ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, Vol 8(23), DOI: 10.17485/ijst/2015/v8i22/73246, September 2015 ISSN (Online) : 0974-5645

An Efficient Architecture for Implementing Syntax Evolution of Embedded System

J. Sasi Bhanu[1*], A. Vinaya Babu[2] and P. Trimurthy[3]

[1]Department of Computer Science Engineering, KL University, Vaddeswaram - 522502, Guntur, Andhra Pradesh, India; sasibhanu@kluniversity.in
[2]Department of Computer Science and Engineering, JNTU Hyderabad - 500085, Hyderabad, Andhra Pradesh India; avb1222@gmail.com
[3]Department of Computer Science and Engineering, ANU, Guntur - 522510, Andhra Pradesh, India; profpt@rediffmail.com

Abstract

Background/Objective: Every embedded system follows a language to communicate with the HOST. Changes do happen to the language which must be affected while the system is up and running dynamically due to criticality reasons. **Methods/ Statistical Analysis:** An efficient architectural modelling is undertaken which includes components that co-exists with ES components and help in dynamically adapting to the changes initiated from the HOST to be carried to the syntax and semantics of the command language used for effecting necessary operations within the embedded system. A tabular method has been used to compare the architectural models and find the most effective architecture that best suits a syntax evolution system. **Findings:** Different kinds of evolution systems that include syntax, semantics, online testing and the communication system have to be supported to make an embedded system adapt to the changes dynamically. Syntax evolution system is the interfacing system that deals with evolving different kinds of commands, changes to the commands and the new commands which are transmitted from the HOST and the target to adapt the same through invoking the related real-time functions. The commands must be resolved and evolved dynamically. Different kinds of methods have been invented which can be used to affect the changes required to the command language dynamically while the system is up and running. The components that are required for implementing the methods have also been invented. The new components have been added to the ES software components and a new architecture has been evolved. The new architecture has been included with all the components related to a Nuclear reactor system which is basically a safety critical system. **Application/Improvement:** A comparative analysis of all the architectures reveal that the new architecture has all the features required for undertaking the dynamic syntax evolution at it also reveals that the new architecture uses all the methods required for dynamic syntax evolution while other architectures supports just one or two methods.

Keywords: Dynamic Evolution, Effective Architecture, Embedded Systems, Syntax Evolution

1. Introduction

An embedded system which is meant for monitoring and controlling a safety critical and Mission critical system must be connected to a HOST situated at a remote location for transmitting control data, references data and the commands required to setup the environment required for an embedded system to function. The HOST and the embedded systems must be situated at long distances due to the mission and safety critical reasons. Figure 1 shows the connectivity between the HOST, embedded system, production system and the test equipment.

The production system is the actual system which is monitored and controlled by the embedded system. The production system that is meant for monitoring and controlling the temperatures within the nuclear reactor systems is used for experimenting the findings presented in this paper. The connection between the production and ES system is established through local short distance

*Author for correspondence

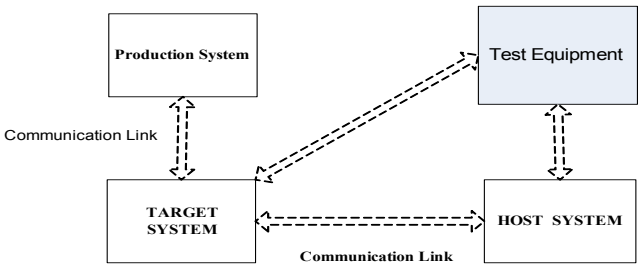

Figure 1. Host connectivity with the target embedded system.

connections which are generally achieved through wired point to point connectivity. Wi-Fi based connectivity can also be used when the feasibility of connecting through the wires does not exist. The issues related to connectivity of ES sensors and actuators to the production system is out of scope of this paper.

The ES system must be connected to the HOST for transmitting control data, reference data and the commands required for setting the ES environment. The HOST however has to be remotely situated from the ES system due to the safety and criticality reasons.

Several methods can be used to effect communication between the HOST and an embedded system. The HOST can be connected to an embedded system through any type of serial communication systems which include RS232C, USB, WiFi, Bluetooth and parallel communication systems that include ESA, PCI and Ethernet. An Embedded system which is fitted with Ethernet interface can be connected to a HOST through Internet. The ES system and the HOST can be placed apart at long distances through establishment of connectivity through Internet. The communication between the HOST and the ES system can be implemented through implementation of Email, WEB services and WEB servers as part and parcel of an embedded system.

The communication between the HOST and the target can be achieved through implementation of one of the following interfaces.

