2010 International Conference on Global Software Engineering

Experience with a New Architecture Review Process using a Globally Distributed Architecture Review Team

Flavio Duarte
Chemtech a Siemens Company
RJ, Brazil

Clarissa Pires
Chemtech a Siemens Company
RJ, Brazil

Carlos A. de Souza
Chemtech a Siemens Company
RJ, Brazil

Johannes P. Ros
Siemens Corporate Research
Princeton, NJ, USA

Rosa M. M. Leão
UFRJ
RJ, Brazil

Edmundo de Souza e Silva
UFRJ
RJ, Brazil

Julius Leite
UFF
RJ,Brazil

Vittorio Cortellessa
University L'Aquila
Italy

Daniel Mossé
University of Pittsburgh
Pittsburgh, PA, USA

Yuanfang Cai
Drexel University
Philadelphia, PA, USA

***Abstract*—We present in this paper our experience with applying a new architecture review process that uses a globally distributed review team to assess architecture risk of a complex mission critical system. The new architecture review process uses aspects of the checklist-based architecture review process and the operational scenario-based architecture review process. We present the architecture review process approach, a summary of the architecture under review and the detailed analysis of the most important operational scenarios. We conclude by presenting a summary of the lessons we learned using the new process.**

I. Introduction

Architecture reviews of mission-critical real-time systems have been routinely used in several companies over the last two decades, with significant money saving being reported for large complex projects [4], [5]. The development of large complex mission-critical software using a globally distributed environment creates a significant challenge for architecture review processes that require the co-location of stakeholders and the independent review teams [2].

In this paper we present our experience with the development and review of the architecture for a Dynamic Positioning System (DPS), that has as its primary function to dynamically monitor and control a vessel's position and attitude (heading). Dynamic Position Systems have very stringent non-functional requirements that must be met with high likelihood, as these systems are usually deployed to control mission-critical vessels, where the violation of real-time or availability constraints could endanger the vessel with significant risk to the operator, to the crew, and to the environment.

We report on the application of a new architecture review process that uses a globally distributed review team of architecture evaluation experts and employs aspects of both the checklist based bottom-up architecture review process [4] and the operational scenarios based top-down architecture review process [5]. The project development team and key stakeholders are located at the software development company headquarters, while the independent review team composed of domain experts on the architecture evaluation measures is globally distributed.

We used bi-weekly teleconference meetings for synchronous communication and a group-ware tool (Gforge, [17]) for storing documents and to record asynchronous forum discussions. We also used a face-to-face meeting to validate the results of the globally distributed team architecture review.

We started our architecture review process by applying the bottom-up checklist based architecture review process [4], which consists of reviewing the architecture document and identifying risks to the successful completion of the project. As we progressed in the review, it became clear to the participants that prioritization and linkage of the identified risks to the business goals that usually occurs informally in face-to-face meetings in the checklist-based architecture review process was very difficult to achieve using a globally distributed review team. As a result, the independent review team decided to extend the bottom-up checklist process with aspects of the top-down scenario-based architecture review process [5] to derive operational scenarios linked to business goals. Once the most important operational scenarios were identified, the architecture risks identified in the bottom-up checklist-based architecture review approach could be prioritized by matching risks to scenarios. A face-to-face meeting stakeholders and reviewers was held to validate the architecture analysis results developed by the globally distributed architecture review team and to identify opportunities for architecture improvement. Therefore, we have developed a new architecture review approach that uses some key aspects of the checklist-based approach and of the scenario-based approach. One of the key aspects of the architecture-checklist approach that could not be implemented using a globally distributed review team was the caucus of the independent review team and readout of the cards to the project team. In this paper we present an empirical study of the application of our new architecture review process to a large complex mission-critical system.

In Section II we present the review of the existing literature. In Section III we describe the architecture review process used. In Section IV we discuss our experience applying this new architecture review process. In Section V we present our experience evaluating the architecture using detailed operational scenarios. In Section VII we summarize the lessons learned, present our conclusions, and provide guidance for future research.

II. Related Work

In [6] empirical studies based on data collected from 50 architecture reviews conducted at AT&T were used to develop project metrics. These metrics were developed to enable the automation of the analysis of risk data generated by architecture reviews, with the objective of distinguishing between high-risk and low-risk projects. In [4] some of the experience with the architecture review process that is routinely used by AT&T, Lucent and Avaya is described. The process adopted by these telecommunication companies is checklist-based and uses an independent team of domain experts to assure impartiality, and to provide an open channel of communication of technical issues to the company upper management. The assessment of the architecture is performed by evaluating the ability of the architecture to address the defined problem statement.

These architecture reviews are conducted openly: project members are invited to attend and all attendees can see the architecture issues and strengths and make notes of the 5x8 cards that are the basis for the architecture review report. These reviews are conducted during several days of face-to-face meetings. The outcome of the review is the set of 5x8 cards and a written review report that contains the prioritized list of review findings. Their experience covers over 700 architecture reviews conducted using the checklist-based architecture review process, from 1998-2005. Therefore, the architecture review process used in these telecommunication companies is a bottom-up process that identifies architecture risks based on detailed analysis of the architecture. The identified risks are later categorized into areas and prioritized according to their impact on the ability of the architecture to solve the business problem.

