A Generic Software Architecture for a Domain Specific Distributed Embedded System

G. Santhosh Kumar, Rameetha K and K Poulose Jacob

Department of Computer Science, Cochin University of Science and Technology, Kerala, INDIA
email: {san, rameetha, kpj}@cusat.ac.in

***Abstract*—In this paper, we have evolved a generic software architecture for a domain specific distributed embedded system. The system under consideration belongs to the Command, Control and Communication systems domain. The systems in such domain have very long operational lifetime. The quality attributes of these systems are equally important as the functional requirements. The main guiding principle followed in this paper for evolving the software architecture has been functional independence of the modules. The quality attributes considered most important for the system are maintainability and modifiability. Architectural styles best suited for the functionally independent modules are proposed with focus on these quality attributes. The software architecture for the system is envisioned as a collection of architecture styles of the functionally independent modules identified.**

***Index Terms*—Software Architecture, Distributed Embedded System, Architectural Style, Domain Specific Architecture**

I. INTRODUCTION

SOFTWARE architecture has emerged as an area of immense interest among researchers and software practitioners. This has resulted in several approaches to deal with architecture definition, description and analysis, architectural styles and domain specific architectures [1] [2]. With the proliferation of embedded systems in the market such as mobile phones and PDAs, more focus has been laid on systematic and architecture based solutions for embedded system software. The objective of our study is to arrive at a robust architecture for the embedded software. Embedded systems are generally long drawn systems with an active life span of over 20 years. Therefore, the software architecture should result in a system that is highly maintainable and modifiable. The complexity of the software that goes into the embedded system has grown manifold over the years [3]. Therefore it is essential to follow quality software architecture to ensure that the complex software meets the functional and performance goals. The software architecture of the system is evolved by organizing the application logic as a composition of modules (functional components) and the definition of their interfaces (connectors) [4]. The architecture is evolved for a class of systems wherein variations to specific systems can be effected by addition/ deletion/ variation of existing components in the generic architecture.

II. PROBLEM DOMAIN

The problem domain is a distributed embedded system used in military applications. These systems have been following custom coded functionality, optimized for speed and other performance parameters with very little thought to long term maintainability and modifiability [5]. The use of state of art of technology - software architecture- for such real time embedded applications are inadequate.

The embedded computer system is mostly heterogeneous and distributed, because modern systems are often composed of existing subsystems, having their own control software and processors [6]. Furthermore, systems must be easily *maintainable*, *scalable* and *adaptable*, to support ever changing functional specifications and evolution of computer hardware.

The system under study is an integrated one comprising of various types of environmental sensors, data acquisition and data processing hardware interfaced to the sensors, application specific embedded processors, general purpose embedded processors, standard communication interface between the embedded processors, display units and console station for operator interaction. The embedded processors communicate over dedicated links and/or over bus using standard communication protocols.

A general operational scenario is described. The system sense the external environment through the array of sensors, carry out necessary preprocessing on the input data stream, performs application specific detailed data processing so as to extract the required information from the data, and displays the information to the operator in the form of various graphical displays. The operator is capable of configuring the system into the required mode of operation by way of the human machine interface. A simplified block diagram of the system is as shown in Figure 1.

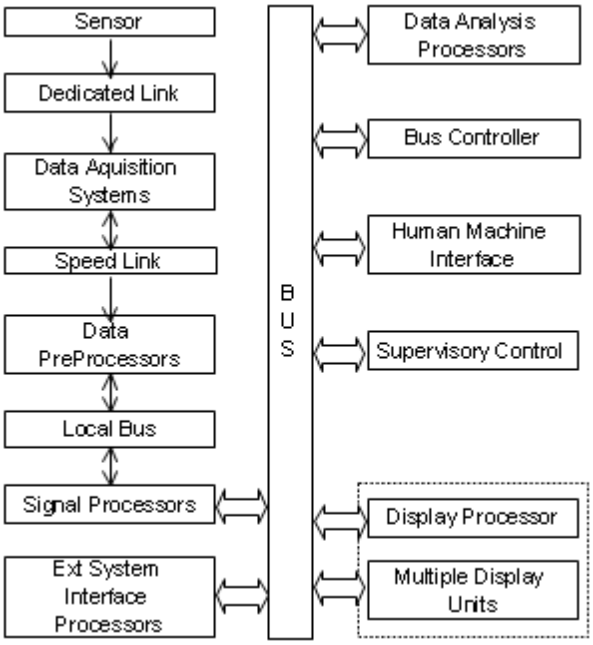

Fig. 1. Block diagram of distributed embedded system.