- Command Language Interface by way of transmitting a string of Text containing the command and the data arguments to the command.
- Remote method Invocation and passing of the Command and the data as arguments.
- Transmission of the objects either way which encapsulates the commands and the data arguments to the commands.

Communicating through transmission of commands is more frequently employed technique. The remote HOST will communicate with an embedded system for several purposes through use of a command language. The commands as such are to be understood both by the HOST and ES. The embedded systems are originally designed with a set of commands, each command meant for a particular purpose.

To realize the above mentioned functions, commands are to be issued from the HOST to the target and the target after processing the commands must pass the results back to the HOST. The set of commands and the associated data arguments can be initially designed using the standards which may change from time to time.

Embedded systems which are meant for monitoring and controlling of the mission critical systems must not be shut down for want of making changes to the ES software as shutting down of the mission and safety critical systems is not practically feasible. Any change needed must be achieved while the ES system is up and running. Any change to the embedded software needs to be done dynamically meaning, the change has to be undertaken while the system is running. The embedded system must be adaptable to the changes dynamically.

A specific standard syntax is used for transmitting the command and the arguments that the commands required for executing the code on the ES (Target) side. Generally UNIX like command and command line arguments standard can be used universally and as such the need for making changes to the standard using which the command strings are presented is not much of a concern.

Changes to the commands used between the HOST and the embedded systems are inevitable. Generally the commands are available as a set with version attached to the set. When new commands are added or changed, new versions of the command sets gets created. It is quite possible that at a given time more than one command set is to be operated. Each command set can be considered like a module operating at a time.

The changes to the commands and the command sets must be undertaken dynamically. The dynamic evolution of the embedded system is called syntax evolution. The syntax evolution is the command language evolution which is used to effect the communication between the HOST and the Target. The communication between the HOST System and the Target system is most important as data moves too and

forth for monitoring and controlling of the Production system and the kind of actions that must be taken when data is received from either end. The execution of functions at either end, based the data received requires that a Command Language interface be implemented which can be dynamically adapted as the changes to the interface takes place over the time. Dynamic Evolution of the interface is required as the Production system of type mission and safety critical nature cannot be shut down and as a consequence the embedded system interfaced with the production system also must be up and running as long as the production system runs.

The commands may change several times in vocabulary or new commands may be added from time to time. The same command may also be existing in several versions. The embedded system should adopt itself to the changes taking place in the commands either due to change in meaning and content of the command or adding more commands as required. The changed scope of the commands and addition of the new commands must be adapted dynamically without the need to shutting down either the embedded system or production system. The vocabulary evolution must be dynamic and on-line upgradable.

Embedded systems run in harsh environment with lot of limitations on the availability of the computational resources such a memory and the processing power. If the changes are to be implemented on-line, the software components that are related to interfacing through command language must evolve as the changes takes place. The evolution must take place without the need for enhancement of the embedded system resources.

Dynamic syntax evolution requires the design of an appropriate architecture that is suitable for making changes to the command language while ES system is up and running. A suitable architecture must be implemented based on the type of change that must be adapted at a given point in time.

2. Problem Definition

Thus the problem of syntax evolution is to evolve command language as the changes takes place from time to time and adapt the revised command language as the ES system is running. The changes to a command language involve operating several versions of the same command and versions of the set of commands. There is a need to investigate various types of architectures that deals with effecting changes to the syntax system and also to find the efficient architecture that seamlessly lead to implementation of syntax evolution system. Several methods have been presented in the past for dynamic extension of the software systems and have offered pragmatic ideas for software evolution using generic Architectures.

D. Notkin et al.[1] have presented novel ideas for dynamic extension of software systems. S. Jarzabek et al.[2] have presented ideas for software evolution using components and generic architectures. Software architecture must the first step in architecting the software evolution P. Oreizy et al3., P. Oreizy et al.[4], M. Shaw et al.[5] and L. Bass et al[6]. Once the issue of software evolution is made part of the software architecture design, then the issue is dealt throughout the rest of the life cycle of the system. Software evolution is generally considered as an issue of software adaption and it has been recommended that the software architectures must be first step to consider the software evolution as the changes takes place. At every step of the Software Life cycle the software evolution issue is to be considered so that the software evolves as the changes evolve.

Nary et al.[7] has considered the software evolution from the point of view of software adoption. Software adoption is a non-functional requirement of any system. Adoption is effecting a change into a system occurring due to the external events. Adaptability is the ability of a system to adapt to the changes in the environment. Adaptability is a Non-Functional requirement and software has to be adaptable if it has to evolve. The adaptability requirement must be part of software requirement specification so that it can be considered as a part of software architecture itself. One has to construct architectures that deal with the issue of software adaptability. Nary et al.[7] has suggested a three tire architectural model which considers establishing the communication between the HOST and the Target in the first tier. Figure 2 shows the architecture presented by them.

