributes are:
  - *Message Size* (Worst, Average and Best). In *Bit_Count* units. The transmission time is obtained by dividing the size by the *Throughput* and by the *Speed_Factor* of the corresponding Network. The default values are very large *Bit_Count* values for the worst and average, and zero for the best message size.

```
Operation (
    Type                              => Message_Transmission,
    Name                              => Identifier,
    Overridden_Sched_Parameters       => Overridden_Sched_Parameters,
    Max_Message_Size                  => Bit_Count,
    Avg_Message_Size                  => Bit_Count,
    Min_Message_Size                  => Bit_Count);
```

7.11 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 behaviour 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. Its default value is zero.

- *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. Its default value is zero.
- *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). Its default value is zero.

```
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. Its default value is zero.

```
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. Its default value is zero.
  - *Average Interarrival*. Average interarrival time. Its default value is zero.
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* (default value) 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. Its default value is zero.
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* (default value) 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. Its default value is zero.
  - *Max_Arrivals*. Maximum number of events that may arrive in the *Bound_Interval*. Its default value is one.
  - *Average Interarrival*. Average interarrival time. Its default value is zero.
  - *Distribution*. It represents the distribution function of the aperiodic events. It can be *Uniform* (default value) or *Poisson*.

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

7.12 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* (default=0), and a reference to the *Referenced Event*.

- *Soft Global Deadline*. Attributes are the value of the *Deadline* (default=0), and a reference to the *Referenced Event*.
- *Hard Local Deadline*. The only attribute is the value of the *Deadline* (default=0).
- *Soft Local Deadline*. The only attribute is the value of the *Deadline* (default=0).

```
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 (default=0).
  - *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 (default=0).
  - *Ratio*. Percentage representing the maximum ratio of missed deadlines (default=5%).

  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,
                                   ...))
```

7.13 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 6 shows the different classes of events.

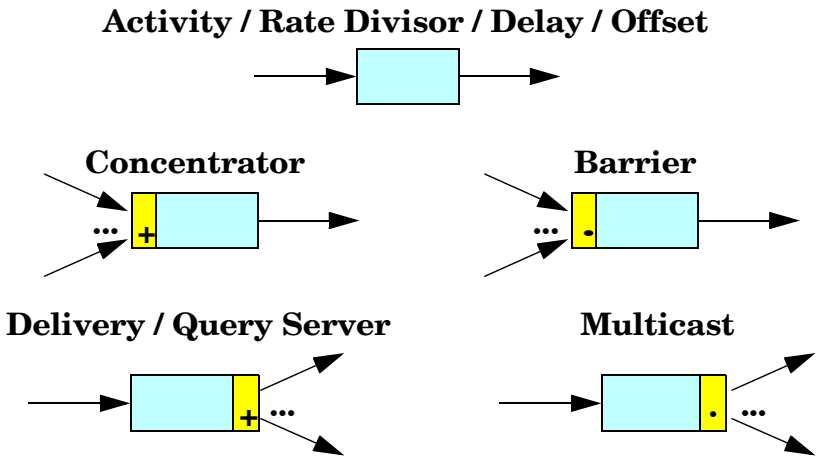


Figure 6. Classes of Event Handlers

- *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,
    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* (default value).

```
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*. The default value is *Scan*.

```
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. Its default value is one.

```
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. Its default value is zero.
  - *Delay Min Interval*. Shortest time interval used to generate the output event. Its default value is zero.

```
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)
```

7.14 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,
                                   ...));
```

7.15 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 analysed. Global information about the real-time situation and the underlying implementation 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.
- System PiP Behaviour: the behaviour of the underlying implementation in regard to the priority inheritance protocol. It can be STRICT, when the implementation strictly follows the original priority inheritance protocol, or POSIX, which allows a more relaxed implementation that can lead to longer blocking times. The default value is STRICT.

```
Model (
     Model_Name                    => Identifier,
     Model_Date                    => YYYY-MM-DDThh:mm:ss,
     System_PiP_Behaviour[1]       => STRICT | POSIX);
```

8. Templates for the MAST File

In the text special-purpose format, the structure of a Mast model file is the following:

```
-- 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,
                                         ...));
```

1. The alternative spelling **System_PiP_Behavior** is also supported

```
      -- 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 text special-purpose format of the results file is described next, and appendix A describes the corresponding XML-Mast Format.

In the text format, the *results file* 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 analysed, and contain a set of results (described below) and the following attributes:

- *Model_Name*: Name of the analysed 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 analysed, 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 analysed, 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 analysed, 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 analysed, 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 analysed system. Usually they are only written to the file if they were automatically calculated by the scheduling parameters assignment tools. See section on “Scheduling Parameters” for a description of their format.

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

- *Synchronization_Parameters*: The synchronization parameters that were used in the analysed system. Usually they are only written to the file if they were automatically calculated by the scheduling parameters assignment tools. See section on “Synchronization Parameters” for a description of their format.

```
Result = (
    Type                             => Synchronization_Parameters,
    Server_Synch_Parameters          => Synch_Parameters)
```

9.6 Shared Resource

The shared resource results are relative to a shared resource in the system that has been analysed, 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)
```

- *Preemption Level*: The preemption level automatically calculated by the MAST tool. Only shared resources of the type *SRP_Resource* may have this type of result.

```
Result = (
    Type                             => Preemption_Level,
```


```
     Level                              => Preemption_Level)
```

- *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. Traces File Format

The traces that are generated by some of the Mast tools are stored in the *traces file.* They contain four data block:

- the real-time situation description:
- Message types description.
- Message sources description.
- Messages list.

