Implementing Syntax Evolution of Embedded Systems

Sasi Bhanu Jammalamadaka*, A. VinayaBabu, P. Trimurthy*****
* Department of Computer Science Engineering KL University, Vaddeswaram, Guntur District
** Department of computer Science and Engineering, JNTU Hyderabad
*** Department of computer Science and Engineering, ANU Guntur

Article Info

Article history:

Received Jul 2, 2015
Revised Aug 20, 2015
Accepted Sep 8, 2015

Keyword:

Command language
Embedded systems
ES architectures
Remote communication
Syntax evolution

ABSTRACT

Safety Critical systems such as Nuclear reactor systems cannot be shut down as restarting is a huge process and incurs heavy cost. The embedded systems which are used for monitoring and controlling the safety criticalsystems cannot be shut down as well. ES system which drives safety critical systems must be communicated from remote location generally through a HOST connected on to an internet. Communication between the HOST and the ES system is done using command language which has to be evolved from time to time. The change to the command language must be undertaken while the embedded system is up and running. The evolution thus must be dynamic. Many architecture have been proposed in the literature for evolving syntax of commandlanguage. The implementation of an efficient architecture as such has not been found place in the literature without which existing architecture as such has no meaning. This paper presents a set of methods using which the implementation of syntax evolution of embedded systems as such can be achieved. The syntax evolution methods presented have been applied to a safety critical system that monitors and controlstemperatures within nuclear reactor systems.

Corresponding Author:

Sasi Bhanu Jammalamadaka,
Department of Computer Science and Engineering,
KL University,
Vaddeswaram, Guntur District, Andhra Pradesh, INDIA 522502.
Email: sasibhanu@kluniversity.in

1. INTRODUCTION

1.1. Background

Majority of the embedded systems are designed to operate on their own without any outside intervention, Mission or safety critical systems requires monitoring and controlling initiated from a remote location.The monitoring and controlling is achieved through adapting the required sensing and actuating mechanisms.

An Embedded system which is meant for monitoring and controlling a safety critical and Mission critical systems must be connected to a HOST which is situated at long distances through Internet for safety reasons. The connection is required for transmitting control data, references data and the commands required to set up the environment for embedded system to function. Figure 1 shows the connectivity between the HOST, Embedded Systems, the safety / Mission critical system and test equipment.

The ES system and the HOST can be placed a part at long distances through establishment of connectivity between the systems through internet. The communication between the HOST and the ES can be implemented through implementation of Email, WEB services and WEB servers as an integral part of the embedded system in addition to the ES application system implemented in it for sensing and actuating mechanism.

The communication between the HOST and the Target can be achieved through implementation of command language interface, remote method invocation and object transmission. Communicating through transmission of commands is more frequently employed technique. The remote HOST will communicate with the embedded systems for several purposes through implementing a command language. The commands as such must be understood both at the HOST and the Embedded system. The embedded systems are originally designed with a set of commands, each command meant for a particular purpose. To realize the core functions of the application system and other functions that gets added due to dynamic evolution, 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.

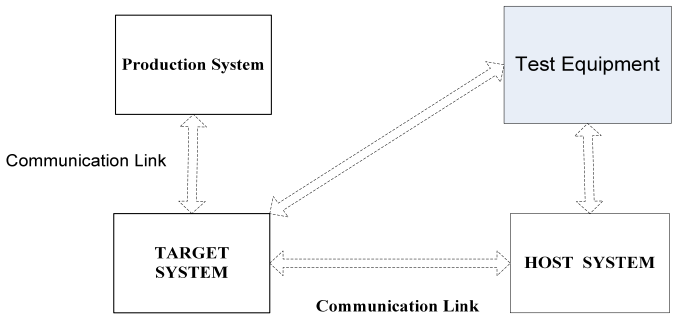

Figure 1. Host Connectivity with the Target Embedded System

Embedded systems which are meant for monitoring and controlling mission/safety critical systems must not be shut down for want of making the 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 generally used for transmitting the commands from the HOST to the Targets. While commands travels from HOST, the results obtained out of execution of the commands are returned to the HOST by the ES system. Generally UNIX like command language as a standard can be used universally and as such, the need for making changes to the standard of formulating the command string is not much of a concern.

Change to the commands used between the HOST and the embedded systems is 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 sets 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 also called as 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 to and forth for monitoring and controlling the safety/mission critical 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 on 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.