The first step in the design of software for the domain is software architecture design. The quality of the final system is heavily dependent on the quality of the architecture. The embedded systems are usually heavily constrained in both hardware and software due to the stringent functional and performance requirements. Therefore it is very essential to prioritize the requirements taking into view the concerns of all the stakeholders of the system. The quality attributes are of critical importance in defining the software architecture. Therefore the stakeholders have to arrive at a consensus regarding the most essential quality attributes. The degree to which the system meets the quality attributes often determines the success or failure of the system [7].

III. Architectural Quality Attributes

The embedded system considered in our problem is expected to be operational for a long period of time. The architecture business cycle of the system defined by the system stakeholders consider maintainability and modifiability as the most important quality attributes.

Maintainability of the system can be defined as the measure of the relative cost of modifying the architecture to accommodate new functionality [8]. It is a function of locality of change and abstraction level of the software.

Modifiability of the system can be defined as the ease with which the system can be modified to meet the change requirements[8]. As the system is put into operation, it is sure to encounter change requirements mostly from the operational point of view. Also as the technology become obsolete, the system may be required to adapt to new operating environments. The ability of the system to incorporate these modifications with ease and without compromising on functionality or performance is a very important quality attribute for systems having long life span.

IV. Software Architecture based design process

The focus of our work is to find out a most suitable architecture for the problem conforming to the above mentioned quality attributes. The Architecture based design process, which has its foundation on iterative functional decomposition of systems, is expected to uphold the required quality attributes. The method is based on three foundations – The decomposition of the system into functional subsystems to the required level of abstraction, the realization of the functional, quality and business requirements of the system through choice of suitable architectural style and the use of software templates [9]. The architecture can be abstracted and reusable units can be identified. The collection of the reusable architecture components forms the domain specific repository for the system. The developing organization can use this repository effectively for future systems. Specific architectures can be instantiated easily from this repository.

The Command, Control and Communication system is iteratively partitioned into functionally independent systems and subsystems. At each level of iteration the decomposed subsystems have well defined functional responsibility. The partitioning can be carried out to the required level of abstraction. The guiding principles for the functional decomposition are: -

Each subsystem

-is a functional unit with well-defined responsibility
-exports functionality to other modules
-imports functionality from other modules [4].

Following these guidelines, the system is partitioned into seven subsystems. These subsystems can be iteratively decomposed into subsystems.

The subsystems are

1. Sensor – Data Acquisition
2. Data Preprocessing
3. Data Analysis
4. Display Processing
5. Supervisory Control
6. Human Machine Interface
7. Communication

Each subsystem has an independent, well-defined responsibility. The complexity of the software in each of the subsystem vary. These factors force us to think of different architectural styles to suit the embedded software in each of the subsystems. The subsystems can be categorized into two: *Hardware intensive subsystems* and *Software intensive subsystems*.

The first two subsystems, Sensor–Data Acquisition and Data Preprocessing are hardware intensive. The software volume is comparatively less in these subsystems. The software work on custom hardware. However, the performance requirements of the software in these subsystems are more stringent due to the necessity of handling high data rates and synchronization with the hardware. The hardware

used in these subsystems is application specific and often the software has to provide custom interface to the hardware

The next four subsystems - Data Analysis, Display Processing, Supervisory Control and Human Machine Interface and Communication subsystems are software intensive and work on general purpose embedded processors. The software has to perform various information extraction algorithms, complex signal processing algorithms, optimization processing etc. to extract the information from the data captured by the sensors.

V. Proposed Abstract Software Architecture

The abstract software architecture for the system is the collection of the architectural styles defined for the subsystems. A class of concrete architecture style for each of the subsystems is proposed. The software architecture is described in terms of components and connectors encapsulated by virtue of their functional responsibility. Details of control flow, synchronization constructs and inter component communication protocols are not assumed. These may be carried out at a finer level of abstraction, when the abstract architecture is instantiated into software architecture for a specific problem.