The data sent by the HOST System is received by the communication block which hands over the string to the command evaluation model which implements the verification of the correctness of the format of the command string. The Command Evaluation module provides for the software that validates the commands received by it and hands over the commands to one of the Command processors after ensuring that the commands received are correct grammatically. A kind of decision logic is implemented by the Command evaluation module based on

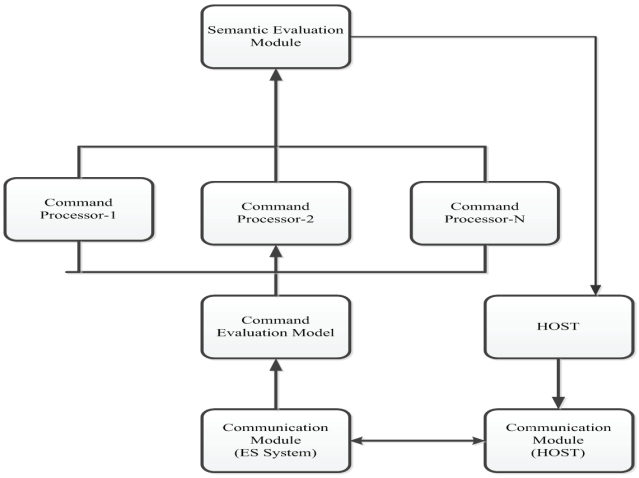

Figure 2. Architectural model-1 for syntax evolution.

which the received command is handed over to one of the command evaluation processors.

The Command processor parses the commands and the arguments and verifies the validity of the Command and the arguments by meaning and the content. The data representing the arguments transmitted along with the command are verified and validated. A response is sent back to the HOST if any of the failures in processing the commands is noticed. The command processor hands over the commands to semantic evolution module once the command is found to be correct after verification. The semantic evolution model will affect the necessary ES application evolution as conveyed through the commands transmitted by the HOST.

The Semantic Evolution block comprises the embedded application which actually implements the commands received by it and send the results directly to the communication block for onward transmission to the HOST. The syntax evolution model also sends back the results to the Communication block onward to the HOST system.

The architecture proposed by Nary et al.[7] is a three tier model which considers the communication, Syntax evaluation and semantic evaluation undertaken in each of the tiers. The architectural models presented in the literature have ignored describing the issues related to implementing the architecture to effect the syntax evolution of the communication between the HOST and the Embedded system using a command language interface. The environment in which the embedded system must function can change from time to time. The embedded software must adopt itself to the changes in the environment initiated from a remote location.

The changes to the embedded system might happen in the communication interface, command language or to the ES code. The changes to the communication block are rather minimal and are assumed to be absent by considering standard and stabilized communication software. The Syntax Environment of the software systems changes continuously and the embedded systems must adapt to such changes for continued success and survival.

Several software architectures have been proposed in the literature but they did not really address the issues related to adapting a system dynamically to the changes taking places to the command language itself. They have not suggested any architectural model that suits to implementation of the dynamic evolution of command language which is used between a HOST and an embedded system.

Nary et al.[7] have considered several techniques for adoption of software components which include standard method, conditional expressions, algorithmic selection, modification of binary code at run time and components porting outside the system. The modification of the binary code at run time is most suitable for testing the mission critical and safety critical embedded systems as on-line adoption do not call for the shutting down of either the production system or the embedded system.

Conditional expressions let a component change its behavior based on the value of an expression. Algorithmic selection involves selecting a different algorithm to adapt to an environmental change. Runtime binary code modification involves changing the binary executable to adapt to an environment change. The porting outside the system method involves moving the component that has to be adapted outside of the embedded system to a more traditional environment. This lets the available adaptation strategies for non-embedded software be used to achieve the adaptation.

Various commands are to be issued from HOST to the embedded systems to facilitate communication and executing the functions at either end. The commands and the scope of the commands while can be designed during the initial development phase of the system, more commands may have to be added subsequently.

The commands and the input data to the commands may change from time to time specifically due to the development of the embedded system application. Every time new functions are added, new commands are to be added. The new functions may

be added either due to addition of new hardware in terms of sensors and the actuators or _bbgrgml md lcu dslargmlq rm rfc embedded system itself. The embedded system must evolve dynamically when changes in the vocabulary of the existing commands or addition of the new commands are necessary. The adoption must be done dynamically without the need for shutting down of any of the systems. If an embedded system is designed using a standard command language interface, then any change in the standard, calls for changes in the syntax of the command language.

