Meeting Nonfunctional Requirements through Software Architecture: A Weapon System Example

Kadir Alpaslan Demir
Department of Computer Science
Naval Postgraduate School
833 Dyer Road, Monterey, CA 93943, USA
kdemir@nps.edu

Abstract

Meeting nonfunctional requirements is as important as meeting functional requirements. A well-designed software system architecture helps to ensure that the necessary quality attributes of the system are satisfied. The goal of this paper is to show how a system's software architecture can be designed to achieve its nonfunctional requirements. The development process is explained using a weapon system example named Mine Neutralization System for navy mine hunting ships. Also, a novel aspect of this paper is the introduction of a new architectural style. The style is described via an example.

1. Introduction

All software systems have a software architecture whether or not it is explicitly spelled out in the design documentation [1]. Architecture is the backbone of the software system. Therefore, it encompasses all the early important design decisions and trade-offs. The software application is built onto it. The modifications in the software architecture later on in the development process cost dearly.

According to Kruchten, software architecture is used for [2]:

- Understanding what the system does and how the system works
- Thinking and working in the pieces of the system
- Extending the system
- Reusing the parts of the system to build other ones

Thus, a software system architecture helps us to answer most important questions related to a product. It also helps us to achieve high quality software.

In this paper, we present a software system architecture with its development process. The development process is explained via a weapon system example, a mine neutralization system for mine warfare ships. Weapon systems are complex and safety-critical embedded systems in general [3]. Analysis and design of these systems pose many challenges [4]. Most of those challenges can be addressed with a well-crafted software architecture. Such challenges and strategies to resolve them are presented with the associated architectural solutions throughout the development of the mine neutralization system example.

A new architectural style, named star-controller, is introduced, as well. An architectural style describes the structure of a pattern that can be applied to a family of systems. Architectural styles also explain the terminology of the components and connections along with a set of rules on how they can be combined [5]. How the style is applied to the architecture development is also provided.

The rest of the paper is organized as follows. Section 2 presents a brief discussion of how nonfunctional requirements are met through software architecture. Section 3 introduces the system and presents the architectural development process along with the system architecture and design diagrams. Section 4 draws the conclusion. Experiences, lessons learned and future work is explained in section 5.

2. Nonfunctional requirements through software architecture

Requirements engineering process provides the main input for the software architecture development. The software architect takes the requirements and develops a software architecture that meets both the functional and nonfunctional requirements. The decisions he makes at this phase of software development establish the boundaries of the system quality attributes such as extensibility, modifiability, adaptability, reliability, safety, maintainability, testability etc. The importance of meeting quality attributes with software architecture is already recognized. For example, Bachmann and Bass present an attribute driven design method for designing the software architecture [6]. Bass and John link the usability to software architecture patterns [7].

In our weapon system example, adaptability, modifiability, maintainability, usability, testability, reliability and safety are the quality attributes that are specifically addressed. These attributes and how to achieve them are presented with specific architectural patterns and solutions. Analysis of the architecture of a software system reveals whether the architecture of that system is capable of meeting the system nonfunctional requirements.

3. Mine Neutralization System (MNS)

Due to developments in electronics and software systems, navies around the world undergo major revisions in their combatant ships. Instead of designing and building ships from the scratch, it is cheaper to revise its combat systems to increase the ship's combat capabilities. The Mine Neutralization System (MNS) is conceptualized and designed to adapt latest technologies in mine warfare without undergoing major changes in a mine hunting ship's original structure. The objective of the MNS is to detect and eliminate sea mines. The system uses a detection sonar to detect an underwater threat, possibly a sea mine. Classification of sonar data input helps the sonar operator to classify the threat type which may be a magnetic or moored mine. The operator of the system eliminates the mines via a remotely operated underwater vehicle (ROV). MNS encapsulates and controls all these main and auxiliary devices including system consoles to achieve sea mine hunting mission for navy mine warfare ships. Figure 1 depicts the use of the system in a mine hunting ship.

3.1. MNS High-Level and User-Level Goals

Requirements engineering is an important success factor in software projects [8]. The high-level and user-level goals of the system were identified through a series of interviews with navy officers who are the major stakeholders for this system. Also, the analysis of business opportunities and technological improvement projections for mine warfare systems guided the most important system requirements.