The architecture of the entire system is a simple and powerful structure in which the system is iteratively decomposed into functional subsystems and suitable architecture styles. The functional, business and quality requirements of the system can be validated using this method. The architecture can be iteratively decomposed to a level wherein the software templates corresponding to each of the lowest level modules can be specified.

The subsystems identified during the Architecture based design process are explained in detail in the subsequent sections. The functional requirements of the system and the best suited architectural styles are identified. The architecture style is illustrated using collaboration diagrams.

A. Sensor – Data Acquisition

The Sensor Data Acquisition subsystem is the front end of the system where the sensors receive input data from external environment. The data may be from a single sensor, an array of homogeneous sensors or from an array of heterogeneous sensors. The data stream is sampled at the required rate and made ready for the subsequent data processing subsystems. The processing function in this subsystem is controlled by the availability of the data from the sensors

The Pipe and Filter style architecture style is proposed for this system[2]. The system has an explicit linear data flow from source to destination. This architecture is the most suitable one to process data streams. The data gets transformed in the filters and between filters the data flows through Pipes.

The sensor data arrive at a fast and steady rate and storage options are limited in the data acquisition subsystem. Further, the ensuing processing has to be carried out in real time. Therefore the data needs to be pushed through the pipeline. The data source pushes the data in a downstream direction. Since the data flow follows the "push strategy" the architecture style is refined further as Push Pipe and Filter style based on how control is exerted on the data. The architecture of the system during sensor data acquisition scenario is shown in Figure 2.

The component types are Sensor (Producer), Filter and the Sink (Consumer). The sensor component produces the data and pushes it into an output port connected to the input end of a pipe. The Filter component receives data from an input port connected to the output end of a pipe, transforms the data and puts the data onto an output port that is connected to the input end of a different pipe. The destination or the sink component gets the data from the input port that is connected to the output end of a pipe and consumes the data. A filter component is a combination of a sink and a Producer. The components in this subsystem are usually realized in hardware with dedicated communication links between the components.

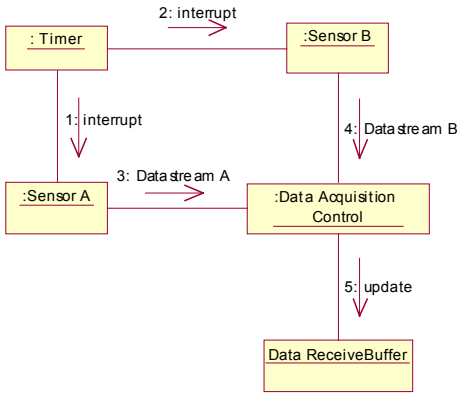

Fig. 2. Sensor – Data Acquisition

The Pipes or connector types in the subsystem are procedure invocations invoked once the data is ready to be pushed down the pipeline to the next filter component. Each of the filter components is a black box with a well-defined functional responsibility. This helps to ensure maintainability, modifiability and reuse which are essential quality criteria for the system.

B. Data Preprocessing

The data collected has to be subjected to various pre processing functions so as to prepare the data for detailed analysis. The processing requirements vary from one or two stages to multiple stages depending on the complexity of the system. Here, the data is transformed in a batch oriented fashion. The architectural style suitable for this system is the batch sequential data flow style. The architecture is an ordered sequence of independent processing steps. Each processing step operates on a predetermined data set and runs to completion before the next processing step. The intermediate data is stored wherever necessary. The data

preprocessing scenario while following this architecture is shown in Figure 3.

The components of this architecture are the distinct processing steps. They are independent software modules having a definite responsibility.

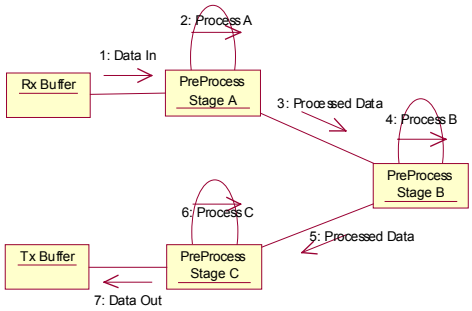

Fig. 3. Data Preprocessing

C. *Data Analysis*

The data from the sensor(s) after the preliminary processing are now ready for detailed analysis so as to extract domain specific information. Depending on the complexity of the system, there may be multiple data analysis processors. All of them need to access the data . The architecture of the system therefore has to be data centered. The blackboard architecture style is proposed for this subsystem. The data analysis scenario following the architecture is shown in Figure 4.