In contrast, the Software Architecture Analysis Method (SAAM) and the Architecture Trade-off Analysis Method (ATAM) [5] are top-down architecture review processes that start from the identification of business drivers and quality attributes and proceeds with the elicitation of operational scenarios. The detailed analysis of the ability of the architecture to satisfy the most important operational scenarios generates risks, trade-offs and sensitivity points that are used to provide recommendations for architecture improvement. In [5], the authors experience with SAAM and ATAM covering 25 architecture reviews over a period of five years is presented. They recommend that architecture reviews should be used as standard practice in the software engineering process for large complex mission-critical processes.

In [3] the results of a large survey to determine industrial practices in architecture reviews was reported. The two leading methodologies used for architecture review are checklist-based and scenario-based reviews, with 54% of respondents reported using scenario-based and 40% of respondents using ckecklist-based reviews. The key learnings from the survey were: the majority of reviews are informal, using external independent reviewers is not a common practice, and practitioners pick and chose specific aspects rather than apply a standard architecture review practice.

The architecture review process presented in this paper is an extension to both the checklist-based [4] and operational scenario-based [5], as it runs both architecture review processes back to back to elicit risks on a bottom-up checklist-based approach, and prioritize risks using a top-down operational scenario-based approach. It uses a globally distributed review team for the initial phases of the architecture review process and one face-to-face meeting with most of the stakeholder present to validate the results of the globally distributed architecture review with the project domain experts.

III. Architecture Review Approach using a Globally Distributed Review Team

In this section we present a brief overview of the architecture review process introduced in this paper.

The important roles in the approach proposed for architecture review are: (1) The system architect is responsible for reviewing the system requirements, documenting architecture alternatives, performing a walk-through of the detailed scenarios over the architecture artifacts, and incorporating suggestions for architecture improvement that are required to address the high risks associated with the high utility scenarios; (2) The system stakeholders are the business customer, managers, users, the application domain experts, and the team members responsible for requirements, architecture, development, testing, documentation, installation and operations; and (3) The independent, globally distributed, review team, is composed of experts on the most important non-functional requirements for the system being reviewed.

The major steps that are required to be executed by architects, stakeholders and the independent review team are:

1) The system architect reviews the system requirements and documents the architecture alternatives in the draft

system architecture document,

2) The independent, globally distributed, review team reviews the architecture document and asynchronously posts comments on the group-ware tool forum (Gforge),
3) The findings of the independent review team are discussed in several teleconferences,
4) The outcome of the findings discussion is compiled by the leader of the independent review team in a risk table,
5) The quality attributes, i.e., the project contribution to the business goals, are defined by the customer representative and stored in the group-ware tool,
6) Operational scenarios, linked to business goals and the quality attributes, are developed asynchronously by the architect and the independent review team, and stored in the group-ware tool,
7) Detailed analysis of the operational scenarios is performed by the independent review team with the objective of assigning risks from the risk table to the specific scenarios,
8) A face-to-face meeting with stakeholders and reviewers is held to validate the results of the detailed analysis and to recommend improvements to the architecture,
9) The system architect reviews the final recommendations of the review team, and the risks associated with the high utility and high risk scenarios to create a risk mitigation document that contains final recommendations for architecture improvement that trades-off cost to implement the change, and risk impact of not implementing the change,
10) The risk mitigation document is reviewed and approved by the project stakeholders.

IV. Case Study

We present in this section our experience with the architecture review process discussed in this paper.

The process was initiated with the detailed work of the system architect to study the system requirements and to write the first draft of the system architecture document. This document was reviewed by the independent review team over a period of two months and a forum discussion was initiated on the group-ware tool (Gforge). Three teleconferences were held to discuss the independent review team findings and a risk table consisting of 30 risks was compiled. The identified risks were divided into several categories depending on the project functional area affected by the risk: requirements engineering, architecture & design and testing. However, the team felt that the bottom-up approach for risk identification did not provide sufficient information to assign severity to the risks.

Therefore, a new forum discussion was initiated to help derive a quality attribute tree that would be linked to the project business goals. As a result, twenty-eight operational scenarios were developed over a period of one-month. These scenarios were rated using the ATAM method [5] by the architect and the independent review team according to the perceived utility to the business and the risk carried by the scenario if the architecture under review was used.

A teleconference was held to help the architect and the independent review team reach consensus on the five scenarios to be selected for detailed analysis. Most of the scenarios selected for detailed analysis were associated with the most important quality attribute for the business: robustness.

The next step was to identify which of the risks stored in the risk table could be associated with the high utility and high risk scenarios. In addition, by performing a teleconference based detailed analysis of the most important scenarios 24 new risks were identified.

A face-to-face meeting was held between the independent review team members and the projects stakeholders to validate the independent architecture review team findings. In this face-to-face meeting 14 additional risks were identified bringing the total number of risks to 69. Of these 69 risks, 22 were categorized as non-risks after detailed discussion with the stakeholders. The remaining 47 risks were categorized into 5 risk themes. Several recommendations for architecture improvement were generated by carefully analyzing these risks. These recommendations are presented in Section V.