The overall system data are relative to a real-time situation that has been analysed, to the tools that has generate the trace and to the time range. It contains the following attributes:

- *Model_Name*: Name of the analysed 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*: Text representing the name of the tool that generated the traces.
- *Generation_Profile*: Text representing the command and options used to invoke the tool for the generation of the traces.
- *Generation_Date*: Date of generation of traces, in the ISO 8601 format `YYYY-MM-DDThh:mm:ss`.
- *Init-Time*: Start time of generation of traces, in the experiment scale time.
- *End-Time*: End time of generation of traces, in the experiment scale time

```
TRACE_FILE (
     Model_Name                         => Identifier,
```

```
Model_Date                      => YYYY-MM-DDThh:mm:ss,
Generator_Tool                  => "Text",
Generation_Profile              => "Text",
Generation_Date                 => YYYY-MM-DDThh:mm:ss,
Init_Time                       => Float,
End_Time                        => Float,
Msg_Type_List                   => (
                                Msg_Type 1,
                                Msg_Type 2,
                                ...)
Src_List                        => (
                                Src 1,
                                Src 2,
                                ...)
Msg_List                        => (
                                Msg 1,
                                Msg 2,
                                ...));
```

The list included in a traces register of a real-time situation traces are:

- *Msg_Type_List*: List of types of messages referenced in the traces register. Each message type is described by:
  - *Mid*: Message type identification that is a natural number.
  - *Type*: Explanatory text of the message type (it is optional).

```
Msg_Type = (
    Mid                         => Natural,
    Type                        => String)
```

- *Src_List*: List of objects of the real-time situation model that are able to generate messages in the trace. Each source object is described by:
  - *Sid*: Source identification that is an integer number.
  - *Name*: Textual identifier of the source object in the model (it is optional).
  - *Type*: Type of the object in the model domain.

```
Src = (
    Sid                         => Integer,
    Name                        => Identifier,
    Type                        => String)
```

- *Msg_List*: List of timed message that build up the traces register. Each message item is described by:
  - *Time*: Time of message generation in the experiment scale time.
  - *Sid*: Identifier of source object that has generate the message.
  - *Mid*: Identifier of the type of the message.


```
Msj = (
      Time                              => Float,
      Sid                               => Integer,
      Mid                               => Natural)
```

11. 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 7.

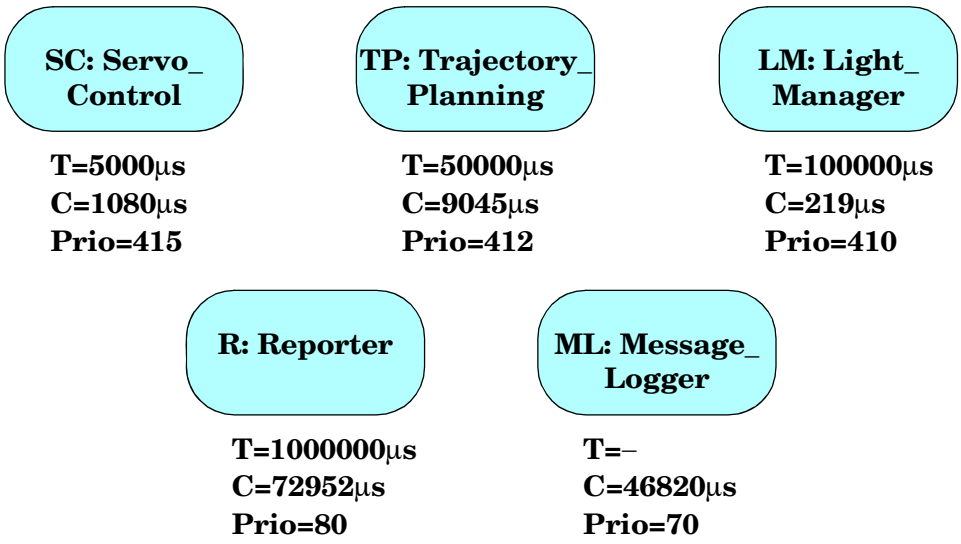


Figure 7. Basic Characteristics of the tasks of the CASEVA controller

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

Shared Resource	Operation	WCET (μs)	Used by
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)));
```

12. 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 behaviour 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 8 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 behaviour.

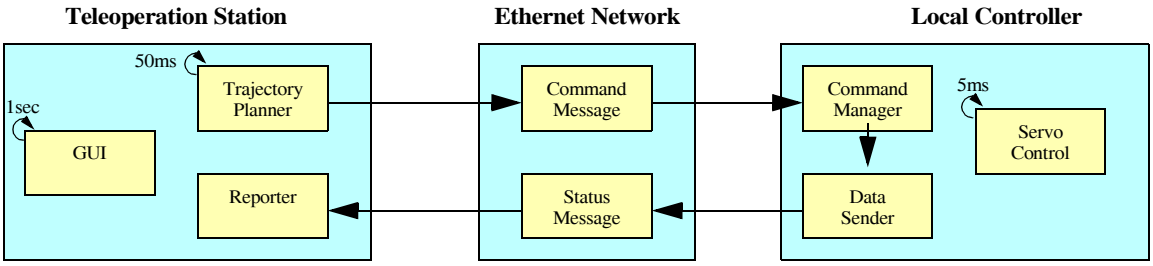


Figure 8. Architecture of the teleoperated robot controller

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 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));
```

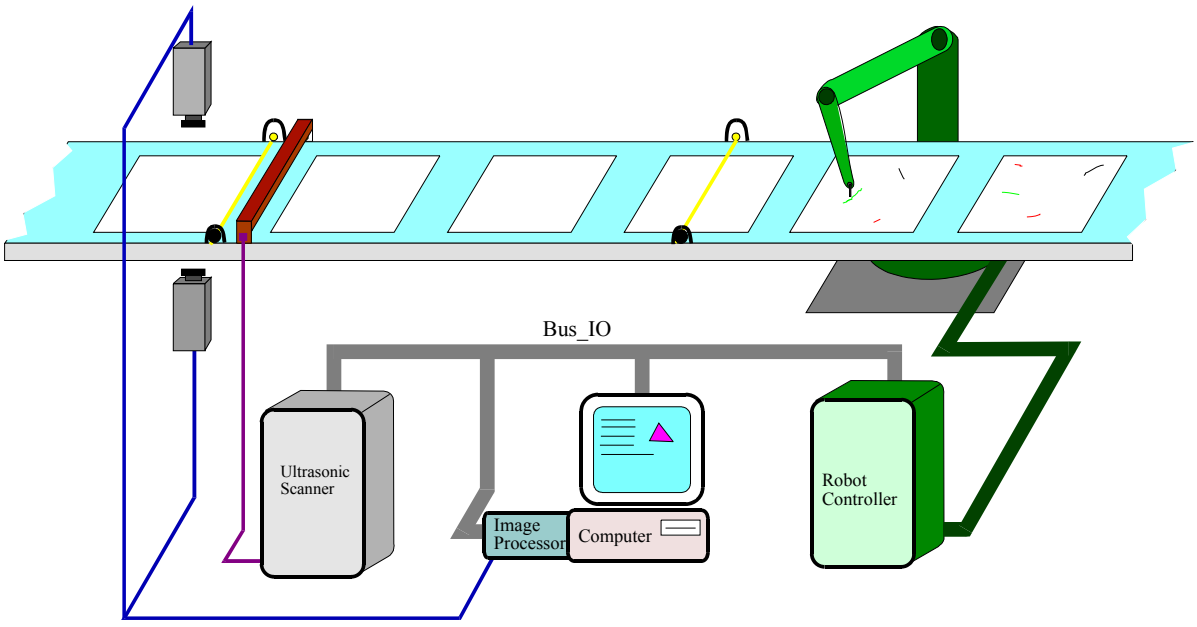
```
Operation (
     Type    => Enclosing,
     Name    => Reporter,
     Worst_Case_Execution_Time => 2086,
     Composite_Operation_List =>
            (Write_Status));

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),
```