The commands may change several times in vocabulary, meaning and content. The changed scope of the commands must be adapted dynamically without the need for shutting down either the embedded systems or the Mission/Safety critical system. The vocabulary evolution must be dynamic and online up-gradable.

Embedded systems run in harsh environment with lot of limitations on the availability of the computational resources such a memory and the processing power. The embedded systems should also meet stiff response time requirements which are either hard real time or soft real time. If the changes are to be implemented online, 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 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 also be implemented based on the type of change that must be adapted at a given point in time. Many architectures have been proposed in the literature and none of them have clearly explained the way, architectures can be implemented which are designed to meet some defined response times and for which use of a real time operating systems (RTOS) is critical.

1.2. Problem Definition

The main problem is to find and implement methods that can effectively adapt a chosen architecture which has been proved to be efficient. The implementation of the architecture should at least be undertaken to the level of syntax evolution of dynamically evolving embedded system. The methods must be implemented under severe resources constraints which is the case with embedded systems. The methods must be running under real time operating system so that real time constraints can also be met with.

1.3 Related Work

Several Architectures have been presented in the past [1], [2] for dynamic extension of the software systems and have offered pragmatic ideas for software evolution using generic Architectures. [3]-[7] have 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 has considered the software evolution from the point of view of software adoption [7]. Software adoption is a non-functional requirement of any system. Adoption is a change in a system to accommodate changes in the environment. Adoptability 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 has suggested a three tire architectural model for establishing the communication between the HOST and the Target [7]. Figure 2 shows the architecture presented by them. Most of the architectures proposed in the literature more or less recommend the same kind of architecture for implementation of syntax evolution. 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 which the received command is handed over to the one of the command evaluation processors.

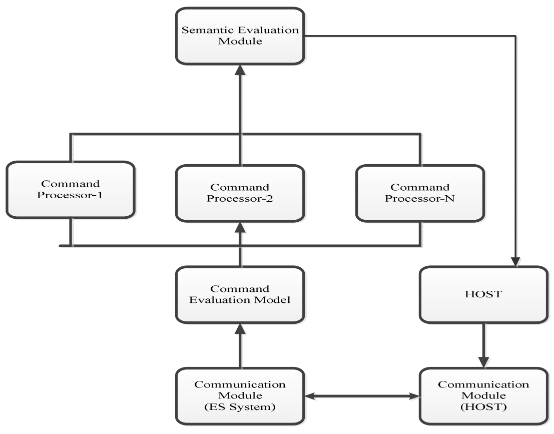
Semantic Evaluation Module
Command Processor-1
Command Processor-2
Command Processor-N
Command Evaluation Model
HOST
Communication Module (ES System)
Communication Module (HOST)

Figure 2. Architectural model-1 for Syntax Evolution

The Command processors 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 system 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 for 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.

These architectures 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 which operates under real-time environment.

Nary has 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 [7]. The modification of the binary code at run time is most suitable for evolving the syntax to communicate with the Mission critical and safety critical embedded systems as online adoption do not call for the shutting down of either the production system or the embedded system.

Conditional expressions let a component change its behaviour based on the value of an expression. Algorithm selection involves selecting a different algorithm to adapt to an environment change. Run-time 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 and vice versa 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 using the incremental model. 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 and the associated hardware on the embedded system side or due to enhancements of the features of the 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 shall include non-functional requirements such as dynamic evolution of command language interface. The requirements are designed by identifying the command evaluation and Command processing programs which are evolution agents for the Syntax Block.

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 et. Al., [8] have detailed in their hand book 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.

Many architectures have been presented in the literature which include Dynamic-Automatic-retroactive, Dynamic-Selection–Proactive, Dynamic-Modification-Proactive, Dynamic-Addition of Binary code, and Distributed Network based. All of these architectures suffers from lack of proper implementation mechanisms to effect the change within an embedded system that operate in real-time environment.

SasiBhanu et. al., have described an architecture that can be used for implementing an embedded system using which the command language to be used for communicating with the remote HOST can be evolved [9], [10]. Figure 3 shows the architecture proposed by them.Several components have been included into the architecture which comprise, 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.

