A Case Study in Performance Evaluation of Real-Time Teleoperation Software Architecture using UML-MAST

Francisco Ortiz, Bárbara Álvarez, Juan Á. Pastor, Pedro Sánchez

francisco.ortiz@upct.es
Universidad Politécnica de Cartagena (Spain)

Abstract. Reference architectures for specific domains can provide significant benefits in productivity and quality for real-time systems development. These systems require an exact characterization based on quantitative evaluation of architectural features refered to timing properties, such as performance, reliability, etc. In this work, an UML-based tool has been used to obtain a measure of performance between two alternative architectures. These architectures share the same functional components with different interaction patterns. The used technique is illustrated with an industrial and real case study in a well-known real-time domain: teleoperation systems. The obtained results show clear differences in performance between two architectures, giving a clear indication of which one is better from this point of view[1].

1. Introduction

Software engineering has demonstrated that much can be gained from developing generic software architectures for application domains during the last decade [5]. Such generic architectures comprise common properties for a family of related applications and can be instantiated for each specific system. Our accumulated experience in several research projects has allowed us to prove the interest of reusing a reference software architecture for teleoperation systems [2]. The use of such architecture has allowed reusing common software components in different applications, making easier their development and decreasing significantly their costs and development times.

In the mentioned architecture, the application domain was limited to applications in which service robots work in structured environments, whose geometrical characteristics are perfectly known before the operation is performed, and in which reactive behaviour is usually limited to sensor-driven safety actions. The ROSA (*Remotely Operated System Arm*) and TRON (*Teleoperated and Robotized System for Maintenance Operation in Nuclear Power Plants Vessels*) systems are an example of this type of applications for maintenance tasks in nuclear power plants [4]. In these systems, the operator is in charge of monitoring and operating the robot according to the

[1] This work has been partially supported by FEDER (TAP-1FD97-0823) and GROWTH project (GRD2-2001-50004)

information provided by the teleoperation system. This system receives commands from the operator and performs the corresponding actions for executing them.

The development of a blasting robot for ship hulls [7][8], whose characteristics differ from the mentioned above, led us to evaluate if the original architecture could be re-used for the new system. The system should work in unstructured environments and reactive behaviour is predominant during some operation phases. Given these requirements it is necessary to include new services (i.e. a computer vision system) or use the existing ones in different ways. As these services were not kept in mind when the original requirements were defined, the ATAM method *(Analysis Trade-Off Architecture Method)* [9] was applied to evaluate if the original architecture could be used for developing applications capable of guaranteeing the new requirements. ATAM was chosen as evaluation method because it is specifically defined to evaluate software architectures and considers a characterization of the quality attribute requirements at a level of abstraction very suitable for our evaluation purposes.

The original architecture considers the existence of a cinematic server capable of both testing a given trajectory and generating collision free trajectories according to the models of the robot and working environment and to the commanded destination point. The cinematic server was not used for on-line collision detection, because the systems were not allowed performing a motion not previously tested by mean of a previous simulation. However, this approach is not feasible for unstructured environments, simply because if the environment is not perfectly defined simulations are not useful except for training purposes.

But the issue here is not if such cinematic server is useful on line, but if any system (likely a vision system) can be used on line for collision detection and avoidance or, in other words, if the architecture is capable of supporting information and control flows not previously considered. For this reason, the preliminary evaluation simply tested if the cinematic server could be used on line while keeping in mind that such server should be sustituted by other. The result of evaluation was that performance requirements could not be met and it was necessary to propose modifications on the original architecture that combine the original components (trying to minimise the impact of the changes) with different interactions patterns.

The model used for representing the temporal and logical elements and real-time requirements of applications has been MAST (*Modeling and Analysis Suite for Real-Time Applications*), developed by the University of Cantabria (Spain) [6]. MAST allows a very rich description of the system, including the effects of event or message-based communication multiprocessor and distributed architectures as well as shared resource synchronization. The model is directly obtainable from a description of the system design using an UML-based CASE tool [11]. Previous works have proved the successfull of the tool over easier applications than the proposed case study here. We present our experience in using UML-MAST in the development of industrial and real applications.

