Timing Analysis of a Generic Robot Teleoperation Software Architecture

Bárbara Álvarez, Alejandro Alonso, Juan A. de la Puente.

Abstract: The development of generic software architectures for specific application domains is an effective way of reusing software. This was one of the aims in the development of an architecture for teleoperating robots, which can be adapted for dealing with different jobs, operational environments and robots. However, these kinds of systems use to have timing requirements. The purpose of this paper is to present a characterization of this architecture for analysing its timing properties using the Rate Monotonic Analysis method. In this way a generic architecture and a method for checking its timing properties is provided.

Keywords: Real-time systems, software architecture, robots control, teleoperation, software engineering.

1. INTRODUCTION

The correctness of real-time systems depends not only on the computational results, but also on the time at which outputs are generated. Hence, when developing a real-time system it is necessary to check that its timing requirements are fulfilled. This makes more complex its development.

Furthermore, the complexity of current computer systems is increasing. The reuse of software (designs, code and documentation) is a way of dealing with this problem. An efficient way of reusing software is the development of generic software architectures for specific application domains. In this way, a reference architecture for teleoperation systems has been proposed, which can be adapted for dealing with different jobs, operational environments and even robots [2]. In order to reach this purpose, the stages of the domain-engineering process have been applied. Domain-engineering process defines the activities to build a common software core for a family of systems [1]. This was the approach taken in the development of the generic software architecture for robot teleoperation systems [2].

In robot teleoperation systems, the timing requirements should be accomplished to ensure that the information that the operator receives is valid and reflects the current state of the robot. Therefore, the system must respond to external stimuli and the response depends not only on the stimulus itself, but also on what has previously happened in the system. These characteristics are typical of a real-time system [3]. Hence when evaluating the feasibility of the architecture for a particular application, it is necessary to check not only that the appropriate functions are included but that the timing requirements are fulfilled as well. The measures of merit in a real-time system include

- Predictably fast response to urgent events.
- High degree of schedulability. Schedulability is the degree of resource utilization at or below which the timing requirements of tasks can be ensured.
- Stability under transient overload. When the system is overloaded by events and meeting all deadlines is impossible, the deadlines of selected critical tasks must be guaranteed.

The Rate Monotonic Scheduling (RMS) theory [4] ensures that as long as the CPU utilization of all tasks lies below a certain bound and appropriate scheduling algorithms are used, all tasks will meet their deadlines without the programmer knowing exactly when any given task will be running. Even if a transient overload occurs, a fixed subset of critical tasks will still meet their deadlines as long as their CPU utilizations lie within the appropriate bound. The Rate Monotonic Analysis (RMA) method is based on this theory.

In general, the proposed reference architecture can be ported to any platform and it can be executed on any operating system that is not necessary a real-time operating system. However, the timing behavior of the system can be analysed using the Rate Monotonic Analysis (RMA) method [5] if certain conditions are met. The most important conditions are the following:
• A real-time operating system is used and tasks scheduling is based on fixed priorities and preemptive [6]. The priorities of tasks are fixed in the sense that they do not vary from one event to the next and the scheduling is preemptive because if a higher priority task becomes eligible to run, the lower priority task will be preempted of the processor.

• Inter-tasks communication is based on a suitable protocol, such as the priority ceiling protocol, for bounding priority inversion [7]. The priority ceiling protocol guarantees that a high-priority task will be blocked by a bounded time of any lower priority task.

The rate monotonic scheduling theory allows the designers to reason with confidence about timing correctness at the tasking level of abstraction and it analyses if the deadlines of the tasks can be guaranteed. In this way, a generic architecture and a framework for analysing its timing response is proposed. This is provided by identifying the sequences of events, actions and resources in the architecture, following RMA [9]. In this way, the designer of a new application can reuse the architecture and can check easily whether the architecture can meet the timing requirements by providing the specific timing requirements of the particular application and the computation time of the activities in the target platform if the above conditions hold.