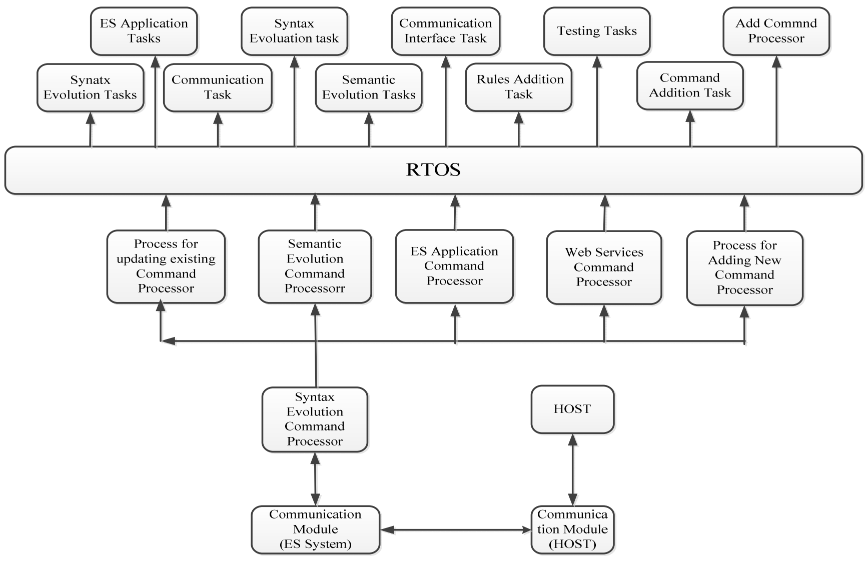
ES Application Tasks
Syntax Evoluation task
Communication Interface Task
Testing Tasks
Add Commnd Processor
Synatx Evolution Tasks
Communication Task
Semantic Evolution Tasks
Rules Addition Task
Command Addition Task
RTOS
Process for updating existing Command Processor
Semantic Evolution Command Processorr
ES Application Command Processor
Web Services Command Processor
Process for Adding New Command Processor
Syntax Evolution Command Processor
HOST
Communication Module (ES System)
Communica tion Module (HOST)

Figure 3. Software architecture for dynamic syntax evolution of embedded system

1.4 Solution

The main implementation solution proposed in this paper is based on use of event handling mechanism supported by any of the real time operating system (RTOS) for chaining, scheduling and un-scheduling the tasks for implementation of a dynamic syntax evolution of embedded systems. The process/Tasks have been identified and it has been shown how the additional processes that gets added to the existing embedded system will implement the dynamic evolution of syntax of commands that are used for effecting the communication between the HOST and the Embedded system

2. METHODS

Many important processes/systems have been used for implementing the syntax evolution more effectively under the influence of an RTOS within the Embedded systems. The subsystems/processes used for implementing syntax evolution include communication system which is achieved through Ethernetconnectivity, memory management, syntax evolution system, event management system, self-adaption process, and addition of more command language processors.

2.1. Memory Enhancement System

Memory management is one of the important issue other than Task creation and deletion that can be implemented by using the RTOS Functions. The location of the Task code within the memory, the free memory available, the addition of external memory etc. have to be supported through a memory management system. Memory management is crucial for dynamic syntax evolution.The memory management system manages memory through a memory lookup table. The format of the memory lookup table as maintained by the memory manager is shown in the Table 1.

Table 1. Memory lookup Table

Name of the Pool	Strat Byte Address	End Byte Address	OCC	Name of the Task Assigned	Name of the Data Assigned
Pool-Z	#0000	#10240	OCC	RTOS	
Pool-A	#10241	#15360	OCC	Temp-1 Processing	
Pool-B	#15261	#20480	OCC	Temp-2 Processing	
Pool-C	#20481	#25600	OCC	Temp Gradient Processing	
Pool-D	#25601	#30720	OCC	Initialization Processing	

Several memory pools are created locating the code and data using the Memory management support provided by the RTOS. The memory pools are used to indicate whether memory is allocated to data or the task code. The lookup table also shows the free memory pools that can be used to create new tasks. When a task is deleted the memory area occupied by it is released. A task is allowed to be deleted only when the task has nothing pending to do at the time of deletion or the deletion is postponed until all the execution that is related to the task is completed. If one memory pool is not sufficient additional memory pool is used for the same purpose

2.2. Implementation of Syntax Evaluation Component

The Syntax Evaluation component receives the command, determines whether the command received is the existing command or a new one based on the first argument that it has received from the HOST. This component also checks whether the string representing the command is syntactically correct. The syntactic validation of the command string is carried by Syntax valuation Task of the system. This task communicates back to the HOST in case of grammar failures that it has traced while validating the string. The module maintains a lookup table that maps the command to a command processor. This task finds the command processor to which a command belongs using the lookup table and hands over the command to the respective command processor. This module maintains the Table shown at Table 2. The event to be triggered for invoking the command processor that should be run is also maintained in this Table.