The paper is organized as follows. In section 2 two alternative interaction patterns are presented in order to compare the performance of both schemes. In section 3 some relevant aspects of MAST are reviewed. Section 4 contains different models that represent different scenarios. In section 5 we describe the results of the analysis. Finally, section 6 gives some conclusions and discusses further work.

2. Architectures for teleoperation systems

Figure 1 shows the components taking into account for performance evaluation and their interaction patterns as they were described in the original architecture. Though the scheme showed in the figure is somewhat oversimplified is enough to ilustrated all the important issues. The components considered are:

- *CinServer*. This server provides the system with operations for checking if a given movement implies a collision between the robot and the operating environment, or with itself. This service was provided before the execution of a movement command in the original architecture. However, as said before, if the robot operates in unstructured environments, *CinServer* needs to receive the actual robot position in execution time from *HighLevelController*. An asynchronous client-server pattern is used in the collaboration diagram of the figure 1. In this scheme, *HighLevelControlle*r sends the current robot position to *CinServer* and retrieves asynchronously the answer telling it if the current motion can cause a collision.
- *UserInterface.* This subsystem 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.
- *LowLevelController*. This module physically actuates the robot to move it and to sense information from the robot in order to evaluate its global state and position. This state information is sent to *HighLevelController* by means of *updateStatus*. In general this subsystem can be ported to any platform, nevertheless its execution has been considered in the same node of the others subsystems to simplify the analysis.

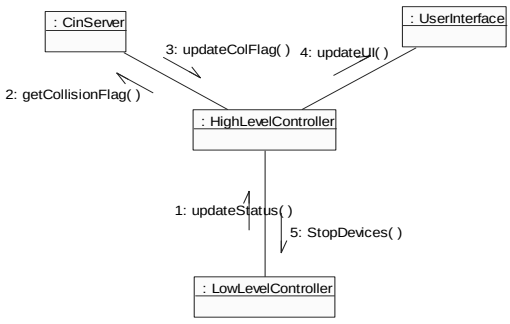

Fig. 1: Asynchronous client-server pattern between CinServer and HighLevelController

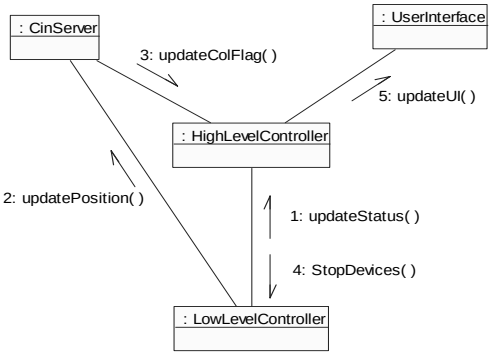

Fig. 2: Observer pattern between CinServer and HighLevelController

The interactions between *CinServer* and *HighLevelController* are a system bottleneck when *CinServer* is used on-line. For this reason the interaction pattern between them was modified according to the scheme showed in figure 2. In this scheme *CinServer* directly receives the robot position from *LowLevelController* and *HighLevelController* is subscribed to the events produced by *CinServer* following an observer pattern. The functional components of both schemes are the same, as well as their interfaces, but not their interaction patterns. It is also possible than *LowLevelController* can be subscribed to events or messages from *CinServer*, but this lead us to further modifications in the architecture and was not considered.

The model used for evaluating the timing requirements was based on the characterization of architecture timing behaviour described in [3]. Such description, based on *Rate Monotonic Analysis* method [10], is very exhaustive and allows the designers to reason with confidence about timing correctness at the tasking abstraction level, and it analyses whether the deadlines of the tasks can be guaranteed. In this way, the designer of a new application can reuse the architecture, and can easily check to whether the architecture can meet the timing requirements, but it assumes the interactions originally defined and not the new ones and is hard to use when new interactions have to be added. So, instead of using this model to compare the performance of the schemes showed in figures 1 and 2, a simpler model, that also uses RMA, was defined using UML-MAST. The proposed scenarios do not seek to carry out an exhaustive study of the performance, but allow us to compare behaviors with different interaction patterns.

