Experience Building Non-Functional Requirement Models of a Complex Industrial Architecture

Daniel Dominguez Gouvêa, Cyro de A. Assis D. Muniz, Gilson A. Pinto
Chemtech a Siemens Company
RJ 20011-030, Brazil

Alberto Avritzer
Siemens Corporate Research
Princeton, NJ 08540

Rosa Maria Meri Leão, Edmundo de Souza e Silva
Federal University of Rio de Janeiro, COPPE
RJ 21941-972, Brazil

Morganna Carmem Diniz
Federal University of the State of Rio de Janeiro, RJ, Brazil

Luca Berardinelli
University of L' Aquila, Italy

Julius C. B. Leite
Universidade Federal Fluminense, Niterói, Brazil

Daniel Mossé
University of Pittsburgh, Pittsburgh, PA

Yuanfang Cai, Mike Dalton
Drexel University
Philadelphia, PA

Lucia Kapova, Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, GE

ABSTRACT

In this paper, we report on our experience with the application of validated models to assess performance, reliability, and adaptability of a complex mission critical system that is being developed to dynamically monitor and control the position of an oil-drilling platform. We present real-time modeling results that show that all tasks are schedulable. We performed stochastic analysis of the distribution of tasks execution time as a function of the number of system interfaces. We report on the variability of task execution times for the expected system configurations. In addition, we have executed a system library for an important task inside the performance model simulator. We report on the measured algorithm convergence as a function of the number of vessel thrusters. We have also studied the system architecture adaptability by comparing the documented system architecture and the implemented source code. We report on the adaptability findings and the recommendations we were able to provide to the system's architect. Finally, we have developed models of hardware and software reliability. We report on hardware reliability results based on the evaluation of the system architecture. As a topic for future work, we report on an approach that we recommend be applied to evaluate the system under study software reliability.

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Statistical methods; Reliability*; D.2.8 [**Software Engineering**]: Metrics—*Performance Measures*

General Terms

Performance

Keywords

Performance, Modeling, Simulation

1. INTRODUCTION

In this paper, we present our experience with the application of performance, reliability, and architecture modeling approaches to the assessment of a Dynamic Positioning System (DPS) architecture. The Dynamic Positioning System (DPS) under study is a software prototype that has been in development at Siemens-Chemtech for several years and is targeted to be deployed to control large mission-critical vessels. Specifically, it is being designed to be deployed for monitoring and controlling deep-water oil-drilling vessels. These systems have very stringent performance and reliability requirements that need to be demonstrated prior to obtaining the required quality certifications. Therefore, modeling of performance and reliability is a very important project objective.

The main contribution of this paper is the presentation of a detailed experience report of the application of several complementary performance, reliability and adaptability models to the architecture assessment of a complex mission-critical system.

In a companion paper [10], we have presented a new architecture review process that used a globally distributed review team to perform architecture risk assessment of this Dynamic Positioning System (DPS). The DPS uses the vessel's thrusters to control the vessel position and heading. We employed a team of experts to identify and categorize the architecture risks related to the performance, reliability and adaptability non-functional requirements.

The results presented in [10] were based on teleconference discussions and face-to-face interviews of the architects and domain experts.

In contrast, in this paper we present experimental results that were obtained by modeling performance, reliability and adaptability using data derived from the implementation of the DPS system architecture. Specifically, the performance modeling results presented in this paper were based on actual measurements performed on the implemented software prototype. The measurement results were analyzed and used to calibrate the models. In one instance, the actual implemented software library was executed inside the simulation model. The reliability modeling results presented for the hardware reliability were based on the analysis of the system architecture using hardware failure rates obtained from the hardware vendors. In contrast, for the software reliability modeling approach, no data was yet available to instrument the model and therefore, we only present the approach to be used on future experiments. The adaptability modeling approach was performed by generating Design Structure Matrixes (DSMs) from the Enterprise Architect (EA) documentation tool and from the actual DPS prototype source code.

In summary, we have built several models to assess different aspects of the DPS architecture. These models were instrumented by measuring the performance of parts of the implemented software. The execution of the models provided both positive and negative feedback to the project. As a result of this effort, the project learned that the tasks as implemented could be proven to be schedulable and that the sensitivity analysis showed that the system would perform well even for vessels with a larger number of thrusters. The evaluation of convergence characteristics of the thruster allocation under six different scenarios was beneficial as it showed good convergence for the six real scenarios that were designed by the domain experts. The difference in convergence behavior for different number of thrusters, indicates a need to test additional scenarios before a production version of the DPS system is certified for production deployment. The framework developed for the thruster allocation evaluation is very efficient and could be used to test hundreds of scenarios. The adaptability assessment evaluation uncovered several discrepancies between documentation and implementation. We were able to provide good feedback to the project about the need to continuously maintain the architecture documentation up to date. The software reliability modeling approach was developed and is presented as a topic for future work, as for accurate software reliability assessment extensive failure data collection is required.

The outline of the paper is as follows. In Section 2 we present an overview of the Dynamic Positioning System architecture and the information flow from the sensors to the thrusters. We describe the most important task types, their responsibilities, and how these tasks are activated. In Section 3 we present the three different approaches that were used to analyze system performance: worst case analysis *Real-Time* modeling, *Stochastic Modeling*, and *Tangram-II* actual implementation simulation modeling. The objectives of the three different performance modeling approaches are:

1. In the *Real-Time* modeling performance sub-section, we report on the results that were obtained from the DPS prototype system. Data was collected on 1,000,000 execution instances, for three different vessel configurations. The worst-case execution times were computed and we were able to show that the system is schedulable under the Rate Monotonic Scheduling discipline (RM),

2. In *Stochastic Modeling* performance sub-section, we report on experiments that were conducted using a *Palladio Component Model (PCM)* based high-level simulation model. This model was instrumented using data that was obtained from measurements of the system tasks execution time. The objective of running experiments using the *Palladio Component Model (PCM)* was to understand the full-distribution of tasks execution time and to assess the impact of number of sensors and thrusters on the tasks execution time.

3. In *Tangram-II* implementation based modeling approach the actual library implemented for the thruster allocation was executed inside the Tangram-II model to evaluate some of the thruster allocation important characteristics such as the number of iterations required for convergence, and the force distribution among the different thrusters as a function of the number of thruster actually used by the vessel. The framework created for running the library is very useful as it allows for a controlled and efficient execution of the thruster allocation model.

In Section 4 we present results of our experiments using an architecture adaptability modeling approach. We compared source code based and design based Design Structure Matrixes (DSNs), and we report on our experience with adaptability assessment of the system architecture. In Section 5 we present experimental results based on a Tangram-II model of hardware reliability. The reliability model was created using the system architecture and it was instrumented with hardware failure data. In the software reliability part of Section 5 we present a proposal for an approach to evaluate the DPS software reliability. Section 6 contains our conclusions and lessons learned.

2. DYNAMIC POSITIONING SYSTEM

The non-functional requirement models presented in this paper were applied to the Dynamic Positioning System (DPS) project. An overview of the DPS project architecture was presented in [10]. The system consists of sensors, controllers (aka IPU), human-machine interfaces (HMI) and thrusters. This paper focus is in the system's main flow, where the IPU plays the major role.