A. Dynamic Positioning System

In this paper we present the proposed architecture review methodology applied to the Dynamic Positioning System (DPS) project. Dynamic Positioning systems were created in the 60's for the offshore oil industry and were originally designed to control the vessel's position and heading by using the vessel's propulsion system. Modern Dynamic Positioning Systems are capable of several other functions like autosail, joystick control, and positioning relative to a moving object (other vessel or remote operated underwater vehicle). In some vessels, the joystick allows the DP operator to move the vessel in a particular direction without changing its heading. Similarly, the joystick also allows the operator to spin the vessel without changing its position.

Quality certification societies have issued rules for Dynamic Positioning ships based on [7]. These rules describe three classes of DP Systems:

- Class 1: Loss of position may occur in the event of a single fault.
- Class 2: Loss of position should not occur from a single fault of an active component or system such as generator, thruster, switchboards, remote controlled valves, etc, but may occur after failure of a static component such as cables, pipes, manual valves, etc.
- Class 3: Loss of position should not occur from any single failure including a complete burnt fire subdivision or flooded watertight compartment.

Every DP system is composed of some basic elements: Sensors, Human-Machine Interface (HMI), Controllers (aka IPU), Generators, and Thrusters.

To comply with a target Class 3 system, redundancy must be added to all basic elements. Therefore, for every sensor type, three sensors of that type will be collecting the same data. Similarly, there are three computers (IPU) receiving data from all sensors and running the control algorithms to find the best thruster configuration required to overcome external forces and control the vessel position. For class 3 DP systems, the number of thrusters must be greater than the number required in any scenario, since thrusters can also fail during a critical operation. Usually, there is one generator for each thruster.

Figure 1 illustrates the physical layout of a DPS, showing the DP components, their redundancies, and their connections. This figure also shows a panel where the DP operator can manually control the vessel in case the DPS crashes. Although the DP operation panel is not technically part of the DPS, it is a mandatory safety component.

The macro view of the architecture is displayed in Figure 2. The system was decomposed in three layers: Human-Machine Interface (HMI) layer, the data layer, and the process layer.

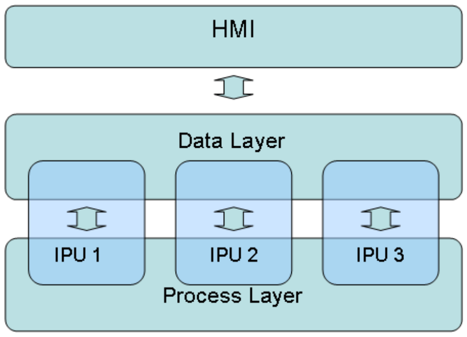

Figure 2. Macro view

The HMI layer is where all information about the system, the vessel and the environment is displayed. It is also through this layer that the operator commands the system and, as consequence, the vessel. This layer runs on dedicated machines and communicates only with the data layer through the redundant network.

The data layer stores all the relevant information for the DP system. It can be viewed as a real-time database. This layer is replicated in all control machines, but this replication is hidden from the operator, so he sees it as a single entity.

The process layer is where the control algorithms run and is responsible for controlling the vessel by issuing commands to its thrusters. Just like the data layer, this layer is also replicated in all control machines. In practice, the control machine is composed of the process and data layers.

Figure 3 illustrates the system data flow, which is composed of the following steps:

1) The system collects positioning and environmental data from the sensors. For each sensor type there are three sensors collecting the same data,
2) The sensors outputs, for each specific type, are fed into the Consolidator module, which produces one result for each sensor type. In this system, the only environmental data measured is the wind. All other environmental forces will be derived by the External Forces Model.

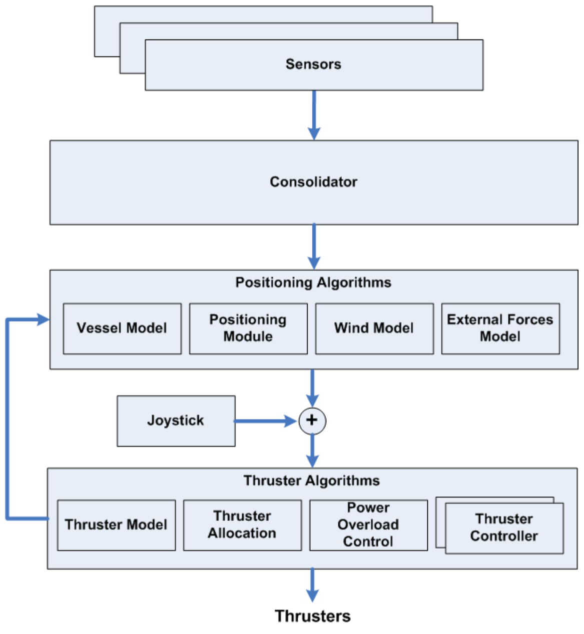

Figure 3. Dynamic Positioning data flow

Figure 3 also shows the system control loop which is composed of the sensors, the consolidator, the positioning and thruster algorithms.

The consolidated values for the positioning sensors are forwarded to the Vessel Model, while the consolidated wind data goes to the Wind Model. The Wind Model uses the measured wind speed and direction to estimate the wind force that is being applied to the vessel by using characteristics of the vessel, such as mass and cross section area.