3. Analysis of the architectures using UML-MAST

The MAST suite defines a model capable of describing the timing behaviour of a large set of real-time systems, including distributed systems and event-driven systems with complex synchronization schemes. In UML-MAST three views are represented:

- **Platform model.** This view allows to model the processing capacity of hardware and software resources which execute the activities of the system. There are two basic components:
    1. *Scheduling servers* to represent schedulable entities in a processing resource. If the resource is a processor, the scheduling server is a process, a task or a thread of control.
    2. *Processing resources* to represent hardware components and software infrastructure.

The platform model is shown in the figure 3. To simplify the model one node has been considered and all the Ada tasks were scheduled according to fixed priority policy.

- **Logical components model.** This view allows to model processing requirements of operations (methods, functions and procedures). In this model, shared re-

sources are defined. These resources are shared among different tasks and must be used in a mutually exclusive way[2]. For example, the data between the different subsystems and *HighLevelController* is exchanged through a shared buffer.

- **Scenario model.** This view allows to model the system as a set of transactions. Each transaction is activated from one or more external events, and represents a set of activities that will be executed in the system. Activities generate events that are internal to the transaction, and that may activate other activities. Special event-handling structures exist in the model to handle events in special ways. Internal events can have timing requirements associated with them. The previous models are common for both collaboration diagrams (figures 1 and 2). However, one scenario model was defined for each collaboration diagram respectively. These scenario models are described in the next section.

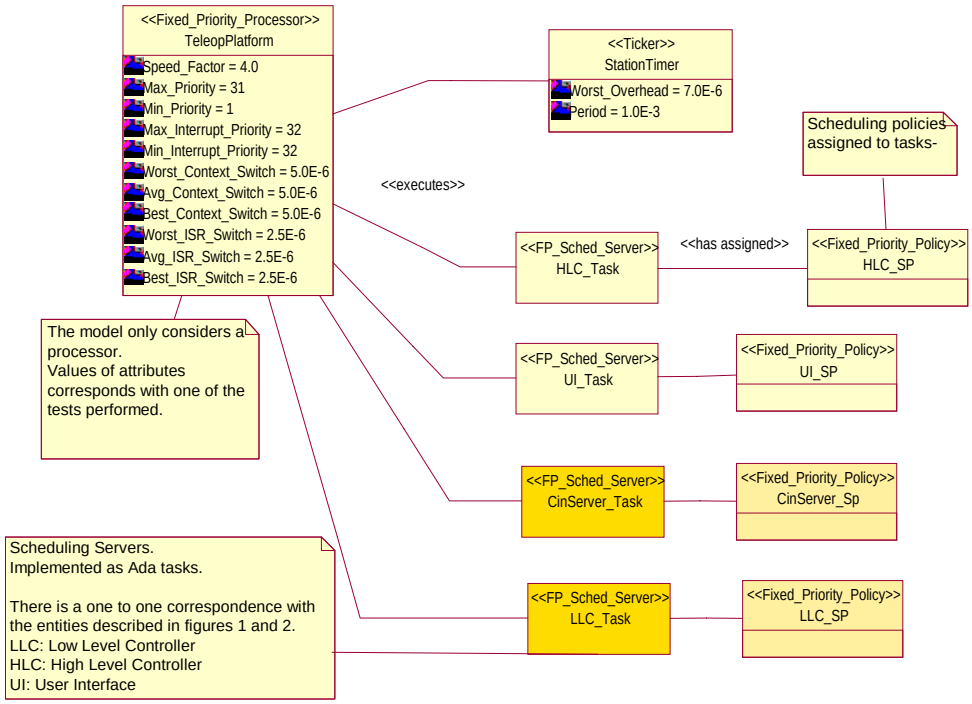

Fig. 3. Platform Model.

[2] The operations and resources for this case are not presented in this work for space reasons. The operations invoked by the tasks of platform model are presented in the scenarios model.