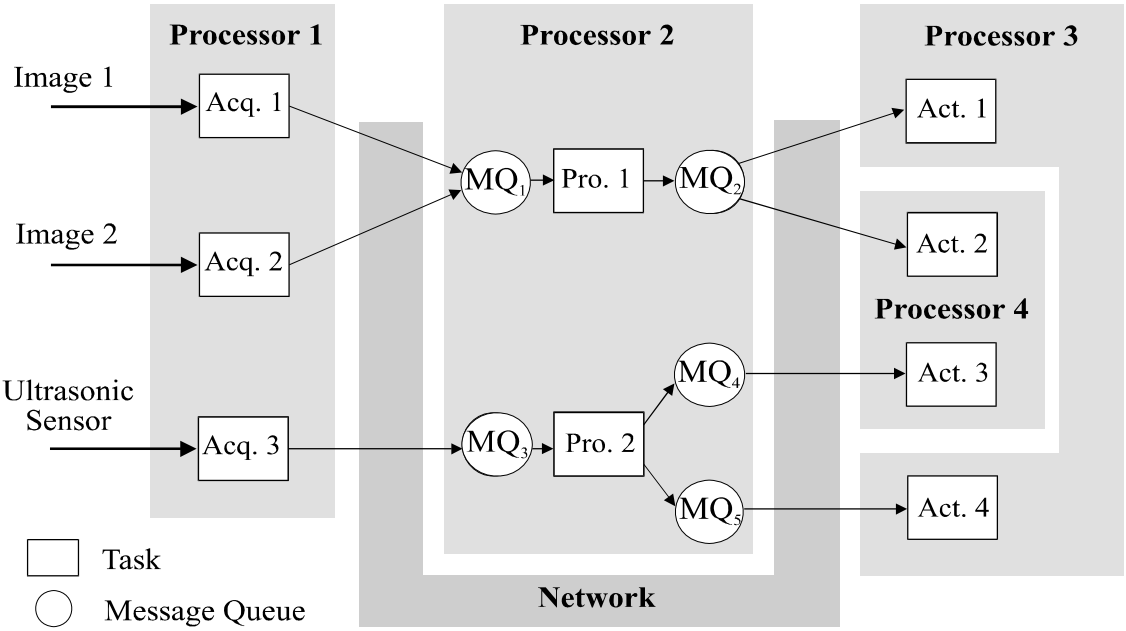
```
            (Type          => Activity,
             Input_Event   => O6,
             Output_Event  => O7,
             Activity_Operation => Reporter,
             Activity_Server=> Reporter)));

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)));
```

13. Example of Multiple_Event_Transactions

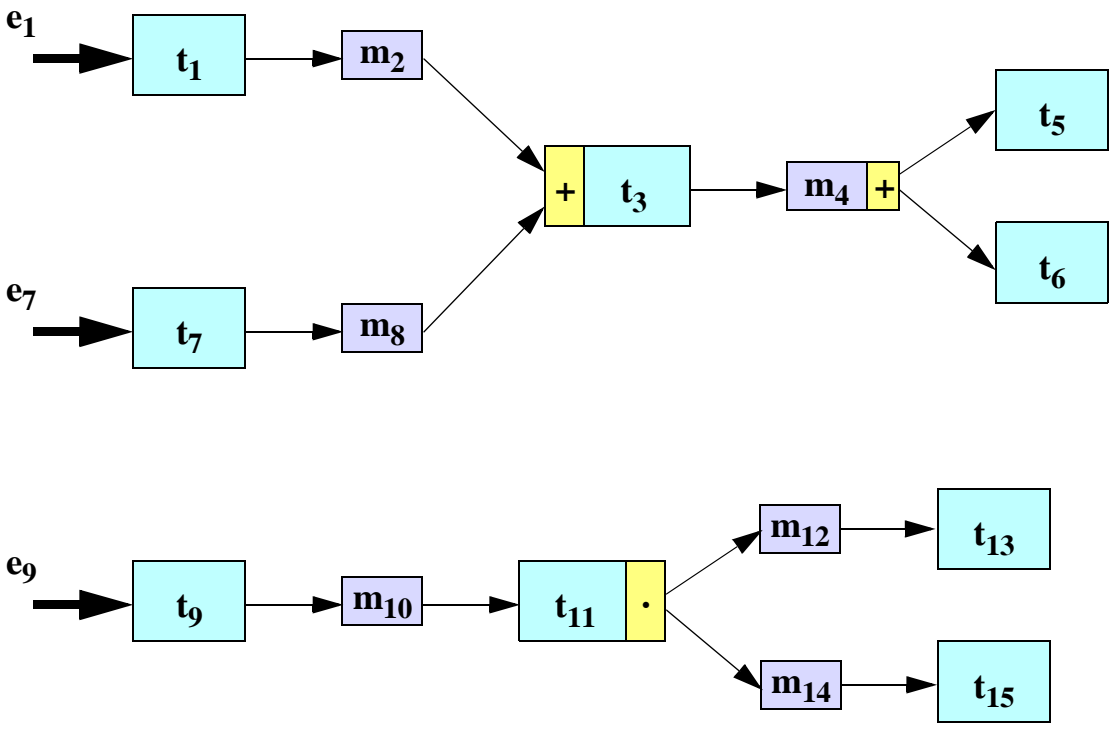
Example of steel bars inspection:



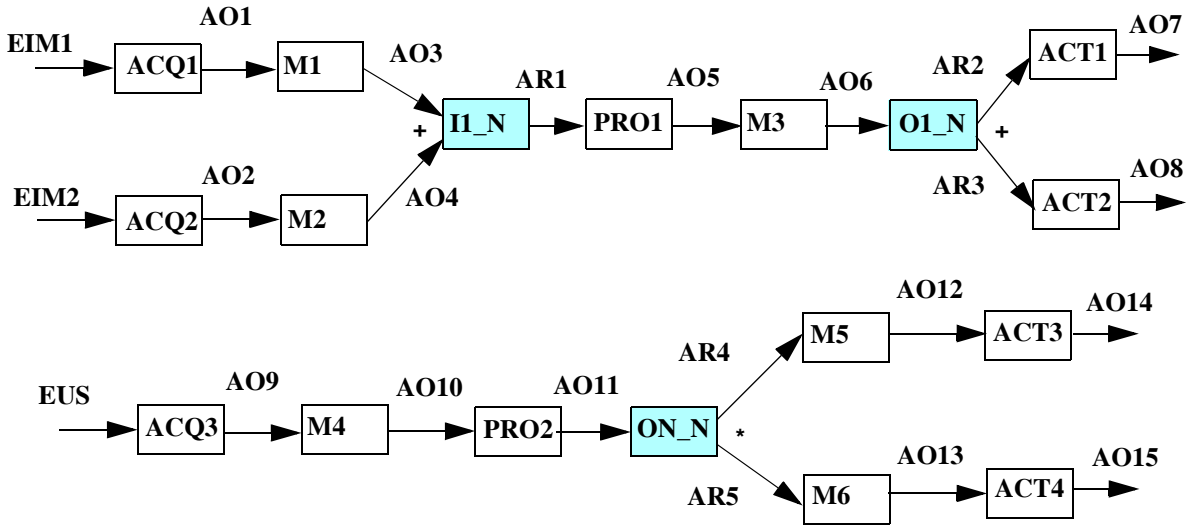
Software Architecture for this example:



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))));
```

APPENDIX A. XML MAST Format

In order to access the wide range of XML free tools that are available to validate, parse, analyse, and process an text file description formatted under XML rules, an XML format has been defined for the MAST model. It is formalized by means of a W3C-Schema.

The keywords of the special-purpose MAST description format have been used as tags in the XML format and therefore both have the same appearance.

A.1. Writing the XML-Mast File

The rules for writing the Mast files with an XML-Mast format are the following:

- The tags used for delimiting the model element are the keywords that are defined as types in the special-purpose MAST description format. There are only a few exceptions, mentioned in Table 3.
- The name attributes are mandatory.
- Blank spaces, tabs and new lines are ignored.

Table 3. Relation between MAST model format and XML tags

special-purpose format type	XML_Mast tag
Regular (Scheduling_Server)	Regular_Scheduling_Server
Fixed_Priority	Fiched_Priority_Scheduler
EDF (Scheduler)	EDF_Scheduler
FP_Packet_Based	FP_Packet_Based_Scheduler
Ticker (System_Timer)	Ticker_System_Timer
Alarm_Clock (System_Timer)	Alarm_Clock_System_Timer
Periodic (External_Event)	Periodic_External_Event
Sporadic (External_Event)	Sporadic_External_Event
Unbounded (External_Event)	Unbounded_External_Event
Bursty (External_Event)	Bursty_External_Event
Singular (External_Event)	Singular_External_Event
Regular (Event)	Regular_Event
Regular (Transaction)	Regular_Transaction
Composite (Timing req.)	Composite_Timing_Requirement

- Identifiers or names follow the Ada rules for identifiers: they begin with a letter, followed by letters, digits, underscores ('_') or periods ('.').
- The identifiers are XML attributes and are always expressed with quotes. There are no reserved words.
- The order of declaration of the modelling elements is not relevant. A name may be referenced in the file before it is declared.
- Floating point types without fractional part can be expressed without the decimal point.
- Comments are like in XML: they begin with the four character pattern ("<!--") and end with the three character pattern ("-->").
- The description is case-sensitive, although the identifiers are not.

A.2. W3C-Schema template for the XML Mast Model format

The XML-Mast format of the model file is defined in agreement with the terms and concepts that have been defined in the previous Section 7.

A.2.1 W3C-Schema definition

See file mast_model_xsd.pdf

A.2.2 Example Using the XML-MAST Format

In this section we show an example of the instance model file with XML-Mast Format of the "Single Processor System: Caseva" that had been described in Section 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Model-File" version="1.1"?>
<mast_mdl:MAST_MODEL
   xmlns:mast_mdl="http://mast.unican.es/xmlmast/model"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://mast.unican.es/xmlmast/model Mast_Model.xsd"
   Model_Name="caseva_with_clock"
   Model_Date="2000-01-01T00:00:00">
   <mast_mdl:Regular_Processor Name="processor_1" Max_Interrupt_Priority="32767"
Min_Interrupt_Priority="1" Worst_ISR_Switch="0.00" Avg_ISR_Switch="0.00" Best_ISR_Switch="0.00"
Speed_Factor="1.00" >
      <mast_mdl:Alarm_Clock_System_Timer Worst_Overhead="50.00" Avg_Overhead="0.00"
Best_Overhead="0.00"/>
   </mast_mdl:Regular_Processor>
   <mast_mdl:Primary_Scheduler Name="processor_1" Host="processor_1"  >
    <mast_mdl:Fixed_Priority_Scheduler Worst_Context_Switch="102.50" Avg_Context_Switch="0.00"
Best_Context_Switch="0.00" Max_Priority="32767" Min_Priority="1" />
   </mast_mdl:Primary_Scheduler>

   <mast_mdl:Immediate_Ceiling_Resource Name="servo_data" Ceiling="32767" Preassigned="NO" />

   <mast_mdl:Immediate_Ceiling_Resource Name="arm" Ceiling="32767" Preassigned="NO" />

   <mast_mdl:Immediate_Ceiling_Resource Name="lights" Ceiling="32767" Preassigned="NO" />

   <mast_mdl:Immediate_Ceiling_Resource Name="alarms" Ceiling="32767" Preassigned="NO" />

   <mast_mdl:Immediate_Ceiling_Resource Name="error_log" Ceiling="32767" Preassigned="NO" />


   <mast_mdl:Simple_Operation Name="read_new_point" Worst_Case_Execution_Time="87.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>servo_data</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>servo_data</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="new_point" Worst_Case_Execution_Time="54.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>servo_data</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>servo_data</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="read_axis_positions" Worst_Case_Execution_Time="135.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>arm</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>arm</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="control_servos" Worst_Case_Execution_Time="99.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>arm</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>arm</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="turn_on" Worst_Case_Execution_Time="74.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>lights</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>lights</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>
```

```
   <mast_mdl:Simple_Operation Name="turn_off" Worst_Case_Execution_Time="71.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>lights</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>lights</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="time_lights" Worst_Case_Execution_Time="119.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>lights</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>lights</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="read_all_alarms" Worst_Case_Execution_Time="78.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>alarms</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>alarms</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="set" Worst_Case_Execution_Time="59.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>alarms</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>alarms</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="notify_error" Worst_Case_Execution_Time="85.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>error_log</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>error_log</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Simple_Operation Name="get_error_from_queue" Worst_Case_Execution_Time="79.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Shared_Resources_To_Lock>error_log</mast_mdl:Shared_Resources_To_Lock>
      <mast_mdl:Shared_Resources_To_Unlock>error_log</mast_mdl:Shared_Resources_To_Unlock>
   </mast_mdl:Simple_Operation>

   <mast_mdl:Enclosing_Operation Name="servo_control" Worst_Case_Execution_Time="1080.0"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
     <mast_mdl:Operation_List>read_new_point read_axis_positions control_servos read_all_alarms
set</mast_mdl:Operation_List>
   </mast_mdl:Enclosing_Operation>

   <mast_mdl:Enclosing_Operation Name="trajectory_planning" Worst_Case_Execution_Time="9045.0"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Operation_List>new_point turn_on turn_off read_all_alarms set notify_error</
mast_mdl:Operation_List>
   </mast_mdl:Enclosing_Operation>

   <mast_mdl:Enclosing_Operation Name="light_manager" Worst_Case_Execution_Time="119.00"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Operation_List>time_lights</mast_mdl:Operation_List>
   </mast_mdl:Enclosing_Operation>

   <mast_mdl:Enclosing_Operation Name="reporter" Worst_Case_Execution_Time="72952.0"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Operation_List>read_axis_positions read_all_alarms</mast_mdl:Operation_List>
   </mast_mdl:Enclosing_Operation>

   <mast_mdl:Enclosing_Operation Name="message_logger" Worst_Case_Execution_Time="46820.0"
Average_Case_Execution_Time="1.000E+100" Best_Case_Execution_Time="0.00">
      <mast_mdl:Operation_List>get_error_from_queue</mast_mdl:Operation_List>
   </mast_mdl:Enclosing_Operation>


   <mast_mdl:Regular_Scheduling_Server Name="servo_control" Scheduler="processor_1" >
      <mast_mdl:Fixed_Priority_Policy The_Priority="415" Preassigned="NO"/>
```

```
      <mast_mdl:SRP_Parameters Preemption_Level="0" Preassigned="NO" />
   </mast_mdl:Regular_Scheduling_Server>
   <mast_mdl:Regular_Scheduling_Server Name="trajectory_planning" Scheduler="processor_1" >
      <mast_mdl:Fixed_Priority_Policy The_Priority="412" Preassigned="NO"/>
      <mast_mdl:SRP_Parameters Preemption_Level="0" Preassigned="NO" />
   </mast_mdl:Regular_Scheduling_Server>
   <mast_mdl:Regular_Scheduling_Server Name="light_manager" Scheduler="processor_1" >
      <mast_mdl:Fixed_Priority_Policy The_Priority="410" Preassigned="NO"/>
      <mast_mdl:SRP_Parameters Preemption_Level="0" Preassigned="NO" />
   </mast_mdl:Regular_Scheduling_Server>
   <mast_mdl:Regular_Scheduling_Server Name="reporter" Scheduler="processor_1" >
      <mast_mdl:Fixed_Priority_Policy The_Priority="80" Preassigned="NO"/>
      <mast_mdl:SRP_Parameters Preemption_Level="0" Preassigned="NO" />
   </mast_mdl:Regular_Scheduling_Server>
   <mast_mdl:Regular_Scheduling_Server Name="message_logger" Scheduler="processor_1" >
      <mast_mdl:Fixed_Priority_Policy The_Priority="70" Preassigned="NO"/>
      <mast_mdl:SRP_Parameters Preemption_Level="0" Preassigned="NO" />
   </mast_mdl:Regular_Scheduling_Server>

   <mast_mdl:Regular_Transaction Name="servo_control" >
           <mast_mdl:Periodic_External_Event Name="e1" Period="5000.00" Max_Jitter="0.000"
Phase="0.000" />
           <mast_mdl:Regular_Event Event="o1" >
              <mast_mdl:Hard_Global_Deadline Deadline="5000.00" Referenced_Event="e1"/>
           </mast_mdl:Regular_Event>
      <mast_mdl:System_Timed_Activity  Input_Event="e1" Output_Event="o1"
Activity_Operation="servo_control" Activity_Server="servo_control"/>
   </mast_mdl:Regular_Transaction>
   <mast_mdl:Regular_Transaction Name="trajectory_planning" >
           <mast_mdl:Periodic_External_Event Name="e2" Period="50000.00" Max_Jitter="0.000"
Phase="0.000" />
           <mast_mdl:Regular_Event Event="o2" >
              <mast_mdl:Hard_Global_Deadline Deadline="50000.00" Referenced_Event="e2"/>
           </mast_mdl:Regular_Event>
      <mast_mdl:System_Timed_Activity  Input_Event="e2" Output_Event="o2"
Activity_Operation="trajectory_planning" Activity_Server="trajectory_planning"/>
   </mast_mdl:Regular_Transaction>
   <mast_mdl:Regular_Transaction Name="light_manager" >
           <mast_mdl:Periodic_External_Event Name="e3" Period="100000.00" Max_Jitter="0.000"
Phase="0.000" />
           <mast_mdl:Regular_Event Event="o3" >
              <mast_mdl:Hard_Global_Deadline Deadline="100000.00" Referenced_Event="e3"/>
           </mast_mdl:Regular_Event>
      <mast_mdl:System_Timed_Activity  Input_Event="e3" Output_Event="o3"
Activity_Operation="light_manager" Activity_Server="light_manager"/>
   </mast_mdl:Regular_Transaction>
   <mast_mdl:Regular_Transaction Name="reporter" >
           <mast_mdl:Periodic_External_Event Name="e4" Period="1000000.00" Max_Jitter="0.000"
Phase="0.000" />
           <mast_mdl:Regular_Event Event="o4" >
              <mast_mdl:Hard_Global_Deadline Deadline="1000000.00" Referenced_Event="e4"/>
           </mast_mdl:Regular_Event>
      <mast_mdl:System_Timed_Activity  Input_Event="e4" Output_Event="o4"
Activity_Operation="reporter" Activity_Server="reporter"/>
   </mast_mdl:Regular_Transaction>
   <mast_mdl:Regular_Transaction Name="message_logger" >
           <mast_mdl:Unbounded_External_Event Name="e5" Avg_Interarrival="1000000.00"
Distribution="UNIFORM" />
           <mast_mdl:Regular_Event Event="o5" ></mast_mdl:Regular_Event>
    <mast_mdl:Activity  Input_Event="e5" Output_Event="o5" Activity_Operation= "message_logger"
Activity_Server= "message_logger"/>
   </mast_mdl:Regular_Transaction>

</mast_mdl:MAST_MODEL>
```

