Technical Report–V1.1

Rania.MZID

July 2022

The design patterns described in this document can be utilized for early verification of real-time distributed systems. We start by providing the pattern catalogue. Then, we go over the different patterns that have been suggested thus far.

1 Pattern catalogue

The current set of pattern categories is shown in Figure 1. We define two categories that allow calculating two real-time metrics: processor utilization factor (PUFC) and task response time (RTAC). The description of the proposed categories are provided in Table 1.

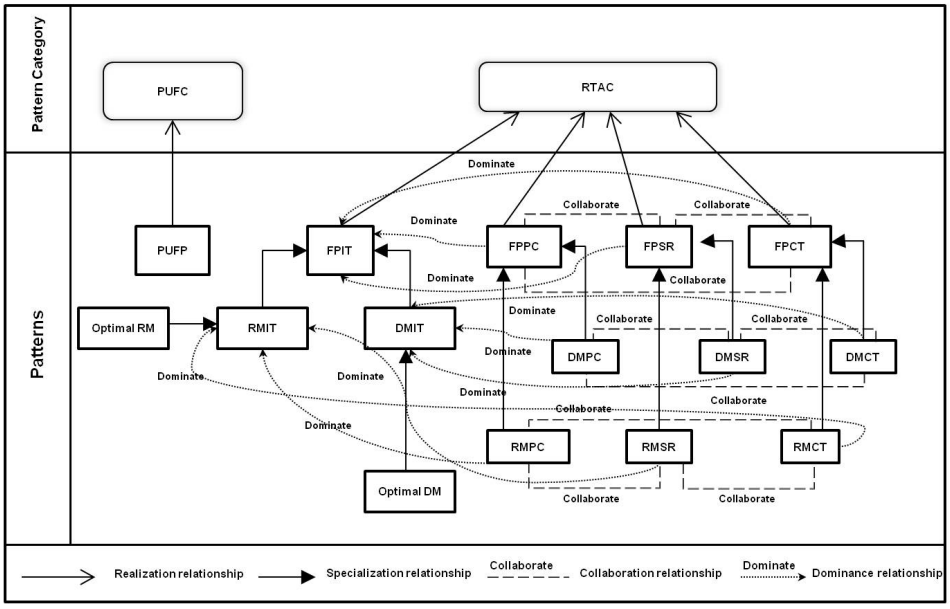

Figure 1: Current design patterns and relationships

Table 1: Current list of pattern categories for real-time verification of critical embedded systems

Pattern category	Description	Patterns
Processor Utilization Factor Category (PUFC)	This category includes patterns that describe resources and their attributes, allowing the processor utilization factor to be calculated.	PUFP
Response Time Analysis Category (RTAC)	This category contains patterns that define resources and their associated attributes and constraints, allowing task response times to be computed.	FPIT, RMIT, DMIT, OptimalRM, OptimalDM, FPPC, RMPC, DMPC, FPSR, RMSR, DMSR, FPCT, RMCT, DMCT

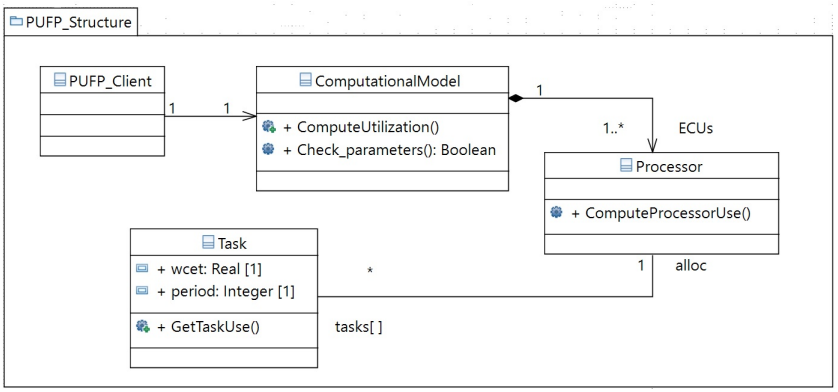

Figure 2: Structural description of the PUFP pattern using the UML class diagram

2 Processor Utilization Factor Pattern (PUFP)

Pattern Name and Classification. Processor Utilization Factor Category (PUFC): Processor Utilization Factor Pattern (PUFP).
Intent. The PUFP describes the computational model assumed by the schedulability analysis techniques to enable utilization-based analysis. This pattern avoids incomplete design models, allowing the processor utilization factor to be computed. As a result, the load on each processor can be calculated to determine whether the design alternatives under consideration are feasible.
Applicability. The PUFP pattern is applicable when the real-time application is described as a collection of tasks (either dependent or independent) that must be carried out on execution nodes (i.e., processors) with the goal of calculating the processor utilisation factor of a certain task set.
Structure. Figure 2 depicts the structural view of the PUFP pattern.
Behaviour. Figure 3 illustrates the behavioral view of the PUFP pattern.

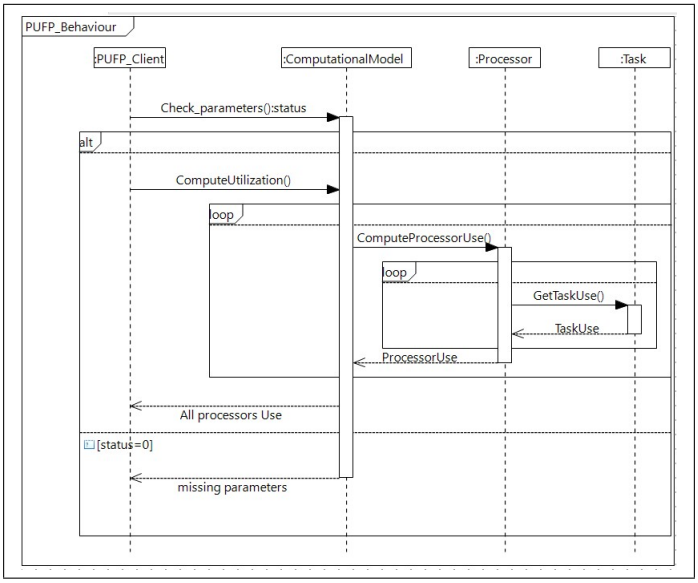

Figure 3: Behavioral description of the PUFP pattern using the UML sequence diagram

3 Fixed Priority Independent Task Pattern (FPIT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Fixed Priority Independent Task Pattern (FPIT).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern is applicable when the software designer aims to compute the task response times to assess the design feasibility. In order to apply this pattern, the real-time application must be defined as a collection of tasks to be executed on a set of processors. The tasks are presumed to be independent. The priorities of the different tasks are considered as inputs (i.e., the software designer aims to assign a predetermined priority to each task).
Structure. Figure 4 depicts the structural view of the FPIT pattern.
Behaviour. Figure 5 illustrates the behavioral view of the FPIT pattern.

4 Rate Monotonic Independent Task Pattern (RMIT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Rate Monotonic Independent Task Pattern (RMIT).
Intent. This pattern avoids using incompatible or incomplete design models