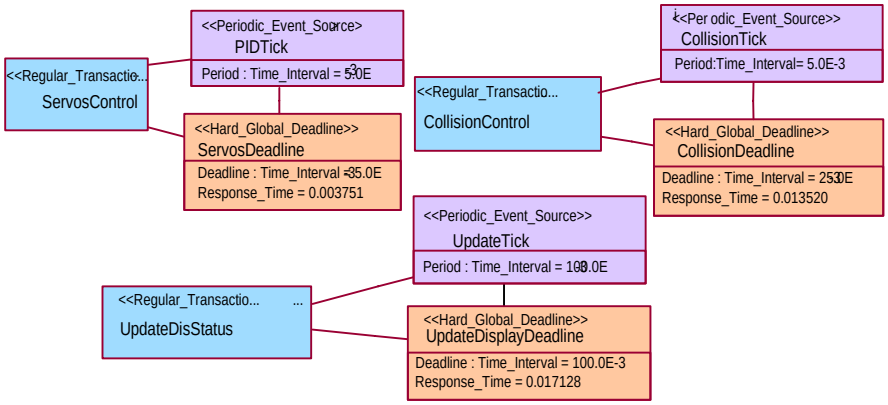

Fig. 4: Scenario Model 1.

4. Scenario models

Two scenario models (figures 4 and 5) have been elaborated in order to characterize the interactions described in the figures 1 and 2. Following the UML-MAST notation, several transactions have been considered and the corresponding timing parameters are attached as UML association relationships. For example, transaction *ServosControl* in figure 4 has attached a periodic arrival pattern, *PidTick*; stereotyped as *Periodic_Event_Source* (T= 5ms) and a deadline, *ServosDeadline*, stereotyped as *ServosDeadline* (D = 5 ms). Because architecture used to be developed before an exact definition of timing requirements is available, periods and deadlines have been assigned taking into account the usual values in the application domain. Moreover, timing requirements will be different along the different applications that will be developed using the architecture. For this reason, at the architectural level of abstraction and for the purposes of the analysis the important issue is not such values themselves, but (1) that they were the same in both schemes to allow the comparisons and (2) that architecture can be adapted to different timing requirements.

Figure 4 shows the scenario model related to the collaboration diagram of figure 1. In this scheme, the services from *CinServer* are explicitely invoked by *HighLevelController* when a new position data is received from *LowLevelController*. Three regular transactions are considered: *ServosControl*, *CollisionControl* and *UpdateDisStatus*.

In UML-MAST, each transaction is described by an activity diagram. Figures 6 and 7 describe the transactions *ServosControl* and *UpdateDisStatus*. The transaction *ServosControl* represents the servo control of the teleoperated mechanisms. The transaction *UpdateDisStatus* gives the updating of data in the graphical user interface of those mechanisms. Figure 8 shows the UML activity diagram associated to the transaction *CollisionControl*. This transaction represents how the collision detection is managed in the collaboration diagram of the figure 1. Each element of the activity diagram has a stereotype that define it in the UML-MAST model. It is beyond the scope of this paper to describe such model and the reader can see [6] for a detailed information. But very roughly:

- Each swingline is labeled with the name of the task that perform the activities of such swingline. So, it is easy to identify the tasks involve in each transaction an their interactions.
- Each transaction starts with the arrival of a timed clock driven event. So, initially the first task involves in a transaction is in a *Wait_State*. A *Wait_State* in the UML-MAST model represents that the task is waiting for an external event for its activation, that in all the cases considered such external event is a clock event. Note the correspondence between the names of the *Periodic_Event_Sourse* in figures 4 and 5 and the names of the *Wait_States* of figures 6 to 10.
- All the activities are stereotyped as *Timed_Activity*. A *Timed_Activity* encloses an operation or a set of operations whose timing behaviour is well described in the model. Such operations can include the access to shared resources. In this case it is possible to follow the original ceiling priority protocol or immediate ceiling priority protocol. It is important to remark that all the transactions can be performed simultaneously, despite they are describe in separate diagrams.
- Finally, transaction ends when reaching a *Timed_Stated* that describes then deadline of the transaction. Note the correspondence between the names of *Hard_Global_Deadline* of figures 4 and the names of the *Timed_Stated* in figures 6 to 10.

The collaboration diagram from figure 2 has associated the scenario model given in figure 5. This model considers the previous transactions *ServosControl* and *UpdateDisStatus*. However, the collisions detection is represented by the transaction *CollisionControl2*. In this case, the robot position information is sent from LowLevelController to CinServer without passing HighLevelController. Figure 9 gives the activity diagram associated to the transaction *CollisionControl2*. Finally, the activity diagram of figure 10 is exclusively associated to the scenario model 2 and represents the transaction *MonitorState.* This transaction includes the activities which are periodically performed by *HighLevelController* to receive new state data from *LowLevelController*. In the previous scheme (figure 1), this task was not necessary because it was included in the transaction *CollisionControl* since robot position and state are managed by *HighLevelController*. The previous models have been intencionally simplified. However, those simplifications make easier the interaction process given by the original scheme (figure 1). Even so, the response times are worse than the given by the another scheme (figure 2).