1.1. System description

A schematic view of the components of the system is shown in figure 1. The operator is in charge of monitoring and operating the robot according to the information provided by the teleoperation system. This system receives commands from the operator and performs the corresponding actions for executing them. For this purpose, it communicates with the robot control unit (ROC), which physically actuates on the robot to move it. The robot control unit makes some sensing from the robot in order to evaluate its global state and to send this information to the teleoperation system, which uses it to represent graphically to the operator the state of the robot and to ensure the correctness of its behaviour. Different tools are attached to the robot for performing the maintenance operations. The tools are operated in a similar way to the robot.

![Figure 1: Scheme of the system](image)

At the system requirements analysis phase, the functional and non-functional requirements are defined [3]. The high-level design of the architecture is shown in figure 2. It was based on an analysis of the system functionality. The controllers interact with the following subsystems:

• Graphical representation: This subsystem is in charge of drawing the environment where the robot is operating and the robot itself. The application can be adapted for working with others robots and tools in different environments. The robot controller module requests a service, and a status of the operation is returned.

• Collisions detection: This server provides the system with operations for checking if a given movement command does imply the collision of the robot with the operating environment or with itself.

![Figure 2: High level architecture description](image)
• User Interface: It is in charge of interacting with the user. It allows him to issue the desired command to the robot and to show the status of its execution.

• Communications: It embodies the communication protocol with the ROC computer. This module returns the received status from robot control unit.

2. PROCESS MODEL.

The development of the teleoperation architecture has been based on the 4+1 view model [8]. It uses different system descriptions in order to contemplate all the important aspects. In this paper, the process model is presented, which is the most important to analyse the timing requirements of the proposed software architecture.

Figure 3 shows the process model of the robot controller. This model describes tasks structure, which is based on the fact that it is not desirable for the controller to be blocked by any reason, because it is in charge of executing the main system functions and, in particular, the safe stop procedure in case of severe errors. For this reason, several tasks have been allocated for communicating with each of the other subsystems and thus decoupling the controller task from having to interact directly with the other subsystems. A queue for storing the events in a prioritized order is used by the other tasks for reporting the results of the interaction with the other subsystems.

The tools controller subsystem was the first example of reuse within this project. The tools are handled as a more simple and specialized robot. In this way, the tool controller shares the same structure and code than the robot controller. With respect to figure 3, the only modification when adding the tool controller, is the need to synchronize both controllers. The rest of the paper is concentrated on identifying the RMA model of the robot controller. The other one is exactly the same

The controllers, the task that receives messages from the ROC (t5) is periodic, because the status robot information is sent periodically from the robot control unit (ROC). When this task is activated, an order for getting this information is sent to the communication module. This information is processed and the results are sent back to the controller buffer for being processed by the robot_controller task.

A task for sending messages to the robot control unit (t4) is activated sporadically from the robot controller. The ROC accepts commands with a minimum separation time between them. In addition, the system does not allow the request for a command, before the handling of the previous one. The only exception to this rule is the emergency stop of the robot.

The controller buffer (s5) is a shared resource and it is based on the priority ceiling protocol. Hence, its priority is immediately higher than the highest priority client. Therefore, the maximum time that a task can be blocked by a lower priority task is the duration of the largest critical region. In our case, the most priority client is the robot controller.

When the robot controller receives a status message, it sends a request to the graphic representation module (s2) through Graphics_Data_Processing task (t2), in order to present the current robot status to the user.

The syntactical and lexical analysis of the operator commands is performed by the user interface subsystem. If a command is correct, it is sent to the User_Data_Processing (t3). This task checks periodically the commands from user, and it sends them to the controller buffer (s5), in order to be handled.

If the user’s command is a robot motion command, the robot controller requests a service to the collisions detection server (s1) for ensuring that there are no collisions with the environment. If this check is successful, the command is sent to the robot. The server (s1) spends some computation time for calculating kinematic robot.

The Kinematic_Data_Processing (t1) and Graphical_Data_Processing (t2) are activated sporadically, and their priorities depend on the timing requirements of the associated motion request.

In the next section, the RMA method is applied and the sequences of events, actions, and resources in the system are identified. In this way, it is possible to predict the worst case response for each of the meaningful system events.
3. SYSTEM MODEL FOR TIMING ANALYSIS.