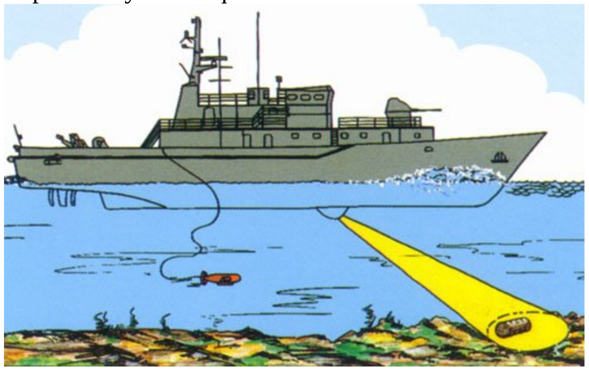

Figure 1. The illustration of the MNS use on a mine hunting ship

The interviews with navy officers revealed important shortcomings of existing mine hunting systems. For example, existing systems require quite a few personnel. In a mine hunting ship, the number of personnel is limited and sometimes operators need to stay on watch for long hours, which poses a threat to the mission. MNS reduces the number of personnel to only one operator. This is one of the important achievements of the system. Another accomplishment of the system is that the system is highly adaptable to the new technological advances in mine warfare. This requirement is derived from the business opportunities.

After a detailed requirement analysis, the high-level goals of the system are identified. Some of them are listed as follows:

- MNS provides a complete solution to satisfy the prospective technological advances in mine hunting warfare.
- MNS is a reliable and safe system that can eliminate the shortcomings of current systems in navies.
- The system is maintainable that is a benefit to both the developer and the customer.
- The system operates in 15-600 ft. depth range which is highly sufficient for the mine hunting operations.
- With the support of the umbilical cable attached to the mine neutralization vehicle (MNV), the length of the mission will not be limited to short periods.
- The system is highly adaptable for future upgrades.
- Emergency mode operation provides flexibility during mission.
- The system can operate with many existing sonar suites currently used in mine warfare operations.
- A large variety of alarms help the system operator to monitor the safety of the system.

The user-level goals are as follows:

- MNS requires only one operator.
- The system needs less training than existing systems.

- MNS has a simple graphical user interface which helps the operator and the commander of the ship to visualize the controls and the state of the system.
- The camera on the MNV provides high reliability for the operation.
- The easy control of the MNV provides an increase in the flexibility of the operation.
- Multilanguage support makes the product usable by many different countries without further training in language.
- Logging features of MNS helps the ship’s crew prepare after-operation review reports.
- Logging features helps the personnel for maintenance of the system.
- Training mode is the same with the operation mode, which provides an excellent training environment for the ship’s crew.

The requirements analysis phase of the system development lead to the following outstanding features of the proposed product:
- One-man operated system
- Highly adaptable to new sonar systems and remotely operated underwater vehicle systems
- A complete solution
- A simple and well-designed interface
- Easy training
- Long-operation support
- Emergency operation mode
- A safe and reliable system
- Multilanguage user interface
- A large variety of alarms and monitoring features
- Enhanced logging of conditions and operation milestones

All these goals and resulting features provide the most important input for the system software architecture development.

3.2. MNS Components

Analysis of similar mine warfare systems, reveals the necessary main components for the MNS. The system is composed of five main components:

- Detection and Classification Sonar Suite: The sonar suite is responsible for the detection and classification of mines. Two different sonars exist in this suite. The detection sonar is a long-range wide-spectrum sonar that is used to detect the presence of an underwater object. The classification sonar is used for further analysis of underwater objects suspected to be a mine threat. This sonar creates a contact for the system. Bat thermograph and echo sounder are the auxiliary devices attached to the suite to provide necessary sea condition data.

- Navigation Unit: This unit provides the precise location information of the ship. The navigation unit consists of a global positioning system (GPS) device and a gyro unit. Both of these devices provide the location data and one of the devices is sufficient for the operation. Therefore, the failure of one of the devices doesn’t compromise the mission.

- Mine Neutralization System Console: This unit is the interface between the operator and the system.

- Mine Neutralization Vehicle (MNV): This unit handles the elimination of the sea mines. It is a remotely operated underwater vehicle and attached to the mother ship with an umbilical cable that carries the communication and power cables. The vehicle carries many devices to achieve the mission. Depth unit, TV camera, light, emergency pinger, umbilical cable, gyro unit and mine neutralization vehicle control unit are only some of them.

- Mine Neutralization System Controller: This unit is the heart of the system. It provides communication between components. It also synchronizes the events within the mine hunting operation. The mine system controller encapsulates the necessary interfaces in case the decision of using existing components in various types of mine hunting ships.

Figure 2 shows the connections between the components and the framework for the system software architecture.

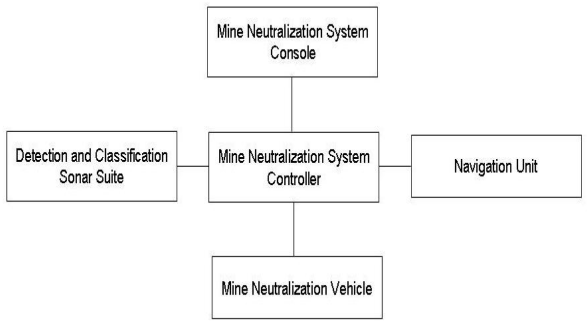
Mine Neutralization System Console
Detection and Classification Sonar Suite
Mine Neutralization System Controller
Navigation Unit
Mine Neutralization Vehicle

Figure 2. The MNS framework

3.3. MNS Software Architecture

Weapon system software development is an expensive and effort-intensive process and these types of systems tend to have long life-cycles. The systems evolve during the years. Older versions are replaced with newer versions in order to keep up with advancing technology. A well-designed software system architecture prolongs the system life-cycle and ease the maintenance effort.