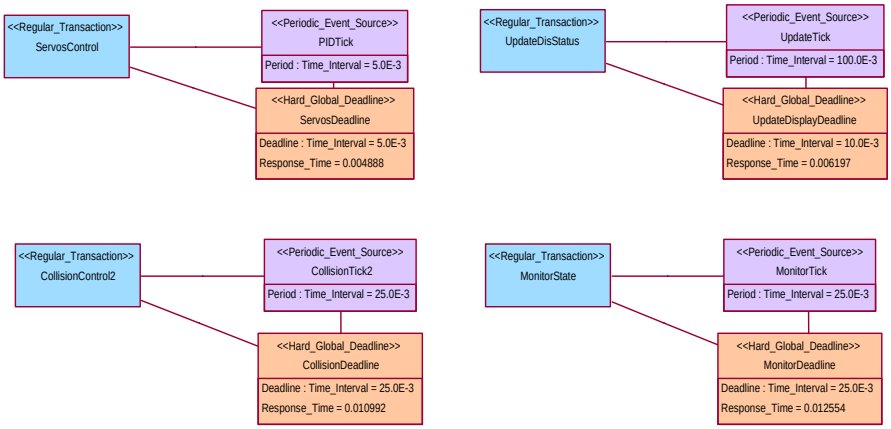

Fig. 5: Scenarios model 2

5. Results of the analysis

Table 1 shows the results of the analysis of the previously described scenario models. The MAST tool automatically sets task priorities following the *Rate Monotonic Scheduling* algorithm. The well known *Rate Monotonic Analysis* method has been considered. As the table shows, the response times for collision detection (*Collision-Control*) and user interface update (*UpdateDisStatus*) are bigger in the scenario 1 than in the scenario 2. However, the response time of the transaction *ServosControl* is a little bigger in the scenario 2 than scenario 1. This last transaction is independent both of the interaction patterns used by `HighLevelController`. It must be highlighted that all the considered simplifications benefit the original scheme of interactions. A more complex model that takes into account explicitly the internal tasks of *HighLevelController* (not included in this paper for space reasons) produced the results showed in Table 2 (as more tasks were considered, CPU capacity was increased respect to the used in Table 1 for avoiding deadlines expiration). It is remarkable that the response times corresponding to scheme 1 worsen rather fast when new tasks are added,while the corresponding to scheme 2 remain more stable. The unique adavantage of scheme 1 according to the tables is that the response time of ServosControl task is slightly shorter.

Transaction	Events arrival pattern	Deadline	Scenario 1 Response time	Scenario 2 Response time
ServosControl	Periodic, T = 5 ms	5 ms	3,8 ms	4,8 ms
UpdateDisStatus	Periodic, T = 100 ms	100 ms	17 ms	6,1 ms
CollisionControl	Periodic, T = 25 ms	25 ms	14 ms	
CollisionControl2	Periodic, T = 25 ms	25 ms		11 ms
MonitorStatus	Periodic, T = 25 ms	25 ms		13 ms

Table 1. Results of simulation.

Transaction	Events arrival pattern	Deadline	Scenario 1 Response time	Scenario 2 Response time
ServosControl	Periodic, T = 5 ms	5 ms	3,1 ms	3.4 ms
UpdateDisStatus	Periodic, T = 100 ms	100 ms	23 ms	8,1 ms
CollisionControl	Periodic, T = 25 ms	25 ms	23 ms	
CollisionControl2	Periodic, T = 25 ms	25 ms		10 ms
MonitorStatus	Periodic, T = 25 ms	25 ms		16 ms

Table 2 Results of simulation.