A.3. W3C-Schema template for the XML Mast Results format

The XML-Mast format of the results file is defined in agreement with the terms and concepts that have been defined in Section 9.

A.3.1 W3C-Schema definition

See file mast_results_xsd.pdf

A.3.2 Example of the XML-Mast Results Format

This section shows an example of a results file with the XML-Mast Format for the “Single Processor System: Caseva” that had been described in Section 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Result-File" version="1.1"?>
<mast_res:REAL_TIME_SITUATION
   xmlns:mast_res="http://mast.unican.es/xmlmast/result"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://mast.unican.es/xmlmast/result Mast_Result.xsd"
   Model_Name="caseva_with_clock"
   Model_Date="2003-03-29T12:00:00"
   Generation_Tool="MAST Schedulability Analysis, version 1.3.7.4"
   Generation_Profile="mast_analysis classic_rm -s /home/mgh/prog/mast/mast_xml/tests/
caseva_with_clock.xml /home/mgh/prog/mast/mast_xml/tests/caseva_with_clock.out.xml"
   Generation_Date="2005-02-22T12:53:31">
      <mast_res:Slack Value="98.44"/>
      <mast_res:Transaction Name="servo_control" >
         <mast_res:Slack Value="214.84"/>
         <mast_res:Timing_Result Event_Name="o1" Num_Of_Suspensions="0"
Worst_Blocking_Time="135.000" >
            <mast_res:Worst_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e1" Time_Value="1620.00"/>
            </mast_res:Worst_Global_Response_Times>
            <mast_res:Best_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e1" Time_Value="0.000"/>
            </mast_res:Best_Global_Response_Times>
            <mast_res:Jitters>
               <mast_res:Global_Response_Time  Referenced_Event="e1" Time_Value="1620.00"/>
            </mast_res:Jitters>
         </mast_res:Timing_Result>
      </mast_res:Transaction>

      <mast_res:Transaction Name="trajectory_planning" >
         <mast_res:Slack Value="257.03"/>
         <mast_res:Timing_Result Event_Name="o2" Num_Of_Suspensions="0"
Worst_Blocking_Time="135.000" >
            <mast_res:Worst_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e2" Time_Value="13540.00"/>
            </mast_res:Worst_Global_Response_Times>
            <mast_res:Best_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e2" Time_Value="0.000"/>
            </mast_res:Best_Global_Response_Times>
            <mast_res:Jitters>
               <mast_res:Global_Response_Time  Referenced_Event="e2" Time_Value="13540.00"/>
            </mast_res:Jitters>
         </mast_res:Timing_Result>
      </mast_res:Transaction>

      <mast_res:Transaction Name="light_manager" >
         <mast_res:Slack Value="39079.7"/>
```

```
         <mast_res:Timing_Result Event_Name="o3" Num_Of_Suspensions="0"
Worst_Blocking_Time="135.000" >
            <mast_res:Worst_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e3" Time_Value="13864.00"/>
            </mast_res:Worst_Global_Response_Times>
            <mast_res:Best_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e3" Time_Value="0.000"/>
            </mast_res:Best_Global_Response_Times>
            <mast_res:Jitters>
               <mast_res:Global_Response_Time  Referenced_Event="e3" Time_Value="13864.00"/>
            </mast_res:Jitters>
         </mast_res:Timing_Result>
      </mast_res:Transaction>

      <mast_res:Transaction Name="reporter" >
         <mast_res:Slack Value="636.72"/>
         <mast_res:Timing_Result Event_Name="o4" Num_Of_Suspensions="0"
Worst_Blocking_Time="79.000" >
            <mast_res:Worst_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e4" Time_Value="139314.00"/>
            </mast_res:Worst_Global_Response_Times>
            <mast_res:Best_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e4" Time_Value="0.000"/>
            </mast_res:Best_Global_Response_Times>
            <mast_res:Jitters>
               <mast_res:Global_Response_Time  Referenced_Event="e4" Time_Value="139314.00"/>
            </mast_res:Jitters>
         </mast_res:Timing_Result>
      </mast_res:Transaction>

      <mast_res:Transaction Name="message_logger" >
         <mast_res:Slack Value="102300.0"/>
         <mast_res:Timing_Result Event_Name="o5" Num_Of_Suspensions="0"
Worst_Blocking_Time="0.000" >
            <mast_res:Worst_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e5" Time_Value="1.000E+100"/>
            </mast_res:Worst_Global_Response_Times>
            <mast_res:Best_Global_Response_Times>
               <mast_res:Global_Response_Time  Referenced_Event="e5" Time_Value="0.000"/>
            </mast_res:Best_Global_Response_Times>
            <mast_res:Jitters>
               <mast_res:Global_Response_Time  Referenced_Event="e5" Time_Value="1.000E+100"/>
            </mast_res:Jitters>
         </mast_res:Timing_Result>
      </mast_res:Transaction>