The Thruster Model determines the net force being applied to the vessel by combining the forces being generated for each thruster. The individual force is computed from the thruster's direction and speed.

The Vessel Model estimates the vessel's position and heading using the consolidated positioning data and the net force generated by all active thrusters. The Vessel Model is based on a Kalman Filter. It generates the estimated vessel's position and outputs the difference between the estimated and measured value of the vessel position. This difference is forwarded to the External Forces Model (EFM), which produces the vector representing all other forces that were not taken into account, such as tidal force. The Vessel Model

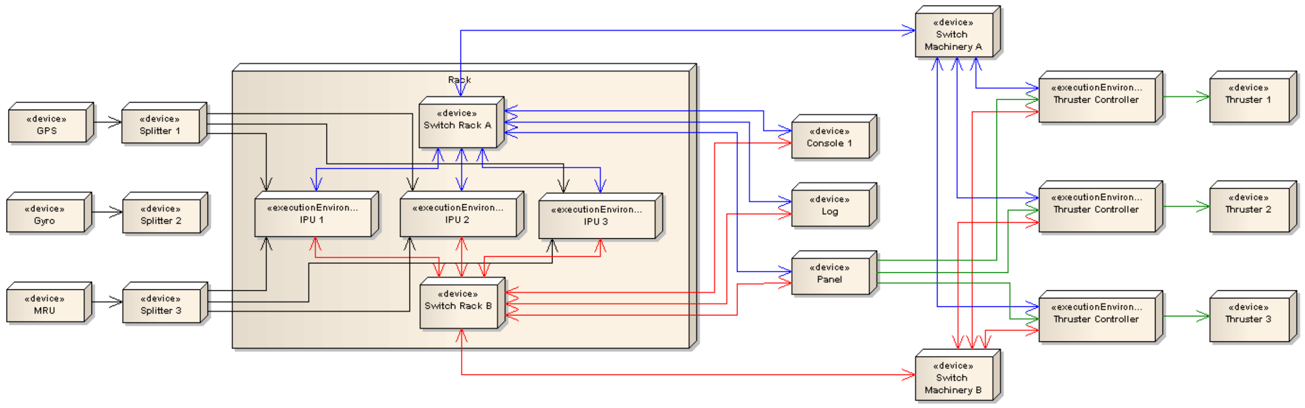

Figure 1. Physical Layout

and EFM form a closed-loop control where the estimated forces by EFM on instant $t-1$ is used by the Vessel Model to estimate the force at instant t.

The Joystick module gets its inputs from the joystick located on the bridge and determines the direction and force intended by the operator.

The Positioning Module component is responsible for driving the vessel to a destination set by the operator and it receives as input the offset between the estimated position and heading and the reference. The output of the Positioning Module is added to the external forces, the wind force, and the force generated by the Joystick module to be used by the Thruster Allocation module, which will compute the optimum thruster configuration that will generate the required force.

The Power Overload Control analyzes the thrusters settings and may decide to initiate one or more additional power generators to supply the necessary power.

The Thruster Controller, upon receiving the appropriate settings from the Thruster model, issues commands to the physical thrusters for the desired speed and angle.

V. Non-Functional Requirements Assessment using Operational Scenarios

This section presents the empirical data produced by following Steps 3 through 8 of the architecture review process as outlined in Section III.

Each sub-section presents a detailed operational scenario analysis for each of the following quality attributes: adaptability, reliability, availability, real-time performance, fault-tolerance, concurrency management, and system performance.

The operational scenarios were produced in Step 6, and this section presents only the high utility, high risk scenarios that were selected by the review team for detailed analysis in Step 7. The risks that were associated with these selected scenarios were obtained from the the risk table as produced by Step 4. These risks are labeled R1, ..., R54.

A. Adaptability

It is highly possible that new features will need to be added and the system will be used in different types of vessels. Adaptability is a property that would allow these new features to be accommodated quickly. The business metrics for this property would be the time needed to deliver these new features, including the time needed for implementation and testing. Technically, we measure adaptability by calculating which and how many components and test cases will be added, deleted, or modified.

The approach adopted in the DP architecture to implement adaptability was to consider the following two specific scenarios:

- *Need to deploy a DP system developed for high cost heavy vessel to a three orders of magnitude lighter vessel*. In practice, this scenario concerns the weight of the vessel. The current Vessel Model is designed for high cost heavy vessels. The weight will influence the algorithm used in position calculation. We would like to design a flexible software that can be used in different types of vessels.

  The Stimulus for this scenario came from changed sales requirement arrived after the deployment of the software. The system should be able to respond to the weight by having an explicit weight parameter and the development and testing time needed for new vessel types should be less than one month.

  We consider two options to achieve this purpose: (1) using a complex vessel model that can vary according to the weight of the vessel; (2) using a different vessel model for lighter vessels. In both options, we decide to make the weight of the vessel an explicit parameter.
- *A new DP feature is implemented during development time*. This does not require major changes to the underlying platform. Required time of integration testing and reliability certification is minimal. For example, “add a sensor for current to the system in less than 1 month calendar time”.