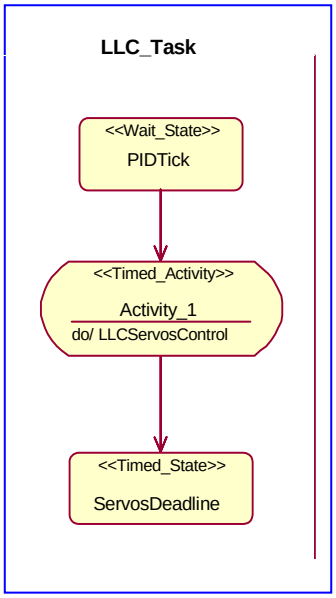

Fig. 6: Scenarios Model 1 and 2. ServosControl transaction

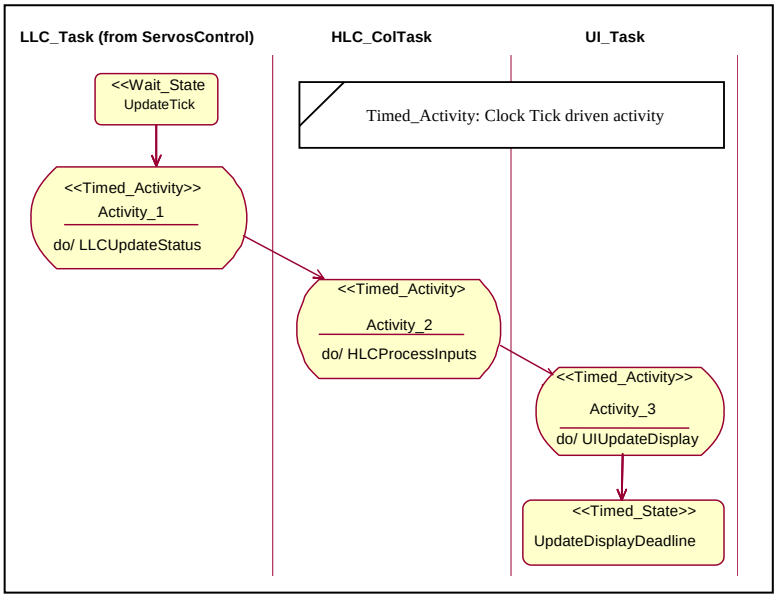

Fig. 7: Scenarios Model 1 and 2. UpdateDisStatus Transaction

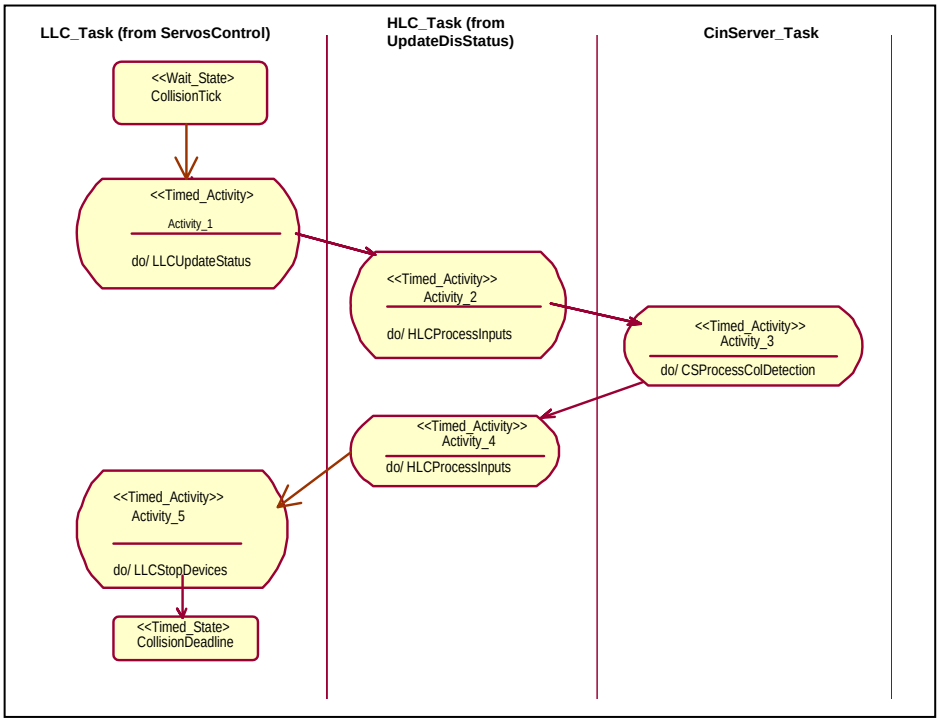

Fig. 8: Scenarios Model 1. CollisionControl transaction

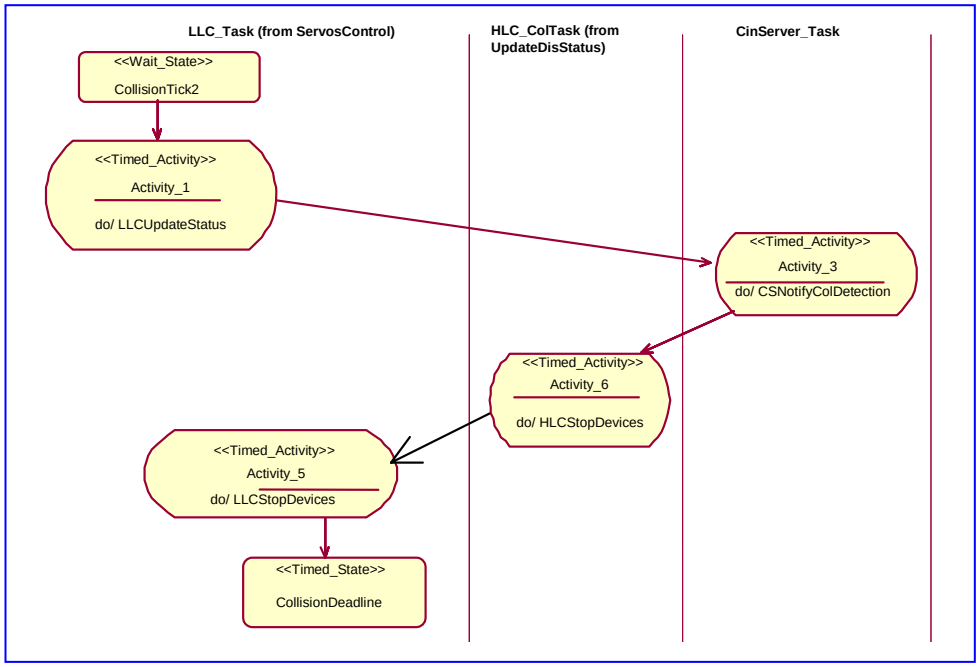

Fig. 9: Scenarios Model 2. CollisionControl2 transaction

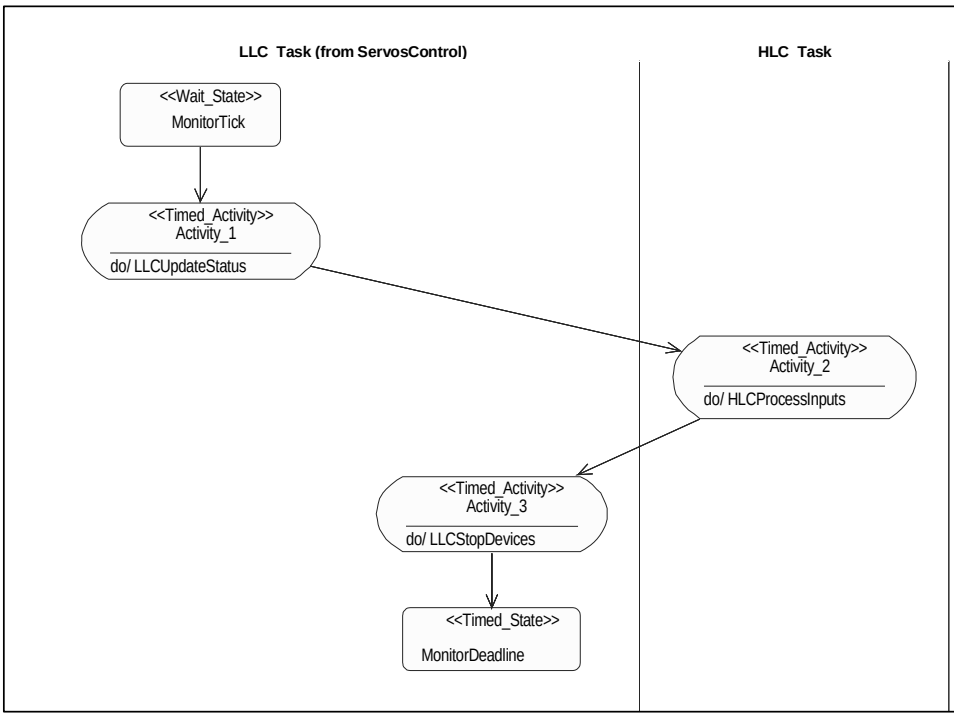

Fig. 10: Scenarios Model 3
`MonitorState Transaction`

6. Conclusions

Teleoperation systems can be very diverse, but this diversity affects more to the interaction patterns among the components than to the functional decomposition. The two interaction schemes introduced in this work share the same functional decomposition, however their timing behaviour is very different. Although from the functional point of view both designs could be suitable, the first (asynchronous client-server) is not acceptable with regard to the performance when *CinServer* has to be used on line. However, the first scheme is very adequated for working in simulation mode as the applications considered for the design of the architecture do. It is even not unrealistic to consider applications that should change from one scheme to another in different modes of operation. The result of the evaluation against performance requirements was that the original architecture was not appropiate (without major modifications) for the new system. However this is not the main conclusion of the evaluation, nor the most useful one at long term, but:

- Most of the relevant trade-offs are referred to the interaction patterns among components and not to their enclosed functionality.
- It would be possible to re-use a significant number of existing components if it would be possible to modify their interaction patterns maintaining their functionality and interfaces.

To summarise, it is much more interesting to define an architectural framework that defines a set of rules that allow sharing the same components among systems with different architectures than try to define a software architecture for large domains in which it is nearly impossible to reach the requirements of all of the potential applications. And the first rule of such architectural framework should be to consider the interaction patterns as design and parametrizable features at the same level that the components. In this way, some original components can be reused to work in non-structured environments when other interaction patterns are selected. In the same way, other components can be replaced by other new ones (i.e. collisions detection subsystem can be replaced by a computer vision subsystem).

The study of the performance during the first design phases is useful to compare different design solutions. At an architectural level such analysis can be of coarse grain and can be completed in later development phases. But even so, it is necessary an automated support of evaluation process and a standard notation as UML, despite all its drawbacks for describing architectures. So, the profile UML-MAST has been an excellent help where two research areas converge: software engineering and real-time systems.

References

1. Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652, 1995.
2. A. Alonso, B.Alvarez, J.A. Pastor, J.A. de la Puente, A.Iborra. "*Software architecture for a robot teleoperation system*". Proceedings of the 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control. Portugal. April 1997.
3. B. Álvarez et al.. "*Timing Analysis of a Generic Robot Teleoperation Software Architecture*", Control Engineering Practice, vol 6(6), pp.409-416. June, 1998.
4. B. Álvarez et al.. "*Developing multi-application remote systems*" Nuclear Eng. International, vol. 45(548). March 2000.
5. L. Bass et al. "*Software Architecture in Practice*". Addison-Wesley, 1998.
6. J.M. Drake et al.. "*Mast Real-Time View: Graphic UML Tool for Modeling Object Oriented Real Time Systems*". Group of Computers and Real-Time Systems. University de Cantabria (Internal Report), 2000. http://ctrpc17.ctr.unican.es/mast.html.
7. Environmental Friendly and Cost-effective Technology for Coating Removal" (EFTCoR). GROWTH project ref. GRD2-2001-50004, 2001.
8. A. Iborra et al. "*Service robot for hull blasting*" . The 27th Annual Conference of the IEEE Industrial Electronics Society (IECON´01), pp. 2178-2183. November, 2001.
9. R. Kazman et al., "*ATAM* ^{SM}: Method for Architecture Evaluation", Technical Report, CMU/SEI-2000-TR-004, 2000.
10. M.H. Klein et al. "*A Practitioner´s Handbook for Real-Time Analysis Guide to Rate Monotonic Analysis for Real-Time Systems*". Kluwer Academic Publishers, 1993.
11. Reference manual. Rational Sw Corp, 2000. Available at www.rational.com.