The requirement analysis of safety and mission critical system includes non-functional requirements such as af_lecq rm rfc command language interface. The requirements pcj_rcb rm command ctmjsrgml and Command processing which are evolution agents for the Syntax Block k sqr `c gbclrgdgcb.

The communication module at the target, first must recognize the command part of the string and then check whether the command is the existing command or a new command. If the command issued is the existing command, the same is issued to the corresponding Command Processor or else the communication block must communicate with the HOST for want of the Hex Code for the new Command Processor which can process the new command issued.

The new Command Processor must be copied to the address location specified by the HOST and dynamically linked. The New Command Processor is then issued with the new command for the processing.

P. A. Laplante[8] have detailed various issues that must be dealt with* in designing real time systems. They have referred some issues related to dynamic evolution that must be considered at the design stage.

Sasi Bhanu et al.[10] have described an architecture that can be used for implementing an embedded system for which the command language to be used for communicating with the remote HOST can be evolved. But the architecture proposed is limited to fixed number command language processors. A new architecture that can be used for implementing syntax evolution considering both self-adaption and enhancement by adding more number of command processors is presented in this paper.

3. Investigations and Findings

Every system must be continuously updated to accommodate changes taking place during life of a system. A system will succeed during its life only when changes to the software are undertaken as the environment for which the software is developed changes.

Software architecture defines the software components and the interaction between the components therefore helps in limiting the changes to the components and making it easy adapt to the changes.

Adaptability for embedded systems often requires techniques different from those used for non-embedded systems. Embedded systems run in a harsh environ-ment – small memory, limited computational resources, and meet fast response to process external events and are usually real-time systems as well. Several examples of embedded systems that require dynamic evolution of embedded systems have been presented in the literature. Various evolutionary strategies are required for implementing dynamic evolution bearing in mind the limited computational power and memory of such systems.

Adaptation is the change to be carried to a system to accommodate changes taking place in the environment. The system can be called adaptable if it can adapt to the changes taking place in the environment. The software built on the requirements specifications that include adaptability requirements would become evolutionary.

The first step in building evolutionary software is the construction of adaptable software architecture. Adaptable software architecture lets the changes be considered in terms of the architectural components of the system that individually are much smaller than the whole system. This significantly reduces the problem of determining the parts of the system to change and effecting the change, each time the system needs to be changed.

One of the main problems that occur is the need of the embedded system to respond to the commands initiated from a HOST and that the embedded system to understand the vocabulary of the commands initiated from the remote HOST.

The main reasons of the evolution of the vocabulary is due to the need for adaption of the standards which may be changed from time to time, need for adding more commands due to change in the scope of ES applications system, the need for the new commands to communicate with older version of the ES modules etc. Syntax

evolution is all about adapting the changes taking place in the command language and dynamic evolution implies making changes to the command language as the system is running.

Many software architecture models related to the dynamic evolution of software have been considered but not much in specific to evolution of command language. Any change required in the software must be carried and dynamically configured. The adaptability of the software can be achieved in different ways which includes Static or Dynamic, Manual or Automatic and Proactive or Retroactive.

In the case of dynamic adoption, the system changes only its run time behavior while its implementation is fixed. Dynamic evolution implies accommodating the change both in behavior and implementation. Adaption and evolution both are to be considered in case of Syntax and Semantic evolution. While Syntax is related to command language interface, Semantics is related to the very embedded software itself.

In the case of static adoption, the changes are made off line, the embedded system is shut down and the new system is moved. The static adaptability of the changes to the either syntax system are semantic cannot be done in the case of production system which are either mission critical or safety critical as they cannot be shut down for want of making changes. The static adaptability is achieved through making changes manually outside the embedded systems whereas in the case of dynamic adoption, the changes are undertaken automatically while the embedded system is up and running.

Proactive adoption shall take place when an embedded system recognize the change in advance and adopt it before the environmental change has taken place, whereas in the case of retroactive adoption the syntax component of embedded system adopts to the change after the change has taken place. In the case of mission critical system the adoption should be dynamic, automatic and at least retroactive.

Several architectures have been presented for syntax evaluation dynamically which are suitable for managing the Mission Critical and Safety Critical systems. The architectures support the dynamic and automatic evolution and the change in the environment is effected either retroactive or proactive. Several architectures can be sued for syntax evolution of embedded systems which include Static-Manual-Proactive, Dynamic - Automatic-Retroactive, Static-Automatic-Retroactive and Dynamic Automatic-Proactive. The architectures that are related to static adaption are not quite useful for dynamic evolution as the changes are to be carried while the ES system is up and running.