It is highly possible that a new type of sensor will be added to the model. The Business Goal is to accommodate new sensors quickly. Its Attribute Concern is time-to-market when such new features are required. We assume that the Stimulus for this scenario came from changed sales requirement arrived after the deployment of the software. The system should be able to accommodate such new features in less than one month.

There are two architectural options to achieve this goal: (1) each time when a new parameter is added, we need to figure out which and how many other components will be affected. (2) employing a "plug-in" architecture so that the impact of adding new sensors can be localized. For the first option, we can use a design structure matrix model to analyze the impact of each new sensor. For the second option, we need to design a new stable interface, that is, design rules, that allows new sensors to be added without influencing other features.

The trade-offs between the two options is flexibility vs. performance. The second option will add extra layer of indirection and may degrade the performance or real-time requirements. On the other hand, the second option may need less testing and implementation time so that the new feature can be delivered quicker.

We analyze the risks and the impact of each option as follows. We first list some of the risks associated with the adaptability option for the scenario *Need to deploy a DP system developed for high cost heavy vessel to a three orders of magnitude lighter vessel*:

R46: Will the cost be too high to implement a complex model?
R47: How the test plan will be affected by having a complex model vs. multiple models?
R48: If having a different vessel model for lighter vessels, then how many different vessel models should be there?
R49: How to ensure reliability for lighter vessels?

We need to change the current architecture model to make the weight parameter explicit before we can answer these questions. To analyze the trade-offs between these two options, the architecture model should allow us to infer: (1) which and how many components in existing architecture will be affected if implementing a complex model? (2) which and how many components will be reused or changed if implementing multiple vessel models. Based on this information, we can further estimate the trade-offs in terms of cost, reliability, performance, robustness and time needed for testing. For example, implementing multiple but simpler models may be less costly in terms of implementation, but some components may have to be reused in both models, while changing these commonly used components may have broader impact on multiple models. On the other hand, the time needed for testing a complex model may be shorter than the time needed for testing multiple models.

The risks associated with the scenario *A new DP feature is implemented during development time* are:

R50: Does the algorithm need to be changed by adding new sensor data?
R51: How to ensure reliability when adding new features?
R52: How to ensure performance when adding new features?
R53: How to isolate the impact of adding new features?
R54: Will the new parameter influence the real-time deadline?

To automate the calculation of adaptability metrics, we have implemented a tool that transforms a component diagram into a decision model called the *Augmented Constraint Network* (ACN) [8], [11], [13], from which a *design structure matrix* can be generated and adaptability metrics can be calculated automatically [10], [12], [9].

In Section VII we discuss the tool support that we are putting in place to automate the analysis of availability, reliability and performance non-functional measures.

B. Reliability/Availability

A certain number of metrics to quantify reliability and availability have been defined and well-assessed in the past decades [1]. The difference between these two non-functional attributes is that the former only considers the correct behavior of a certain system (either within an interval of time or for a certain number of invocations), whereas the latter also consider system recoveries. For sake of DP system analysis we will make use of metrics from both attributes.

The approach adopted in the DP architecture to address reliability/availability was to target a specific scenario that claims: *A limited number of recoveries from single failure events allowed per year. System is required to operate without failures after a single failure event for the average repair time for the failed sub-system or component.* Such scenario needs to be evaluated from an overall model that takes into account both the components failures and repairs. From that one may obtain the probability of a system failure during the mission time and the expected number of repairs during a specified time interval, for instance.

For this type of scenario validation two approaches can be taken at different phases of the system development, that are: (i) a model-based approach before the system deployment, and (ii) a monitoring-based approach after the system deployment. In this project we primarily intend to tackle a model-based approach, due to the project duration and the development status of DPS. In a longer-term objective, such approach shall be compared with monitoring-based results in order to refine the models built and achieve a better capability of availability estimation while the deployed system will evolve.

The construction of a vessel model useful for availability validation starts from the identification of a fault and repair model. The system should be somehow tolerant to single failure events, but where these failure can come from? What types of DPS components or subsystems can be subject to failures? What kind of failures is reasonable to consider (i.e. byzantine, crash, etc...)?

Once agreed on a fault model, a reliability/availability model can be introduced that represents the whole system as an assembly of its components or sub-systems. The validation consists in solving this model for different values of input parameters (such as the system load, the failure parameters of components, the error propagation features, etc.) and system configuration (intended as the degree of fault tolerance achieved by introducing specific redundancy mechanisms) to analyze the sensitivity of availability indices of the whole system.

The results of this analysis are not only represented by the non-functional indices of the DPS model, but also by the identification of critical components, that are the ones that more heavily can affect the system reliability/availability. This identification is crucial to provide architectural feedback to software designers that, on the basis of these results, may decide to spend a higher testing/development effort on such critical components.

The risks that were encountered by performing the detailed operational scenario analysis for reliabililty/availability were:

R3: Missing quantitative specification of Non-Functional Requirements
R4: Need to estimate cost and robustness impact of alternative architectures for concurrency management.
R7: There is no level of traceability of NFR requirements
R13: Possible loss of synchronization between distributed components due to a component failure?
R16: How to identify interaction between features developed independently?
R28: What is the mechanism to implement a FT (Fault-tolerance) manager module?
R29: Which are the fault and repair models?