Table 2. Mapping Command processors to Memory addresses

Processor serial	Command Processor Name	Description of the purpose	Start Address of Command processor
1.	SynTaxEvl	To deal with commands related to Syntax Evaluation	#40961
2.	SenaticEvol	To deal with commands related Semantic Evolution	#51201
3.	ESAPPL	To deal with Commands related to ES' Application	#51201
4.	ONLTEST	To deal with Commands related to Online Testing' Application	#61441
5.	WEBSERCOM	To deal with commands related to WEB servers based Communication	#74543

Table 3. Mapping Commands to the Command processor

Serial Number of the Command	Command Name	Command Description	Component Name	Command Processor / Process Name	Command Argu-1	EVENT NAME	Task name
1.	CMDALLINT	Process for Communicating with HOST	Communicating with Host	BEGIN		STRAT	COMMUNICATION
2.	RULEADD	Adding new rule to the existing command			NAMECOM	ADDRULE	RULEUPDATE
3.	CMNDADD	Adding New command to existing Command Processor	Self-Adaption	SELFADAPT	NAMEPROC	ADDCMD	COMMNDUPDATE
4.	ADDCMNDPROC	Adding new Command Processor	Add-Command-Processor	NEWPROCESS	NAMEPROC	ADDCPRO	PROCESSADD
5.	CMDALLSYNEVL	ALL	Syntax Evolution	SYNTXEVOL PROCESS	NON	SYNVAL	EVOLSYNTAX

Table 3 Maps the commands to the command processors. All the commands that must be processed by the system are classified based on the processing that must be carried when a command is received. To start with the commands that are related to Syntax Evolution, Semantic evolution, ES application, Online testing and the ES system communicating with HOST have been identified. The Syntax Evaluation Manger, finds the command processor responsible for processing the command that it has received from communication Task and having found that the received command is valid. This task also updates Table 3 automatically when a new command is added to one of the existing command processor or whoever new Command processor is added to the system.

2.3. Implementation of Self-adaption of the Command Processor through Update Process

The process of adaption of the rules by a command processor is achieved through mapping of the rules to the command that must be adapted at run time by the command processor. This task is essentially responsible for maintaining rules to the command. Any number of rules can be mapped to a single Command. A command processor self-adapts itself through maintenance of rules to a command. Self-adaption is achieved through mapping rules to the command. HOST sends the command along with the rule that must be used for execution. The command processor also shall have intelligence to fetch the rules by comparing the existence of process states when compared to the rules that are enforced. The maintenance of the rules is achieved through specific commands initiated from the HOST. The update process maintains the rules to the commands. Table 4 shows the mapping of the rules to the commands.

Table 4. Mapping Rules to the Commands

Serial Number of the Command	Command Name	Command Description	Rule Code	Rule Description
1.			REF1GT	If reference-1 is Greater Than
	RECVREF1	Receive Reference Temp-1	REF1LT	If reference-1 is less than
			REF1EQ	If reference-1 is =
2.			REF2GT	If reference-2 is Greater Than
	RECVREF2	Receive Reference Temp-2	REF2LT	If reference-2is less than
			REF2EQ	If reference-2 is =
3.			TEMP1GT	If Temp-1 is Greater Than
	SENDTEMP1	Send sensed Temp-1	TEMP1LT	If Temp-1 is less than
			TEMP1EQ	If Temp-1 is =
4.			TEMP2GT	If Temp-2 is Greater Than
			TEMP2LT	If Temp-2 is less than
	SENDTEMP2	Send sensed Temp-2	TEMP2EQ	If Temp-2 is =
			IFOFF	If Buzzer is off

The Syntax evaluation module invokes the command processor related self-update process whenever a new rule to be added or updated is received. The HOST sends the command along with rules code that must be adapted when the command is initiated for execution.