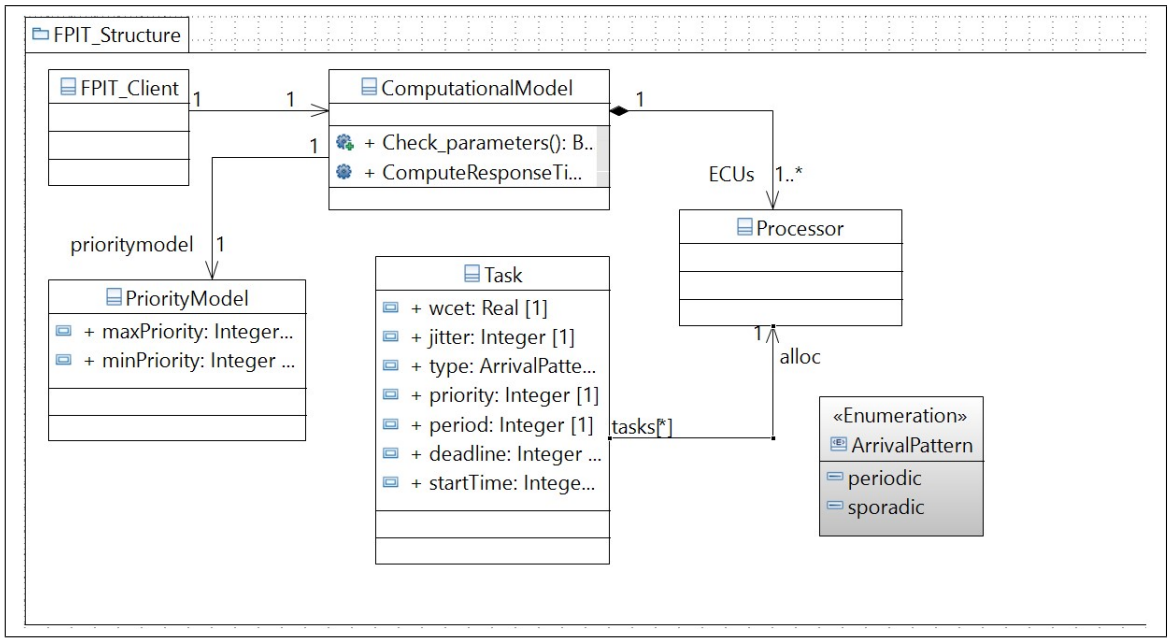

Figure 4: Structural description of the FPIT pattern using the UML class diagram

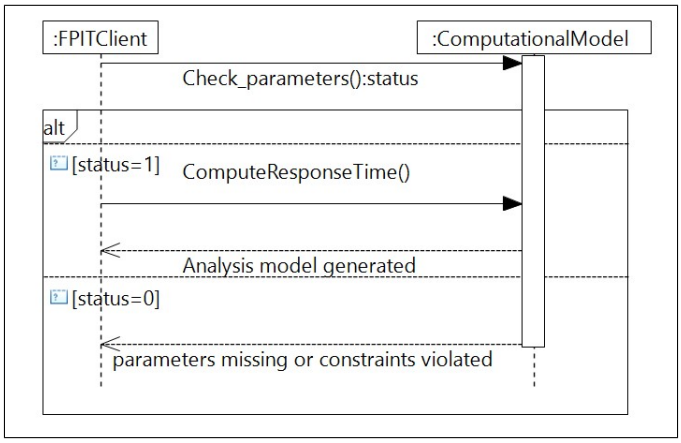

Figure 5: Behavioral description of the FPIT pattern using the UML sequence diagram

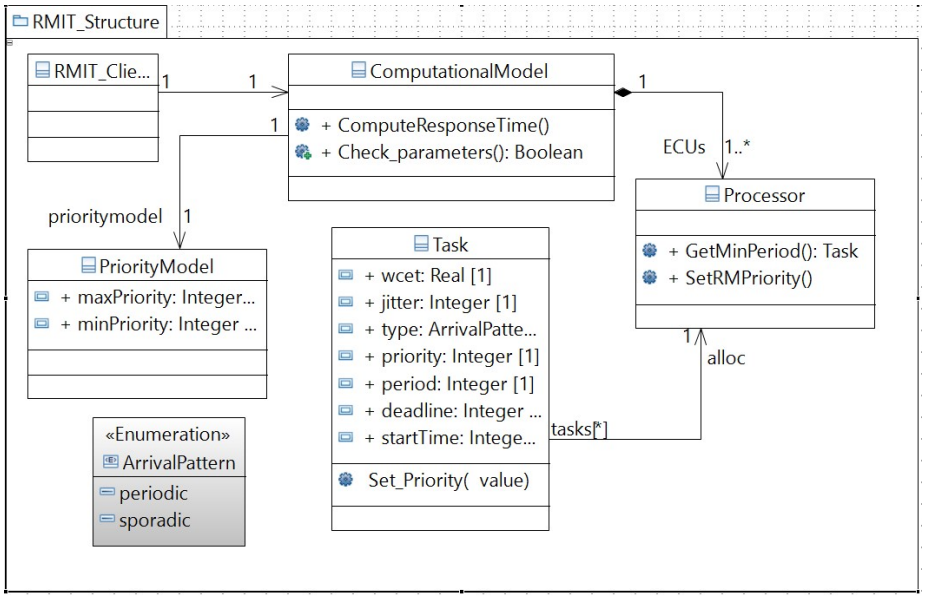

Figure 6: Structural description of the RMIT pattern using the UML class diagram

and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.

Applicability. This pattern is applicable when the software designer aims to compute the task response times to assess the design feasibility. In order to apply this pattern, the real-time application must be defined as a collection of tasks to be executed on a set of processors. The tasks are presumed to be independent. The rate monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.

Structure. Figure 6 depicts the structural view of the RMIT pattern.

Behaviour. Figure 7 illustrates the behavioral view of the RMIT pattern.

Related real-time verification patterns. This pattern is a specialization of the Fixed Priority Independent Task (FPIT) pattern.

5 Deadline Monotonic Independent Task Pattern (DMIT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Deadline Monotonic Independent Task Pattern (DMIT).

Intent. This pattern avoids using incompatible or incomplete design models

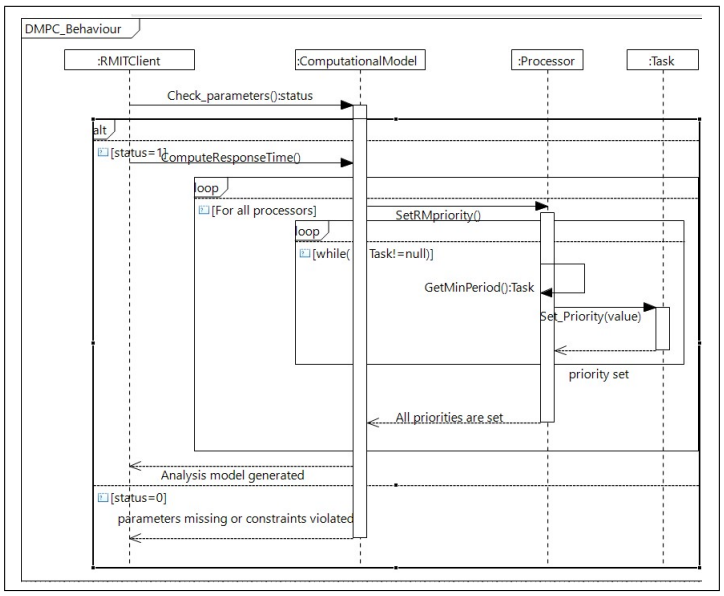

Figure 7: Behavioral description of the RMIT pattern using the UML sequence diagram

and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.

Applicability. This pattern is applicable when the software designer aims to compute the task response times to assess the design feasibility. In order to apply this pattern, the real-time application must be defined as a collection of tasks to be executed on a set of processors. The tasks are presumed to be independent. The deadline monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.

Structure. Figure 8 depicts the structural view of the RMIT pattern.

Behaviour. Figure 9 illustrates the behavioral view of the RMIT pattern.

Related real-time verification patterns. This pattern is a specialization of the Fixed Priority Independent Task (FPIT) pattern.

6 Optimal Rate Monotonic pattern (OptimalRM)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Optimal Rate Monotonic pattern (OptimalRM).

Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being con-

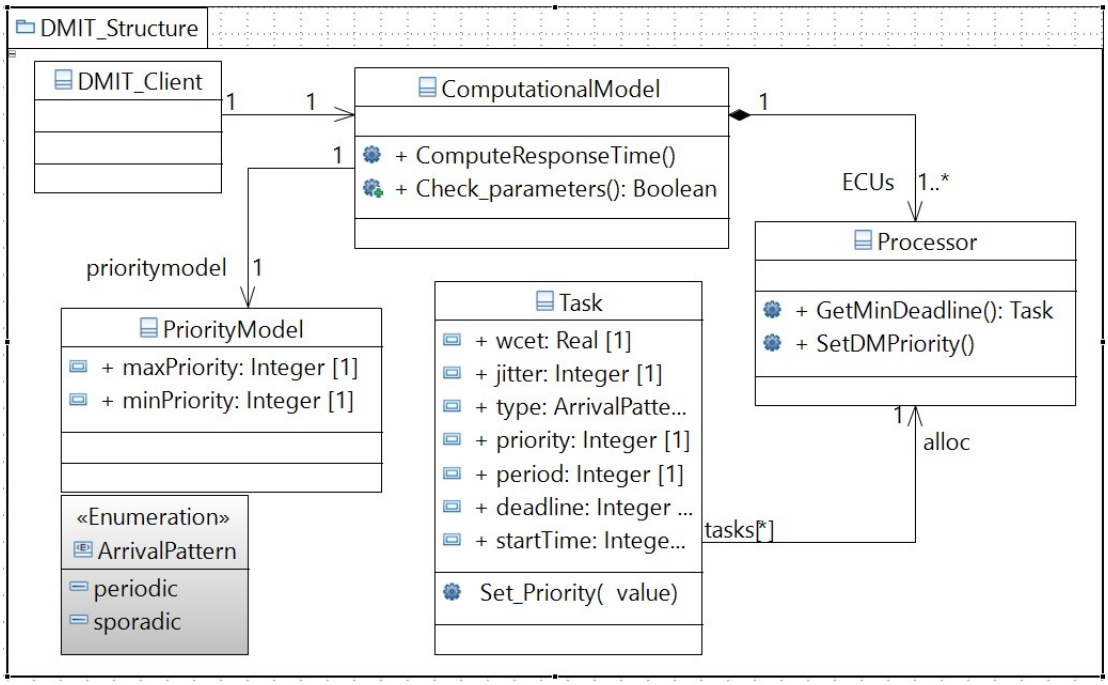

Figure 8: Structural description of the DMIT pattern using the UML class diagram

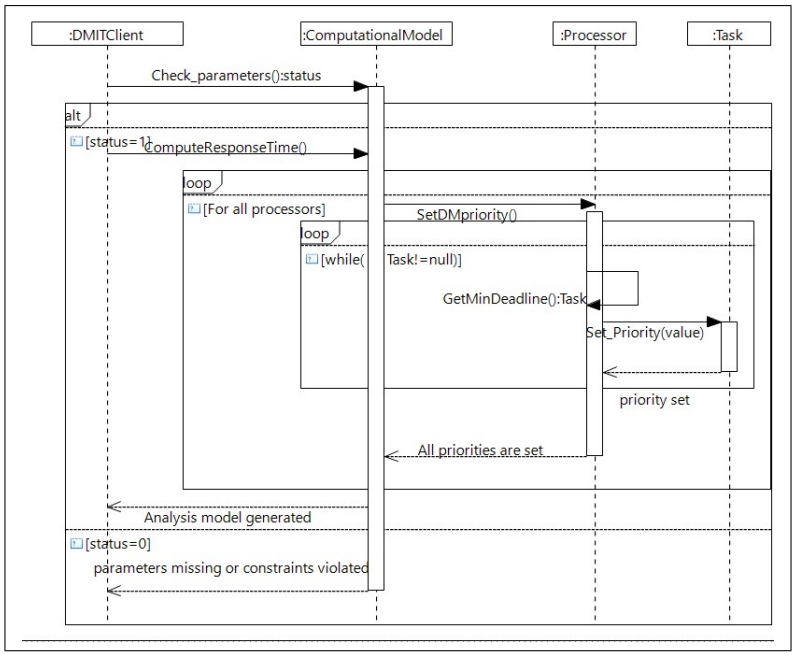

Figure 9: Behavioral description of the DMIT pattern using the UML sequence diagram

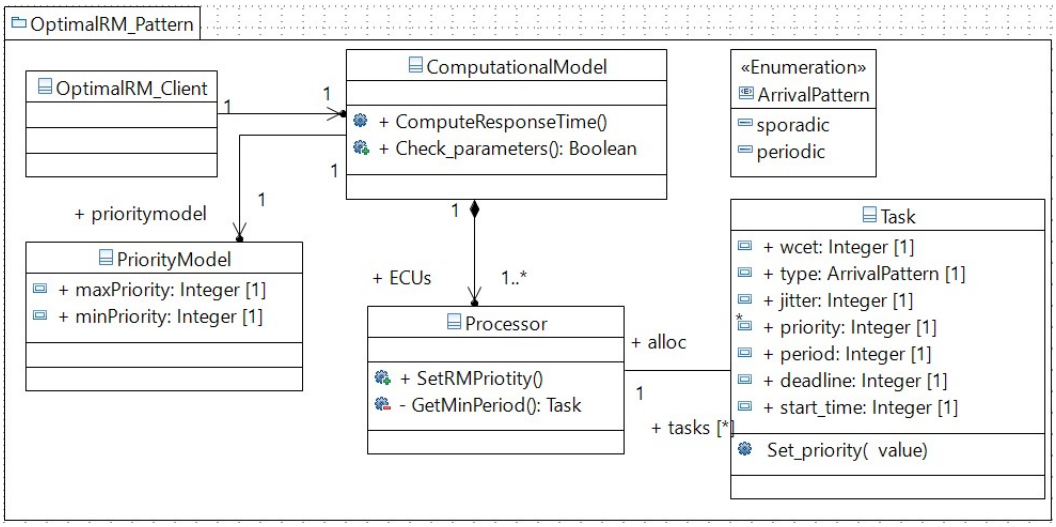

Figure 10: Structural description of the OptimalRM pattern using the UML class diagram

sidered.

Applicability. This pattern is applicable when the designer wants to compute the task response times to evaluate the feasibility of the designer's choices. It could also help with the selection of the scheduling model by exploiting the optimatilty property. This pattern is applicable when the real-time application consists of a set of independent, periodic or sporadic tasks to be executed on a collection of processors. The different tasks must be released and ready for execution at *t=0*. All task deadlines must be equal to their periods. The rate monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.

Structure. Figure 10 depicts the structural view of the OptimalRM pattern.

Behaviour. Figure 11 illustrates the behavioral view of the OptimalRM pattern.

Constraints. When using this pattern, the considered design model must verify the rate monotonic algorithm's optimality assumptions. As a result, we add the following OCL constraint to this pattern:

```
context t: Task
inv: (t.start\_time=0) and (t.jitter=0)
and (t.deadline=t.period)
```

Related real-time verification patterns. This pattern is a specialization of the Deadline Monotonic Independent Task (RMIT) pattern.

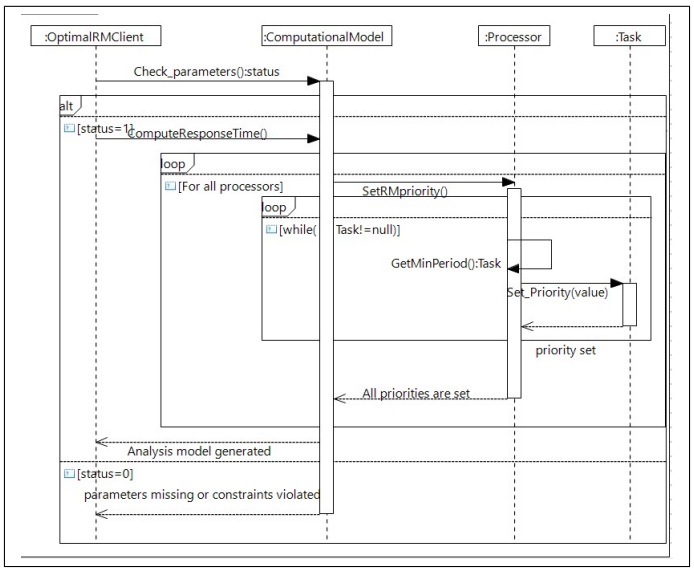

Figure 11: Behavioral description of the OptimalRM pattern using the UML sequence diagram

7 Optimal Deadline Monotonic pattern (OptimalDM)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Optimal Deadline Monotonic pattern (OptimalDM).
Intent. This pattern describes the computational model assumed by the schedulability analysis when the Deadline Monolithic algorithm is optimal. This pattern avoids incomplete and incompatible design models, allowing the analysis models to be generated automatically. As a result, task response times can be computed to assess the feasibility of the design options under consideration. This pattern could also help the designer with the selection of a scheduling model by exploiting the optimatilty property.
Applicability. This pattern is applicable when the designer wants to compute the response time to evaluate the feasibility of the designer's choices. It could also help with the selection of the scheduling model by exploiting the optimatilty property. This pattern is applicable when the real-time application consists of a set of independent, periodic or sporadic tasks to be executed on a collection of processors. The different tasks must be released and ready for execution at *t=0*. All task deadlines must be lower or equal to their periods. The deadline monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.

