Journal of Theoretical and Applied Information Technology
20th August 2015. Vol.78. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT ARCHITECTURE FOR SEMANTIC EVOLUTION OF EMBEDDED SYSTEM

SMT. J. SASI BHANU[1], A. VINAYA BABU[2], P. TRIMURTHY[3]

[1] Assistant professor, Dept. of Computer Science and Engineering, KL University, Guntur, India

[2] Professor, Dept. of Computer Science and Engineering, JNTU, Hyderabad, India

[3] Professor, Dept. of Computer Science and Engineering, ANU, Guntur, India

E-mail: [1]sasibhanu@kluniversity.in, avb1222@gmail.com, profpt@rediffmail.com

ABSTRACT

Embedded systems (ES) are being used these days for monitoring and controlling Safety or mission critical system. Mission and safety critical systems as such will not be shut down for the reasons of safety and also due the cost of restarting. Changes are inevitable for any system either due to the reasons of updating the modules contained in the ES software or addition of new modules for the reasons of adding new functionality. The changes to the ES software required are to be undertaken online while the ES system is up and running. Ability to make changes to the software while the system is running is called dynamic semantic evolution. For any software architecture is the basis. Modules are to be reflected in the architecture that support dynamic evolution of the embedded software. Many architectures have been proposed in the literature that cater to the dynamic semantic evolution of loaded systems. Every architecture proposed incorporated only one method of undertaking the dynamic evolution. Many methods have been in existence for dynamically evolving the embedded systems but all the available methods have not be brought under the same architecture. In this paper a comprehensive architecture is presented that considers all available approaches that help implementing dynamic semantic evolution of the ES software.

Keywords: *ES Architectures, Dynamic Semantic Evolution, Monitoring And Controlling Safety Critical Systems, Software Evolution*

1. INTRODUCTION

Semantic evolution of a running software is dynamically updating a computer program, while it is executing. Dynamic updating is crucial in applications where the cost of stopping and restarting the program makes doing so impractical. Mission critical and safety critical systems such as nuclear reactor systems cannot be stopped for making changes and therefore the changes must be made while the system is running. Computer software changes constantly. Change may be necessary because new features were to be added to a program or because bugs were discovered in the current version or improvements to the existing program units are to be carried.

Sometimes the changes have to be carried while the system is running as bring down the system for want of making changes would be expensive. Examples of such systems include a airline reservation system or a telecommunications switching system or a computer-controlled life-support system or an air-traffic control system. The ability to dynamically update a program, i.e., load a new version of a program without stopping the currently running version, could alleviate the costs involved in making changes to a running program.

Prior approaches to the problem of replacing portions of computer programs without stopping them can be classified into three main categories which include hardware-based, service- oriented, and procedural based. Several strategies have been presented in the past to undertake dynamic evolution of the software. The techniques include hardware based redundant systems, abstract data type systems, client server based systems, Module based systems, Architectural based systems etc.

The architecture level based systems deals with dynamic evolution as the capability of modeling architectures in which the number of components, connectors, and bindings may vary when the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

software system is executed. Dynamism has also been defined as an aspect of configuring and allowing replication, insertion and removal, and reconnection of architectural elements either statically and dynamically. Support for dynamic reconfiguration at run-time is supported in terms of reconfiguration manager . A supervisor can send directives in a script language to the system that invoke dynamic reconfiguration. Every element can be either added, removed, or replaced dynamically but from external tools only. The operations for describing change include adding components, removing components, upgrading or replacing components, changing the architecture topology by adding or removing connections between components, altering the mapping of components to processing elements, querying properties of architectural elements, obtain versioning information and querying the current architectural topology. A typical change will require several modification operations.

Every system must be adaptable considering the application architecture or in the way the data is exchanged. The new software modules must be able to interact with the old software module considering the changes that might come up in the very programming languages using which the software is developed. When changes are made to the software it is also necessary to consider other issues that include fault tolerance, scalability and performance. The scalability of the Hardware and data must also be taken into account.

Specific architectures are required for implementing dynamic evolution of the embedded systems for that matter any type of a system. The architectural models that evolve at run rime are called dynamic and the models that do not get changed after the software is implemented are called static architectural models. Static architectural modeling notations do not support constructs that are required to express runtime change. Notations and tools must be added to describe run-time architectural changes. When the notations and the tools are added to static architectures they become dynamic architectures.

Dynamic architectures require formalisms and tools beyond those of static architectures. Adequate notational constructs are needed to describe runtime change, analysis tools are needed to help verify their unique properties, and runtime support libraries are needed to reduce the costs associated with their implementation.

The architectural modifications can occur during four periods of time which include design time, pre-execution time, constrained run-time and run-time. Changes made during constrained based run time are safe and generally do not disturb the ongoing process. When modifications are carried that cross architectural boundaries, system integrity may be lost. **M**echanisms that maintain system integrity in light of architectural modifications are needed. Modification constraints provide a means to specify limits on what aspects of an architecture may change. They may restrict change based on modification operation, particular components, modification time, or a combination thereof. They may also require that functional properties be verified before a change is committed. Constraints must be related to behavioral properties of the system and must allow trade-offs if all constraints cannot be met simultaneously.

Architectural languages are required to support run time changes. The existing architectural languages describe static architectures. Two additional descriptions are required which include modification language (AML) and constraint language (ACL) are also required. Architecture plays a vital role to define the way the dynamic evolution of the ES software is to be undertaken.