Hofmeister et. al. describes the four views of the software architecture[1]. These four views are conceptual, module, code and execution view. The conceptual view deals with the issues relating to the application domain. One of the most important questions answered with the conceptual view is how the system fulfills its requirements. How the functionality partitioned to the conceptual components is also explained with the conceptual view. The module view explains how the conceptual components are mapped to subsystems and modules. In this view, the conceptual solution is realized with today's software platforms and technologies. The execution view describes the runtime interactions of the software application. It also deals with how subsystems and modules are mapped to the hardware platforms. The code view deals with how runtime entities are mapped to the deployment components such as executables, libraries etc. Each view acts an input for another view and helps the software architect to analyze trade-offs.

Developing the views of the system software architecture starts with a global analysis.

3.3.1. Global Analysis. The global analysis is the process of identification of factors that influences architectural design. The goal of the process is to develop strategies for each identified factor. The factors related to the development of MNS are listed as follows:

1. The quality of the product is more important than the schedule.
2. The system must be easily modifiable.
3. Because MNS is a weapon system, safety and reliability is extremely important.
4. The system must have a friendly user interface that minimizes operator errors.
5. MNS must be an adaptable system and incorporating COTS products must be easy.
6. The system is intended for many countries. Therefore, the user interface should easily be adaptable for different languages.
7. The system must be a maintainable system.
8. A one-man operated system is a must.
9. MNS should meet performance criteria.
10. The system design should support a 20-25 year life cycle.

During MNS development, strategies are laid out to provide solutions for each identified factor. It is important to cover each of the factors with at least one strategy. Table 1 shows factors and corresponding strategies. For example, factor 2 enforces the system to be easily modifiable. Encapsulating the features into separate components is selected as a strategy to ensure a solution for the specific factor. In the MNS framework, the navigation unit handles all the navigation tasks and the unit is only connected to the system controller. If an upgrade becomes necessary in the navigation features, the navigation unit can easily be replaced with a newer version. Such modification doesn't affect other components of the system. This is also how one of the corresponding high-level user goals is achieved.

The next step in the process is the development of the conceptual view of the system. The framework of the system presented in figure 2 is used as an input for the development of the conceptual view.

Factors	**Corresponding Strategy**
1,2,4,7,9,10	Use of well-known patterns
1,3,5,7,9,10	Build instead of buy and/or build products similar to the ones in the market
2,4,5,6,7,10	Make it easy to add and remove features
2,3,5,7,8,9,10	Use a central controller component
1,2,3,4,5,6,7,10	Use standards
2,4,5,6,7,8,9	Separate components and modules along dimensions of concerns
4,6,8	Decouple the user interaction module
2,7,9,10	Encapsulate features into separate components

Table 1. Factors and corresponding strategies

3.3.2. Conceptual view. In the global analysis, we decided to use the well-known patterns as a strategy to address some of the identified factors. For our product's conceptual view, we determined to use the Model-View-Controller pattern. The suggested context for this architectural pattern is interactive applications with a flexible human-computer interface [9]. The model-view-controller architectural pattern divides an interactive application into three components. The model contains the core functionality and data. Views display the information to the user. The views and controller together compromise the user interface. A change-propagation mechanism through controller ensures consistency between the user and the model. Figure 3 shows the conceptual view and how the

architectural pattern is applied to the MNS framework. The rationales for selecting the pattern are as follows:
- The product market is intended for the Navies around the world. This requires complying with the existing user interfaces from all over the world forcing the user interface of the application to be flexible. Like multiple language support, different look and views.
- Even if the model changes, the users will require the same information from the system. So the pattern enables us to decouple the model from the view. For example, an upgrade in the navigation unit will not affect the interface. The navigation unit is one of the components of the model and the mine neutralization system console, which is the interface to the user, is the view in the pattern.
- The product is a weapon system and therefore, it is a real-time interactive application.
- Abstracting the controller enables us to focus on the synchronization of the events in the system in a real-time environment.
- The system is an adaptable system which requires modification when necessary in each component of the pattern.
- Easy addition of views and controllers will benefit the maintenance of the product.
- The architecture of the product will base a framework for future versions and similar products. It is important to remember the necessity of the long life-cycle of the product.