Structure. The structural view of the OptimalDM pattern is depicted in Figure 12.
Behaviour. Figure 13 illustrates the behavioural view of the OptimalDM pattern.

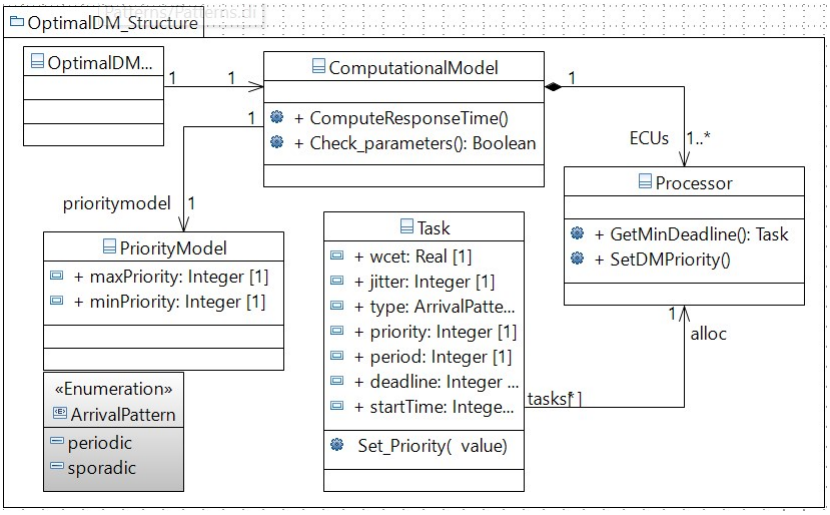

Figure 12: Structural description of the OptimalDM pattern using the UML class diagram

Constraints. When using this pattern, the considered design model must verify the deadline monotonic algorithm's optimality assumptions. As a result, we add the following OCL constraint to this pattern:

```
context t: Task
inv: (t.start\_time=0) and (t.jitter=0)
and (t.deadline<=t.period)
```

This constraint ensures that the different tasks in this model must be released and ready for execution at *t=0* (i.e., *start_time* and *jitter* must equal to 0 for all tasks). In addition, the deadlines for all tasks must be lowered or equal to their periods.
Related real-time verification patterns. This pattern is a specialization of the Deadline Monotonic Independent Task (DMIT) pattern.

8 Fixed Priority Shared Resource (FPSR)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Fixed Priority Shared Resource (FPSR).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern is applicable when the designer wants to compute

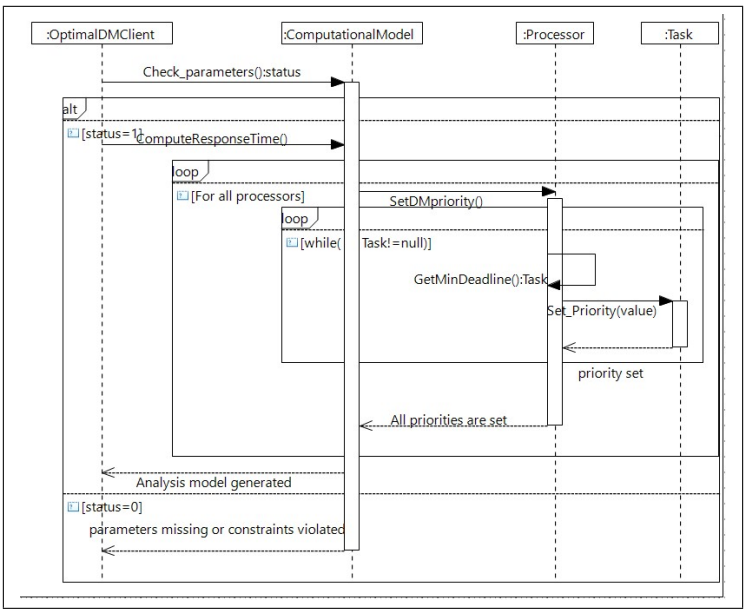

Figure 13: Behavioral description of the OptimalDM pattern using the UML sequence diagram

the task response times to evaluate the feasibility of the designer's choices. To apply this pattern, the real-time application must consist of a set of tasks to be completed on processors. At least two tasks from the task set are expected to be dependent on having mutual access to a critical section (i.e., a shared resource). In this pattern, tasks that share resources must be assigned to the same processor. The priorities of the different tasks are considered as inputs (i.e., fixed priority).

Structure. The structural view of the RMSR pattern is depicted in Figure 16.

Behaviour. Figure 17 illustrates the behavioral view of the FPSR pattern.

Constraints. In a distributed architecture, dependent tasks (tasks that share a resource) must then be assigned to the same processor to enable response time computation and avoid potential errors during the analysis phase. To that end, we add the following OCL constraint to this pattern in order to ensure this propriety.

```
context r: Resource
inv:
([r.tasks->collect(t:Task | t.alloc)]->size()=1)
and
([r.tasks->collect(t:Task | t.alloc)]= r.alloc)
```

Related real-time verification patterns. This pattern has collaboration relationships with the Fixed Priority Precedence Constraint (FPPC) and the Fixed Priority Communicating Tasks (FPCT) patterns.

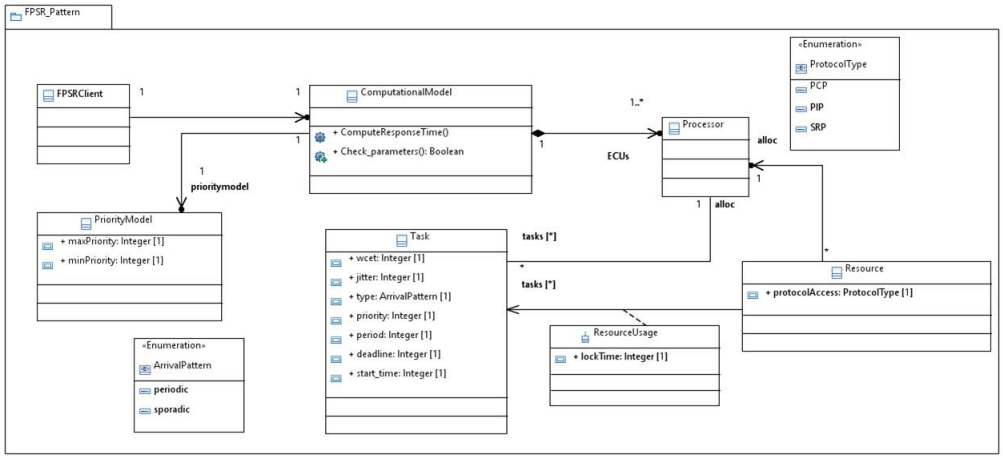

Figure 14: Structural description of the FPSR pattern using the UML class diagram

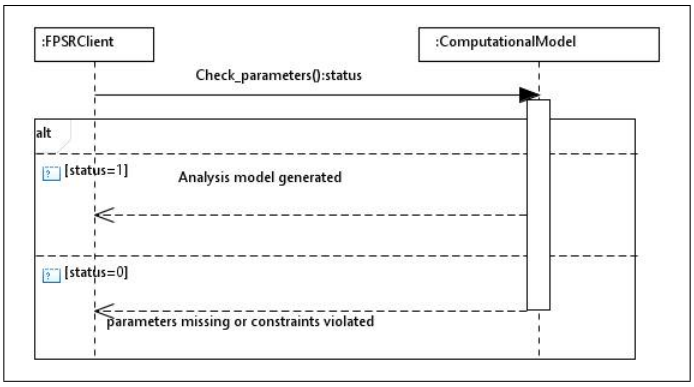

Figure 15: Behavioral description of the FPSR pattern using the UML sequence diagram