Thus many methods and several architectures have been in existence for undertaking the dynamic evolution of the loaded systems. But the limitation is that every architecture just supported only one method. Very few architectures and many methods have been presented which are related to dynamic evolution of embedded system. Every architecture that suits embedded systems also have been incorporated with just one method. Thus there is a requirement of presenting a comprehensive architecture that incorporates all the existing methods into a single architecture so that all aspects of dynamic evolution of embedded software can be undertaken.

2. PROBLEM DEFINITION

Embedded systems are quite frequently used for monitoring and controlling Mission and safety critical systems. The embedded systems are driven by a HOST which is located at a long distance connected through Internet. Embedded software keep changing either to improve response time, implement efficient sensing and actuating system, add more tasks, delete the existing tasks, improve the algorithm or processing implemented through

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

existing tasks etc. The embedded systems which monitor and control the mission or safety critical system cannot be shut down for want of making changes due to the nature of criticality attached to it. The changes to the ES software thus must be undertaken while the system is up and running. Any change required must be initiated from the remote host and the same has to be effected within the ES software without actually bringing the ES system down

Very few architectural model exists and even these few implements only one method of dynamically evolving the ES software. Many software components are to be added into the architectural models which are all required to support various methods. No architectures as such has been recommended ever that considers dynamic evolution under the ambit of an RTOS.

The main problem thus is to find the architectural models that can be implemented for the dynamic evolution of ES software. The architectural models should consider all the methods and the related components, tasks, processes etc., which are required to implement dynamic evolution of ES software. It is also necessary to consider all those components that are required for dynamic evolution of ES system that runs under the control of a real time operating system.

3. LITERATURE SURVEY

Lui Sha et. al., 1996][9] have presented a Simplex Architecture that considers upgrades to the existing systems using new technologies and ensuring safe, reliable, with negligible down time. The Simplex Architecture was originally developed to support safe online upgrades to a feedback control and radar systems regardless of the faults that might exist in the modeling, designing and implementation of the new software and hardware. The simplex Architecture is considered the evolution of the architecture itself while the system which is developed using the very architecture is evolved. The simplex architecture considered dynamic evolution by incorporating independent modules that ensures timing, fault tolerance through handling exceptions, monitor system status, configure the system as required and to provide the interface for the users to manage changes online. To manage the changes online, advanced real time resources management technology has been incorporated into the architecture. The simplex architecture considers, real time process management primitives.

Peyman Oreizy[4] presented that run-time evolution of the software require dynamic architectures that evolve as the system runs. The evolution is done online. Most of the systems require online updates without bringing down the system which is already running. Static architectures will not be able to cater to incorporate dynamic changes. End-user customizability and extensibility also requires runtime evolution.

Huw evans [3] have presented an architecture titled DRASTIC that supports run-time adaptability through implementation of run-time type changes and system configuration. The architecture uses a kind of abstraction called “ZONE” for effecting the evolution. DRASTIC divides the system into smaller and more easily managed sub-domains called zones. The change is encapsulated into the Zones and a change in one zone will not affect another zone. Zones to start-with are recognized at design stage and become explicit components at run time. Software in one zone is evolved autonomously from software in other zones, even though the source code may originally been shared by components in many zones. **Peyman Oreizy[4]** have presented an architecture-based approach based on which software evolution is undertaken through use of software connectors. The implementation of the architecture is undertaken through use of the tool called ArchStidio.

Software architectures D. E. Perr [8], M. Shaw [10] can provide a foundation for systematic runtime software evolution. An architecture-based approach to runtime software evolution has been proposed that includes an explicit architectural model, which is deployed with the system and used as a basis for change, preservation of explicit software connectors in the system implementation, and an imperative language for modifying architectures. A tool suit has also been presented that supports runtime software evolution at the architectural level. The architecture proposed by them considered various aspects related to change management that include change policy, change scope, separation of concerns, level of abstraction at which change must be carried and the type of change that must be made.

Peyman Oreizy [4] have presented an approach which uses architecture of the software as a basis. The architecture as such describe and reason about the systems behavior D. E. Perr[8], M. Shaw [10].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Change can be effectively managed by exploring the system level knowledge of the architect. Even the off-the-shelf components can be accommodated if no restrictions are imposed on the component internals. The change application policies are encapsulated into connectors which are independent of the functional components making it possible to separate change policy independent of the functional support.

Peyman Oreizy [4] have presented, the way the architectures support different types of software evolution and the circumstances which leads to making changes to the software. As such they have considered three characteristic type of evolutions that include corrective, perfective and adaptive. Software faults are removed through corrective evolution. Enhancement of the application functionality can be achieved thorough perfective evolution and adaptive evolution changes makes the software adapt to a new environment.

Many design styles are in existence using which components can be added to the running system. Observers design Pattern E. Gamma [5] allows addition of more observers with minimal impact. In the mediator design approach, new mediators can be introduced that maintain relationships between independent components. Design approaches that consider the implicit invocation methods D. Garlan [6] are more effective to add components at run-time. Invoking components are generally unaware of the other components running at the time of invoking. Methods have been added which help new components discover the state of the system and perform necessary actions to synchronize the new components with the ongoing state of the system. When new components are to be added, structural changes to be made to the architecture of the software must be specified. The changes to the structure some-times are implicit or derived from externally visible properties of the components.