2.4. Implementation of Command Processor

The primary responsibility of the command processor is to receive the command and the rules that must be enforced while the command is in execution. Every command is related to Task which is used as a basic design principle. The command processor prepares the data required to set the environment under RTOS required to execute the command through invocation of its related task. The command processor invokes its related component which is meant for setting the environment required to execute the task related to the command. Every command is mapped to its related task and the mapping of a task to the command is shown in Table 3**.** The command processor fetches the Task related to the command, rule that must be effected while executing the command from the Table 5 and prepares the data required for setting the environment required for executing the command.

2.5. Setting the Environment for Executing the Command

For setting the environment required to execute the command, the data that must be set, event that triggers a TASK to be moved from blocked state to execution state are required. The data required for setting the environment is generated from a specific rule that must be enforced. By default every task will be waiting for a specific event to take place. The event that must be triggered to execute the task related to the command is fetched from Table 3. The event triggering is undertaken by calling Appropriate function of RTOS.

2.6. Event Handling Mechanism

The dynamic evolution is built around event handling capability of RTOS. Events are handled through the functions supported by µCOS Operating systems. An event is essentially a Boolean flag that can be set or reset and that other tasks can wait for. When an event is triggered, the task that is waiting for the event to take place must be moved from the blocked state to ready state so that it can run. When an event occurs, the related hardware initiates an interrupt and from the interrupt routine the event occurring signalling can be implemented. The signalling sets a flag for which a particular task is waiting. A task related to the event will be in blocked state and as the event occurs, the Task is moved from blocked state to ready and then thetaskis made to run. Dynamic evolution is implemented extensively using the event handling capability of RTOS.To start with a handle for event handling is created which is in a way, providing the memory space for storing the details of various events, the related tasks and the state of the tasks. The declaring of the handle is done as shown below:

AMXID amxidHandle;

Various events that are related to the tasks are to be registered by calling RTOS functions. Table 3 will have all the details of the tasks and the events that are used by the Tasks. Table 3 is maintained by command evaluation component of the embedded system.Set and reset variables are hash defined for each of the event as detailed below for the Tasks related to the ES application. SET variable is used to signal the occurrence of the event and RESET is used to signal the completion of handling the event.

```
define AUTN-SET 0x0001
#define AUTN-RESET 0x0000
#define INIT -SET 0x0001
#define INIT -RESET 0x0000
#define RREF1-SET 0x0001
#define RREF1-RESET 0x0000
```

A handle to all the events shown in the Table 2 that can happen within the system has to be created by calling the following functions related to the RTOS.

ajvcre (&amixdTrigger, 0, “EVTR”)
The occurrence of an event can be signaled using the following function supported by RTOS
Ajevsig (amxidTrigger, AUTN-SET)
Ajevsig (amxidTrigger, INIT -SET)

These functions are called from any of the ES Task to trigger a communication that the related task that has been waiting for the event to commence its execution. The completion of handling the event is signalled by the TASK concerned by using the following concerned functions supported by RTOS. Every task has to wait for occurrence of an event and as it occurs, the task must be executed and then the event must be reset. Every Task can wait for the occurrence of an event by calling the RTOS function.

2.7. Making ES Application Tasks Event Driven

Various Tasks that are related to TMCNRS have been identified and presented in the Table 5. A Task lookup table is created. This table gets enhanced as and when more tasks gets added. This table is an indication of active tasks that runs under RTOS. This table gives control on creation and deletion of the tasks.

Table 5. Tasks/Components developed for pilot project

Task serial	Task Name	Task description
1.	RECVREF1	Task to receive Reference Temperature-1
2.	RECVREF2	Task to receive Reference Temperature-2
3.	AUTHENTICATE	Task to implement Authentication
4.	SENDTEMP1	Task to process and send Temperature-1 to HOST
5.	CONPUM1	Task to Control Pump-1