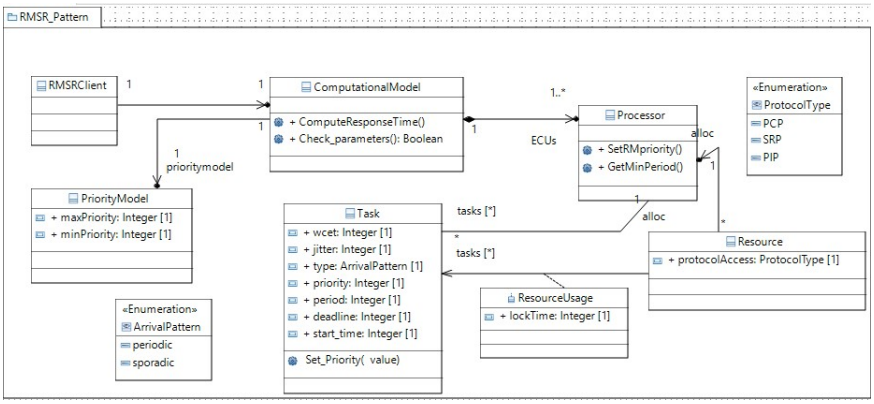

Figure 16: Structural description of the RMSR pattern using the UML class diagram

9 Rate Monotonic Shared Resource (RMSR)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Rate Monotonic Shared Resource (RMSR).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern is applicable when the designer wants to compute the task response times to evaluate the feasibility of the designer's choices. To apply this pattern, the real-time application must consist of a set of tasks to be completed on processors. At least two tasks from the task set are expected to be dependent on having mutual access to a critical section (i.e., a shared resource). In this pattern, tasks that share resources must be assigned to the same processor. The rate monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.
Structure. The structural view of the RMSR pattern is depicted in Figure 16.
Behaviour. Figure 17 illustrates the behavioral view of the FPSR pattern.
Constraints. In a distributed architecture, dependent tasks (tasks that share a resource) must then be assigned to the same processor to enable response time computation and avoid potential errors during the analysis phase. To that end, we add the following OCL constraint to this pattern in order to ensure this propriety.

```
context r: Resource
inv:
([r.tasks->collect(t:Task | t.alloc)]->size()=1)
and
([r.tasks->collect(t:Task | t.alloc)]= r.alloc)
```

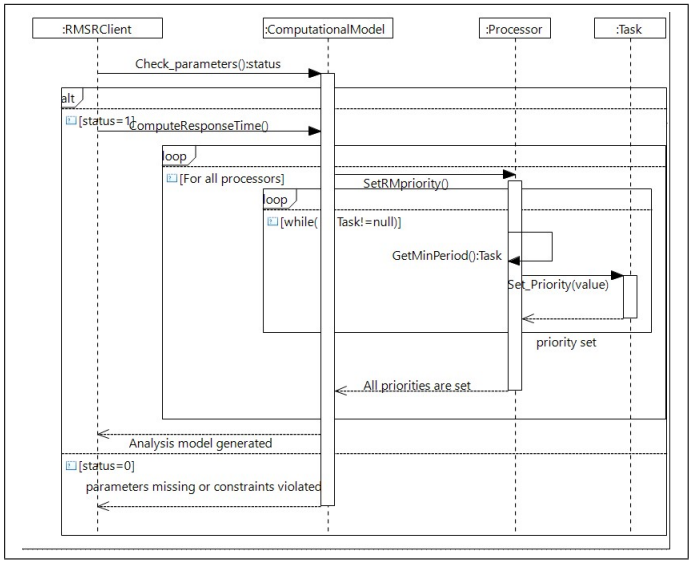

Figure 17: Behavioral description of the RMSR pattern using the UML sequence diagram

Related real-time verification patterns. This pattern has collaboration relationships with the Rate Monotonic Precedence Constraint (RMPC) and the Rate Monotonic Communicating Tasks (RMCT) patterns.

10 Deadline Monotonic Shared Resource (DMSR)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Deadline Monotonic Shared Resource (DMSR).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern is applicable when the designer wants to compute the task response times to evaluate the feasibility of the designer's choices. To apply this pattern, the real-time application must consist of a set of tasks to be completed on processors. At least two tasks from the task set are expected to be dependent on having mutual access to a critical section (i.e., a shared resource). In this pattern, tasks that share resources must be assigned to the same processor. The deadline monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.
Structure. The structural view of the DMSR pattern is depicted in Figure 18.
Behaviour. Figure 19 illustrates the behavioral view of the FPSR pattern.
Constraints. In a distributed architecture, dependent tasks (tasks that share

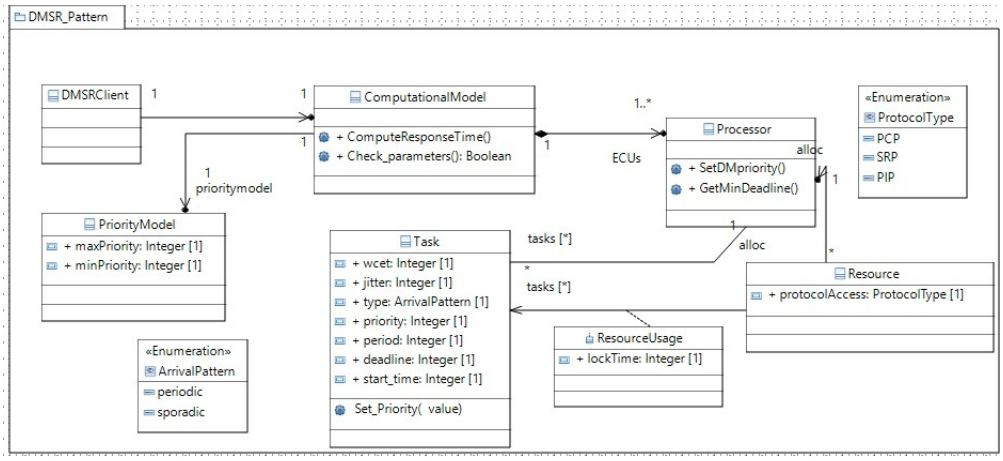

Figure 18: Structural description of the DMSR pattern using the UML class diagram

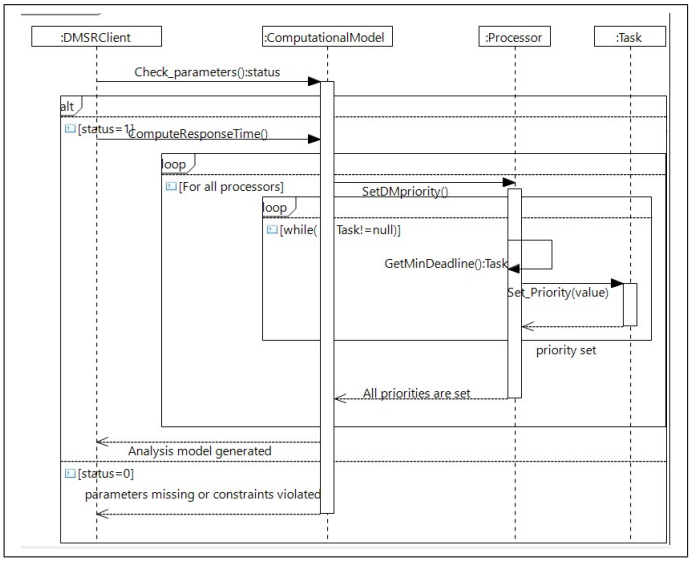

Figure 19: Behavioral description of the DMSR pattern using the UML sequence diagram

a resource) must then be assigned to the same processor to enable response time computation and avoid potential errors during the analysis phase. To that end, we add the following OCL constraint to this pattern in order to ensure this propriety.

```
context r: Resource
inv:
([r.tasks->collect(t:Task | t.alloc)]->size()=1)
and
([r.tasks->collect(t:Task | t.alloc)]= r.alloc)
```

Related real-time verification patterns. This pattern has collaboration relationships with the Deadline Monotonic Precedence Constraint (RDMPC) and the Deadline Monotonic Communicating Tasks (DMCT) patterns.