Another approach, exemplified by the Simplex architectural style Gilsi Hjalmtysson [9], incorporates an "operational model" in the implementation. The model rejects upgraded components when they do not satisfy explicit performance and accuracy requirements. Many systems cannot tolerate loss of system state. In such cases the state of the system must be preserved during the change. More of the considerations are to be included in additions to those related to addition and removal of the components. Several methods have been proposed that preserve the state and communication when run time changes are to be undertaken.

Structural reconfiguration of the architecture supports recombining existing functionality to modify overall system behavior. Data-flow architectures, such as UNIX's pipe and filter style and Weaves M. M. Gorlick[7], provide substantial flexibility through static reconfiguration of existing behaviors. Runtime reconfiguration can be performed by altering connector bindings since connectors mediate all component communication.

Lawrence Chung[11] have presented how semantic evolution of a remotely controlled embedded system can be carried. They have presented a three tire architecture using which the semantic evolution of the embedded systems can be carried. The overall architecture proposed by them contains syntax evolution block, sematic evolution block and a communication interface. **Run-time module adaption** is a very powerful technique that allows the system to change its behavior in many different ways. New modules are generated at run-time to enable the system to adapt to the change in environment. Code generation algorithms are required in this case to generate code based on the commands received. Based on the command received, one of the module is linked. The module generation could be undertaken by taking instance of a template class or including the new class in the class library or by loading the class using the class loader.

A system is adaptable if an adaptation function exists. Adaptability then refers to the ability of the system to make adaptation. Adaptation involves three tasks which include ability to recognize, ability to determine the change to be made to a system, ability to effect the change in order to generate the new system. The adaption involves verification and validation to ensure the correctness of the modified system.

Dominic Duggan[12] has presented that hot swapping of running modules as one of the most important requirement to be supported for achieving dynamic evolution essentially within server based software based systems. Under such a system a new module be able to change the types exported by the original module while preserving the type safety. Type based approach of swapping running modules is the most important approach that got evolved over the time. Programmer

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

defined version adapters at run-time adds type sharing constraints to the type system.

Michael Hicks [13] has developed a dynamic evolution system considering flexibility that takes into account the time at which the change is effected, robustness considering type safety, completeness, well-timed, simplicity, rollback enabled, efficiency and ease-of-use. Dynamic evolution has been presented as practical and general purpose when compared to the other solutions presented in the literature which are specific purpose and application dependent.

Narayanan [14] have presented several architectures that can be used for evolution of vocabulary of the embedded system which is nothing but evolution of command language which is used to effect communication between the HOST and the Embedded System. GARP [15] has proposed an architecture that combines configurable hardware with a standard MIPS processor on the same die. The architecture can be used for configuring the Hardware in microseconds. The GARP compiler can implement instruction level parallelism (ILP) from C code and directly compile selected loops to the reconfigurable array.

M. Kau [20] proposed a SPARCS framework includes a synthesis tool that estimates system resources and latency. An Integer Linear Programming(ILP) model is formulated to solve spatial and temporal partitioning problems. This flow is complete and well defined and mostly dependent on traditional hardware. Sasi [19] have presented a basic architectural model that caters for syntax evolution of the embedded systems.

Many presentations have been made for dynamic evolution of loaded systems. Few contributions have been related to dynamic evolution of embedded systems. An embedded system can be evolved by using different methods which include, invoking the tasks which are in dormant stage at the run time, rule based self-adaptability, Online code generation, online simple update and online update considering the criticality of the update etc. Each dynamic evolution method presented in the literature looks into only one kind of dynamic evolution. It is necessary to comprehend different kind dynamic evolution methods that can be used and come out with a comprehensive model into which all kinds of dynamic evolution methods are incorporated considering the embedded systems.

4. INVESTIGATIONS AND FINDINGS

Several kinds of archliberal models can be designed which can be implemented for achieving dynamic evolution of the embedded systems. The architectural models that can be used include Architecture based on stored data model. Architecture based rule based model, Architecture based on code generation, Architecture based on code migration, architecture that considers code updating considering both critical and non-critical updates

4.1 Architecture Based on Stored Data Model

In the stored data technique all the code components are pre-identified for each of the change expected in the environment and appropriate code is used based on the state of the embedded system. The change in this case is to be pre-identified and components are to be provided and made available right in the beginning before reset of the micro controller. A state machine keeps track of current state of the system. The change in the environment is recognized as change of state of the embedded system. Every change is recognized as the state of the machine.

An embedded system can be considered to be in a state when a particular external or internal event is being executed. The execution of a Task which is a unit of code will be based on the occurrence of an event. The code units which are future additions or updates to the existing tasks are to be identified with pre-defined units. The commands to activate the new code units are pre-identified and when such commands are received the relevant Tasks are invoked. The tasks that are invoked will be made to be waiting for the occurrence of its related event.