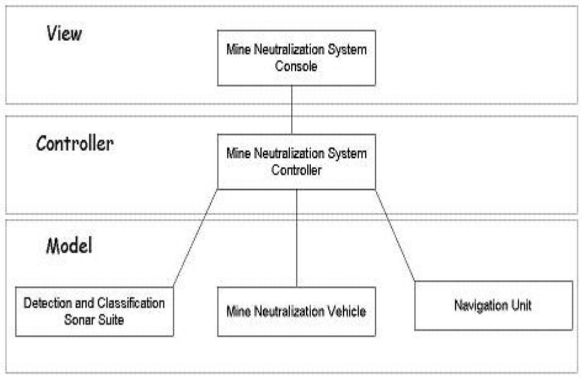
View
Mine Neutralization System Console
Controller
Mine Neutralization System Controller
Model
Detection and Classification Sonar Suite
Mine Neutralization Vehicle
Navigation Unit

Figure 3. Conceptual view

Selection of the model-view-controller pattern helped us to conceptualize an adaptable and maintainable system. This is how important nonfunctional requirements can be achieved in the conceptual view.

3.3.3. Module view. The main purpose of the module view is to simplify the system's implementation in software. It helps us to overcome the complexity of the system. In the module view, all the application functionality, control functionality, and meditation are mapped to subsystems, modules and connections.

For the module view, we developed an architectural style named star-controller architecture. The style resembles to a star network topology in structure. The style benefits from the well-known design decomposition principle. The system is carefully partitioned to subsystems which are strictly loosely coupled with each other. In this architectural style, the system is divided into two types of components: controllers and subcomponents. The controllers handle the control functionality and the subcomponents handle the application functionality in the module view. The architectural style follows two basic rules:

1. A controller can be connected to controllers and subcomponents.
2. Subcomponents can only be connected to controllers.

Figure 4 shows the star-controller architectural style.

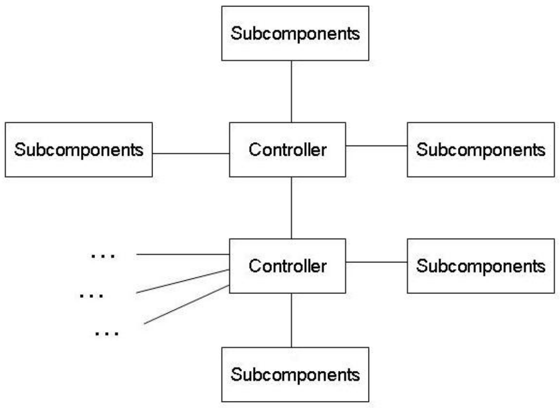

Figure 4. The star-controller architectural style

The style helps us to reduce the development effort for interfaces and similar subcomponents. It also enables the independent development of subcomponents or easy addition of existing subsystems which is enforced to MNS with one of the high-level goals. This architecture ensures to achieve an adaptable and maintainable system.

In this architectural style, faults can easily be identified and localized to some specific portion of the system. Subsystems are tested separately and integration testing is achieved as new subsystems are added to the system. The style follows design for testing principle in this perspective.

The star-controller architecture has a simple structure. Synchronization and the control flow of information are handled by controllers. The information is produced by subcomponents. The controllers' solemn task is to ensure reliable communication and synchronization which are important considerations for real-time systems. In this architecture, high cohesion is achieved via attributing a part of functionality per controller.

The nonfunctional requirements of MNS require the system to be safe and reliable. Ease of testing and a simple design is essential for achieving safety and reliability.

The major drawback of the style is that the failure of one controller disables all the subcomponents attached to it. In the MNS example, the solution is provided with the redundancy in hardware. The MNS has a system self-checking mechanism built in its design. Every controller constantly monitors the attached subcomponents and another controller. Whenever a failure is detected in a subcomponent or in a controller, the system immediately switches to the redundant hardware. Only some of the key elements have this redundancy. Another solution to this problem may be redundancy through software. A software module having the same functionality may be designed differently and installed to the redundant hardware. However, only hardware redundancy exists in the MNS. Figure 5 shows how star-controller architecture is applied to the mine neutralization vehicle subsystem. Note the self-checking mechanism is accomplished via status attributes in classes.