C. 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 tasks relevant to the DP system were tested so that the 1 second behavior can be guaranteed.

The scenario considered to exemplify the real-time attribute was *control loop should not take longer than 1 second*. The Business Goal for it is performance, and the Attribute Concern is latency. The components in control loop estimate where the vessel will be, given its current position, heading and speed, as well as the external forces (wind and tide) and resulting forces due to the thrusters. The control loop receives its inputs at an 1s interval, and should produce an output within a 1 second deadline.

The stimulus for this scenario analysis was the module not generating an output within 1 second, and that the stimulus Sources were incorrect input values, or that the original temporal testing was incomplete. For this scenario, it was assumed that the vessel was operating at normal environmental conditions. Further, we assumed that a missing output could be detected by a fault manager component noticing a time overrun, and then taking a fault recovery action.

The risks associated with this scenario are summarized below:

R1: What are the consequences of missing a real-time deadline?
R2: What is the Vessel Model performance bottleneck?
R8: What are the architecture styles of each processing component?
R11: Proprietary extensions to hardware components (thruster, power management system, etc.) could require a change in the vessel model.
R12: Setting up integration test, system test, load test and production environment test (interface simulators, load generators, test tools).
R26: What is the real-time impact of data loss?

Aside from the risks we also identified trade-offs in the architecture. These trade-offs are present in the original list of items that were discussed by the team in the analysis phase; examples follows. The trade-off between interoperability vs. performance is related to R11, because a new proprietary component (e.g., more advanced hardware) could command a new, different interface, thus reducing interoperability, and at the same time increasing performance.

D. Fault-tolerance

Fault-tolerance is the mechanism designed to prevent faults from becoming service affecting failures. The approach adopted in the DP architecture was to use replication, redundancy and voting of some components such as: network, sensors, controllers and thrusters. Three scenarios were selected to check the architecture for compliance with the fault-tolerance non-functional requirement. These scenarios were designed to inject failures into the following critical components: the controllers, the network and the thrusters.

The scenarios used for the fault-tolerance quality attribute were:

- *No significant loss of position after a single IPU crash.* This scenario was envisioned to help in the analysis of the system behavior and to estimate system recovery time after a single point of failure controller crash. The vessel's mass and inertia are such that a reaction time of under one minute is acceptable. The scenario stimulus is a crash in one of the controllers and the architecture artifacts activated by the scenario are the controllers and the leader election component.
  The architectural approach used to implement fault-tolerance is to build a distributed leader election algorithm to manage the redundant components shown in Figure 1. The redundant components receive the same inputs and perform the same computations, so they provide the same results if no failures occur. The distributed leader election algorithm chooses one component to be the master and to control the vessel, while the other components are used as back-up controllers. The distributed leader election algorithm runs every second. If one of the back-up controllers crashes, there

is no system impact. In contrast, if the leader crashes it will fail to participate in the next leader election cycle and another component will be chosen as leader.

- *No significant loss of position after a single point of failure network crash.* This scenario analyses how the system deals with network related failures. The stimulus is a failure in one and only one network component and the artifacts are the network elements: network cards, switches and cables.
  The proposed DP system uses two independent networks and a driver which replicates the outgoing packets and filters out the incoming duplicated packets. The network protocol in consideration is UDP, which uses a checksum to detect and discard packets with errors in its bits. Bit errors that are not detected by the checksum are negligible and will not be considered here. A failure in one network element implies that packets will arrive in only one interface and thus the network driver will not have packets to discard.
- *Reaction to one thruster failure is recovered in time to avoid a significant loss of position for a heavy vessel.* This scenario explores the system behavior after failure in one of the thrusters. The stimulus is a failure in one of the thrusters and the artifacts are the thrusters and the controllers. Again the mass and inertia involved are such that the vessel will not change its position quickly. Therefore, a reaction time of one minute was established.
  When a thruster fails, it triggers an alarm to its thruster controller to reset its parameters and to avoid using the faulty thruster. The reaction time will be close to 1 second since this alert message is scheduled to be delivered as soon as it is generated.

The risks associated with these scenarios are summarized below:

R1: What are the consequences of missing the 1 minute real-time deadline?
R13: Possible loss of synchronization between distributed components due to a IPU or Network crash?

E. Concurrency Management

Concurrency in the DPS architecture context is the ability to execute several tasks at the same time.

The scenario chosen to be evaluated concerning concurrency management was the architecture ability to deal with two controllers trying to control the thrusters. The stimulus in this scenario is two controllers sending commands to the thruster controllers and the artifacts involved in this scenario are the controllers, thrusters, thruster controllers, leader election component, sensors and network elements.

This scenario can happen if for some reason the communication to and from the master controller is interrupted. In this case, the current master controller would not report to the leader election, and thus a new master would be elected. Since the communication to the current master machine is down, the other 2 IPU could conclude they are the only controllers working, and each one would elect itself as master, creating an environment where the DP system has two masters that would send commands to the thrusters and try to configure the sensors.

The risks associated with this scenario are:

R13: Possible loss of synchronization between distributed components due to a crash on the leader election component?
R14: How to efficiently encapsulate state re-synchronization in the data layer?
R32: Tight synchronization mechanism to assure all IPUs are performing the same computations.