Journal of Theoretical and Applied Information Technology
20th August 2015. Vol.78. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

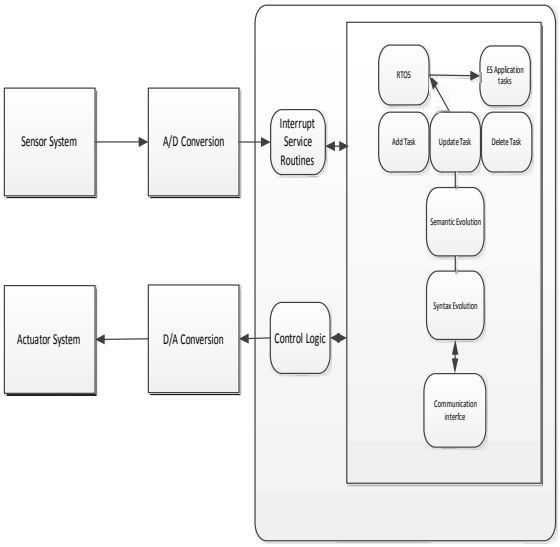

Figure 1 Semantic Evolution Based Stored Data

Figure 1 shows the architecture for the stored data model. In this architecture also all the components required for catering to the changed environment are to be identified and make available right in the beginning of the development of the system before the same are to be migrated to the Target System. The code elements that are not required or in dormant state initially and they are brought into the main stream based on the command that it receives. When a code element is initiated to be brought into the main stream, the address of the code element, event that triggers the Task is to be sent along with the command to invoke. Invoking is like adding new task. Chaining new task with other tasks is achieved through making the task to wait for a particular event.

Stored data means, the code elements which are pre-stored and made available in the address space of the entire ES application. The pre-stored code elements are only made to be active at run time. The stored data model is not a true dynamic evolution model even though code elements which are not active are activated at run time. A program structure is required to be followed such that all variables will be global variables.

4.2 Architecture Based on Rule Based Model

Rule based model recognizes a set of rules and the rules recognizes the changes that should be effected within the semantics of ES application. The component dealing with the rules will parse the rules set the environment data and invoke the set of tasks that comply and fulfil the rule. This approach is specifically suitable when control of the production system must be done based on the configuration data and the conditional logic. Fig 2 shows the architecture suitable for rule based adoption.

The remote HOST shall provide the command and associated rules to be adapted to syntax evolution model and the rules are handed over to the semantic evolution component. The semantic evolution component hands over the rules to be adapted to the rules manager. The rules manger will process the rules, parses the same and generates some internal events and the tasks that are waiting for those events are executed. The rules manager also sets the environmental data generally maintained as global variables. The rules are maintained in a lookup table and the table which is maintained dynamically through commands issued by the remote HOST. If an existing rule is the one to be adapted then the HOST just transmits the command and the name of the rule to be adapted. The control data required for the rules to be adapted sometimes gets acquired automatically through the sensors which triggers the interrupts required. In this case also all the Tasks must have identified and have been built along with ES application right in the beginning. Rules implements dynamic evolution to certain extent when overall functioning of the embedded systems must be changed based on rules fed from a remote location, the control data for which is either obtained through the sensors or transmitted from the remote HOST.

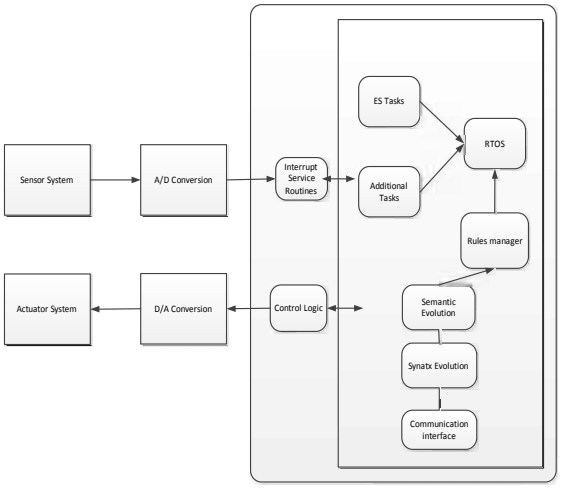

Figure 2 Rules Based Dynamic Adaption

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.3 Architecture Based on Code Generation

Runtime module generation is a very powerful technique that allows new modules to be generated at run time to enable the system to be adapted at run time. Generation of new modules is a complex process. Code generation is a time taking process and also it is difficult to generate efficient code based on the inputs fed from a remote HOST about the changed situation. Huge repository has to be built within the embedded system which is required for generating the code

Figure 3 shows the architecture for Run-Time Module Generation Technique. The details required for code generation are transmitted from the HOST and the same is maintained within the embedded system. Two processes are to be included into the architecture one for maintaining Code generation repository and the other for generating code. The memory module will make available the address space required for storing the code that is generated. After creating the code, a task is created along with the event for which the task must be waiting. The tasks that are related to Repository building, code generation and memory management are invoked by the semantic evolution modules triggering their related events.

4.4 Architecture Based on Moving Code from HOST

Yet another important technique is to receive the Task code through a command received form the HOST and copy the code to the respective address spaces and recognize the code separately identified with the Real Time operating system. This kind of method helps in dynamically creating new tasks without the necessity of any code generation. As such code developed at the HOST is migrated to the ES system.