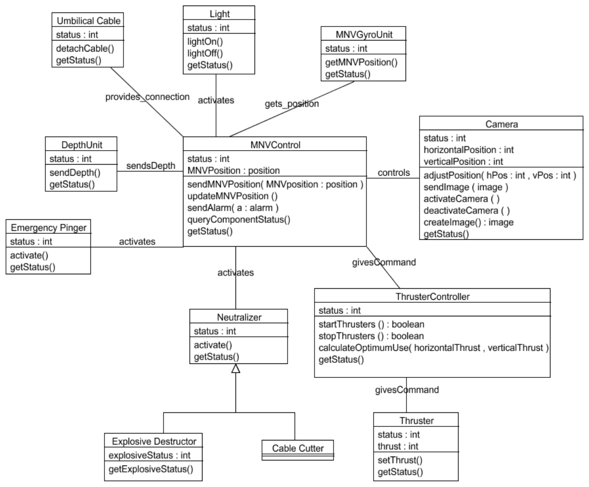

Figure 5. The mine neutralization vehicle subsystem

Rationales on choosing the star-controller architectural style are listed as follows:
- Easy elimination of synchronization problems will increase the system's reliability and safety.
- Functions and controls are separated. Therefore, modifications in functionality will not affect the control aspects.
- Easy addition/removal of modules and functionality, thus support for adaptation and modification.
- Easy localization of errors will reduce the testing effort.
- Elimination of errors and fault propagation increases the system safety and reliability.

A system may have orthogonal software architectures that address different concerns. Because high reliability and safety are important concerns for MNS, we used an additional software architecture addressing communication and synchronization issues. A layered architectural pattern is selected. Layered architectures help us to structure applications that can be decomposed into groups of subtasks. These subtasks are at a particular abstraction. In the MNS example, the system is divided into two layers. The first layer, networking layer, handles the communication between modules as well as establishing the protocols and checking messages for errors. The networking layer corresponds to the physical and data link layer in open system interconnection model (OSI). The second layer is named system layer and it is responsible for all other application-related communications in the system. Figure 6 shows the layered architecture of the MNS.

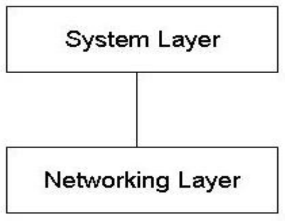

Figure 6. The layered architecture of the MNS

The next phase in the process is the process is the development of code and execution views of the system. However, these are detailed design views and are will not be addressed in here.