The Rate Monotonic Analysis [9] is based on the identification of the system events and the associated actions, for obtaining conclusions about the timing behaviour of the real-time system. An event occurs when the system or environment state changes. Since an event can occur repeatedly, RMA refers to recurrence of events as an event sequence, and a computation that is performed as a consequence of an event is generally referred to as the response to the event. An event sequence is classified by its type, which is based on its origin. Events can originate outside the system, inside the system, or because of the passage of time:

- **External events sequence (environment events):** It appears when a change occurs in the environment. For example, a key depression from a console or a message from another computer.
- **Internal events sequence:** It appears when a change is detected in the system. For example, when an alarm is detected as result of processing data or a tracking algorithm predicts a collision of the robot with its environment.
- **Timed events sequence:** When the event is generated by a system clock. For example, controlling the temperature of a reactor may require that the temperature be sensed every certain time.

Event sequence arrivals are characterized by their arrival pattern and by the mode of the system during which the events can occur. The arrival pattern and modes of an event sequence are important because they identify when the event will be competing for resources. For a particular event sequence, it is critical to be able to characterize the event’s pattern of occurrence as a function of time. This is called the arrival pattern under the RMA method.

In general, there are three parameters which characterize the timing behaviour of robot teleoperation systems: (1) Minimum separation for sending commands to the robot unit control (ROC), (2) minimum separation between user commands and (3) status robot information is sent periodically from the ROC. In this work, these general parameters are illustrated for a specific product.

The controller task is the most critical one. It is in charge of handling system events. A queue for storing the messages associated to the events that the controller receives from others subsystems is used. In this way it is possible to handle the events based on their priority. Whenever a new message is inserted in the queue, the event Proc_Message is raised. It is necessary to bound the maximum number of these events that can occur in a certain interval of time, in order to be able to analyse the timing requirements.

If the next event can occur arbitrarily close to a previous event, but the number of events over a specified period of time is restricted, then the event arrival pattern is called bursty under RMA method. Bursty arrival patterns are characterized by the length of time over which the burst restriction applies and the number of events that can occur during that duration of time. In our case, there is a limit to the number of events generated between the generation of two messages to the ROC (60 ms)\(^1\). The events that the controller can receive during this time are the following:

- **Status_msg:** a new status message has been received from the ROC.
- **Command_msg:** a command has been introduced by the operator.
- **Kin_msg:** a result of the operation for calculating collisions detection is returned to the controller.
- **Graphics_msg:** a result of operation for updating graphical representation is returned to the controller.
- **Error_msg:** an error message indicates to the controller a failure for sending to ROC.
- **Sinc_msg:** a synchronization message from tool (robot) controller is received by the robot (tool) controller. The synchronization between robot and tool controllers is required because some tools (robot) operations can only be done when the robot (tools) is in a certain state, as for example the load and unload of tools.

The generation of these kind of messages by the robot controller implies the generation of new ones, for activating appropriate activities in the rest of the subsystems. These internal events are the following:

- **Send_User:** The user interface must be updated. Any action for controlling the robot is limited by the time of reception of commands in the ROC (60 ms). Therefore, it is not necessary a higher frequency for updating the user interface.
- **Send_Command:** A command must be sent to the ROC. The minimum separation time between two of this messages to the ROC is 60ms.
- **Coll_Operation:** A motion command has been received and the controller checks its feasibility by requiring this service to the Collisions Detection module. The minimum separation time between two commands of motion is 2s, which is highest period for the operator for sending commands to the robot.
- **Graphical_Update:** A new robot position must be updated in the screen. This event is generated when a new robot status is received, therefore it occurs with a minimum separation time equal to 250 ms.
- **Automatic_Action:** The controller decides to execute an action as a consequence of the current state of the system. Any action for control-

---

1. For illustrating purposes, the timing requirements of the product developed for performing maintenance operations in the bowl a steam generator are included.
When a change is detected in the environment, the following events can be generated:

- Capt_Status: A status message is received periodically from ROC. The period of reception is 250 ms.
- Capt_Command: A task checks periodically the introduction of command by the operator. The period of reception is 2s.

Every event sequence has an associated response. A response is the computational work that must be performed as a consequence of the occurrence of an event. An action is the smallest decomposition of a response. In this way, a response consist of an ordered collection of actions and lower level responses. In the next sections, the sequences of actions that are executed for handling system events are described.