The IPU houses the system core, being responsible for consolidating the data from all sensors, computing the force needed to keep the vessel in the desired position, commanding the thrusters, and making important process data available. The IPU uses the QNX Neutrino Real-Time Operating System [16]. There are four kinds of tasks running in the IPU:

1. The *DataRetrieval* task (DR) is responsible for getting data from a specific sensor. There are three kinds of sensors in the system, each one of them with triple redundancy, which gives a total of 9 DataRetrieval tasks,

2. The *DataLayer* task (DL) stores the real-time values collected from sensors, from control tasks, and from the operator. The *DataLayer* task makes the collected data available for every other tasks that needs to access this data. It is essentially a memory-resident database that can be used to synchronize the tasks (passing information from one task to another),

3. The *DynamicPositioning* task (DP) is the main system task. It is responsible for running the mathematical algorithms [13] that compose the dynamic positioning system (in particular the thruster allocation algorithm), and also for sending commands to the thruster system,

4. The *IPUManager* task (IPUM) is responsible for coordinating the IPU redundancy. The *IPUManager* is responsible for selecting one and only one IPU to be the master, leaving the other two as backup stations. Since this selection is done periodically it is also used to keep all three IPUs synchronized. However, we do not consider the IPUM task in the timing analysis since it has a very low execution time.

The IPU information flow is illustrated in Figure 1. A *DataRetrieval* task is instantiated for each sensor that is connected to the system. The measurements collected by the *DataRetrieval* tasks are transferred to the *DataLayer* task and are routed to the *DynamicPositioning* task.

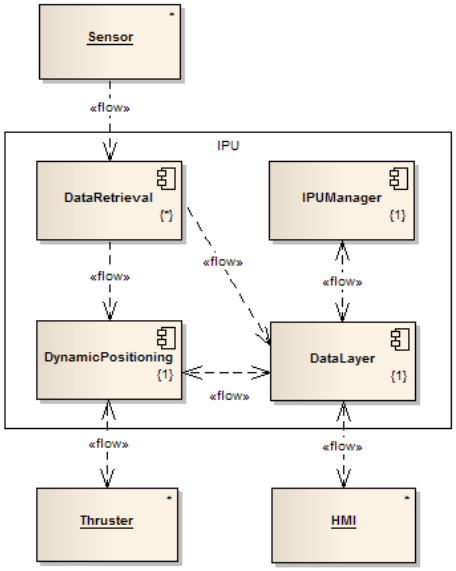


Figure 1: IPU Information Flow

The control loop is the flow that begins in the Sensors, passes through the DataRetrieval and the DynamicPositioning tasks and ends in the Thrusters. The diagram in Figure 2 shows the relationships between the tasks.

3. PERFORMANCE

In this section we present the three performance modeling approaches that were developed to assess real-time performance, task execution time as a function of the number of system interfaces, and the thruster allocation convergence characteristics.

The real-time task model level of abstraction assesses the impact of worst case control loop execution time on the system ability to satisfy the real-time requirement, as the the control loop must be executed in one second. The stochastic analysis model studies the impact of the number of sensors and thrusters on the control loop execution time distribution. The actual implementation approach executes the thruster allocation module inside a performance model to assess the impact of the number of thrusters on the convergence characteristics of the thruster allocation library, because the most critical task in the control loop is the thruster allocation algorithm. The thruster allocation algorithm is based on an iterative solution of an optimization problem. It is a small part of the control loop, but the investigation of the convergence characteristics of the thruster allocation algorithm is a very important modeling objective as bad convergence characteristics could have a significant impact on the ability of the DP system to control the vessel's position.

3.1 Real-time Performance

Real-time is a property that allows reasoning about time and temporal characteristics of the system. In particular, for this DP system, the approach adopted in the DP architecture to implement real-time was to use a periodic task set that repeats execution at specific moments in time. The fixed-priority scheduling discipline used is known as Rate Monotonic Scheduling [4].

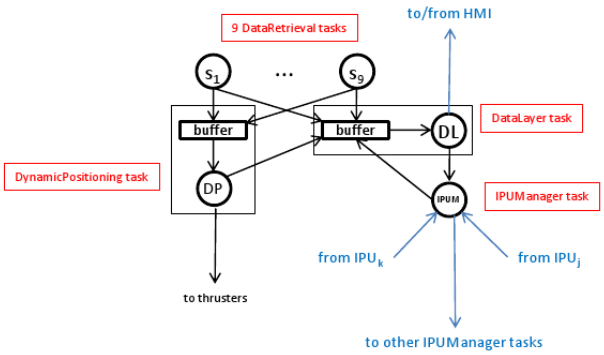


Figure 2: DP tasks relationships

All the described tasks are periodic, with a $1s$ period, and this period was chosen also as the cycle execution. Experiments were conducted with a prototype system that executes each of the tasks individually and independently, and measures the execution time (ET) of a task for each instance executed. We measured the ETs of each task from 1,000,000 executions instances, for 3 different vessel configurations. These configurations are based on the number of thrusters in the vessel, and we used typical values of 4, 6, and 8 thrusters. In all cases, only the DP task has different worst-case execution time (WCET), DL and DR keeping the same execution times. The DP WCETs obtained were, approximately, $106.0ms$, $107.8ms$, and $168.9ms$, for 4, 6, and 8 thrusters, respectively. In Table 1, we show the worst-case, the average case and other statistical measures of the ETs of the tasks, for an 8 thrusters configuration, i.e., the most demanding one.

Table 1: Execution times (ET) (μs)

Measure	DL	DR	DP
Worst-case ET	2039.6	1038.8	168875.0
Minimum ET	21.8	71.1	97596.0
Average ET	155.0	152.6	109268.7
Standard deviation	55.2	102.3	4841.7
Median	145.1	110.8	108877.0
Percentile (0.99)	286.27	394.7	122783.0

As indicated in the Table, in 99% of the cases the execution times are well below the WCETs, and this indicates the existence of an additional spare capacity for running background (non-critical) tasks. We note that even if there is a major reduction in execution times of the DR and DL tasks, the DP task is the dominant in terms of WCET and therefore dwarves the other tasks.

The OS used in the DP system is QNX, which allows for specification of real-time tasks, that is, critical tasks that have fixed high priorities and can pre-empt lower priority tasks. It should be noted that the tasks are independent. Since the system is periodic and all tasks have the same period, we can simply add their WCETs and

compare the sum with the period/cycle length (task switching overhead is negligible in this system). In this case, the total execution time is less then $181ms$ for the 12 tasks (9 DR, 1 DL, 1 DP, and 1 IPUM tasks), corresponding to a total CPU utilization of less than 19%.

More precisely, it can be shown that for a set of independent periodic tasks, under the RM scheduling discipline, if the utilization restriction below is attained [4]:

$$\sum_{i=1}^{n} \frac{WCET_i}{T_i} \leq n(2^{1/n} - 1) \quad (1)$$

where T_i is the task period and n the number of tasks, then the system is guaranteed to be schedulable (the restriction expressed in Equation 1 is a sufficient condition). For large values of n, the right-hand side of the Equation converges to 69.3%, that is greater than the computed DP utilization of 18.03%. Therefore, the DP system is schedulable and all deadlines will be met.

3.2 Stochastic Performance Analysis

In addition to reasoning on the worst-case execution time as presented in the previous section, we study the execution time *distribution* for the tasks on the IPU node. Studying the execution time distribution gives additional insight into the timing behaviour of the system, and enables us to estimate how quickly the control loop executes in most cases (e.g. in 95% of all cases). For systems where rare misses of the deadline are acceptable, stochastic analysis can give less conservative estimates for performance and thus avoid oversizing of resources. In the DPS system, rare misses would be acceptable because the system can continue to function with the results of the previous control loop iteration.