This style is characterized by two types of components. A central data store component, that represents the system state and stores the data to be processed, and a set of independent components that operate on the central data store. Each of the processing subsystem is capable of operating as a separate independent thread of control. These subsystems are partial solution providers and are triggered by the current state of the data store.

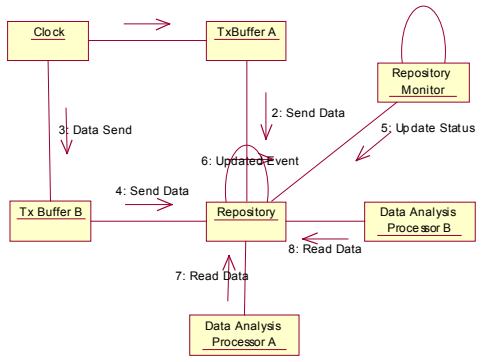

Fig. 4. Data Analysis

D. *Display Processing*

Display processing system is responsible for the presentation of the results of data analysis. The system is crucial in the domain of Command, Control and Communication systems as the operator decision making is solely dependent on the information presented on the display. Therefore, the subsystem has to emphasize on systematic organization of information for easy assimilation.

The Display processing system has dual responsibility at this level of abstraction. (1) Reception of the results of Data Analysis and (2) Display specific processing of the data and converting them into video objects. It can be decomposed into distinct systems at the next level of system decomposition.

The information to be displayed are held by various data analysis subsystems. Not all the information from all the data analysis subsystems needs to be displayed at the same time. This depends on the selected mode of the system and the current processing configuration.

Display subsystem has to follow an architecture whereby only the information required to be displayed in the current mode of the system is received from the Data Analysis subsystems. The display processing scenario while following this architecture style is shown in Figure 5.

The architecture style suitable for this processing is that of independent interacting processes where the processes interact by way of event based implicit invocation. Here, The display processing system will broadcast the current display mode and configuration. The data analysis subsystems that are information providers for this mode and configuration will respond to the event by sending data. The display is not unnecessarily loaded with unwanted data. This architectural style is characterized by the style of communication between components. Rather than invoking procedures directly or sending messages, the components announce events. Components register interest in an event by associating a procedure with the event. When the event is announced the system implicitly invokes all the procedures associated with the event.

The components in this style are modules whose interfaces provide a collection of procedures / methods and a set of events that it may announce. The connectors are bindings between the event announcements and procedure / method calls. The components that announce the event do not know which components will be affected by the event. Also the components cannot make assumptions regarding the order of processing or what processing will occur as a result of its events.

This style adopted for the display subsystem provides strong support for modifiability and asset reuse. Display system is most susceptible to user change requests. The architecture style facilitates modifications/ enhancements with ease by registering/ deregistering components. The style also upholds the maintenance attribute by easily adding and replacing components with minimum affect on other components in the system.

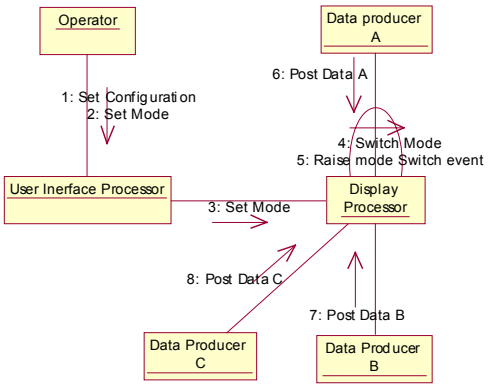

Fig. 5. Display Processing

E. *Supervisory Control System*

The supervisory control system has the role of a system supervisor. The embedded system is capable of operating in different modes. Each mode of operation has its associated configuration defined by its control parameter set. The control parameter sets corresponding to the different configurations are provided by the user and are stored in the Human Machine Interface subsystem. Upon selecting a particular mode/configuration, its control parameter set needs to be loaded into all the other concerned controllers. The configuration needs to be pulled from the Human Machine interface subsystem. The user provided configuration also need to be transformed into control words which has to be communicated to other subsystems so as to effect the configuration.