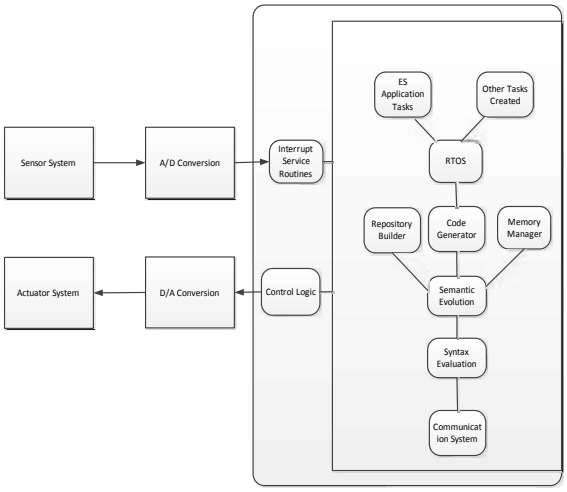

Figure 3 Semantic Evolution Based On Code Generation

Figure 4 shows the architecture for run rime copying and reconfiguring of the Application code. The remote interface receives the new code or modified code developed on the HOST as an input to a command and the copy component copies the code to the address space specified by the command. The location where the code must be written is obtained by the memory manager through calling the RTOS functions. The code may be sent from the HOST either for the purposes of creating new TASK in which case the code is copied and a Task is created under RTOS and the TASK is made to wait for an event to be initiated from the Semantic Evolution Task.

Code also can be transmitted as data to the Embedded system in respect of a TASK which needs to be updated. This code also can be copied to the address location returned by the memory manager. After copying the code another task can be created under the operating system. But the main issue is that the old task which is already scheduled and running must be deleted and the state information of the old Task should be transferred. The state information gets automatically transferred to the new updated task as both share the same global variables. If the code is modified the task is killed and the task is created within the real time operating system if the task does not deal with any of the control function. The priorities with which the code must be activated are set at the time of creating the task under the control of the operating system.

Journal of Theoretical and Applied Information Technology
20th August 2015. Vol.78. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.5 Dynamic Task Updating

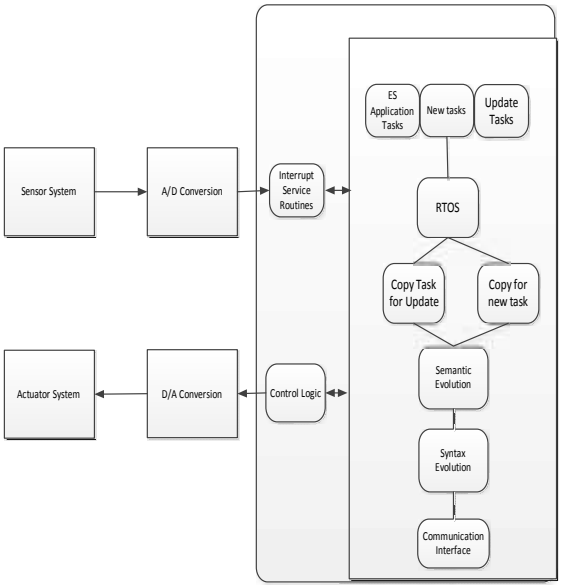

Figure 4 Run Time Module Copy And Reconfigure Architecture

The main issue is to determine the time at which the old task shall be deleted. If the tasks are mundane tasks, the deletion of the old task is straight forward. If the Task to be deleted is a Task that undertakes an actuating / controlling function, then there should be a guarantee that the new task will do the actuating function exactly in the similar manner as the old task or else addition of new task and deletion of the existing task will make the entire system to get jeopardized. To avoid this a check must be done by a different task and the task will have the logic which compares the output of new task with old task over a predefined period of time. If the output is same over a period of time, the old task is deleted and the new task will take over the new task using its output for handling the control mechanism. If the output is not same, a message is sent to the HOST and the new update Task shall be deleted and the memory occupied by it will be released.

[**Sha et. al, 2001**] have proposed an architecture for online update of real time embedded system which is termed as simplex architecture. The Figure 5 shows the simplex architecture for online update of an ES Application.

In this architecture all critical tasks are performed by simple and verifiable components and the output of new components (Complex Components) which are counter parts to the simple verifiable components is fed as input to the simple components.

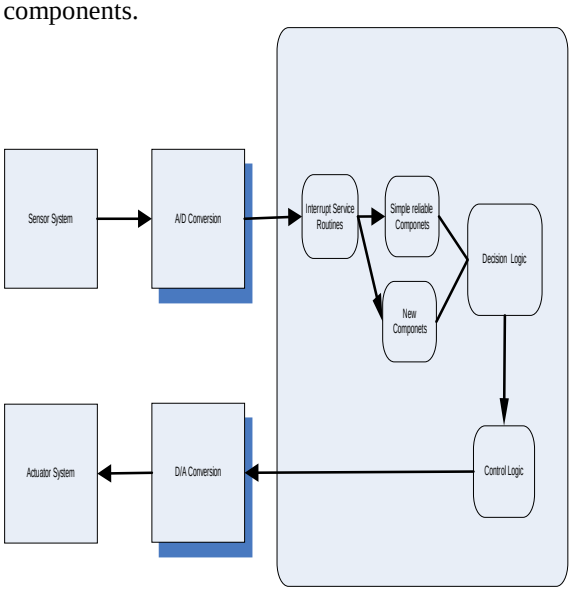

Figure 5 Simplex Architecture For Online Updates

The new components may not have been tested fully. The input data is fed to the Micro Controller by an A/D converter which communicates using a specified communication protocol such as I^2C. An interrupt service routine is activated to read the data and place in a shared memory. The memory is accessed by the simple reliable task and the Complex new task as soon as the data is placed in the shared memory. The tasks shall be waiting for the arrival of the data. The output generated by both the tasks is fed to Decision logic. The architecture includes a Decision and Logic Component that tests the output from the complex component and also checks whether the system shall be in the recovery region once the output from the complex system is released to the rest of the system. The Decision logic components keep track of the recovery region of the application in terms of the peripheral and memory boundary values of various data elements, the timing requirements, the sequencing requirements etc. The output from the complex components shall be verified with safety critical region variable values. If the output produced by the complex region is within the limits of the safety of the systems, the output shall then be allowed to be committed to the safety memory area of the application. If the output of complex components are not confirming to the safety region requirements of the application then the output of the simple component shall be considered and the output is subjected to the critical region.