11 Fixed Priority Precedence Constraints (FPPC)

We describe in this section the content of each field in the pattern template for the Fixed Priority Precedence Constraints (FPPC) pattern.
Pattern Name and Classification. Response Time Analysis Category (RTAC): Fixed Priority Precedence Constraints (FPPC).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. The FPPC pattern is applicable when the real-time application is defined as a set of tasks and processors with the goal of computing the response time of tasks. At least two tasks in this pattern are assumed to be dependent (i.e., tasks that must be executed in a specific order since they depend on each other's results). Dependent tasks must be assigned to the same processor. This pattern is applicable when the designer wants to manually set the priorities of the various tasks (i.e., fixed priority).
Structure. The structural view of the FPPC pattern is depicted in Figure 20.
Behaviour. Figure 29 illustrates the behavioral view of the FPPC pattern.
Constraints. A precedence-constrained task's release may be delayed until all immediate predecessors have been terminated. Several models have been proposed in the literature to turn this scenario into a computational model that can be analysed. To ensure that a task never runs before in this pattern, it must have a lower priority than its predecessors and a release jitter that is greater than or equal to that of its predecessors. In order to enable response time computation and avoid potential errors during the analysis phase, we add the following OCL constraints to this pattern in order to ensure these two properties.

```
context Task
inv: self.preds->forAll (t: Task|
self.priority < t.priority)
```

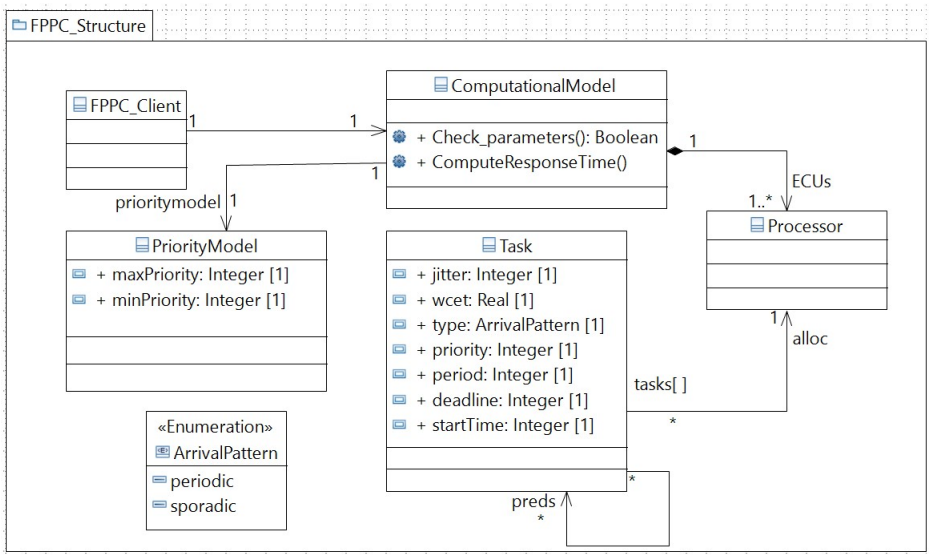

Figure 20: Structural description of the FPPC pattern using the UML class diagram

```
context Task
inv: self.preds-> forAll(t: Task|
self.jitter >= t.jitter)
```

In addition, since in this patterns precedence-constrained tasks must be assigned to the same processor, the following constraint is also defined:

```
context Task
inv: self.preds -> forAll(t: Task|
self.alloc = t.alloc)
```

Related real-time verification patterns. This pattern has collaboration relationships with the Fixed Priority Shared Resource (FPSR) and Fixed Priority Communicating Tasks (FPCT) patterns.

12 Rate Monotonic Precedence Constraints (RMPC)

We describe in this section the content of each field in the pattern template for the Rate Monotonic Precedence Constraints (RMPC) pattern.
Pattern Name and Classification. Response Time Analysis Category (RTAC): Deadline Monotonic Precedence Constraints (RMPC).
Intent. This pattern prevents the use of incompatible or incomplete design models and enables the automatic generation of analysis models. As a result,

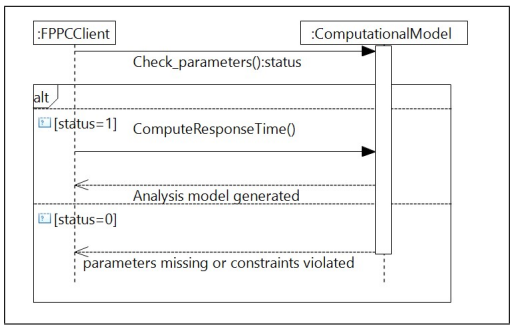

Figure 21: Behavioral description of the FPPC pattern using the UML sequence diagram

task response times can be estimated in order to evaluate the feasibility of the considered design.
Applicability. When the model representing the real-time application at the design level is described as a set of tasks and processors with the aim of determining the task response time, the DMPC pattern is applicable. In this scenario, the model is expected to include at least two dependent tasks (i.e., tasks that must be executed in a specific order since they depend on each other's results). The same processor must perform all dependent tasks. Furthermore, this pattern can be used when the designer wants to define task priorities based on the deadline monotonic scheduling algorithm.
Structure. Figure **??** depicts the structural view of the RMPC pattern.
Behaviour. Figure 23 illustrates the behavioral view of the RMPC pattern.

13 Deadline Monotonic Precedence Constraints (DMPC)

We describe in this section the content of each field in the pattern template for the Deadline Monotonic Precedence Constraints (DMPC) pattern.
Pattern Name and Classification. Response Time Analysis Category (RTAC): Deadline Monotonic Precedence Constraints (DMPC).
Intent. This pattern prevents the use of incompatible or incomplete design models and enables the automatic generation of analysis models. As a result, task response times can be estimated in order to evaluate the feasibility of the considered design.
Applicability. When the model representing the real-time application at the design level is described as a set of tasks and processors with the aim of determining the task response time, the DMPC pattern is applicable. In this scenario, the model is expected to include at least two dependent tasks (i.e., tasks that must be executed in a specific order since they depend on each other's results).

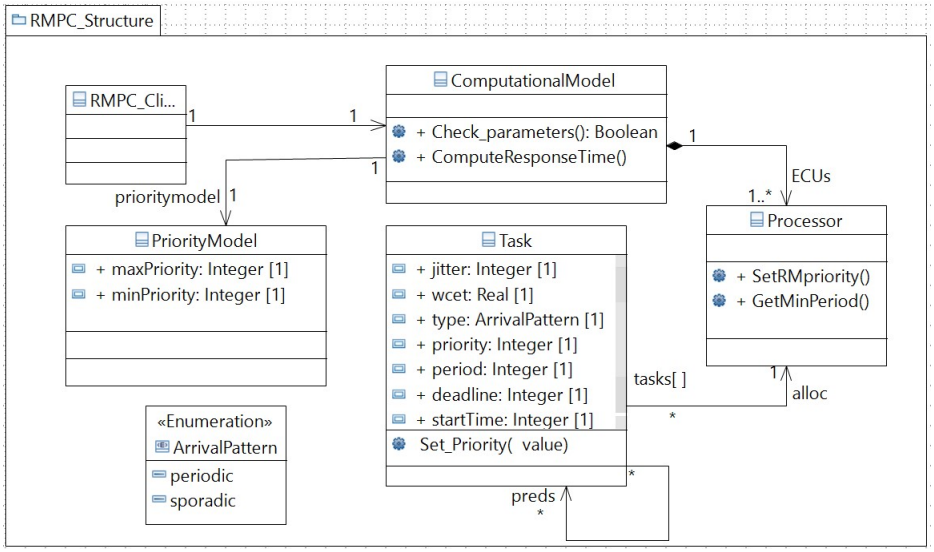

Figure 22: Structural description of the RMPC pattern using the UML class diagram

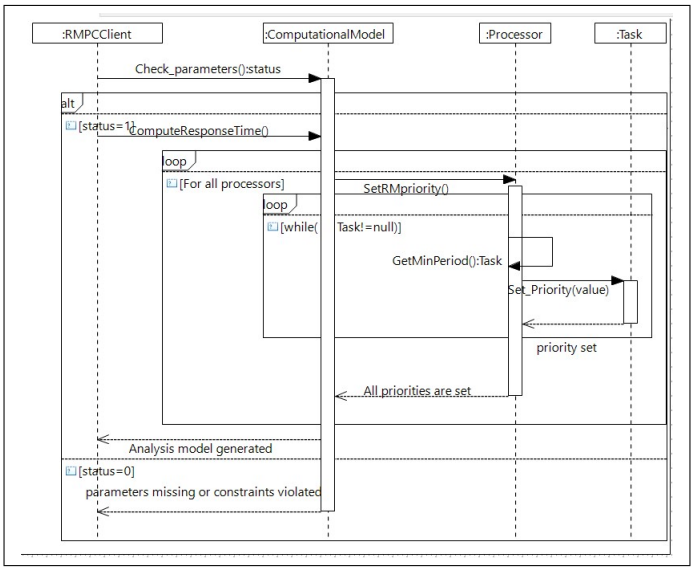

Figure 23: Behavioral description of the RMPC pattern using the UML sequence diagram