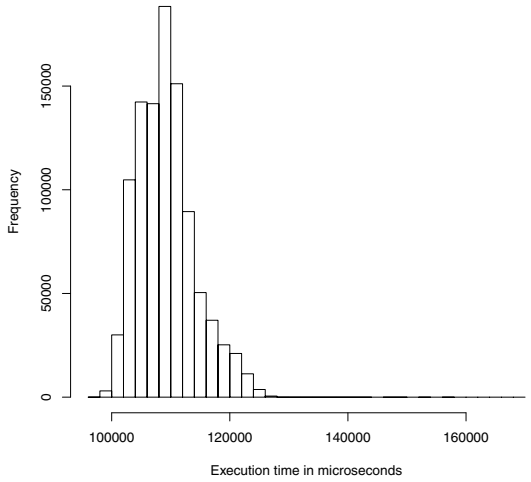
For the current system design the results from the previous section show that the control loop deadline is satisfied in the worst case. The analysis in this subsection provides additional results concerning the sensitivity of the control loop execution time to changes in the vessel configuration.

The implementation of our approach is based on an architectural modeling language called *Palladio Component Model (PCM)* [3, 14]. The PCM is a modelling language specifically designed for performance prediction of component-based systems, with an automatic transformation into a discrete-event simulation of generalised queuing networks. Its available tool support (PCM Bench) allows performance engineers to predict various performance metrics, including response time, throughput and resource utilization.

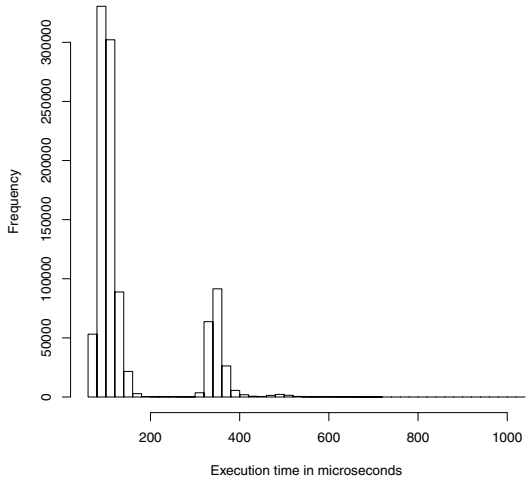
In a PCM model, time consumptions of single tasks can be modelled as generalised distribution functions, approximated using stepwise functions as shown in Figure 3. Thus, accurate distributions of the overall performance metrics can be derived by simulation.

We modeled the DataRetrieval, DataLayer, and DynamicPositioning components using measurements of the IPU tasks on a PCM model. The input data were execution time measurements for the DataRetrieval tasks of three different types of sensors (Gyro, GPS, Anemometer), for the DataLayer setDataPoint operation, and for the DynamicPositioning task, which contains the thruster allocation. We approximated the measured execution time distributions by step functions (some are shown in Fig. 3) and fed them into the PCM model. Figure 4 visualises an excerpt of the resulting PCM model: each component's behaviour is modelled and annotated with the measured execution time distributions.

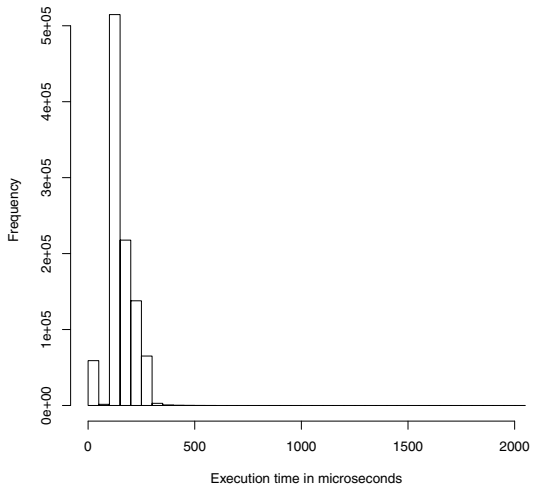
Simulation results with the predicted execution time distribution are shown in Figure 5. The execution time varies between 110 ms and 130 ms. The distribution has two modes and positive skew (i.e.



(a) Dynamic Positioning (Thruster Allocation)



(b) Data Retrieval Gyro



(c) Data Layer

Figure 3: Measured Execution Times of Tasks for Stochastic Performance Analysis

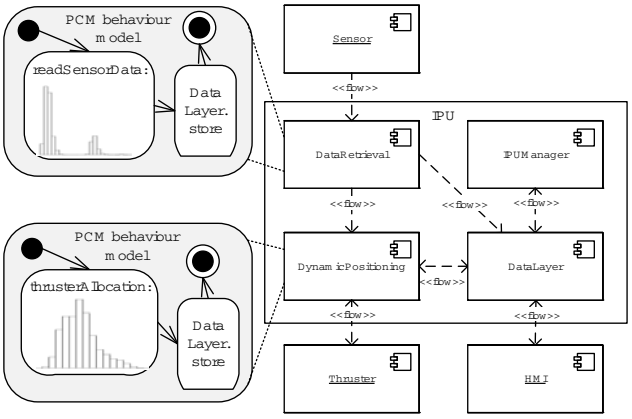


Figure 4: Visualization of an PCM model excerpt: Measured execution times from Fig. 3 are annotated to the PCM behaviour model

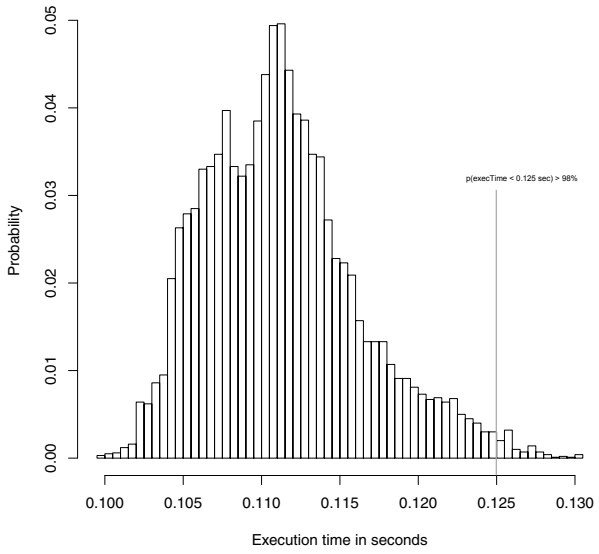


Figure 5: Execution time distribution of the control loop with 3 sensors and 8 thrusters

a longer tail on the right side). The quantiles of the distribution tell us how likely it is for the execution time to be below a given threshold. For example, the execution time is lower than 125ms in more than 98% of all cases (marked in the figure).

3.3 Sensitivity Analysis

In this section, we analyse the impact that added components have on the tasks execution times, when, (1) new types of sensors are added for further calculations, which leads to an increase of input data and messages, (2) the number of thrusters of the vessel varies.

For both extension scenarios, we perform a sensitivity analysis by first re-evaluating the real-time performance and then determining the execution time distributions by running the PCM simulation.

If new types of sensors are added to the system (first extension scenario), more DataRetrieval tasks (one per added sensor) have to be executed on the IPU node. As every sensor is triple redundant, adding one more functional sensor would require the addition of three new physical sensors. Domain experts estimate that up to six different functional sensors could be required on a vessel, leading to up to 18 physical sensors. We assume that new types of sensors will have similar computational demands as the existing ones. In the PCM model, we vary the number of functional sensors from the currently existing three sensors to a maximum of nine sensors and study system performance. The number of thrusters is set to eight.