3.1. Sequence for capturing robot status

**Description:** This sequence handles the Capt_Status event and it is initiated by the timer of the reception messages task (t5), and it finishes when a robot status message is inserted in the controller buffer (s5). When the events arrive at constant intervals have an arrival pattern called periodic under RMA method.

**Type:** External, periodic.

**Response:** a5.1 $\rightarrow$ a5.2 $\rightarrow$ a5.3

- a5.1: This action is executed by the communications module for obtaining the robot status from the ROC.
- a5.2: The received status is converted in a controller message.
- a5.3: The message is inserted in the controller buffer

3.2. Sequence for handling user commands

**Description:** This sequence is executed when the operator issues a command (Capt_Command event). Its an aperiodic sequence, but the minimum separation between commands is fixed and limited. This event arrival pattern is called bounded aperiodic under RMA method.

**Type:** External, bounded aperiodic.

**Response:** a3.1 $\rightarrow$ a3.2

- a3.1: The User_Data_Processing task analyses the received user command and converts it in a controller message.
- a3.2: The message is inserted in the controller buffer.

3.3. Sequences for processing control messages

**Description:** The event Proc_Messages is generated when a new message is inserted in the controller buffer. As a result the controller_robot task gets and analyses its contents and as a result it executes the appropriate activity. The messages associated to the event can occur arbitrarily close to each other, in the worst case. However, the maximum number of event occurrences over a specific period (60 ms) is restricted to six. This specific period is fixed by the reception time of commands by the ROC.

**Type:** Timed, bounded aperiodic.

**Response:** ac.1 $\rightarrow$ [ac.2.. ac.7]

- ac.1: A controller message is extracted from buffer.
- ac.x: The controller message is processed by robot controller and sent to the appropriate task.

3.4. Sequence for updating user interface.

**Description:** This sequence is initiated by the task (t6) that is in charge of interacting with the user interface. The associated event (Graphical_Update) is issued by the robot_controller task.

**Type:** Timed, bounded aperiodic.

**Response:** a6.1 $\rightarrow$ a6.2

- a6.1: A message is processed for sending to user interface.
- a6.2: The user interface is updated.

3.5. Sequence for kinematic calculating.

**Description:** This sequence is initiated by task (t11) that is in charge of interacting with the collision detection server, and it finishes when a status of operation is inserted on controller buffer. This sequence is executed in response to the Coll_Operation event.

**Type:** Timed, bounded aperiodic.

**Response:** a1.1 $\rightarrow$ a1.2 $\rightarrow$ a1.3 $\rightarrow$ a1.4

- a1.1: A message is sent to the server.
- a1.2: The robot kinematic calculation is made.
- a1.3: The result of the operation is converted into a message for the controller.
- a1.4: A message is inserted in the buffer.

3.6. Sequence for graphical updating

**Description:** As mentioned above, the status information from the robot control unit is received periodically. When this information is received, a service to the graphical representation module for updating the robot representation is requested. The sequence handles the event Graphical_Update and is initiated by task (t2), which is in charge of interacting with the graphical representation subsystem.

**Type:** Timed, bounded aperiodic.

**Response:** a2.1 $\rightarrow$ a2.2 $\rightarrow$ a2.3 $\rightarrow$ a2.4

- a2.1: The status is processed and sent to the graphical representation module.
- a2.2: The graphical representation is updated.
- a2.3: The graphical representation answer is converted in a controller message.
- a2.4: The message is inserted in the controller buffer.
3.7. Sequence for sending commands to ROC.

**Description:** The sequence is initiated by task (t4) that is in charge of interacting with the ROC. It is the response associated to the Send_Command event.

**Type:** Timed, bounded aperiodic.

**Response:** a4.1 → a4.2

- a4.1: The robot command is formatted for being sent to the ROC.
- a4.2: The command is sent.


**Description:** At any time, the controller can send a command to the robot. Its an aperiodic sequence but the minimum separation between commands is fixed and limited by the robot control unit.

**Type:** Internal, bounded aperiodic.

**Response:** ac.8

- ac.8: A robot message is generated for sending to ROC.

3.9. System situation table

The essence of a real-time situation is expressed in a situation table. In table 1, the system events and the associated information are shown. The system has only one operation mode (Normal). For example, Proc_Messages is a timed event. Its arrival pattern is bursty. The time interval in which the arrival burst will occur is 60000. This event’s deadline is hard.