      <mast_res:Processing_Resource Name="processor_1">
         <mast_res:Slack Value="88.78"/>
      </mast_res:Processing_Resource>
      <mast_res:Shared_Resource Name="servo_data" >
      </mast_res:Shared_Resource>
      <mast_res:Shared_Resource Name="arm" >
      </mast_res:Shared_Resource>
      <mast_res:Shared_Resource Name="lights" >
      </mast_res:Shared_Resource>
      <mast_res:Shared_Resource Name="alarms" >
      </mast_res:Shared_Resource>
      <mast_res:Shared_Resource Name="error_log" >
      </mast_res:Shared_Resource>
   </mast_res:REAL_TIME_SITUATION>
```

A.4. W3C-Schema template for the XML Mast Trace format

The XML-Mast format of the trace register file is defined in agreement with the terms and concepts that have been defined in Section 10.

A.4.1 W3C-Schema definition.

See file mast_trace_xsd.pdf

A.4.2 Example of the XML-MAST Trace Format

This section shows an example of a trace file using the XML-Mast Trace Format for the "Single Processor System: Caseva" example that had been described in Section 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<?mast fileType="XML-Mast-Trace-File" version="1.0" ?>

<mast_trace:TRACE_FILE
   xmlns:mast_trace="http://mast.unican.es/xmlmast/trace"
   xmlns:t="http://mast.unican.es/xmlmast/trace"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://mast.unican.es/xmlmast/trace"
   Model_Name ="Caseva_With_Clock"
   Model_Date ="2000-01-01T12:00:00"
   Generation_Tool ="Sim_MAST"
   Generation_Profile ="sim mast simulator 1.00000E+09 VERIFICATION 512 1024
C:\Temp\caseva_with_clock.xml C:\Temp\caseva_with_clock"
   Generation_Date ="2003-07-24T11:41:21"
   Start_Time = "0.0"
   End_Time = " 1.00000000000000E+09">

   <mast_trace:Msg_Type_List>
      <mast_trace:Msg_Type Mid=" 0" Type="PROC_FREE_ST"/>
      <mast_trace:Msg_Type Mid=" 1" Type="PROC_SWITCHING_ST"/>
      <mast_trace:Msg_Type Mid=" 2" Type="PROC_SCHEDULING"/>
      <mast_trace:Msg_Type Mid=" 3" Type="PROC_ATTENDING_APPLICATION_ST"/>
      <mast_trace:Msg_Type Mid=" 4" Type="PROC_ATTENDING_TIMER_ST"/>
      <mast_trace:Msg_Type Mid=" 5" Type="PROC_ATTENDING_DRIVER_ST"/>
      <mast_trace:Msg_Type Mid=" 6" Type="NETWORK_FREE_ST"/>
      <mast_trace:Msg_Type Mid=" 7" Type="NETWORK_SYNCHRONIZING_ST"/>
      <mast_trace:Msg_Type Mid=" 8" Type="NETWORK_TRANSMITTING_ST"/>
      <mast_trace:Msg_Type Mid=" 9" Type="SHARED_RSRC_FREE_ST"/>
      <mast_trace:Msg_Type Mid=" 10" Type="SHARED_RSRC_USED_ST"/>
      <mast_trace:Msg_Type Mid=" 11" Type="TIMER_IDLE_ST"/>
      <mast_trace:Msg_Type Mid=" 12" Type="TIMER_AWAITING_PROC_ST"/>
      <mast_trace:Msg_Type Mid=" 13" Type="TIMER_MANAGING_ST"/>
      <mast_trace:Msg_Type Mid=" 14" Type="ACTIVITY_IDLE_ST"/>
      <mast_trace:Msg_Type Mid=" 15" Type="ACTIVITY_SCHEDULED_ST"/>
      <mast_trace:Msg_Type Mid=" 16" Type="ACTIVITY_SUSPENDED_ST"/>
      <mast_trace:Msg_Type Mid=" 17" Type="ACTIVITY_AWAITING_TIMER_ST"/>
      <mast_trace:Msg_Type Mid=" 18" Type="ACTIVITY_AWAITING_ACTIVATION_ST"/>
      <mast_trace:Msg_Type Mid=" 19" Type="DISPATCHER_IDLE_ST"/>
      <mast_trace:Msg_Type Mid=" 20" Type="DISPATCHER_AWAITING_ACTIVATION_ST"/>
      <mast_trace:Msg_Type Mid=" 21" Type="DISPATCHER_SUSPENDED_ST"/>
      <mast_trace:Msg_Type Mid=" 22" Type="DISPATCHER_READY_TO_TRANSFER_ST"/>
      <mast_trace:Msg_Type Mid=" 23" Type="DISPATCHER_PACKET_TRANSFERING_ST"/>
      <mast_trace:Msg_Type Mid=" 24" Type="DISPATCHER_AWAITING_SENDER_ST"/>
      <mast_trace:Msg_Type Mid=" 25" Type="DISPATCHER_AWAITING_RECEIVER_ST"/>
      <mast_trace:Msg_Type Mid=" 26" Type="SCHED_SRVR_IDLE_ST"/>
      <mast_trace:Msg_Type Mid=" 27" Type="SCHED_SRVR_AWAITING_PROC_FOR_RSRC_ACCESS_ST"/>
      <mast_trace:Msg_Type Mid=" 28" Type="SCHED_SRVR_AWAITING_RSRC_WITH_PROC_ST"/>
```


```
    <mast_trace:Msg_Type Mid=" 29" Type="SCHED_SRVR_AWAITING_RSRC_ST"/>
    <mast_trace:Msg_Type Mid=" 30" Type="SCHED_SRVR_AWAITING_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 31" Type="SCHED_SRVR_SCHEDULED_ST"/>
    <mast_trace:Msg_Type Mid=" 32" Type="SCHED_SRVR_SCHEDULING_WITHOUT_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 33" Type="SCHED_SRVR_SCHEDULING_WITH_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 34" Type="SCHED_SRVR_AWAITING_POLLING_ST"/>
    <mast_trace:Msg_Type Mid=" 35" Type="SCHED_SRVR_INITIAL_AWAITING_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 36" Type="SCHED_SRVR_ATTENDING_POLLING_TIMER_ST"/>
    <mast_trace:Msg_Type Mid=" 37" Type="SCHED_SRVR_SCHEDULING_AFTER_POLLING_ST"/>
    <mast_trace:Msg_Type Mid=" 38" Type="SCHED_SRVR_CHECKING_SEGMENT_END_WITH_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 39" Type="SCHED_SRVR_CHECKING_SEGMENT_END_WITHOUT_PROC_ST"/>