From a hard real-time point of view, where just WCETs are taken into account, a configuration with 8 thrusters and 9 different kinds of sensors (and thus 27 DataRetrieval tasks) would imply in a maximum processor utilization of 19.9%, and this would satisfy the condition indicated by Equation 1.

The results of the stochastic analysis are shown in Figure 6: the number of sensors has small impact on the overall execution time of the three IPU tasks. Again we observe that, even if 9 functional sensors are used, the execution time stays well below the one second deadline. Extrapolating our predictions linearly, we estimate that the critical amount of deployed sensors for the required 1 second deadline lies higher than 1000 functional sensors.

For the case that the numbers of thrusters varies (second extension scenario), the time required for the call to the thruster allocation algorithm changes. Domain experts estimate that vessels have typically from 4 to 8 thrusters.

We measured the execution time of the calls to the prototypical thruster allocation algorithm for 4, 6, and 8 thrusters and fed the different measured execution time distributions into the PCM model. The execution time distribution as a function of the number of thrusters was estimated from the simulation model. The number of sensors is set to six.

The results of the stochastic analysis are shown in Figure 8: the number of thrusters has a higher impact on the execution time of the IPU node tasks. Thus, vessels with less thrusters than the initial configuration with eight thrusters require significantly less execution time; increasing the number of thrusters to more than eight would significantly increase execution time.

3.4 Thruster Allocation Algorithm Analysis

Sections 3.1, 3.2 and 3.3 analyze the worst-case execution time and the execution time probability distribution for the control loop of the DP system. The control loop analysis is based on measurements taken from experiments conducted with a prototype system. One of the conclusions of that analysis was that the control loop execution time is impacted by the number of vessel thrusters.

In contrast, in this sub-section we simulate several realistic ves-

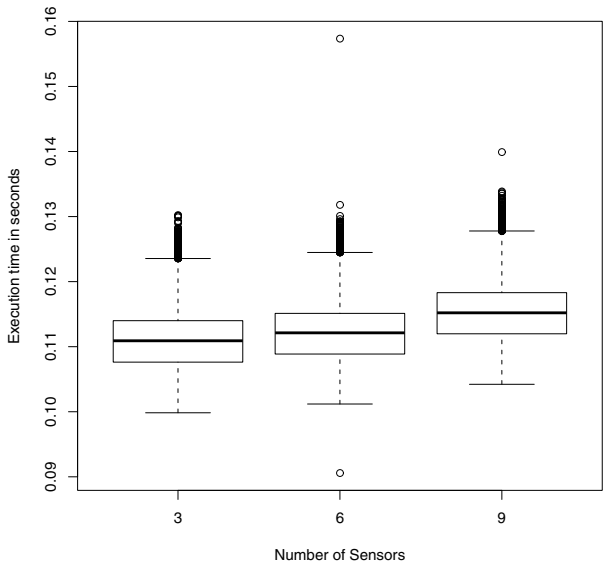


Figure 6: The execution time distribution of the IPU tasks is shown for three different amount of functional sensors. Increasing the number of functional sensors to 6 (middle box) or 9 (right box) leads to an increased execution time, which is well below the deadline of 1 sec.

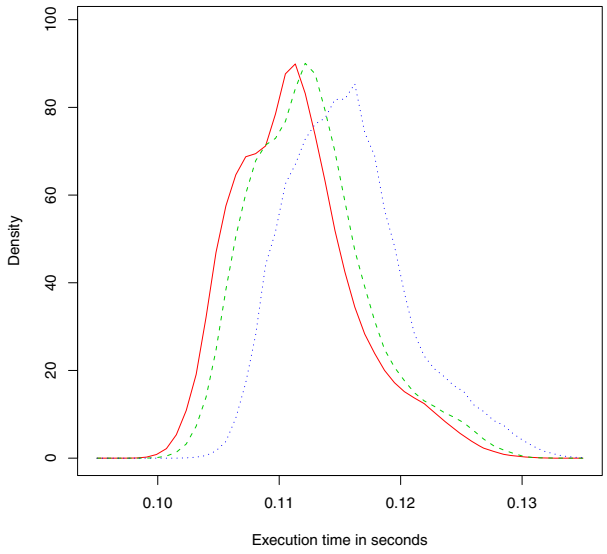


Figure 7: Approximated Density Functions of Sensor Number Change for 3 sensors (red solid line), 6 sensors (green dashed line), and 9 sensors (blue dotted line).

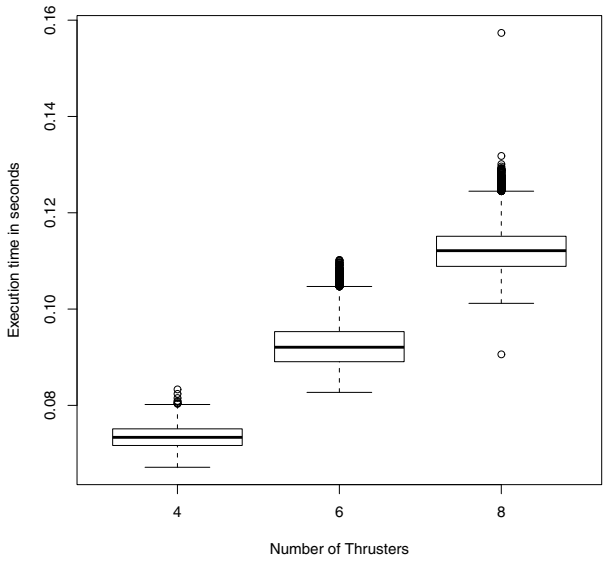


Figure 8: Impact of Thruster Number Change in a Boxplot: The execution time distribution of the IPU tasks is shown for three different amount of thrusters. The current system has eight thrusters (rightmost box). Decreasing the number of thrusters to 6 (middle box) or 4 (left box), as expected for some vessel types, lead to an decreased execution time.

sel scenarios, and analyze the number of steps required for the vessel thruster to converge to the desired position. The analysis contained in this sub-section takes into account the interactions between the force that needs to be applied to turn the vessel and the number of thrusters the vessel can use to produce the required force. Therefore, the goals of this sub-section are to study the thruster algorithm convergence characteristics under different scenarios and to analyze the distribution of the force among the thrusters when the number of working thrusters varies with time.

The Dynamic Positioning (DP) task is the main task of the system and is responsible for running the thruster allocation algorithm, which is based on an iterative solution of a stochastic algorithm. Since the DPS task is the main task of the system, it is important to study the thruster algorithm performance considering several scenarios. This is one of the goals of this section. Our analysis is based on a simulation model which uses the algorithm implementation.

The main goal of the algorithm is to compute the force and the angle that each thruster must have to meet the resultant force demanded. The desired resultant force is represented by three components: (i) Surge: the vessel movement in forward or backward directions, (ii) Sway: the vessel movement in left or right directions, and (iii) Yaw: the vessel rotational movement in clockwise or counterclockwise directions on the plane formed by surge and sway. These parameters are the inputs of the algorithm.

We used the Tangram-II modeling environment [9, 12] in our evaluation. The tool provides the ability to construct from simple to complex models, and solve these by several analytical or simulation methods and to support experimentation via active measurements in computer networks. A large set of methods to calculate the measures of interest is also available. In addition, there are features that help the user to visualize the evolution of the model variables with time, useful both for developing an analytical or simulation model. A rich set of analytical solution techniques, both for steady state and transient analysis, are available, as well as, event driven and fluid simulators.