The decision and logic system sufficiently checks the upgrade of the simple components and on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ensuring the correctness and efficiency shall configure the new component as the simple components. The architecture includes a runtime reconfiguration subsystem which shall have the functions to carry the switching between the testing mode and the normal operation mode and commit the upgrade of sufficiently tested software.

[Kihwal Lee, 2005] has extended the simplex Architecture to carry online upgrading of the Embedded Application and proposed eSimplex architecture. Lee proposed that the simplex components and the Complex components be resident in different address space. Soft real time processes and non-critical processes also run in different address space. The communication between all the processes is achieved through shared memory and communication wrapper where necessary. **Figure 6** shows the interaction of components through shared memory to achieve online upgrade of the system.

ESimplex uses the priority based scheduling of the processes and helps the isolation of processes in the timing domain. The process level isolation is achieved through process level protection offered by the operating system. The memory protection and isolation is achieved through static compiler checks. [Lee et. al., 2005] proposed code safety without runtime checks for control system through usage of operating feature of Process protection. The replacing of the simplex code with Complex components and restarting of the application is achieved through the concept called Process Resurrection (PR). PR is a fast and efficient mechanism for restarting and replacing a process. PR mechanism is used to switch between the production mode and testing mode.

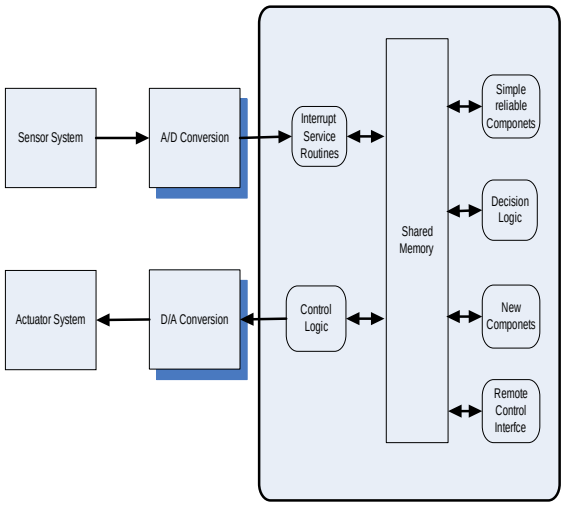

Figure 6 Online Upgrades Through Shared Memory

The new component image can replace the simple component and the mode can be changed from Testing to Production. In the production mode only the verified and tested components will be made to run. The runtime sub system must be able to reconfigure the code and switch between the production and testing in real time meaning the activities related to reconfiguration and switching must be predictable and schedulable. Figure. 7 illustrates the usage of the concept called process Resurrection. Process resurrection is a feature supported by most of the real time operating systems. The mode switch is achieved through the process Resurrection function which will map the new code to the address space of the simple reliable component address space and also setting the control data a value realized by the control logic to indicate that a mode switch has been effected.

The system runs in the normal operation mode when there is no need for testing or upgrade. In this mode only simple reliable functional modules shall be running. The switching overhead is very negligible and as such there is no CPU overhead and little extra storage is used to store the data related to changing the modes between productions and testing & upgrading. When online testing of new software is needed the PR feature converts the regular processes into Esimplex enabled processes and the new software modules can be uploaded for testing. When the new software is unproven and its features are needed then the system can be made to run at the cost of reduced response time of non-real time and non-critical tasks. When the new modules are believed to be running then the system is switched to normal mode of operation through the Process Resurrection.

The updating methods considering the criticality of the tasks, simple tasks and complex tasks proposed by Lee, Sha and Kowshik are quite complicated and does not fit into the overall structure of dynamic semantic evolution of the embedded system. The process of updating must be event driven. Both the old process and the new process must be made to be running and the new task will be made to continue to run while old task will be deleted when sufficient guarantee could be achieved that the new and updated task will meet all the criticality conditions. Figure 8 show the new architecture that deals with simple comparator that runs under the control of a Real time operating system. Both will share the same memory two sets of global variables which will mirror each other. Changeover of the tasks will only require swapping of the variable. When the changeover is undertaken

Journal of Theoretical and Applied Information Technology
20th August 2015. Vol.78. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the original set of global variables will only be used for undertaking all the control actions