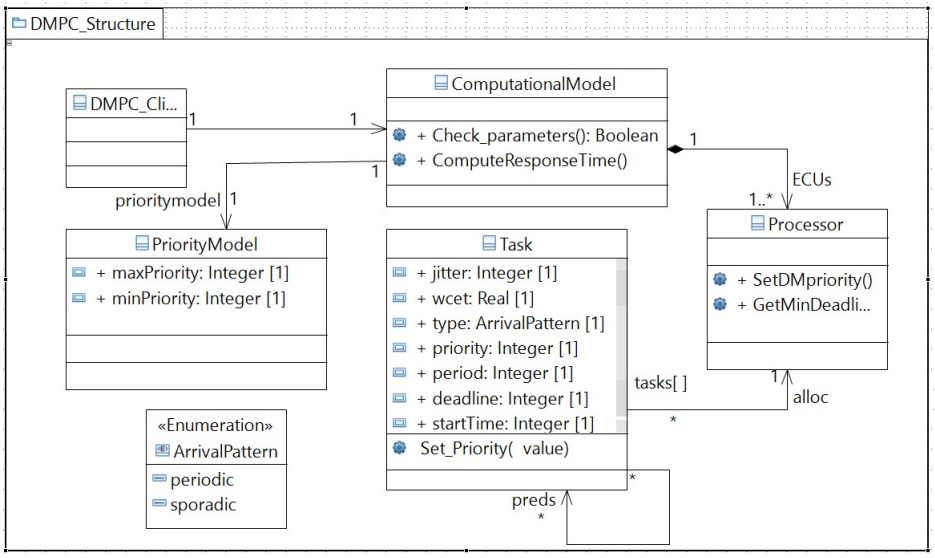

Figure 24: Structural description of the DMPC pattern using the UML class diagram

The same processor must perform all dependent tasks. Furthermore, this pattern can be used when the designer wants to define task priorities based on the deadline monotonic scheduling algorithm.
Structure. Figure 24 depicts the structural view of the DMPC pattern.
Behaviour. Figure 25 illustrates the behavioral view of the DMPC pattern.
Constraints. The constraints defined by this pattern are the same as those defined by the Fixed Priority Precedence Constraints (FPPC) pattern.

Related real-time verification patterns. This pattern has collaboration relationships with the Deadline Monotonic Shared Resource (DMSR) and Deadline Monotonic Communicating Tasks (DMCT) patterns, and a specialization relationship with the Fixed Priority Precedence Constraints (FPPC) pattern.

14 Fixed Priority Communicating Tasks (FPCT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Fixed Priority Communicating Tasks (FPCT).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern may be used when the designer aims to evaluate

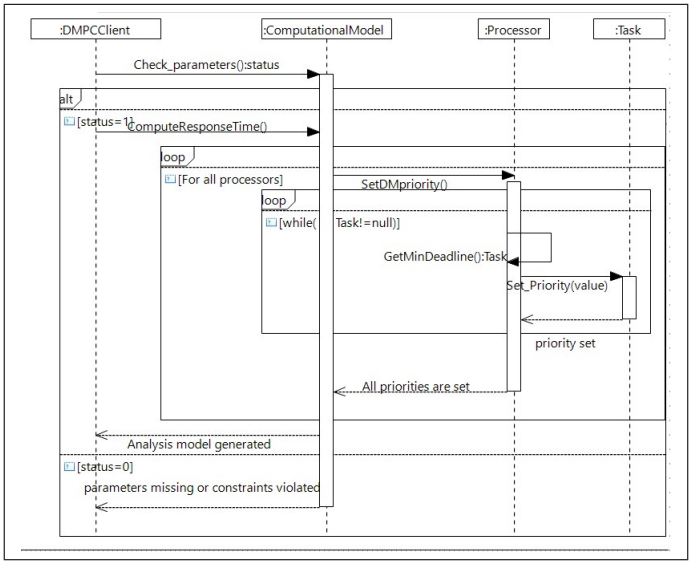

Figure 25: Behavioral description of the DMPC pattern using the UML sequence diagram

the feasibility of the real-time application by performing response-time analysis. The real-time application consists of a set of tasks to be executed on processors. The processors are connected via a network. The real-time application is expected to include at least two dependent tasks (i.e., tasks that must be executed in a specific order since they depend on each other's results). Dependent tasks must be assigned to distinct processors. Thus, they exchange messages via the network. The priorities of the different tasks are manually set by the designer in this pattern.

Structure. The structural view of the FPCT pattern is depicted in Figure 28.

Behaviour. Figure **??** illustrates the behavioral view of the FPCT pattern.

Constraints.

```
context Task
inv: self.preds -> forAll(t: Task| self.alloc <> t.alloc)
```

Related real-time verification patterns. This pattern has collaboration relationships with the Rate Monotonic Communicating Tasks (RMCT) and Deadline Monotonic Communicating Tasks (FPSR) patterns.

15 Rate Monotonic Communicating Tasks (RMCT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Fixed Priority Communicating Tasks (RMCT).

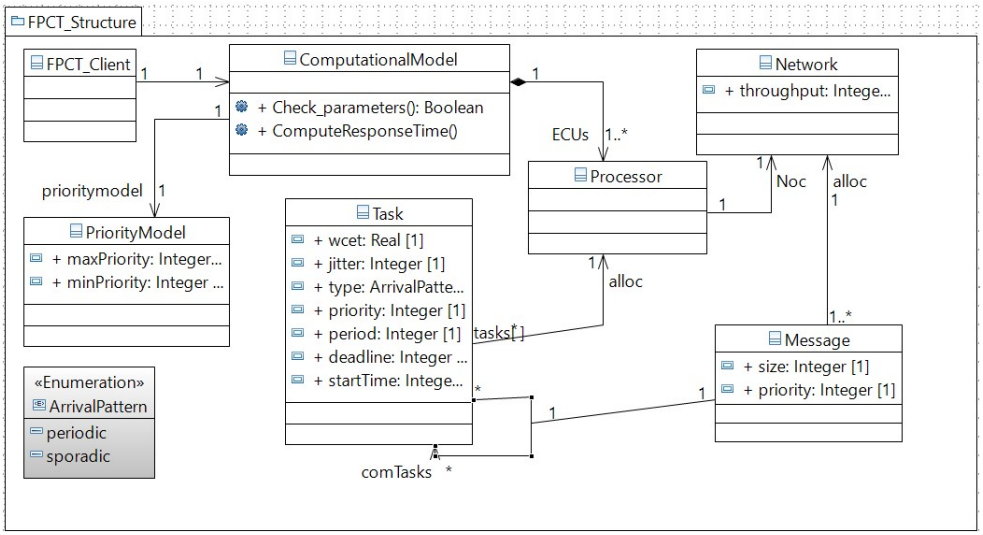

Figure 26: Structural description of the FPCT pattern using the UML class diagram

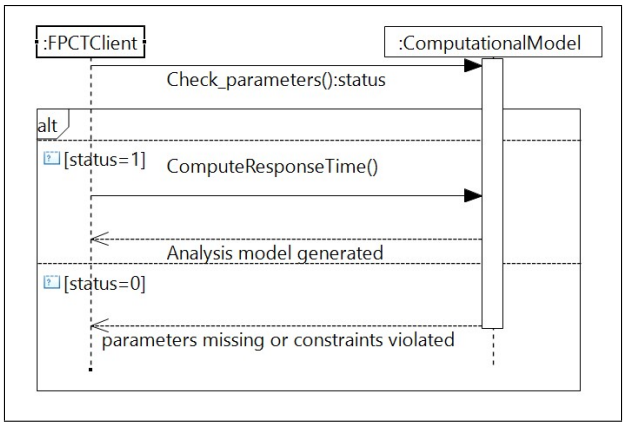

Figure 27: Behavioral description of the FPCT pattern using the UML sequence diagram