The data flow architecture style – Pipe and filter - is proposed. The data source here is the Human Machine Interface subsystem and the data sink is the Supervisory Control subsystem. When the operator changes the system mode the data sink pulls the corresponding configuration data from the source . Therefore the architectural style of the subsystem is refined as "Pull variant" of Pipe and Filter style. The Supervisory control system informs the transformed configuration control words to the subsystem responsible for effecting the configuration change. The Supervisory control scenario while following this architecture style is shown in Figure 6.

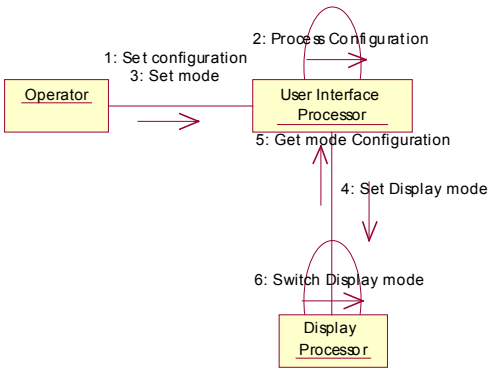

Fig. 6. Supervisory Control

F. *Human Machine Interface*

The Human Machine interface subsystem is responsible for providing operator interaction with the system. The system is configured and controlled through this interface. The subsystem is also responsible for providing interface to a variety of Input/Output devices. The I/O devices, its dedicated hardware units, embedded processors and the HMI control software together comprise the Human Machine Interface system and is housed ergonomically in the operator console.

In the system operating scenario, the operator configures the system into the various modes by providing the control parameters through the keyboard and other input devices. The Human Machine interface processor processes the configuration data, checks for its permissibility and also maintains consistency of the configuration data. The same data is used to generate multiple views.

The Model View Controller architecture style is proposed for this subsystem [2]. The Human Machine interface scenario while following this architecture style is shown in Figure 7.

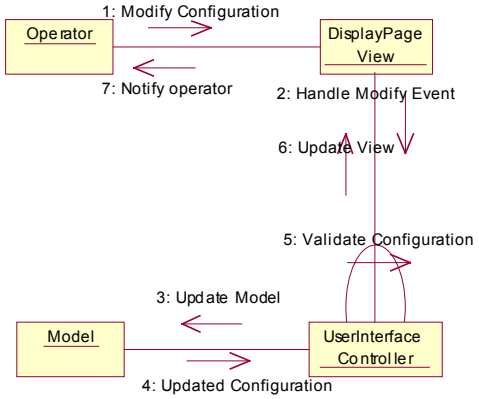

Fig. 7 Human Machine Interface

Following the MVC architecture, the Human Machine Interface system is decomposed into three components. (1) A model containing the functionality and the data (2) Multiple views of the information to the user (3) Controller to handle the asynchronous user inputs. In this architecture each change in the control parameter set is propagated into the data model thus ensuring consistency. This architecture ensures high maintainability and modifiability due to the division of the user interface into the MVC components.

G. *Communication*

The subsystems carry out their functions with the help of an effective communication backbone. The components building up the system were broadly classified as hardware intensive and software intensive subsystems. The role of communication subsystem in this domain is two dimensional. The communication within and between hardware intensive subsystems is by way of dedicated communication channels. The emphasis of these systems on performance constraints is very high and these systems cannot afford overheads involved while following standard interfaces. Software

intensive subsystems communicate by way of standard Bus network. A dedicated embedded processor assumes the role of the bus controller in such systems.

The Bus controller follows a hierarchically organized layered architecture style within. The operating system dependent functions, Low level utilities for bus control, Bus control software and the application level utilities provided for communicating with other embedded processor connected over the Bus follow an "Onion skin" model.

The embedded processors connected over the bus use Call and Return architecture style for communicating with each other.

VI. Generic Software Architecture of the System

The software architecture of the system is derived by systematic organization and representation of the architecture styles of the constituent subsystems. At the topmost level the subsystems identified during the first level of functional decomposition along with data and control interfaces are represented. This forms the high-level software architecture. (Figure 8) Each of these components are exploded subsequently and the internal architecture confirming to its style is represented. This is continued to the required level of abstraction. Hierarchical organization of these representations forms the software architecture of the system.