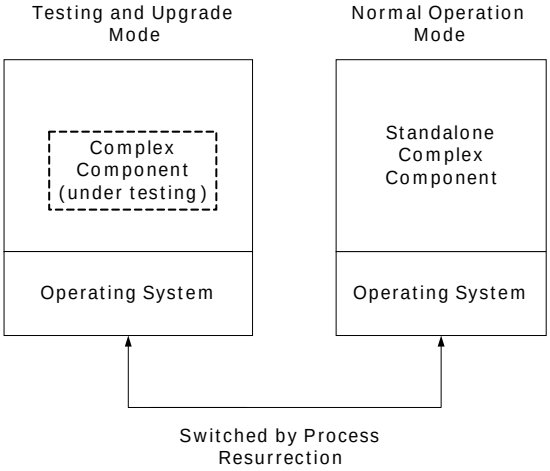

Figure 7 Process Resurrection For Mode Change Of The Embedded Systems

Copy task for update component will trigger an event for which the compare task will be waiting. The compare Task will take the charge of making the tasks (**old and the modified**) run simultaneously each writing the output to the mirror set of variables. The verification is undertaken after completion of each cycle of execution of the old Task. A task is considered to complete a cycle of execution when the task moves from Blocked state to run sate and then to blocked state again.

4.6 Comprehensive Architecture

All the methods discussed above help in semantic evolution of the Embedded Software in one way or the other. The Application code changes when the methods modify the existing code or when new code is created due to the reasons of environment changes and the reasons for adoptability. In the case of mission critical and safety critical system the production system cannot be shut down for want of making changes to the embedded system which helps in monitoring, and controlling of the embedded system. It is necessary that changes by way of modifications or additions must be done at the runtime meaning that when the embedded system is in ruining mode. The modified code and the new code can be developed in the host machine and then have to be moved to the target machine in online mode while the embedded system is in running mode.

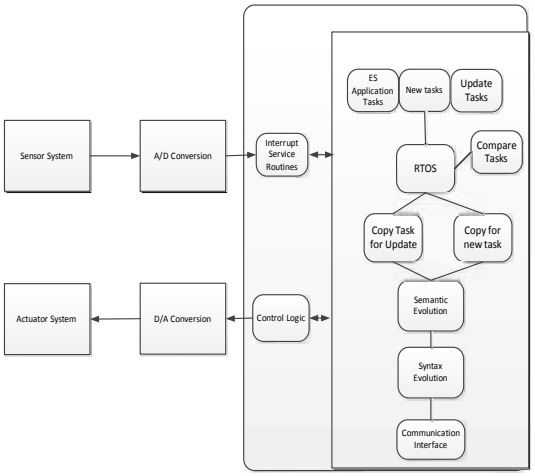

Figure 8 Architecture For Updating The Critical Tasks

The architectures presented above have merit in each of them. It is necessary to consider all the architectures and provide a unified architecture and the kind of dynamic evolution to be adapted should be decided by the semantic evolution task. The comprehensive architecture that caters for all types of dynamic evolutions of embedded systems is shown in the Figure 4.9. Dynamic semantic evolution of the embedded system thus can be achieved through Data Approach (Invoking and deleting the existing Tasks as per the desired functionality), Rule based Approach (Invoking the task execution as per the rules for which a repository can be maintained), Module generation and invoking approach (Generating new modules based on HOST sent specification), Module copy approach (Creation of the new modules as per the code sent from the remote HOST), Module update approach (Creation of update module and swapping the old module with the Update module), Module update with attached criticality assessment Approach (Create update modules and subject the module for criticality assessment).

The overall architecture is efficient that it accommodates any kind of evolution both simple and complex evolution systems. It implements all kinds of approaches using which all possible types of evolutions can be undertaken. The type of evolution that must be taken up can be dictated from the HOST through transmission of appropriate command. Semantic Evolution block can implement the kind of semantic evolution that needs to be implemented as per the request initiated from the HOST.

Journal of Theoretical and Applied Information Technology
20th August 2015. Vol.78. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

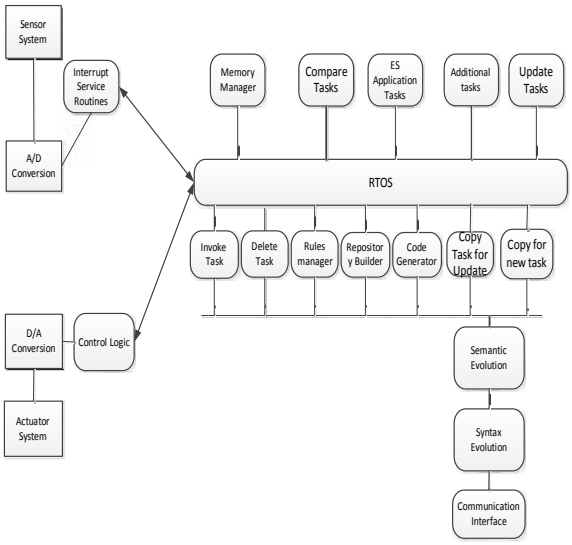

Figure 4.9 Overall Semantic Evolution Architecture

5. COMPARATIVE ANALYSIS OF ARCHITECTURAL MODEL

A comparison is made to show the coverage of dynamic evolution by different architecture proposed by different authors. From the **Table 5.1** it could be seen that the comprehensive model presented by Sasi et. al., is most effective and extremely suitable for dynamic semantic evolution within an embedded system which is interfaced with a remote HOST.