In the case of the architecture that includes dynamic, automatic and retroactive adaption, each generation of evolving vocabulary is assumed to have a basic structural difference that lets the Syntax Evaluation system decide to which Command set that the command sent by the HOST belongs to. For each command set which is in a way the generation of the vocabulary, there should be associated Command Processor pre-identified and available as part of the syntax evaluation system. In this case, the adaptability is done after the change in the vocabulary has taken place and the syntax evolution system has to have the logic to recognize the change and also should have the logic to determine the Command Processor that is suitable for the new version of the vocabulary transmitted by the HOST. The process flow diagram to affect dynamic, automatic retroactive syntax evolution is shown in the Figure 3.

The main drawback of this architecture is that the prior knowledge on the evolution vocabulary quite in advance is necessary and that a suitable Command Processor be pre-identified and made part of the syntax evaluation system. This model is helpful if the increments of the software that will be added to the system are known quite in advance.

The syntax evolution is done in this case by finding the best matching Command Processor explicitly. However the limitation is that the Command Processor is pre-identified and made available as a part of the Syntax evaluation system right at the time of implementation of embedded system for the first ime time In the case of

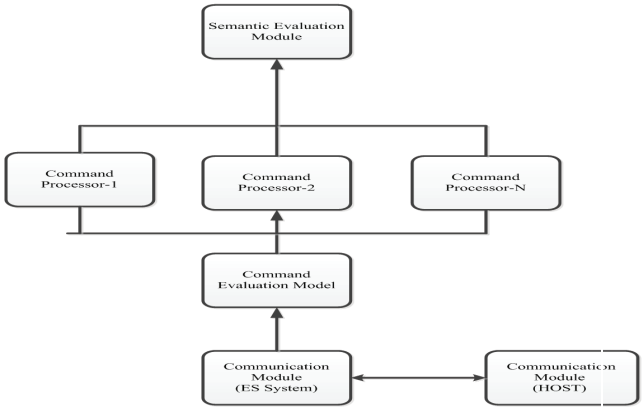

Figure 3. Dynamic, automatic and retroactive syntax evolution architecture.

mission critical system and safety critical system, it is not possible to predict in advance the kind of software changes required or the uncommon failures that can occur or the kind of process required for investigating the failure. The changes to the embedded system contained in all the three layers (Communication, Syntax and Semantics Layers) cannot be foreseen in advance as the operating environment of such as system is dynamic.

Another architecture has been proposed that adapts to the changes dynamically and automatically and the adoption implemented is proactive meaning, the system adopts quite ahead of the occurrence of the change. In this architecture, the Command Processor modifies itself dynamically based on the changes taking place in around the ES system. The occurrence of any change is pre-identified by the HOST and communicates the same to the target. The change is effected either through selection or modification.

In the selection architecture, the HOST first informs the syntax evaluation system that a new command is being sent. The Command evaluation system (parser) selects the correct Command Processor that will parse the new command. The architecture is shown in the Figure 4.

The command evaluation system makes necessary changes to the environment like making an entry into the lookup table related to the command processor indicating the command, the related ES application code, the logic which should hold good for making the related ES component to be activated whenever such a new command is received from the remote HOST.

In this case also it is expected that all the Command Processors are available and the new commands are added in increments to the existing command processors.

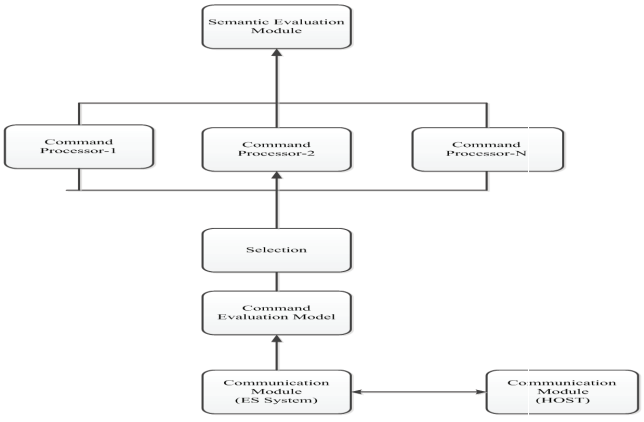

Figure 4. Dynamic evolution through selection – and proactive.

The kind of actions that must be taken when such a new command is to be processed, some rule based specification is made available to the selected command processor and the processor adapts the new command through its internal logic. Thus the selected command processor will be able to process the new command. When a new command has to be added to the existing set, the same is added to the existing command processor through implementation of a kind of lookup table that indicates that a new command has been added. In this case, the code of original Command Processor is not modified. Only the environment under which new processor must operate will change.