2.8. Tasks Related to TMCNRS (Temperature Monitoring and Controlling of Nuclear Reactor System)

Several tasks related to TMCNRS system have been considered which include Initialisation, Temp1 processing, Pump1 Processing. Temp2 processing, Pump2 processing, Temperature gradient processing and the same are made event driven. The code that is related to Pump1 processing is placed below:

```
CONPUM1 TASK ( )
{
while (true)
(
class CompareTemp1Task
{
        char t1;
        char ref1;
        char componetType = “S”
        float latency;
        OS_STKCompareTemp1TaskStk [3000];
        friend void hex2Ascii (unsigned char);
        ComapreTemp1withRef (char tt1, char rref1)
        {
        t1 = tt1;
        ref1 = rref1;
        ProcessPump1pp1 = ProcessPump1 ();
        t1=t1 / 2.55;
        if (t1>ref1)
        {
        pp1.PUMP = HIGH;
        }
        else
        {
        Pp1.PUMP =LOW;
        }
 }
}

// wait for the event to take place
Ajevwat(amxidTrigger, SP1- SET)
CompareTemp1Taskcmpt1;
Cmpt1. ComapreTemp1withRef (temp1, intref1);
Ajevsig(amxidTrigger, SP1- RESET)
// Chain to the next task for sending Temperature -2 to the HOST
Ajevsig(amxidTrigger, ST2-SET)
 }
```

2.9. Communication Task

The Communication Task is an ever ending Task for which Highest Priority is set and the task works in round-robin fashion. The main purpose of this task is to read the command from the HOST and store the same in a Global String and invokes an event that triggers the Symantec evolution component. This component also reads the data stored in a Global string Variable and sends the data to the remote HOST.Grammar especially to find the correctness of the command if it is the existing one and if the command is the new one to check whether its related command processor has already been created is verified by the syntax evolution block. If the command is the existing one it is passed to the concerned command processors (Semantic Evolution, ES application, Testing application, update process and for creating new command processor) by writing the string to a Global Variable and then creating an event for which the command processor has been waiting.

To start-with 5 command processors have been identified which belongs Syntax Evolution, Semantic evolution, ES application, on-line testing, and HOST communicating with the TARGET under WEB services mode. More processors can be added as required while the system is running over its life cycle.Communication task chains to syntax evolution component through triggering its related event. Syntax Task waits for its event to happen and spans to one of the command processors based on the command it has received. The syntax Block also spans to other Tasks which are meant for either self-adaption or creation of a new command processor when a need to handle new version of the command set arises.

2.10. Implementation of Self-updating of Syntax

This process will be activated for adding a command or adding a rule to a command by the Syntax evolution process through triggering the related ADDCOND or ADDRULE events. These events are triggered by syntax evolution process. The events are processed through a separate task specially designed for it. The self-adoption of various tasks is implemented through either adding commands to be processed by a process via their respective tasks or by adding rules for implementation of the commands. The commands are entered into the command lookup table and the mapping of the rule to the rule lookup table.

The self-adoption Task spawns to Rule update Task for adding rules to the commands and spawns to command addition Task for adding commands to the respective lookup tables. The self-updating task thus spawns to either command add or rule add tasks. The command read from the HOST is held in a global variable and various commands that have to be handled by the syntax evolution system are also held in a global look table built around two dimensional array of objects. The mapping of the commands to rules are also stored as global lookup table. The command is first added and the rule that should be used by the command is added next. Self-adaption thus implemented through addition of commands and the rules to the commands.

2.11. Adding Commands for Self-adaption

The command to be added is made available as third argument of the command line. No order is required in placing a command to command lookup table. The task will wait for ‘add command” event to be triggered. The command is added as when the event is triggered.

2.12. Adding Rules to Commands for Self-adaption

The rule to be added is made available as third argument of the command line. No order is required in placing a rule to Rule lookup table. The task will wait for the add rule event to be triggered. The rule is added as when the event is triggered.

2.13. Adding a New Command Processor

The Syntax Evolution process spans to PROCESS add Task whenever a new command processor is be added to the system. This Process to add a new command processor will be waiting till the time the syntax evolution task triggers “PROCESSADD” event. The process meant for adding a new command processor looks for the availability of the memory. If the memory is not available the message is sent back to the HOST that the memory is not available if the memory is available the start address of the memory is fetched and the same is stored in the global memory. This task then spans to another task that will read the code from the HOST and writes the code into the memory area. A Task is created and added into the memory and the Task is added to the Task lookup Table.