F. System Performance

The main metric used to evaluate the DP System performance is the control loop response time, since the control loop must be executed in one second. Then it is important to evaluate the tasks that affect the above metric. The approach taken to study the system performance is to build a performance model during the system development and to monitor the system after the system deployment.

The scenarios that will be evaluated concerning the performance quality attribute were: (i) to guarantee the performance when new features are added to the system and (ii) no significant loss of performance after a failure in a task in the control loop.

In the first scenario, we must investigate the impact in the system performance when adding new components such as a sensor or a thruster. New components may generate new messages that have to be processed by the system, and the adverse impact of these components in system performance should be evaluated.

The most critical task of the control loop is the thruster allocation algorithm. Then, the second scenario aims at evaluating alternatives for this algorithm. The algorithm is based on an iterative solution of an optimization problem. One problem with this solution is that the convergence time may be greater than the required deadline. We plan to evaluate the algorithm using different input parameters to identify the scenarios where the convergence time is greater than the deadline. In these scenarios, the architectural approach adopted is to use additional algorithms based on an analytical method as a backup solution.

The risks associated with this scenario are summarized below:

R44: How to ensure performance when adding new features? Will the new parameter influence the real-time deadline?
R57: Need a fast algorithm using an analytical approach as back-up to address real-time deadline overrun.
R67: Need to measure control loop iteration response time.

G. Recommendations for Architecture Improvement

As a result of the detailed operational scenario analysis at the face-to-face meeting we documented the following architecture decisions and recommendations for architecture improvement:

- Use an architecture artifact to detect and take action after real-time overrun
- Use redundant IPU, HMI, and data paths
- Traffic to be mirrored on both networks
- Architecture must be constantly pooling the thrusters to get their statuses and redundant thrusters must be employed whenever possible
- Architecture must ensure timely thruster failure detection
- Use fault-tolerance mechanisms and reliability testing to detect faults before deployment
- Each time when a new parameter is added, we need to figure out which and how many other components will be affected
- Employing a "plug-in" architecture so that the impact of adding new sensors can be localized
- Use a parameterized model that can be applied to heavy and light vessels
- Add a security artifact to the system for operations that control vessel
- Add an architecture artifact to enable DP operator override feature
- Add an architecture artifact to perform consequence analysis
- Re-tune and test for each vessel instance, given vessels are not the same
- Develop a low cost approach (simulator) to be able to tune the algorithms
- Manage system state (PID integral, Kalman matrices, etc.) and synchronize state after detection of IPU divergence
- Add an architecture artifact to intelligently manage and control thrusters for failure and special situations, e.g. stopping a platform, find best paths for thrusters on long thruster trips

These recommendations represent the result of Step 8 of the architecture review approach as outlined in Section III. Steps 9 and 10 are currently being executed by the project.

VI. Tools to evaluate the performance/dependability metrics

In order to support the model based analysis of the paper a set of tools will be adopted. One such tool is the Tangram-II [14] modeling environment. The tool integrates different environments for developing and analyzing computer systems and is based on a unique modeling paradigm for describing the model. It implements a rich set of analytical solution techniques, both for steady state and transient analysis and a large set of methods to calculate the measures of interest is also available.

Tangram-II will be used to obtain the availability and performance metrics of the DP system. Figure 4 shows a preliminary availability model we built using the tool. We represent the main DP components, their redundancies, and their connections. One important result we can obtain with this model is the fraction of time the DP system is operational during a given observation interval $(0, t)$. In addition, metrics such as the probability of a DP system failure during the mission time and the expected number of repairs can easily be calculated and one can easily check if the system's dependability and performance requirements are met.

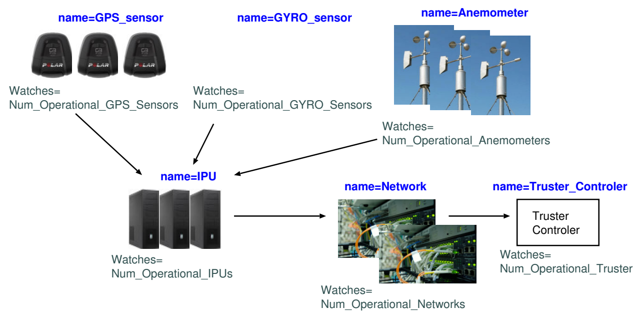

Figure 4. DPS Availability Model

A Tangram-II performance model will also be defined to evaluate the control loop response time which must be equal to one second. We plan to build a performance model of the system tasks that are executed in the system control loop. The most critical task is the truster allocation algorithms. These algorithms are iterative and the convergence time may be greater than the control loop interval. One important modeling feature of Tangram-II is the possibility of integrating in a simulation model an algorithm in C language. We plan to incorporate in the Tangram-II performance model the C code of the thruster allocation algorithms and, from that, obtain metrics such as the fraction of of the control loop intervals that did not make the deadline.

We also plan to use in the future the COBRA (COmponent-Based Reliability Analysis) tool that is under development at University of L'Aquila [16] to automatically transform software architecture artifacts into reliability/availability models. COBRA generates reliability models from UML models appropriately annotated with reliability parameters.