The simulation engine of Tangram-II has a feature called the modeling tool kit (MTK), which is a framework where users can develop customized algorithms as self contained plugins and use those in the simulation engine. Just like a class in the object oriented paradigm, each plugin is composed of attributes and methods, which all models created from that plugin share, and which can be accessed or executed by the user. Users can create and delete the MTK plugins (called MTK objects), set and get their attribute values, and execute their methods. In this way, users can develop complex algorithms in C++ and execute them. The simulation engine sees any MTK plugin as a black box, which corresponds to a new Tangram variable type (the MTK object).

We built a MTK plugin for the thruster allocation algorithm. The MTK plugin contains the C++ code of the algorithm implemented in the prototype system.

Table 2 shows the scenarios evaluated. The scenarios were designed to test the behavior of the algorithm in critical situations. The first six scenarios represent a sequence of vessel movements. The objective of these scenarios is to analyze the convergence time of the algorithm. In scenarios one to four, after each one of the movements, there is always a stop command. The goal is to study the algorithm behavior if the vessel stops after each movement. Scenario 6 evaluates the case when the vessel does not stop between two consecutive movements. In the seventh scenario we evaluate the behavior of the algorithm when the vessel rotates and some thrusters are turned off. Our goal is to analyze the case where some thrusters stop working.

The second column of Table 2 shows the vessel movement and the force in that direction. These are the input parameters of the algorithm. In the first scenario, for example, the resultant force demanded is represented only by the sway parameter, and in the fourth scenario, the demand is represented by the sway and surge parameters.

For each scenario and required movement, the algorithm is executed a certain number of times (It is an iterative algorithm.). The stop condition is the difference between the resultant force demanded and the resultant of the allocated force computed by the algorithm. When this difference is less or equal to 10^{-1}, the algorithm is stopped. The algorithm output is the force and the angle to be applied by each one of the thrusters and the sum of these vectors, i.e. the resultant vector.

For example, for the first movement of scenario 1 (right(40)), the input parameters were surge=0, sway=40, and yaw=0, the required number of iterations was equal to 3, and the output was surge=0.01, sway=40, and yaw=0.

Figure 9 shows the thrusters position (Cartesian coordinates) in the vessel and the thruster ID. Each thruster can produce a force which varies from 1 to 10, in unitary steps, and rotate 360ř, in 30ř steps.

We simulate each scenario 100 times and compute a 95% confidence interval. We choose not to show the confidence intervals in the figures to make the plots more readable.

Figure 10 displays the mean number of algorithm iterations for the scenarios one to six varying the number of working thrusters from 4 to 8. We consider that when the number of working thrusters is equal to n, the working thrusters are $1, 2, ..., n$ (see the thruster ID in Figure 9). For all the results in Figure 10, the length of the confidence interval is less than 6%.

Note that for the majority of scenarios, the number of iterations when only 4 thrusters are working is greater than when all thrusters are operational. The increase in the number of iterations goes from 10% (scenario 3) to 70% (scenario 1). Scenario 5, where the vessel needs only to maintain its position, is the only one where the

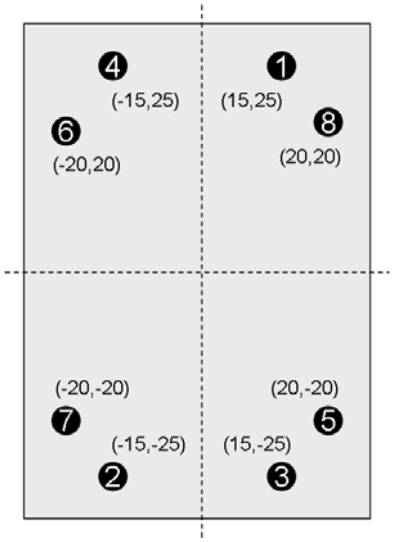


Figure 9: Vessel's thrusters position.

number of iterations does not vary with the number of working thrusters. These results shows that there is a trade off between the number of working thrusters and the number of algorithm iterations. On the one hand, all thrusters must be operational to get a faster algorithm response, on the other hand, there will be more power consumption.

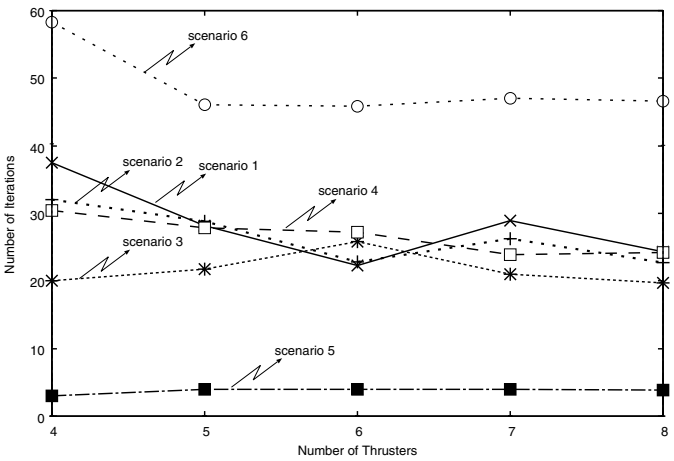


Figure 10: Number of iterations for each scenario.

Figure 11 displays the mean force that each thruster must produce considering scenario 7, i.e., when the vessel rotates counterclockwise and each one of the thrusters is turned off and after turned on. For all the results in Figure 11, the length of the confidence interval is less than 13%.

The main goal of this analysis is to evaluate the distribution of the force among the thrusters when the number of working thrusters varies with time. Note that when only thrusters 1 and 2 are working, the force they must produce is almost the same and near the maximum value, on the other side, when all thrusters are working, each thruster have to produce a force near the minimum value. The results show that the algorithm tries to uniformly distribute the force among the working thrusters.

Table 2: Scenarios

Scenario	Description	Required Movement(Force)
1	Left to right	right(40), left(40), right(40), left(40), right(40)
2	Forward and backward	forward(40), backwards(40), forward(40), backwards(40), forward(40)
3	Rotation	counterclockwise(10), clockwise(10), counterclockwise(10), clockwise(10)
4	Left-backwards and right-forward	forward (26.7) and right(26.7), backwards(26.7) and left(26.7), forward (26.7) and right(26.7), backwards(26.7) and left(26.7), forward (26.7) and right(26.7)
5	Keep position	stop(0)
6	All movements	left(40), right(40), forward(40),backwards(40), left(40), right(40), forward(40), backwards(40)
7	Rotation turning in and off thrusters	counterclockwise(5) during all the scenario; all th on, turn off th 8, turn off th 7 turn off th 6, turn off th 5, turn off th 4, turn off th 3, turn on th 3, turn on th 4, turn on th 5, turn on th 6, turn on th 7, turn on th 8

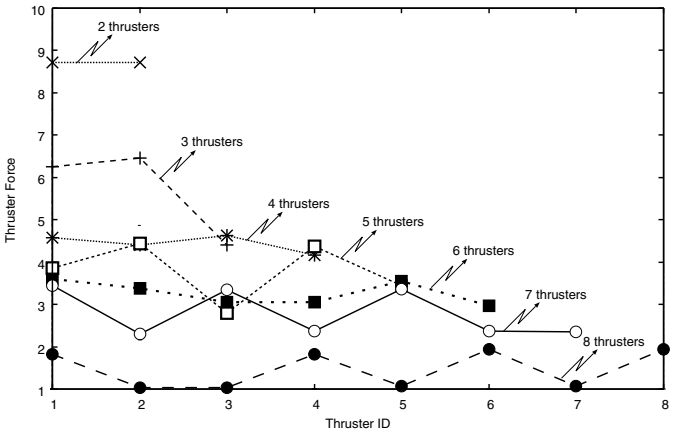


Figure 11: Thruster force.