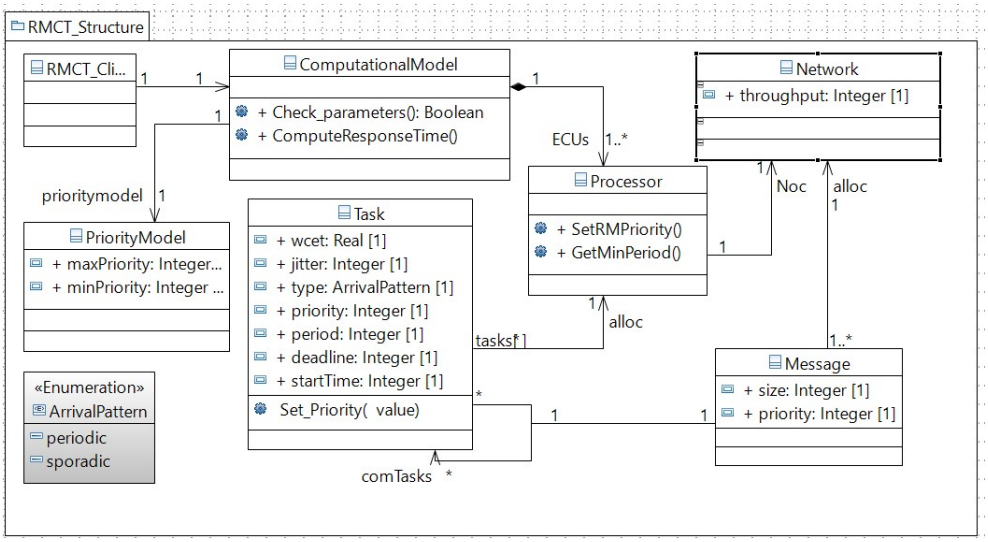

Figure 28: Structural description of the RMCT pattern using the UML class diagram

Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern may be used when the designer aims to evaluate the feasibility of the real-time application by performing response-time analysis. The real-time application consists of a set of tasks to be executed on processors. The processors are connected via a network. The real-time application is expected to include at least two dependent tasks (i.e., tasks that must be executed in a specific order since they depend on each other's results). Dependent tasks must be assigned to distinct processors. Thus, they exchange messages via the network. The rate monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.
Structure. The structural view of the RMCT pattern is depicted in Figure **??**.
Behaviour. Figure **??** illustrates the behavioral view of the RMCT pattern.
Constraints.

```
context Task
inv: self.preds -> forAll(t: Task|
self.alloc <> t.alloc)
```

Related real-time verification patterns. This pattern has collaboration relationships with the Rate Monotonic Shared Resource (RMPC) and Rate Monotonic Communicating Tasks (RMSR) patterns.

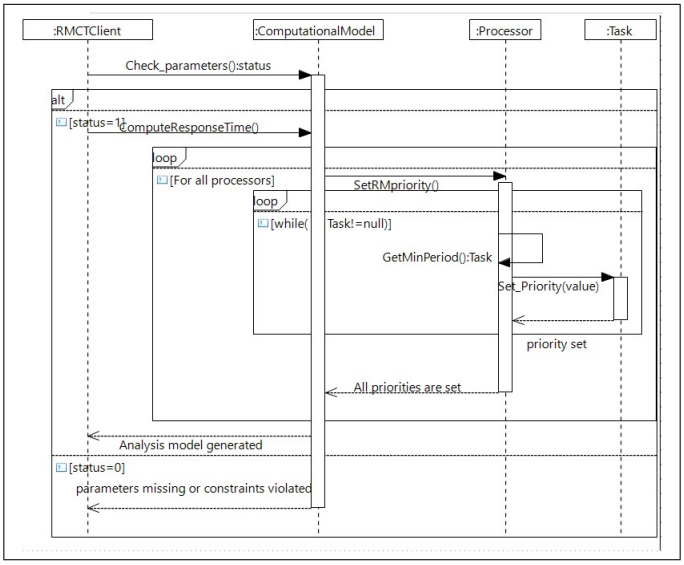

Figure 29: Behavioral description of the RMCT pattern using the UML sequence diagram

16 Deadline Monotonic Communicating Tasks (DMCT)

Pattern Name and Classification. Response Time Analysis Category (RTAC): Deadline Monotonic Communicating Tasks (DMCT).
Intent. This pattern avoids using incompatible or incomplete design models and allows analysis models to be built automatically. As a result, task response times can be calculated to assess the viability of the design options being considered.
Applicability. This pattern may be used when the designer aims to evaluate the feasibility of the real-time application by performing response-time analysis. The real-time application consists of a set of tasks to be executed on processors. The processors are connected via a network. The real-time application is expected to include at least two dependent tasks(i.e., tasks that must be executed in a specific order since they depend on each other's results).Dependent tasks must be assigned to distinct processors. Thus, they exchange messages via the network. The deadline monotonic scheduling algorithm is used to automatically assign the priorities of the various tasks in this pattern.
Structure. The structural view of the DMCT pattern is depicted in Figure 30.
Behaviour. Figure 31 illustrates the behavioral view of the DMCT pattern.
Constraints.

```
context Task
inv: self.preds -> forAll(t: Task| self.alloc <> t.alloc)
```

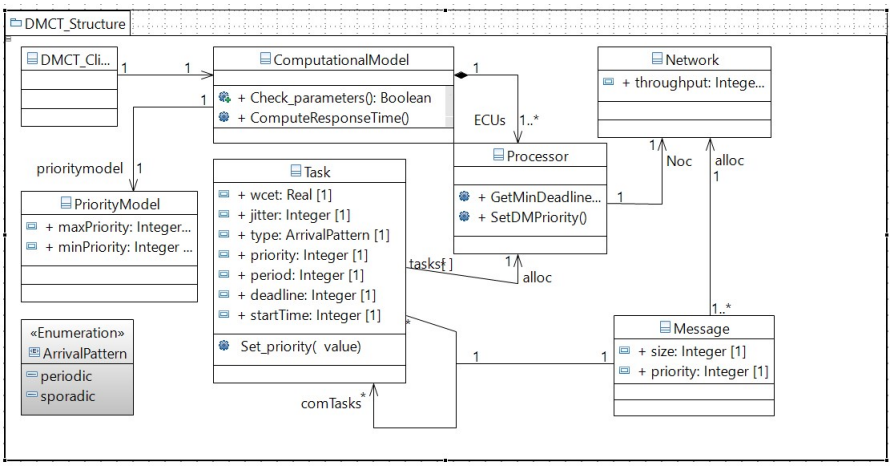

Figure 30: Structural description of the DMCT pattern using the UML class diagram

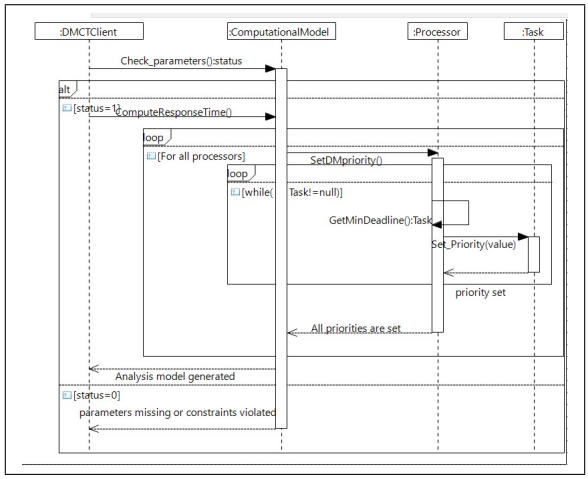

Figure 31: Behavioral description of the DMCT pattern using the UML sequence diagram

Related real-time verification patterns. This pattern has collaboration relationships with the Deadline Monotonic Shared Resource (DMPC) and Rate Monotonic Communicating Tasks (DMSR) patterns.