This architecture apparently will not be suitable if the command Processor has no logic to process a new set of commands which are not foreseen and built into the ES application. This is the case when the very structure of command processor is not suitable for processing the new command submitted to it for processing. It is not quite possible to select a command processor that can most optimally be able to adapt to a new command when issued to it. The actual issue is that, the new commands that will come up in future cannot be foreseen at the analysis stage and that all the Command Processors required to process the future commands cannot be built right at the time of implementing the system at the first time.

In the case of dynamic Evolution through modification, the code can be modified through a set of rules that are maintained into a Lookup Table. A look up table can be maintained that relate a command to the rules that must hold good for the command to be processed by the command processor. Many commands can be added into the lookup table along with the rules by the Command evaluation system and the Command processor automatically adapts the new commands that were included into the lookup table. The modification architecture is shown in the Figure 5.

Adding more command processors is yet another architecture which requires addition of new code into the system, the binary code related to command processor can be dynamically changed to accept evolved commands. Addition of a new command processor is achieved through a special command issued from the HOST which not only sends the Command but also the code that must be written to the Memory. The syntax evolution system writes the new processor to a memory location which has the inbuilt logic to recognize the new command and the kind

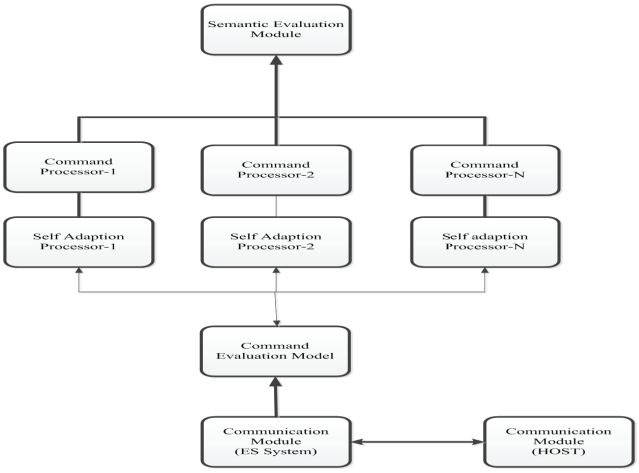

Figure 5. Dynamic syntax evolution through modification architecture.

of communication it should do to process the command. The Command evaluation system must be able to recognize the new commands and re-direct the command to the new command processor. The architecture that deals with addition of new Command processor is shown in the Figure 6. The command evaluation system activates a process that reads the code and copies the code to a memory location, creates a task out of it and makes the task known to the RTOS through creation of a task dynamically for RTOS to schedule the same. The command evaluation system creates necessary lookup tables required by the new command processor. The mapping of new commands to the command processor is undertaken by the command evaluation system, meaning the lookup table that shows the relationship between the command processor and the commands is updated.

Thus it is evident that dynamic evolution of syntax is supported through either making the existing command processors adapt new commands and process them through new lookup tables that relate the processing rules through which the new commands are processed or by adding new command language processors to process new version of the set of commands.

As said earlier, the embedded systems are limited in resources and adding more command language processors or the lookup tables used by the existing processors through a challenge. The code enhancements if any have to be contained within the limited resources with which the ES system has been originally designed. Adding hardware resources can be done within the expansion capability of the Hardware using which ES system can be built.

Dynamic evolution can also be done by adding independent additional Embedded systems with new command processor built into it and connecting the new Embedded processor with the original embedded system through establishing a network of embedded systems with the original embedded system still monitoring and controlling the mission/safety critical system. The Environment setup required for executing the command however is transmitted by the original embedded system. The Dynamic syntax evolution based on the networking of Embedded systems is shown in the Figure 7.

All the architectural model discussed above assumes that the ES system has enough resources to add/Update the lookup tables or adding more code to cater for

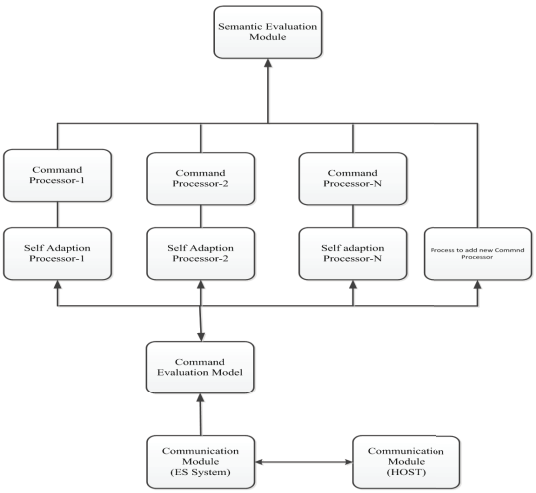

Figure 6. Dynamic syntactic evolution through addition of new Command processors.