In the table 2, the actions of the system are shown. Each action is characterized by its attributes. The timing behavior of an action is affected by:

- Which resource it uses.
- The priority of the action on that resource. Priority is a value associated with an action that is used by an allocation policy to resolve contention for shared resources and can be expressed in integer values or symbolic values.
- The amount of time that it will be using that resource.
- The policy being used to allocate the resource. An allocation policy is characterized by the type of resource being used and the specific policy name.
- Whether the action requires exclusive use of the resource. This is called atomic action. An action may be atomic on one resource, such as a data object, and simultaneously use another resource, such as the CPU, in a non-atomic manner.
- Whether the action has to be performed with limited jitter. Some actions within the response to a periodic event require that the action, usually input/output, be performed without jitter. Jitter is a measure of deviation between the desired time for an input/output is performed and the actual time that an input/output is performed. A specification of n/a means that there is no requirement to control jitter.

In this way, the jitter policy, use of resources, time computation, user task, and priority are described for each action. The fields that can vary between different implementations of the situation are left as general parameters, such as C11 or App[P]. The designer of a new application can analyze whether the architecture can meet the timing requirements by providing the specific values for general parameters.

Finally, table 3 shows the resources, their type and policy for dealing with simultaneous events. Resource types include: CPU, data object, device, backplane bus, and so on. The processor, or CPU, is perhaps the most common shared resource for software systems. The scheduling policies or set rules that determines which of all the actions that are ready to execute will be allocated the CPU are generally implemented by an operating system. In this case, the scheduling policy is called fixed-priority policy which uses priorities of actions are fixed in the sense that they do not vary from one event to the next.

Other software mechanism that is frequently used as a resource is the data object. When data objects are shared in a mutually exclusive fashion between concurrent entities like tasks, one task can cause delays for other tasks. The allocation policy for the data object will determine the size of the delay. In this case, the highest locker protocol is used. The user of the data object executes at the priority of the highest priority task that can access the data object. In this way, the table 3 shows fixed-priority for CPU and highest locker for buffer.

Although a system has been already implemented based on this architecture, it has not been possible to use the presented framework. This product is not based on a real-time operating system, because the timing requirements does not force it. In addition, other requirements, such as the need for using commercial tools for collisions detection and graphical representation, precludes the above option. However, the architecture is being used for the development of other products and it is intended to use the presented framework for checking their timing behaviour.

4. Conclusions

The Rate Monotonic Analysis has been applied to a generic software architecture for teleoperation systems. In order to analyze the timing requirements of the system a set of assumptions have been made. In this way, the timing behaviour of the system can be predicted.

This framework allows the designer to evaluate the suitability of the generic architecture for fulfilling the timing requirements of a particular application.

The presented analysis provides a pattern for identifying the sequences of events, actions and shared resources in the architecture. Then, the designer can provide the required timing information and check the guarantee of the application deadlines. This required timing information consists of:

- general requirements which characterize the timing behaviour of robot teleoperation systems,
• Minimum separation for sending commands to the robot unit control,
• minimum separation between user commands, and
• status robot information is sent periodically from the robot unit control,
• time computation of the activities in the target platform,
• the priority of the action on each resource.

In this work, these general parameters are illustrated for a specific product developed and are 60, 2000, and 250 ms, respectively, but the designer of other application has provide this timing information.

Acknowledgments

This work has been funded by the Spanish Government Committee for Science and Technology (CICYT), projects no. TIC96-0614.