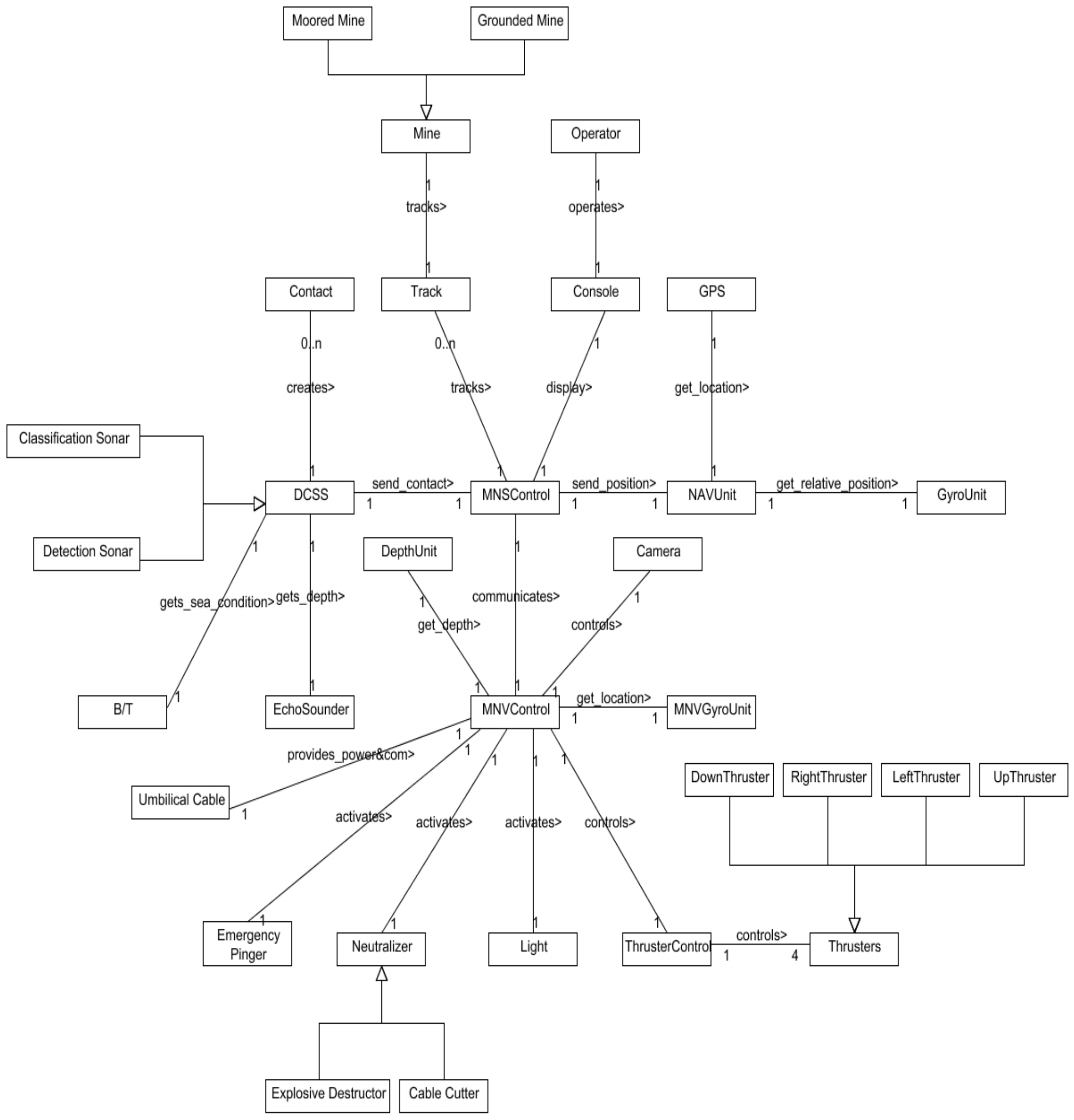

Figure 7. The domain model of the MNS

3.4. MNS High-Level Software Design

The inputs from different views of MNS architecture are used in the high-level design of the system. It is imperative that the design follows the architectural design decisions. A smooth transition from one activity to another activity is achieved in the MNS example using the architectural decisions and rationales. Figure 7 shows the derived domain model of the MNS. Note the structural similarity of the domain model and the star-controller architectural style introduced earlier. Figure 8 and 9 show the respective high-level design examples from the system.

4. Conclusions

In this paper, we presented the development of a real-world weapon system software architecture example. Weapon systems development is a long and expensive process. These types of systems are generally complex safety-critical embedded systems.

Because of these properties, high quality is imperative to accomplish in weapon systems. Only, a well-designed architecture can achieve all the necessary nonfunctional requirements.

First, we identified the high-level and user-level goals through interviews with navy officers and analysis of similar existing systems. Also, analysis of similar systems revealed the necessary components for the mine neutralization system. Then, the global analysis helped us to identify factors that influence our architectural design decisions. The strategies to resolve the factors are determined. The global analysis guided the development of the conceptual and module views of the system software architecture. The commonly-known patterns applicable to the product are used and a new architectural style is developed to meet the specific properties imposed by the previously identified factors. Finally, we showed how the architecture formed as an input for the high-level system design.

We introduced the star-controller architectural style within the system architecture development. This style has the advantage of

- Being simple and easily testable
- Achieving low-coupling and high-cohesion
- Having increased control over synchronization and communication needed in real-time systems.

The major drawback of the style is that the failure of a controller also disables the subsystems attached to it. However, in our context this drawback is not an issue. In weapon systems we require a fully functioning system. Impaired system functionality is not acceptable. Hardware redundancies are used overcome this drawback.

In the example, we used model-view-controller architecture pattern and star-controller architecture to achieve usability, extensibility, adaptability, modifiability, testability, maintainability, safety and reliability. The layered architecture is used to increase maintainability, safety and reliability.

5. Experiences, Lessons Learned and Future Work

During the system architecture development, we understood that we won’t able to satisfy all the nonfunctional requirements with one particular architecture pattern. The requirements we had forced us to use multiple software architectures for different nonfunctional requirement sets. This was a major finding and experience we had during development. Some of the lessons learned can be listed as follows:

- Developing software architecture is an organized and planned activity which takes the nonfunctional requirements into consideration as well as the functional ones.
- Paying special attention to the requirements gathering phase is a good promise of a successful software architecture development. The views of the navy officers about the system proved to be very critical at this phase.
- Partitioning the task into different architectural views, each addressing separate concerns, proves to be very useful in meeting both functional and nonfunctional requirements and reducing the cost of software development process.
- A well documented conceptual view ensures that the problem at hand is understood by all the stakeholders. Communication among developers is improved this way and misunderstandings between customers and engineers are reduced if not eliminated completely.
- Conceptual components ease the way we create module view which identifies static structures and layers of the system being developed. Conceptual view also establishes a starting point for the identification of a simple execution view.

Future work may include:

- One of the challenges we had was the necessity of incorporating redundancy. We eluded software redundancy by using hardware redundancy. However, we would like to see how software redundancy interacts with software architectures.
- Researching architecture description languages (ADLs) have been the focus of software architecture community in the recent years [10]. It is possible to analyze software architectures with ADLs. We would like to analyze the star-controller architectural style with ADLs and get an in depth understanding of the style.
- We would also like to see how the proposed style would be beneficial in other systems.

6. References

[1] Hofmeister, C., Nord, R., and Soni, D., *Applied Software Architecture*, Addison-Wesley Object Technology Series, New Jersey, 2000.