6. CONCLUSIONS

Embedded systems that are used for monitoring and controlling safety or mission critical systems must be evolved dynamically for incorporating the changes to the running systems as it is not possible to shut down the ES system for want of making changes. Many architectural models have been presented in the literature and most of them are not quite suitable for implementing dynamic evolution of embedded software and as such each architectural model has been incorporated with just one method while many methods have been in existence for dynamically evolving the embedded software. Few of the architectural models have been presented that have incorporated just one method for dynamically evolving the ES software. A comprehensive architectural model that supports different methods and runs under the ambit of an RTOS that includes all components that are required for dynamic semantic of the ES software is needed and the same is presented in this paper.

Further research needs to be conducted for dynamically semantic evolution of distributed embedded systems.

REFERENCES

[1]. Lui Sha, Ragunathan Rajkuman, and Michael Gagliardi, "Trans. Software Engineering, *CMU technical report CMS/SEI-95-TR-005*, 1995

[2]. Peyman Oreizy, "Issues in the Runtime Modification of software architecture, *Technical report submitted to university of California – Irvine*, 1996

[3]. Huw Evans and Peter Dickman, ``DRASTIC: A Run-Time Architecture for Evolving, Distributed, Persistent Systems", *Lecture Notes in Computer Science*, *vol.* 1241, pp. 243 – 249,1997

[4]. Peyman Oreizy, Nenad Medvidovic Richard, N. Taylor, "Architecture-Based Runtime Software Evolution", *Proceedings of the International Conference on Software Engineering*, 1998

[5]. E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns", *Addison-Wesley*, 1995

[6]. D. Garlan, G. E. Kaiser, D. Notkin, "Using tool abstraction to compose systems", *IEEE Computer*, Vol. 25, Iss. 6, pp. 30-38, 1992

[7]. M. Gorlick, R. R. Razouk, "Using weaves for software construction and analysis Proceedings of *13th International conference on software engineering*, 1991

[8]. D. E. Perry, A. L. Wolf, "Foundations for the study of software architecture", *Software Engineering Notes*, Vol. 17, Iss. 4, 1994

[9]. L. Sha, R. Raj Kumar, M. Gagliardi, "Evolving dependable real-time systems", *IEEE Aerospace Applications Conference*, New York, pp. 335-446, 1996

[10]. M. Shaw, R. DeLine, D. V. Klien, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for software architecture and tools to support them", *IEEE Transactions on software engineering*, pp. 314-336, 1995

[11]. Lawrence Chung, Nary Subramanian, "Architecture-Based Semantic Evolution: A Study of Remotely Controlled Embedded Systems", *IEEE International Conference on software Maintenance*, 2001

[12]. Dominic Duggan, "Type-Based Hot Swapping of Running Modules", *Intl. Conf. on Functional Programming*, pp. 62-73, 2001

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[13]. Michael W. Hicks, Jonathan T. Moore, and Scott Nettles. "Dynamic Software Updating", *In SIGPLAN Conf. on Programming Language Design and Implementation*, pp. 13-23, 2001

[14]. Narayanan (Nary) Subramanian and Lawrence Chung, "Architecture - Driven Embedded Systems Adaption for supporting Vocabulary Evolution, *IEEE explorer reference 0-7695-0906-1/01 IEEE*, 2001

[15]. J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable co-processor", *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 12-21, 1998

[16]. Lee K and Sha L, "Process Resurrection: a fast recovery mechanism for real-time embedded systems, *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005

[17]. L. Sha,, D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The Simplex Architecture for Safe On-Line Control System Upgrades", *Proceedings of the American Control Conference*, 1998

[18]. Kihwal Lee and Lui Sha, "A Dependable Online Testing and Upgrade Architecture for Real-Time Embedded Systems", *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005

[19]. Sasi J, Sastry KR Jammalamadaka, Rajasekhar Rao K, "Architectural models for syntax evolution of safety critical embedded systems, *International Journal of systems and technologies*, vol. 1, iss. 2 pp. 103-113.

[20]. M. Kaul, R. Vemuri, S. Govindarajan, and I Quaiss, "An automated temporal partitioning and loop fission approach for FPGA based reconfigurable synthesis of DSP applications", *Proc. Design Automation Conference*, pp. 616-622, 1999

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 5.1 Comparative Analysis Of Architectural Models Related To Dynamic Evolution

Serial Number	Author	Main theme	Code Invocation	Rule Based	Code gen
1.	Lui Sha	Adding Independent Modules			
2.	Peyman Oreizy	Online Update			
3.	Huw evans	System Configuration	√		
4.	Peyman Oreizy	Software Connectors	√		
5.	D. E. Perr	System Connectors	√		
6.	M. Shaw	System Connectors	√		
7.	D. E. Perr	System Level Knowledge		√	
8.	M. Shaw	System level Knowledge		√	
9.	Peyman Oreizy	Theoretical evolution			
10.	E. Gamma	Mediators approach			
11.	D. Garlan	Adding Components at run-time			
12.	Gilsi Hjalmtysson	Addition of performance and optimization methods into the new components			
13.	M. M. Gorlick	Run time reconfiguration			
14.	Lawrence Chung	Run-Time module generation			√
15.	Dominic Duggan	Hot swapping			
16.	Michael Hicks	General purpose dynamic update having many features			
17.	Narayanan	Vocabulary Evolution			
18.	J. R. Hause	Configurable Hardware			
19.	M. Kau	Spatial Portioning			
20.	Lee, Sha and Kowshik	Updating considering Criticality			
21.	Sasi	Basic Evolution Architecture			
22.	Sasi	Comprehensive Evolution	√	√	√