3.5 Lessons Learned

The three different approaches for performance analysis provided a comprehensive performance assessment of the DPS architecture.

The real-time analysis has shown that all tasks are schedulable. The sensitivity analysis of tasks schedulability was performed by increasing the number of thrusters and sensors. Even for these larger configurations we have shown that all the tasks are schedulable and all deadlines will be met, as shown in Section 3.3.

The control loop execution time is impacted when control loop components are extended with additional functionality. The stochastic performance analysis can be used to assess the impact of component changes on the control loop execution time.

We have shown in this paper that embedded real-time systems can be modelled with the PCM component-based approach, when task precedence can be simplified into to a task sequence. In addition, stochastic performance analysis can complement worst-case predictions. However, tasks with more complex scheduling behaviour could not be predicted with the PCM approach without taking into account task priorities.

We used the Tangram-II tool to simulate several realistic vessel scenarios, and analyse the number of steps required for the vessel thruster to converge to the desired position. One distinct feature of the approach is to allow the use of the real implementation code as a black box and embedded in the simulator tool. Thus, we can use the full power of the simulation engine to test the code.

4. ADAPTABILITY ASSESSMENT

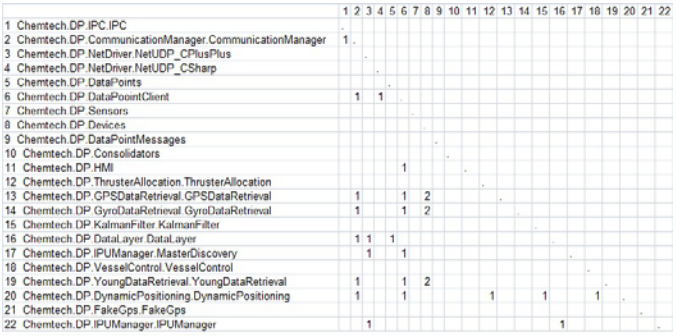


Figure 12: Design Level DSM

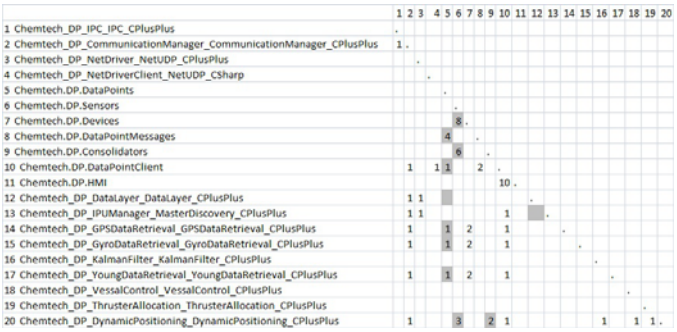


Figure 13: Source Level DSM

It is highly possible that the requirements of the system will change. For example, the program may need to be deployed in different types of vessels or a new type of sensor needs to be installed. We need to assess if the architecture is adaptive enough so that these new features can be accommodated quickly. There can be business metrics to assess the time needed to deliver these new features, for example, by specifying the maximum number of months that can be spent on implementing, testing, and deploying new features. In order to make such estimations, we first need to calculate which and how many components will be added, deleted, or modified, given a specified change. Neither design-level component diagrams nor the source code support such calculation directly. We thus leverage the *design structure matrix* (DSM) [2] model to achieve this purpose.

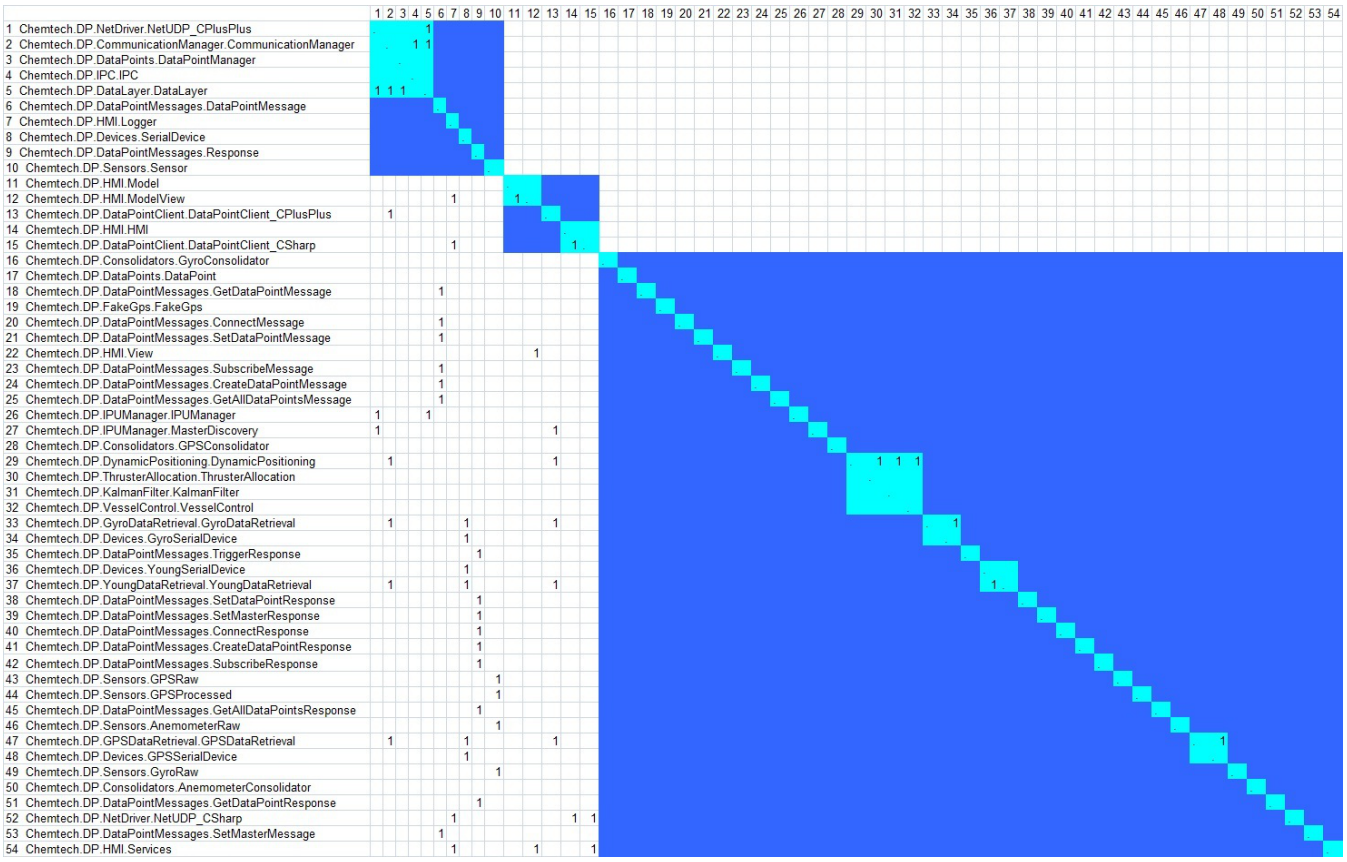


Figure 14: Design DSM Expanded

A design structure matrix is a square matrix where the columns and rows are labeled with the same set, in the same order, of design dimensions where decisions are needed. If a cell is marked, it means that the decision on the row depends on the decision on the column. *Modules* are represented as blocks along the diagonal. A group of variables in a DSM can be clustered into a *module*, which in turn can be represented as a row/column in the DSM. Figure 12 and 13 are two DSMs representing the design and implementation of the dynamic positioning system. In both DSMs, elements 5 (Chemtech.DP.DataPoints) to 11 (Chemtech.DP.HMI) represent modules clustered according to the namespaces. For example, Chemtech.DP.HMI contains all the components within the Chemtech.DP.HMI namespace.