    <mast_trace:Msg_Type Mid=" 40" Type="SPORADIC_SRVR_MANAGER_NORMALSCHEDULED_ST"/>
    <mast_trace:Msg_Type Mid=" 41" Type="SPORADIC_SRVR_MANAGER_NORMALNONSCHEDULED_ST"/>
    <mast_trace:Msg_Type Mid=" 42" Type="SPORADIC_SRVR_MANAGER_BACKGROUNDSCHEDULED_ST"/>
    <mast_trace:Msg_Type Mid=" 43" Type="SPORADIC_SRVR_MANAGER_BACKGROUNDNONSCHEDULED_ST"/>
    <mast_trace:Msg_Type Mid=" 44" Type="DELAY_IDLE_ST"/>
    <mast_trace:Msg_Type Mid=" 45" Type="DELAY_DELAYING_ST"/>
    <mast_trace:Msg_Type Mid=" 46" Type="DELAY_AWAITING_REF_EV_ST"/>
    <mast_trace:Msg_Type Mid=" 47" Type="FLOW_EVENT"/>
    <mast_trace:Msg_Type Mid=" 48" Type="SCHED_REQ_EVENT"/>
    <mast_trace:Msg_Type Mid=" 49" Type="SCHED_ACCEPT_EVENT"/>
    <mast_trace:Msg_Type Mid=" 50" Type="SCHED_ASSIGN_EVENT"/>
    <mast_trace:Msg_Type Mid=" 51" Type="SCHED_UNASSIGN_EVENT"/>
    <mast_trace:Msg_Type Mid=" 52" Type="RSRC_REQ_EVENT"/>
    <mast_trace:Msg_Type Mid=" 53" Type="PROC_REQ_EVENT"/>
    <mast_trace:Msg_Type Mid=" 54" Type="RSRC_ASSIGN_EVENT"/>
    <mast_trace:Msg_Type Mid=" 55" Type="PROC_ASSIGN_EVENT"/>
    <mast_trace:Msg_Type Mid=" 56" Type="PROC_UNASSIGN_EVENT"/>
    <mast_trace:Msg_Type Mid=" 57" Type="RELEASE_EVENT"/>
    <mast_trace:Msg_Type Mid=" 58" Type="AWAITING_EVENT"/>
    <mast_trace:Msg_Type Mid=" 59" Type="PRTY_ASSIGN_EV"/>
    <mast_trace:Msg_Type Mid=" 60" Type="TIMER_EVENT"/>
    <mast_trace:Msg_Type Mid=" 61" Type="STATE_OFF"/>
    <mast_trace:Msg_Type Mid=" 62" Type="GLOBAL_DEADLINE_MISSED"/>
    <mast_trace:Msg_Type Mid=" 63" Type="LOCAL_DEADLINE_MISSED"/>
    <mast_trace:Msg_Type Mid=" 64" Type="OVERRIDDEN_PACKET"/>
    <mast_trace:Msg_Type Mid=" 65" Type="OVERRIDDEN_MESSAGE"/>
  </mast_trace:Msg_Type_List>

<mast_trace:Src_List>
   <mast_trace:Src Sid="-13" Name="Servo_Control.E1" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
  <mast_trace:Src Sid="-99" Name="Message_Logger.E5" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-83" Name="Reporter.E4" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-72" Name="Light_Manager.E3" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-40" Name="Trajectory_Planning.E2"
      Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-15" Name="Servo_Control 0" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-22" Name="Servo_Control 1" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-27" Name="Servo_Control 2" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-30" Name="Servo_Control 3" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-35" Name="Servo_Control 4" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-14" Name="Servo_Control.O1" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-42" Name="Trajectory_Planning 0" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-49" Name="Trajectory_Planning 1" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-54" Name="Trajectory_Planning 2" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-57" Name="Trajectory_Planning 3" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-62" Name="Trajectory_Planning 4" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-65" Name="Trajectory_Planning 5" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-41" Name="Trajectory_Planning.O2"
      Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-74" Name="Light_Manager 0" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-73" Name="Light_Manager.O3" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
   <mast_trace:Src Sid="-85" Name="Reporter 0" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-92" Name="Reporter 1" Type="SIM.EV_C.INSTANCE"/>
   <mast_trace:Src Sid="-84" Name="Reporter.O4" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/>
```

```
      <mast_trace:Src Sid="-101" Name="Message_Logger 0" Type="SIM.EV_C.INSTANCE"/>
     <mast_trace:Src Sid="-100" Name="Message_Logger.O5" Type="SIM.EV_C.MAIN_FLOW_CH.INSTANCE"/
>
     </mast_trace:Src_List>

   <mast_trace:Msg_List>
      <t:E T=" 9.99675335666572E+08" S="-30" M=" 47"/>
      <t:E T=" 9.99675391380435E+08" S="-35" M=" 47"/>
      <t:E T=" 9.99675858696526E+08" S="-14" M=" 47"/>
      <t:E T=" 9.99680000000000E+08" S="-13" M=" 47"/>
      <t:E T=" 9.99680097958756E+08" S="-15" M=" 47"/>
      <t:E T=" 9.99680134392436E+08" S="-22" M=" 47"/>
      <t:E T=" 9.99680222603902E+08" S="-27" M=" 47"/>
      <t:E T=" 9.99680234018401E+08" S="-30" M=" 47"/>
.............
.............
      <t:E T=" 1.00000000000000E+09" S="-83" M=" 47"/>
      <t:E T=" 1.00000000000000E+09" S="-13" M=" 47"/>
      <t:E T=" 1.00000000000000E+09" S="-72" M=" 47"/>
      <t:E T=" 1.00000000000000E+09" S="-40" M=" 47"/>
   </mast_trace:Msg_List>
</mast_trace:TRACE_FILE>
```