Table 6. Experimental results – Dynamic Syntax Evolution

		Test Case			**Test Results**	
Experiment Number	Command Sent	Command Argument-1	Command Argument-2	Command Argument-3	Argument-1	Argument-2
1.	RULEADD	REF1 (Name of the Command)	If Temp1>RefTemp1 + 2	Rule is added to the command	Ref1 =35	Temp1 = 32
2.	CMNDADD	TEMP12 (Command)	SEMANTICEVL (Processor name)	ADDCMND(Task name)	CommndTEMP12 is added	-
3.	ADDCMNDPROC	EMAIL (Process to add Email Extension processor	-	-	EMAL Extension processor is added	#93186 (Address Location at which the email processor is added
4.	RULEADD	REF2	If Temp2 >RefTemp2 + 3	-	Ref2 =35	Temp2 = 32
5.	CMNDADD	TEMP123 (Command)	SEMANTICEVL (Processor name)	ADDCMND(Task name)	CommndTEMP123 is added	-
6.	ADDCMNDPROC	WEBSER (Process to add Web server for effecting communication between the HOST and the ES system	-	-	WEBSER processor is added	#13170 (Address Location at which the email processor is added

A function of the RTOS is called to create the Task which represents the new command processor.The third parameter of the command line will be the name of the command processor which needs to be created.

2.14. Writing New Command Processor Code into the Memory

This Task will read the code from the HOST and writes the same till the end of code is transmitted by the HOST**.**

3. EXPERIMENTATION AND RESULTS

Experiments have been conducted on the development of the ES by sending different kinds of commands and the effect of execution is noticed through results seen on the HOST. Table 6 shows the experimental results. From the results it can be seen that syntax evolution through self-adaption of command processors and creation of the new command processor as the new version of the commands are released, has been achieved quite effectively and correctly implemented. The experimental results are seen through display on LCD connected to the ES system and also through display on the HOST system.

4. CONCLUSION

The Dynamic evolution of command interface between the Target and the HOST requires that the command interface be evolved dynamically and be adoptable without the necessity of shutting down any of the systems involved in the application. It is also not possible in the case of embedded systems that are related to mission critical system to fore see any of the changes that take place in future.

An efficient architecture that considers all the components that are required to implement the dynamic syntax evolution of embedded system has been used to effect the dynamic syntax evolution of the embedded system. The components required, the process of implementation, the use of event handling mechanism, the communication between the components that all put together the way the dynamic evolution of the embedded systems is achieved are presented in the paper. Even handling mechanism is the most suited method for implementing dynamic evolution of syntax within the embedded systems under the influence of an RTOS.

REFERENCES

[1] D. Notkin and W. G. Grisworld, “Extension and Software Development”, *Proceedings of 10th International conference on Software Engineering*”, pp. 274-283, 1998

[2] S. Jarzabek and M. Hitz, “Business-Oriented Component based Software development and Evolution”, *International workshop on Large-Scale Software Composition*, Vienna, Austria, pp. 784-788, 1998

[3] P. Oreizy, M. MGorlick, R. N. Tylor, D. Heimbigner, G. Jhonson, N. Mdevidovic, A.Quilici, D.S Rosenblum and A. L Wolf, “An Architecture-Based Approach to Self-Adaptive Software”, *IEEE Intelligent Systems,* pp. 54-62, 1999,

[4] P. Oreizy, N. Medvidoic, and R. N Tylor, “Architecture Based Runtime Software Evolution”, *Proceedings ofInternational conference on software engineering,* Koyato Japan, pp. 177-186, 1998

[5] M. Shaw and D. Garlin, “Software Architecture- Perspectives on an Emerging Discipline”, Prentice Hall, 1996.

[6] L. Bass, P. Clements and R. Kazman, ”Architecture in Practice, SEI series in Software engineering”, Addison Wesley, 1998.

[7] N(Nary) Subramanian and Lawrence Chung, ”Architecture-Driven Embedded Systems Adoption for supporting Vocabulary Evolution”, *Proceedings of the International Symposium on the principle of Software Evolution (ISPSE’00},* 2000.

[8] P. A. Laplante, “Real-Time Systems Design arid Analysis - An Engineer's Handbook”, IEEE Press, 1993.

[9] J. sasiBhanu, A VinayaBabu, SastryJKR, P Trimurthy, “Dynamic Evolution of Syntax for Communicating with Embedded Systems”, *International Transactions on Electrical, Electronics and Communication Engineering,* Vol. 2, No. 4, pp. 36-43: 2012.

[10] J. sasiBhanu, A VinayaBabu, P Trimurthy, “An Efficient Architecture for implementing Syntax Evolution of Embedded Systems”, *Submitted for publication in Indian Journal of science and Technology,* 2015.