To address the problem that manually constructing a DSM can be time-consuming and error-prone, Cai and Sullivan developed the *Augmented Constraint Network* (ACN) model to represent the dependency relations among design dimensions using logical expressions [5, 6, 17]. From an ACN, the semantics of *pair-wise dependency* can be formally defined and a DSM can be automatically derived. In order for designers trained with UML modeling to leverage these techniques, Cai and her students have formalized and implemented the transformation of prevailing design models and software artifacts, such as UML class diagram, UML component diagram and source code into ACN models, then the structures of these artifacts can be automatically represented as DSMs. The DSMs shown in Figure 12, 13 and 14 are all derived from ACN models. The design ACN model is transformed from the component diagrams modeled using a commercial tool called Enterprise Architect (EA), and the source DSM is transformed from the implemented source code.

To further analyze the modularity and adaptability of the system, we also clustered the DSMs into a special hierarchy [17], as shown in Figure 14. The top level of the hierarchy (element 1-10) are the top level design elements that only influence other parts of the system but are not influenced by them. As we can see, the first level contains the communication structure and the abstract interfaces for sensors and devices. The light blue blocks are the independent modules within the layer. There are no dependency between the modules within a layer. The second layer contains the framework that supports the Model View View Model architecture in the HMI subsystem. The third, and final, layer of the hierarchy contains modules that only depend on the first two layers but not depend on each other.

From the DSM in Figure 14 that shows the designed architecture, we can see that the elements that have significant, crosscutting influences are all at the first two layers, which are mainly communication infrastructure or the high-level architecture framework. About 72% of the components are in the third layer, which means that these modules can be implemented and changed independently from each other.

To assess how well the design can accommodate changes like adding or changing a sensor or adding an error masking mechanism to the vessel control system, we just need to see which and how many other components will be affected. From the design DSM, we can see that changes to sensors only affect at most two other components, and adding error masking mechanism only affects the main function. As another example, if a YoungSerial de-

vice is added or changed, then only the YoungDataRetrieval component will be affected. From these analyses, we conclude that the system is designed to be well modularized and adaptive.

Next we assess whether the implementation of the system is consistent with the design and maintains the same level of adaptability. Figure 13 shows the source code DSM clustered in the same way as the design DSM. In this DSM, the cells with dark background indicate the discrepancies between design and implementation. If a dark cell is empty, it means the dependency exists in design, but not in implementation. If a dark cell is not empty, it means that dependency in the source code doesn't exist in design. Figure 13 only shows the discrepancies among clusters for the sake of space. The numbers in the cell are the total number of dependencies between modules clustered according the namespace.

4.1 Lessons Learned

The analysis shows that the main discrepancies are that in the implementation the data points and sensors are accessed by more components than designed. These discrepancies are caused by the following reasons: more dependencies are found to be necessary during implementation than recorded in the component diagram, the system have evolved in code but the design is not updated, or a part of the design has not been implemented yet. However, the majority of the source code realized the design faithfully. When comparing the source code DSM to the design DSM, we found that the source code also has a layered structure, and each type of sensor only had at most one more dependency than designed. After we clustered the source code DSM using the hierarchy as shown above, about 71% of the components in source are in the third layer, indicating similar level of modularity and adaptability between design and implementation.

5. RELIABILITY/AVAILABILITY

5.1 Hardware

In this section we analyze the reliability of the overall DP system hardware architecture ($R_h(t)$). Our goal is to obtain the repair rates that satisfy the condition: *system reliability is greater than 0.9999 for a mission time equal to 3 months*. In other words, we compute the probability of the DP system being operational at t = 3 months for some values of repair rates. We select the repair rates where $R_h(3) \geq 0.9999$.

Figure 15 shows the Tangram-II model used in our analysis. The model has five different components: gyro sensor, GPS, anemometer, IPU and local network. All components, except the local network (which is dual redundant), are triple redundant.

Figure 16 shows the reliability of the system in the interval from 1 hour to 3 months. We note that for the repair rates equal to 1 day and 15 days, the probability of the DP system being operational at t = 3 months is 0.999999 and 0.9999, respectively. For the repair rates equal to one and three months, $R_h(3) \leq 0.9999$. We conclude that the system reliability goal is met for a repair rate less or equal to 15 days.

5.2 Software

In this sub-section we present our suggestion for an approach to be used to estimate the DPS system software reliability [7]. However, the measurements required to estimate the DPS software reliability are not yet available as the software prototype reliability evaluation is a topic for future work. Therefore, in this section we present the preliminary models and the associated parameters required by the reliability analysis methodology described in [7]. Figure 18 shows the component diagram of the DP Process layer

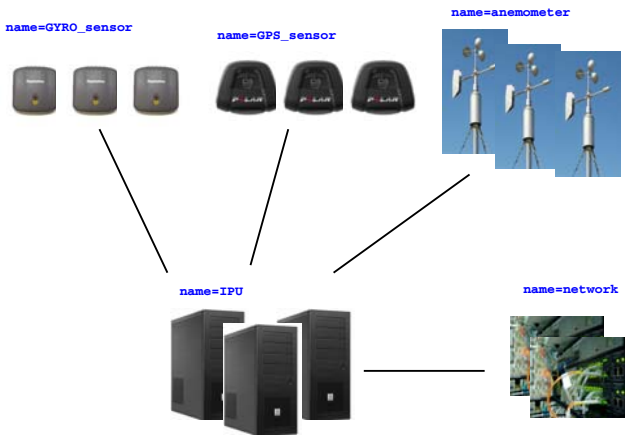


Figure 15: System reliability model.

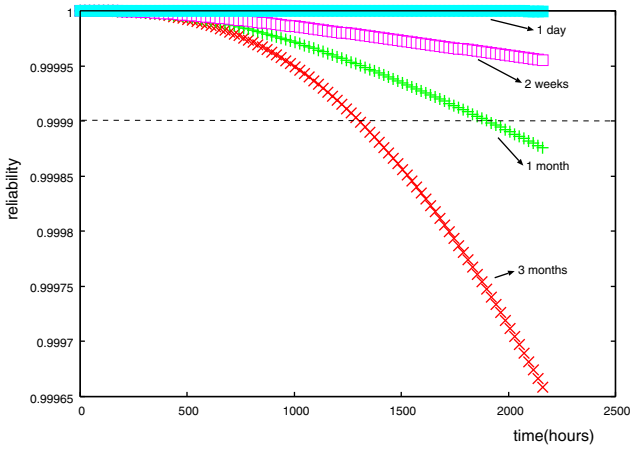


Figure 16: System reliability.