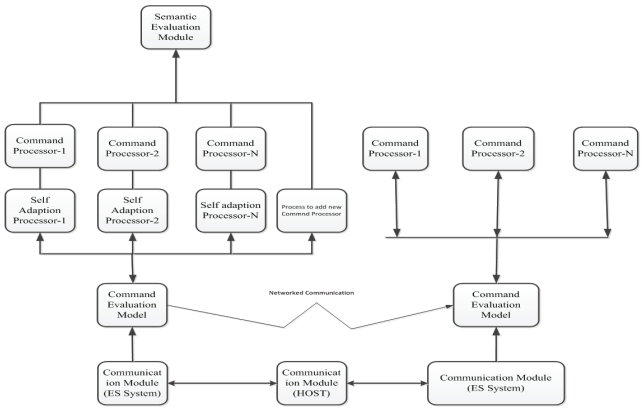

Figure 7. Network dynamic evolution of embedded systems.

the new command processors especially the need for additional memory. Memory can be added through the expansion slots exposed outside. But the code required to drive the memory added from outside has also to be added to the ES application. In addition the code required to interface with the driver has also to be added. Memory pools are to be created mapping to the external memory units through using RTOS functions. All this complicated process has to be undertaken while the ES system is up and running.

The dynamic modification of the Command Processor shall help in accommodating the change in the vocabulary of the system. However the dynamic modification of the Command Processor is restricted to the existing Command Processors only. The dynamic modification of the Command Processor has a limit that the code size cannot exceeds the limits specified at the design stage as any of the excess code may overwrite other areas of the code.

Thus the architectural models suggested in the literature really do not help when system resources are to be expanded when more command processors are to be added. The requirement of additional system resources has to be informed to the HOST when there no space available for accommodating more command processors initiated from the HOST.

While there are many architectural recommendations in the literatures, no recommendations have been made regarding the implementations of the same. Sometimes the need to support syntax evolution leads to the need of supporting semantic evolution.

Embedded systems are real time in nature and predominantly maneuvered by the real time operating systems. The syntax evolution described in various architectural models has to be developed and implemented under the scope of real time systems. No attempt in this regards has been done so far by anybody.

3.1 Implementation of the Dynamic Evolution Models for Embedded Systems

Thus there are only two alternatives for syntax evolution of the embedded system for adapting to the changes taking place in language with which the communications is effected between the HOST and the embedded systems. The alternatives include making changes to the existing command language processor or add more command language processors for accommodating new commands and versions of the exiting commands. The implementation of the architecture chosen is one of the most important challenges that one has to meet. The implementation of the chosen architecture under a chosen real -time operating system under which a ES application has been built through further challenge. The sections below addresses these issues.

3.2 Command Language Specification

The most important aspect of command language implementation is the very design of the command string specification which is communicated from the HOST to the EMBEDDED SYSTEM. The specification must include command name and various arguments to the command which can be indicated using the format shown below:

Command Name, Number of Arguments, Arg-1, Arg-2, ARg-3, Arg-X

The first argument **Arg-1** will indicate whether the command is New/Existing, the **Arg-2** will indicate the version of the Command and the rest of the arguments will indicate the argument values.

3.3 Implementation Architecture of Syntax Evolution for Embedded Systems

The dynamic syntax evolution of embedded systems must adapt all the architectures that have been explained to make it more versatile. The architecture must cater for expansion of existing command processors or addition of more command processors, that meets the twin objectives of self-adaption and modifying the existing command processors and addition of new processors is shown in the Figure 8.

Several components have been included into the architecture which comprises of a communication component, Syntax evaluation component, a set of command processors each meant for self-adaption, semantic evolution, ES application related command processor, processor for adding more command processors, communication related processors etc. All these components are essentially the tasks that are scheduled to be executed by triggering their related events which are to be effected under the control RTOS. The component that needs to work under the influence of an RTOS is shown in the Figure 9.

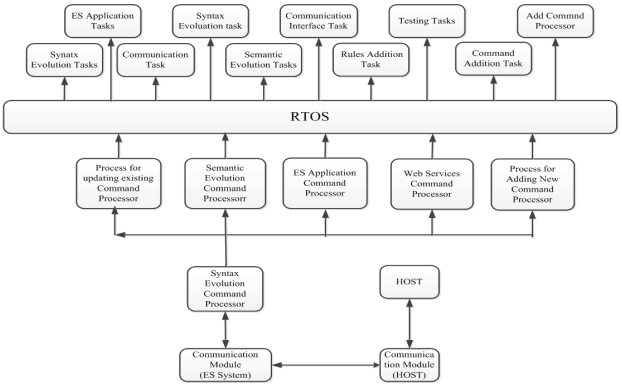

Figure 8. Software architecture for dynamic syntax evolution of embedded system.