References


### TABLE 1. Events table

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Type</th>
<th>Mode</th>
<th>Arrival Pattern</th>
<th>Timing req.</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capt_Status</td>
<td>External</td>
<td>Normal</td>
<td>Periodic, (250)</td>
<td>Hard, n/a</td>
<td>a5.1 → a5.2 → a5.3</td>
</tr>
<tr>
<td>Automatic_Action</td>
<td>Internal</td>
<td>Normal</td>
<td>Bounded, (60)</td>
<td>Hard, n/a</td>
<td>ac.8</td>
</tr>
<tr>
<td>Proc_Messages</td>
<td>Timed</td>
<td>Normal</td>
<td>Bursty, (60, 6)</td>
<td>Hard, n/a</td>
<td>ac.1 → [ac.2, ac.3, ac.4, ac.5, ac.6, ac.7]</td>
</tr>
<tr>
<td>Capt_Command</td>
<td>External</td>
<td>Normal</td>
<td>Bounded, (2000)</td>
<td>Hard, n/a</td>
<td>a3.1 → a3.2</td>
</tr>
<tr>
<td>Send_Command</td>
<td>Timed</td>
<td>Normal</td>
<td>Bounded, (60)</td>
<td>Hard, n/a</td>
<td>a4.1 → a4.2</td>
</tr>
<tr>
<td>Send_User</td>
<td>Timed</td>
<td>Normal</td>
<td>Bounded, (60)</td>
<td>Hard, n/a</td>
<td>a6.1 → a6.2</td>
</tr>
<tr>
<td>Coll_Operation</td>
<td>Timed</td>
<td>Normal</td>
<td>Bounded, (2000)</td>
<td>Hard, n/a</td>
<td>a1.1 → a1.2 → a1.3 → a1.4</td>
</tr>
<tr>
<td>Graphical_Update</td>
<td>Timed</td>
<td>Normal</td>
<td>Bounded, (250)</td>
<td>Hard, n/a</td>
<td>a2.1 → a2.2 → a2.3 → a2.4</td>
</tr>
</tbody>
</table>

### TABLE 2. Actions

<table>
<thead>
<tr>
<th>Action ID</th>
<th>Jitter</th>
<th>Resource ID</th>
<th>Atomic</th>
<th>Time used</th>
<th>User ID</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>a5.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c51</td>
<td>t5</td>
<td>App P</td>
</tr>
<tr>
<td>a5.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c52</td>
<td>t5</td>
<td>App P</td>
</tr>
<tr>
<td>a5.3</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c53</td>
<td>t5</td>
<td>App P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFFER</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c31</td>
<td>t3</td>
<td>App P</td>
</tr>
<tr>
<td>a3.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c32</td>
<td>t3</td>
<td>App P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFFER</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a6.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c61</td>
<td>t6</td>
<td>App P</td>
</tr>
<tr>
<td>a6.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c62</td>
<td>t6</td>
<td>App P</td>
</tr>
<tr>
<td>a1.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c11</td>
<td>t1</td>
<td>App P</td>
</tr>
<tr>
<td>a1.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c12</td>
<td>t1</td>
<td>App P</td>
</tr>
<tr>
<td>a1.3</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c13</td>
<td>t1</td>
<td>App P</td>
</tr>
<tr>
<td>a1.4</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c14</td>
<td>t1</td>
<td>App P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFFER</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a2.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c21</td>
<td>t2</td>
<td>App P</td>
</tr>
<tr>
<td>a2.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c22</td>
<td>t2</td>
<td>App P</td>
</tr>
<tr>
<td>a2.3</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c23</td>
<td>t2</td>
<td>App P</td>
</tr>
<tr>
<td>a2.4</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c24</td>
<td>t2</td>
<td>App P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFFER</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a4.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c41</td>
<td>t4</td>
<td>App P</td>
</tr>
<tr>
<td>a4.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>c42</td>
<td>t4</td>
<td>App P</td>
</tr>
<tr>
<td>ac.1</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc1</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUFFER</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ac.2</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc2</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.3</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc3</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.4</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc4</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.5</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc5</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.6</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc6</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.7</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc7</td>
<td>tc</td>
<td>App P</td>
</tr>
<tr>
<td>ac.8</td>
<td>n/a</td>
<td>CPU</td>
<td>No</td>
<td>cc8</td>
<td>tc</td>
<td>App P</td>
</tr>
</tbody>
</table>

### TABLE 3. Resources table

<table>
<thead>
<tr>
<th>Resource ID</th>
<th>Type</th>
<th>Policy</th>
<th>Switch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>CPU</td>
<td>Fixed Priority</td>
<td>n/a</td>
</tr>
<tr>
<td>BUFFER</td>
<td>Data</td>
<td>Highest Locker</td>
<td>n/a</td>
</tr>
</tbody>
</table>