[10] where each component *i* has to be annotated with the following parameters:

- the *internal failure probability* [11] (*intf(i)*, `DaComponent::failure`) is the probability that the component generates a failure caused by some internal fault,
- the *error propagation probability* [1] (*ep(i)*, `DaComponent::errorProb`) is the probability that the component propagates to its output interface an erroneous input it has received,
- the *propagation path probabilities* (*p(i,j)*), one for each connected component *j*, are the transition probabilities from the output interface of component i (`GaStep::prob` on the implementing operation) to the input of component j.

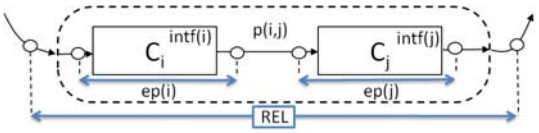


Figure 17: Reliability attributes of the software architecture.

These attributes (*intf(i), ep(i), p(i,j)*) are included as parameters [15] ($-prefixed terms in Figure 18). In particular:

- the *internal failure probabilities* can be roughly estimated from the KLOC (thousands of lines of code) of the corresponding source code artifacts. This and other estimation techniques are described in [11],
- the *error propagation probabilities* are set to 0 for the components that are lacking error masking mechanisms so that a component always propagates to its output interface an erroneous input it has received, or are set to 1 when the best error masking mechanism is implemented for the corresponding component so that no errors propagate to to the output interface;
- the *propagation path probabilities* can be derived at early design stages from the system models that are available at that time, or from software artifacts (e.g. UML interaction diagrams), possibly annotated with probabilistic data about the possible execution and interaction patterns [8].

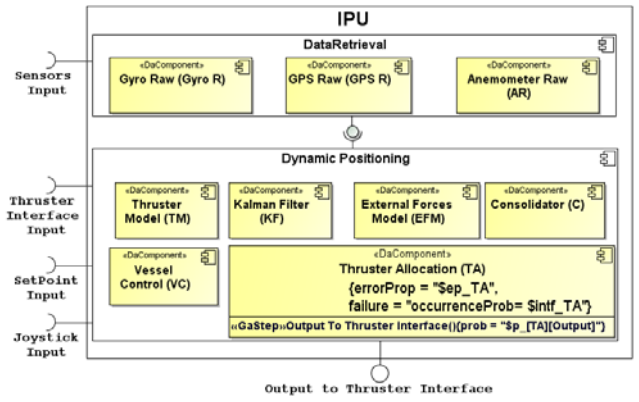


Figure 18: The software architecture of the process layer.

Given the reliability profile for each component, we denote (Rel) as the software reliability of an application, where $err(Input)$ is the probability that the application completes its execution producing an erroneous output, i.e., the application reaches the *Output* component given that the execution started at component *Input*:

$$Rel = 1 - err(Input) \quad (2)$$

Therefore (2) is the probability that the application completes its execution and produces a correct output.

5.3 Lessons Learned

The development of a more accurate software reliability modeling and analysis is left as a topic for future work because the required reliability parameters, i.e. the actual internal failure probability and the error propagation probability of each components, have not been derived yet. We have presented a methodology that is applicable to the DP system to measure each component impact on the overall software reliability and to help prioritize software testing.

6. CONCLUSIONS

We have presented several performance, reliability and adaptability models that were used to comprehensively assess the Dynamic Positioning System architecture. The three performance models presented were instrumented using data collected from the Siemens-Chemtech software prototype. These modeling activities were conducted after an extensive system architecture review uncovered several architecture risks. These risks were reported in a companion paper [10]. The results obtained from the experiments using the non-functional requirement models presented in this paper were of great value to the project in several ways. As a result of the extensive performance modeling experiments reported in this paper, the project has now an increased understanding of the system's ability to meet its real-time deadlines, of the impact of different system configurations on the control loop execution time distribution, and the system configurations impact on the convergence characteristics of the thruster allocation algorithm. The real-time analysis has shown that all tasks are schedulable and that the system will meet its deadlines. The Tangram-II implementation based simulation approach was able to execute several tests using the thruster allocation module and confirmed that the thruster algorithm is stable and generates a well-balanced mean force allocation. The Tangram-II based framework could be used to execute several additional test scenarios to further test the Dynamic Positioning System. In addition, as a result of the system architecture adaptability assessment, the project has received feedback on the importance of maintaining the system architecture documentation up to date. The adaptability assessment model has shown that the system is very modularized, has a layered structure, and that the system implementation complies with the guidelines provided by the system architecture document. This assessment certifies that the system architecture is adaptable and extensible. As a topic for future research, we would like to be able to estimate the Dynamic Positioning System software reliability using actual system testing results.

7. ACKNOWLEDGMENTS

We thank FINEP for partial financial support of the project.

8. REFERENCES

[1] W. Abdelmoez, D. E. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H. H. Ammar, B. Yu, and A. Mili. Error propagation in software architectures. In *IEEE METRICS*, pages 384–393. IEEE Computer Society, 2004.

[2] C. Y. Baldwin and K. B. Clark. *Design Rules, Volume 1: The Power of Modularity*. MIT Press, Cambridge, MA, USA, 2000.

[3] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.

[4] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, USA, 4th edition, 2009.

[5] Y. Cai. *Modularity in Design: Formal Modeling and Automated Analysis*. PhD thesis, University of Virginia, August 2006.

[6] Y. Cai and K. J. Sullivan. Simon: modeling and analysis of design space structures. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *ASE*, pages 329–332. ACM, 2005.

[7] V. Cortellessa and V. Grassi. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In H. W. Schmidt, I. Crnkovic, G. T. Heineman, and J. A. Stafford, editors, *CBSE*, volume 4608 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2007.

[8] V. Cortellessa, H. Singh, and B. Cukic. Early reliability assessment of UML based software models. In *Workshop on Software and Performance*, pages 302–309, 2002.

[9] E. de Souza e Silva, D. R. Figueiredo, and R. M. Leão. The TANGRAMII integrated modeling environment for computer systems and networks. *SIGMETRICS Perform. Eval. Rev.*, 36(4):64–69, 2009.

[10] F. Duarte, C. Pires, C. A. de Souza, J. P. Ros, R. M. M. Leão, E. de Souza e Silva, J. Leite, V. Cortellessa, D. Mosse, and Y. Cai. Experience with a new architecture review process using a globally distributed architecture review team. In *The 5th IEEE International Conference on Global Software Engineering (ICGSE 2010)*, pages 109–118, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[11] J. B. Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Trans. Computers*, 38(6):775–787, 1989.

[12] Federal University of Rio de Janeiro. Tangram-II website. http://www.land.ufrj.br/tools/tangram2/tangram2.html, 2010.

[13] Gilson A. Pinto et al. Advanced control and optimization techniques applied to dynamic positioning systems. In *Rio Oil & Gas Expo and Conference*, Sept. 2010. in press.

[14] L. Kapova and R. Reussner. Application of advanced model-driven techniques in performance engineering. In A. Aldini, M. Bernardo, L. Bononi, and V. Cortellessa, editors, *Computer Performance Engineering*, volume 6342 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-15784-4_2.

[15] Object Management Group (OMG). UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems (formal/2009-11-02). http://www.omgmarte.org/, 2009.

[16] QNX Software Systems. QNX Neutrino RTOS. http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html, 2010.

[17] S. Wong and Y. Cai. Improving the efficiency of dependency analysis in logical decision models. IEEE Computer Society, 2009.