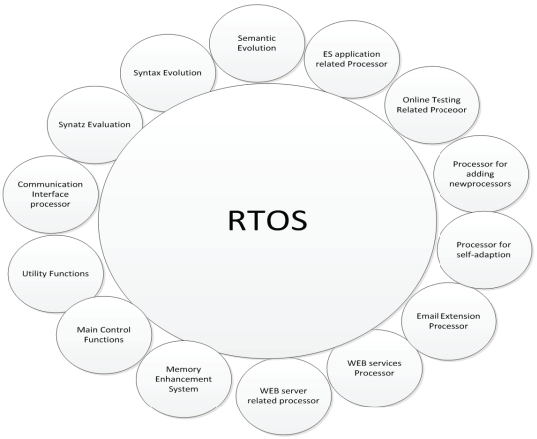

Figure 9. Syntax evolution under an RTOS.

Table 1. Comparison of architectures related to Syntax evolution

Comparison parameter	Static-Manual-Proactive	Dynamic-Automatic-Retroactive	Static-Automatic-Retroactive	Dynamic-Automatic-Proactive	New Architecture
Issue of comman ds from remote location	X	X	X	X	√
Self-Adaption	X	√	X	√	√
Addition of new command processors	X	X	X	X	√
Adaptability under a RTOS	X	X	X	X	√
Extension of adaptability as the system resources are added	X	X	X	X	√

4. Comparative Analysis

A comparison of various types of architectures that can be used over several prevailing parameters has been made and presented in Table 1. It could be seen from the Table 1 that the new architecture presented in this paper meets all the requirements that are to be met for archiving the syntax evolution for safety critical systems.

5. Conclusion

Dynamic evolution of syntax used by the HOST to communicate with the Target is absolutely necessary as it is not possible to shut down the embedded system that monitors and controls a safety and mission critical system for making changes and then restart the system again. Existing architectural models are suitable for syntax evolution of loaded systems and the models are included with one or two evolution methods. The models as such are not quite suitable for embedded systems. New methods are identified that are quite suitable for evolving the syntax used by a Target for a HOST to communicate with it. The new architectural model avoids the necessity of pre-identifying the changes that are to be incorporated in future. It is not easy or possible to find the future changes required right at the time of designing a system. Changes do arise while the system is up and running and they have to be adapted as they arise. The architectural model presented truly implements the dynamic syntax evolution to implement the HOST-Target interface dynamically. The architecture is designed to achieve self-adoption and addition of new command processors as more versions of the command processors are realised in future.

6. References

1. Notkin D, Grisworld WG. Extension and software development. Proceedings of 10th International Conference on Software Engineering; 1998 Apr; p. 274–83.

2. Jarzabek S, Hitz M. Business-oriented component based software development and evolution. International Workshop on Large-Scale Software Composition. Vienna, Austria. 1998 Aug 25–28; p. 784–88.
3. Oreizy P, Gorlick MM, Tylor RN, Heimbigner D, Jhonson G, Mdevidovic N, Quilici A, Rosenblum DS, Wolf AL. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems. 1999 May/Jun;14(3):54–62
4. Oreizy P, Medvidoic N, Tylor RN. Architecture based runtime software evolution. Proceedings of International Conference on Software Engineering; 1998 Apr 19–25; Kyoto Japan; p.177–86.
5. Shaw M, Garlin D. Software architecture- perspectives on an emerging discipline. Prentice Hall; 1996.
6. Bass L, Clements P, Kazman R. Architecture in Practice, SEI series in Software engineering. Addison Wesley; 1998.
7. Subramanian N, Chung L. Architecture-Driven embedded systems adoption for supporting vocabulary evolution. Proceedings of the International Symposium on the principle of Software Evolution (ISPSE'00); 2000; Kanazawa; p. 144–53.
8. Laplante PA. Real-Time systems design arid analysis - An engineer's handbook. IEEE Press; 1993
9. Anbazhagu UV, Praveen JS, Soundarapandian R. A proficient approach for monitoring induction motor by integrating embedded system with wireless sensor network. Indian Journal of Science and Technology. 2014 Nov; 7(S7):174–9.
10. Sasi Bhanu J, Vinaya Babu A, Sastry JKR, Trimurthy P. Dynamic evolution of syntax for communicating with embedded systems. International Transactions on Electrical, Electronics and Communication Engineering. 2012; 2(4):37–41.