[2] Kruchten, P., *The Rational Unified Process: An Introduction,* Addison-Wesley, Reading, MA, 1999

[3] Demir, K.A., *Analysis of TLCharts for Weapons Systems Software Development,* Master’s Thesis, Naval Postgraduate School, Monterey, CA, USA, December 2005

[4] Drusinsky, D., Shing M., Demir, K. ”Test-Time, Run-Time, Simulation-Time Temporal Assertions in RSP, *Proceedings of 16th International Workshop on Rapid System Prototyping, (RSP’05),* Montreal, Canada, June 2005, pp. 105-110

[5] Garlan, D., and Shaw, M., “An Introduction to Software Architecture”, *Advances in Software Engineering and*

Knowledge Engineering, World Scientific Publishing Componay, December 1993, pp. 1-39

[6] Bachmann, F., and Bass, L., “Designing Software Architecture for Quality: the ADD Method”¸ OOPSLA¸ Tampa Bay, Florida, USA, October 2001

[7] Bass, L., and John, B. E., “Linking usability to software architecture patterns through general scenarios”, *The Journal of Systems and Software*, Vol. 66, 2003, pp. 187-197

[8] H.F. Hofmann, and F. Lehner, “Requirements Engineering as a Success Factor in Software Projects”, *IEEE Software*, July/August 2001, pp. 58-66.

[9] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., *A System of Patterns,* John Wiley & Sons ,West Sussex, England, 1996.

[10] Medvidovic, N., and Taylor, R. N., “A Classification and Comparison Framework for Software Architecture Description Languages.” In *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp.70--93, 2000.

Acknowledgements and Disclaimers

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any affiliated organization or government.

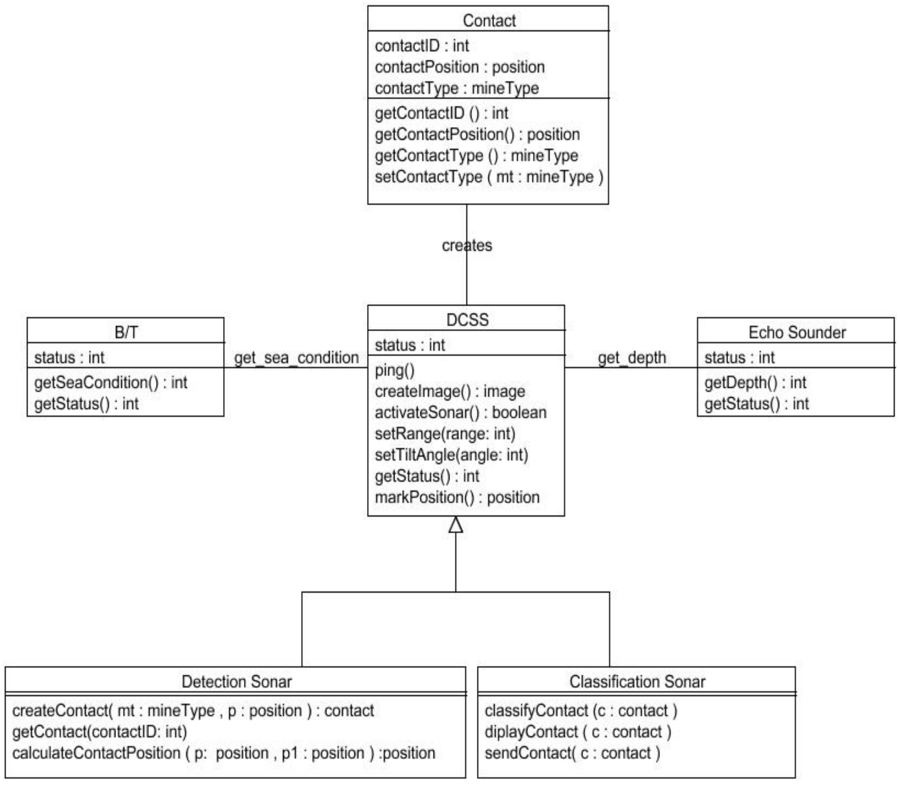
Contact
contactID : int
contactPosition : position
contactType : mineType
getContactID () : int
getContactPosition() : position
getContactType () : mineType
setContactType (mt : mineType)
creates
B/T
status : int
getSeaCondition() : int
getStatus() : int
get_sea_condition
DCSS
status : int
ping()
createImage() : image
activateSonar() : boolean
setRange(range: int)
setTiltAngle(angle: int)
getStatus() : int
markPosition() : position
get_depth
Echo Sounder
status : int
getDepth() : int
getStatus() : int
Detection Sonar
createContact(mt : mineType , p : position) : contact
getContact(contactID: int)
calculateContactPosition (p: position , p1 : position) :position
Classification Sonar
classifyContact (c : contact)
diplayContact (c : contact)
sendContact(c : contact)