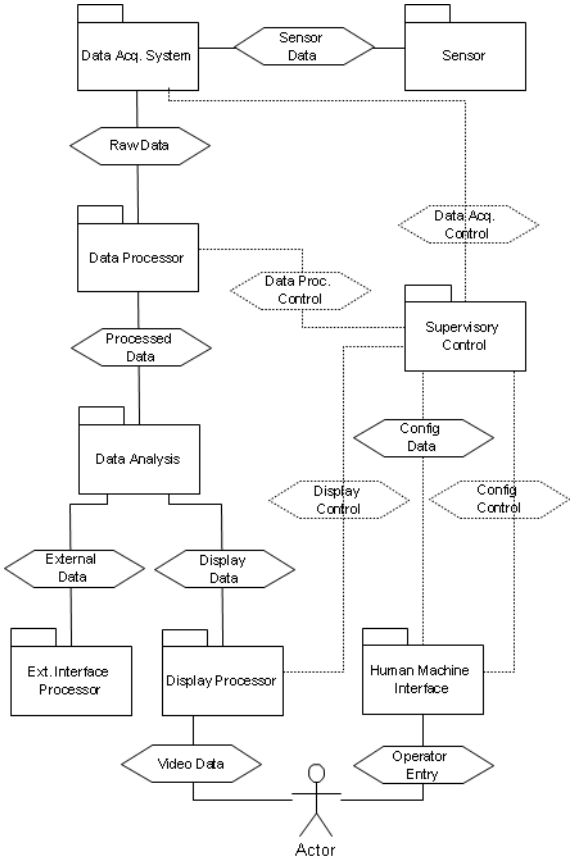

Fig. 8. High level Software Architecture

VII. Conclusion

We have proposed a domain specific reference architecture for distributed embedded systems. The significance of the proposed architecture is in partitioning the system into functionally independent modules following the Architecture Based Design method guidelines. The methodology can be applied to decompose the system iteratively to the required level of abstraction. The modules are mapped onto suitable architectural styles that realize the functional, quality and business requirements of the system. These components form abstraction of the system, which can be reused across systems in the domain. They form the constituent elements of a software architecture repository in the domain of distributed embedded systems. It can be used necessary variation when building specific products. ABD methodology can be applied until the system is decomposed into concrete components and software templates. The component architecture style and software templates and have to be logged into the domain specific repository for future reference.

In the next phase of the work we are planning to build a prototype system instantiating the proposed architecture

References

[1] Len Bass P Clements and R. Kazman, *Software Architecture in Practice.* Addison Wesley, 1998.

[2] Mary Shaw and David Garlan, *Software Architecture: Perspectives on an emerging discipline*. Prentice Hall, 2002.

[3] Nenad Medvidovic, "Software Architectures and Embedded Systems: A match made in Heaven?" , *IEEE Software*, Sept/Oct 2005.

[4] Gerd Frick, Barbara Scherrer, and Klaus D.Muller-Glaser., "Designing the Software Architecture of an Embedded System with UML 2.0", Proceedings of the UML 2004 Workshop on Software Architecture Description & UML, October 11-15, 2004, Lisbon, Portugal.

[5] Michael W DaBose, "A layered software architecture for Hard Real Time Embedded Systems", Doctoral Thesis, Naval Postgraduate School., Monterey, California, March 2002. http://handle.dtic.mil/ 100.2/ ADA401651

[6] H. Kopetz, *Real-Time Systems: Design principles for distributed embedded systems*, Kluwer Academic Publishers, 1997.

[7] S. Jarzabek, B. Yang, S. Yoeun, "Addressing quality attributes in domain analysis for product lines", *IEEProc-Software,* Vol. 153, No.2, April 2006.

[8] Henrik Baerbak Christensen, "Using Software Architectures for Designing Distributed Embedded Systems", Technical Report, University of Aarhus, Denmark. http://www.cfpc.dk/publications/

[9] Felix Bachmann, Len bass, Gary Chastek, Patrich Donohow, Fabio Peruzzi, "The Architecture Based Design Method", Technical Report CMU/SEI, Jan. 2000 http://www.sei.cmu.edu/publications/ documents/ 00.reports/00tr001.html