VII. Conclusions

We have presented in this paper our experience applying a new architecture review process that uses a globally distributed review team. We are very encouraged by our initial results as we were able to implement a cost effective review and provide mean-full recommendations for architecture improvement. The review process was carried out over a period of several months using group-ware, e-mail and teleconference, with one face-to-face meeting at the end of the review process. The main conclusions and lessons learned from our experience are:

- The face-to-face meeting to validate the globally distributed review team conclusions with the project stakeholders was fundamental to validate the risk assessment, to create a list of architecture improvement recommendations and to obtain buy-in from the stakeholders to implement the review team recommendations. We believe this final meeting must be held face-to-face with the maximum possible number of reviewers and stakeholders present.
- Analysis could be carried out at a faster pace in a shorter time span than the time it took for this project. This would cause higher level of commitment from the participants, and would move the process along, benefiting the end-user customer, but also increasing costs.
- There could be an hierarchical approach to the ATAM part of the analysis (after the initial discussion of the architecture as a whole, to give a global view of the problem to the participants): starting with smaller, more focused sub-teams to analyze the scenarios, then analyzing relevant scenarios together, and then finally putting all sub-teams to discuss the project as a whole. For example, start with the real-time aspects review team, join the fault-tolerance aspects review team, and then the entire team.
- There could be a more systematic approach to risk identification, perhaps using discussions on why listed/identified risks (trade-offs, non-risks, etc.) are not actual "risks" in each scenario. That is, not only discuss the risks chosen, but justify why the other risks were NOT selected. A written register (explaining why a given risk was assumed as such or not) should be made for each item in the Risks List.
- Individual scenario analysis could be done with the supervision of the system architect and the review team leader. Care should be taken that the system architect not influence the analysis, but that s/he is present to clarify certain aspects of the evolving design. The review team leader is needed in the beginning of the process to help streamline and clarify the analysis process in smaller sub-teams.

As topics for future research we are considering to create an automated tool infrastructure for the modeling and assessment of the architecture non-functional aspects. Model-based analysis, in perspective, can be widened to the trade-off analysis among different non-functional attributes. For example, in order to achieve a certain degree of software reliability the development diversity techniques can be introduced to achieve software redundancy. However, such techniques can be very expensive in terms of costs and system performance, therefore a trade-off analysis would be appropriate to support engineers' decisions.

References

[1] lgirdas Avizienis, Jean-Claude Laprie, Brian Randell and Carl E. Landwehr. *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Trans. Dependable Sec. Comput. 2004, pp. 11–33.

[2] M. A. Babar. *A Framework for Supporting the Software Architecture Evaluation Process in Global Software Development*, Proceedings of the Fourth IEEE International Conference on Global Software Engineering. July, 2009, pp. 93-102.

[3] M. A. Babar and I. Gorton. *Software Architecture Review: The State of Practice*, IEEE Computer. July, 2009, pp. 26–32.

[4] J. F. Maranzano, S. A. Rozsypal, G. H. Zimmerman, G. W. Warken, P. E. Wirth, and D. M. Weiss. *Architecture Reviews: Practice and Experience*, IEEE Software. March/April, 2005, pp. 34–43.

[5] R. Kazman and L. Bass. *Making Architecture Reviews Work in the Real World*, IEEE Software. January/February, 2002, pp. 67–73.

[6] A. Avritzer and E. J. Weyuker. *Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews*, Empirical Software Engineering. 4(3):199-215, 1999.

[7] International Maritime Organization *Guidelines for Vessels with Dynamic Positioning Systems*, MSC/Circ. 645, June, 1994.

[8] Yuanfang Cai *Modularity in Design: Formal Modeling and Automated Analysis*. University of Virginia, August, 2006.

[9] Kanwarpreet Sethi, Yuanfang Cai, Sunny Wong, Alessandro Garcia and Claudio Sant'Anna *From Retrospect to Prospect: Assessing Modularity and Stability from Software Architecture* Proc. 8th. September 2009, pp. 269–272.

[10] Yuanfang Cai, Sunny Huynh and Tao Xie *A Framework and Tools Support for Testing Modularity of Software Design* Proc. 22nd. November 2007, pp. 441–4.

[11] Yuanfang Cai and Kevin J. Sullivan *Simon: A Tool for Logical Design Space Modeling and Analysis* Proc. 20th. November 2005, pp. 329–332

[12] Alberto Avritzer, Daniel Paulish and Yuanfang Cai *Coordination Implications of Software Architecture in a Global Software Development Project* Proc. 7th. February 2008, pp. 107–116

[13] Sunny Wong and Yuanfang Cai *Improving the Effeciency of Dependency Analysis in Logical Models* Proc. 24th. November 2009, pp. 173–184

[14] Edmundo de Souza e Silva and Daniel R. Figueiredo and Rosa M. M. Leão. *The TANGRAMII integrated modeling environment for computer systems and networks*, SIGMETRICS Perform. Eval. Rev. 36(4):64-69, 2009.

[15] http://people.ee.duke.edu/ kst/ *SHARPE*.

[16] http://sealabtools.di.univaq.it/ *SEALAB*.

[17] http://www.gforge.org/ *GForge*.