Figure 8. The detection and classification sonar suite high-level design

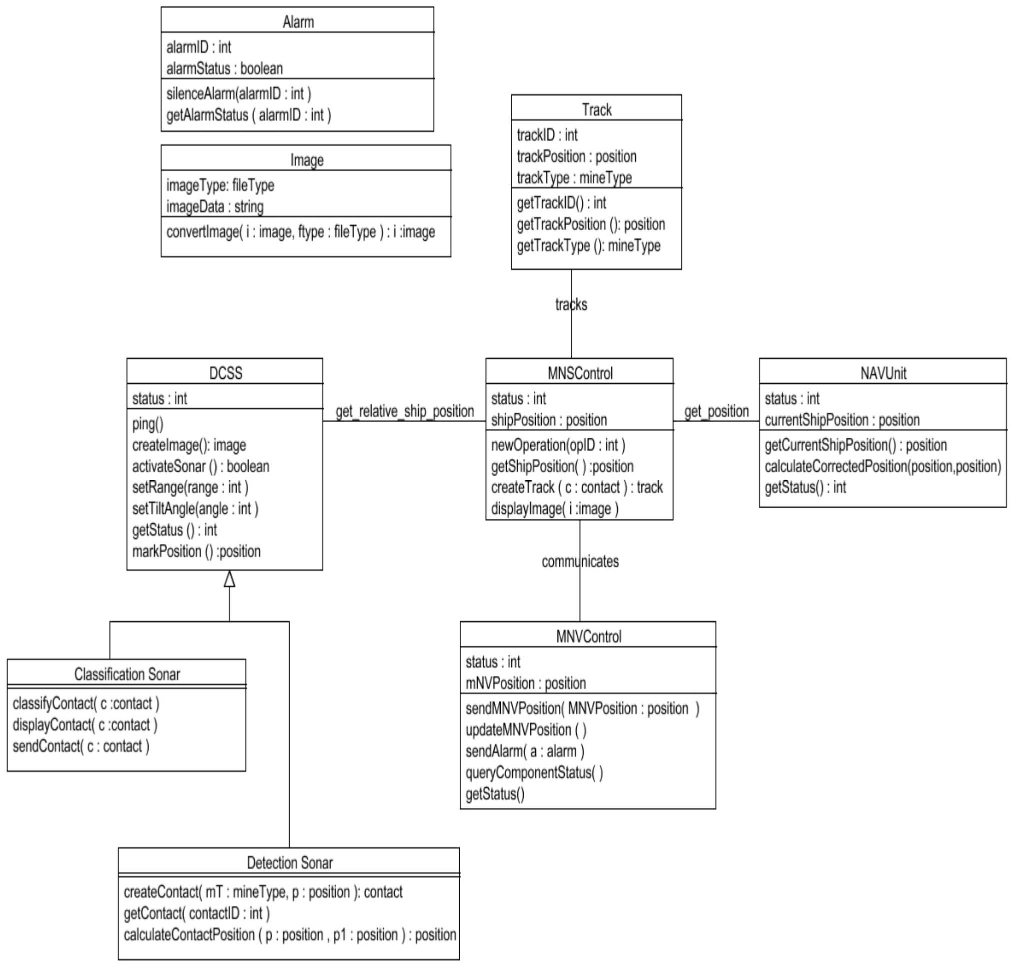
Alarm
alarmID : int
alarmStatus : boolean
silenceAlarm(alarmID : int)
getAlarmStatus (alarmID : int)
Image
imageType: fileType
imageData : string
convertImage(i : image, ftype : fileType) : i :image
Track
trackID : int
trackPosition : position
trackType : mineType
getTrackID() : int
getTrackPosition (): position
getTrackType (): mineType
tracks
DCSS
status : int
ping()
createImage(): image
activateSonar () : boolean
setRange(range : int)
setTiltAngle(angle : int)
getStatus () : int
markPosition () :position
get_relative_ship_position
MNSControl
status : int
shipPosition : position
newOperation(opID : int)
getShipPosition( ) :position
createTrack (c : contact) : track
displayImage(i :image)
get_position
NAVUnit
status : int
currentShipPosition : position
getCurrentShipPosition() : position
calculateCorrectedPosition(position,position)
getStatus() : int
communicates
MNVControl
status : int
mNVPosition : position
sendMNVPosition(MNVPosition : position)
updateMNVPosition ( )
sendAlarm(a : alarm)
queryComponentStatus( )
getStatus()
Classification Sonar
classifyContact(c :contact)
displayContact(c :contact)
sendContact(c : contact)
Detection Sonar
createContact(mT : mineType, p : position): contact
getContact(contactID : int)
calculateContactPosition (p : position , p1 : position) : position

Figure 9. The mine neutralization system controller high-level design