

A Control Theory Based Framework for Dynamic Adaptable Systems

João W. Cangussu
cangussu@utdallas.edu

Kendra Cooper
kcooper@utdallas.edu

Changcheng Li
changcheng@utdallas.edu

Department of Computer Science
University of Texas at Dallas
Richardson-TX 75083-0688, USA

ABSTRACT

The increasingly complex environments in which systems need to execute has led to the need for tools and techniques to systematically design dynamically adaptable systems. A new framework for the design of these adaptive systems is proposed here. The framework, named SMART (State Model Adaptive Run Time), is based on the mathematics of control theory and system identification techniques. This foundation allows the system to accurately predict constraint violations in the environment, such as a memory overflow, and avert them by selecting components that better utilize a particular resource. The result is a more robust system. An example of the application of SMART is presented to show the need and applicability of the framework. Experimental results demonstrate the framework has an 86% accuracy in predicting and averting memory constraint violations. These results indicate the SMART Framework is feasible and has the potential to be a useful design solution for dynamic adaptable systems. Improvements to the framework are proposed as future work.

Categories and Subject Descriptors

D.2.10 [Software Engineering]: Design; D.2.11 [Software Engineering]: Software Architectures—*Domain-specific architectures*

General Terms

Design

Keywords

Run time adaptable systems, control theory, system identification, component based architecture

1. INTRODUCTION

The environments in which software products are executing today have considerably increased in complexity as software

needs to support a large number of simultaneous users on a variety of distinct platforms; each platform has different resource constraints such as memory, disk space, bandwidth, etc. The dynamic interaction among these elements constitutes the basis for this complex environment. A question arises from this scenario: “How can we design software to cope with such dynamic environments?” Recently, dynamic adaptive systems have been proposed as a solution to this problem. A dynamic adaptive system is one in which the system adapts itself at run-time according to changes in the environment [1].

When designing a dynamic adaptive system, new issues arise. For example, some of the inputs may not be observable such as the amount of unused memory available, the behavior is non-deterministic resulting in executions that are unique, the monitoring and control of the system occur continuously, and the adaptation of the system is at run-time rather than at compile-time [2].

One approach to the design of adaptive systems has focused on resource allocation, such as an operating system scheduling processes [3], QoS guarantee [4], and fault tolerance [5]. However, many systems are now requiring the adaptation based not only on resource allocation but also on the switch to alternative design choices that cope better with resource constraints. The expectation that the same application can be executed on very distinct platforms exemplifies this need. For example, web applications are now having to handle the problem of being used not only on personal computers, but also on cell-phones and personal digital assistants [6]. The capacity of these platforms range drastically from very powerful workstations to low memory, low performance pocket devices. The {re}allocation of resources, performed by the operating system, does not appear to be a solution under such distinct scenarios. For example, a real-time operating system cannot do anything if an application requires 10MB of memory and the total memory on the system is 5MB. An adaptive application should be able to identify such environmental constraints and swap to an alternative, though functionally equivalent, component that can be executed under the current conditions. The same adaptation approach can be used when run-time bottlenecks are predicted or detected. In addition, robustness can be improved if some properties of the alternative choices can be guaranteed, such as the maximum memory required under any scenario.

In this work we propose a new approach to the design of adaptive systems called the SMART (State Model Adaptive Run Time) Framework. SMART innovatively integrates two areas of leading edge research in software engineering. The first is the use of feedback control theory to support the continuous monitoring and control of the system in a dynamically changing environment. The second area is the effective specification, matching, and selection of off-the-shelf (OTS) components. SMART provides a knowledge-based repository of components. The functional and non-functional behavior of the components are rigorously specified in an extensible markup language (XML); the system can run XML queries that identify components. Once identified, the components can be swapped into the system to better meet the needs of the users.

This paper is organized as follows. An overview of control theory background is presented in Section 2. The objectives of the SMART Framework are presented in Section 3.1; the framework is described in Section 3.2. An example is used to illustrate and empirically evaluate the SMART Framework in Section 3.3. After the presentation of related work in Section 4, the conclusions we have drawn regarding the SMART Framework and future work are delineated in Section 5.

2. CONTROL THEORY BACKGROUND

Linear state feedback models have provided useful representations for large classes of engineering, biological, and social processes [7, 8]. In this section we present concepts and definitions of state models and system identification techniques that are the most relevant to our approach.

The general format of a LTI (Linear Time Invariant) state model is presented below:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \quad (1)$$

where $x(t)$ is the state vector representing the dominant variables characterizing the process; $u(t)$ is the input signal; $y(t)$ is the output/measurable variables; and A , B , C , and D are the coefficient matrices.

A model capturing the behavior of a system must be available if such system is to be controlled. A model that can capture the dominant behavior of different aspects of any adaptive system is unlikely to be statically created due to the variety of scenarios and environments where these systems execute. However, based on the observation of the behavior of a system executing in a specific environment a model can be dynamically customized for this scenario. Ljung states “System Identification deals with the problem of building mathematical models of dynamical systems based on observed data from the system.” [9]

An adaptive system can be designed in a such a way that any or some specific inputs/outputs are shared with other applications at the same time environment resources are monitored. Under this scenario, the ingredients to apply System Identification techniques are present and a model can be inferred from the observations. A number of techniques, such as Least-Square and Markov Parameters [9], are available to identify state space models. In order to implement a fast

prototype of SMART, the Least-Square identification procedure available in MATLAB was chosen; a concise description of this technique is presented below.

Let A , B , C , and D be the matrices in Eq. 1 and organize them as indicated below.

$$Y(t) = \begin{bmatrix} x(t+1) \\ y(t) \end{bmatrix} \quad \Theta = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \Phi(t) = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \quad (2)$$

Using the definitions of $Y(t)$, Θ , and $\Phi(t)$ above, we can rewrite Eq. 1 as $Y(t) = \Theta\Phi(t)$. The observation of the system resources and the inputs/outputs of the application are used to construct the vectors $Y(t)$ and $\Phi(t)$. Then, a least square approach as in Eq. 3 can be used to compute an approximation ($\hat{\Theta}_N^{LS}$) for the matrix (Θ).

$$\hat{\Theta}_N^{LS} = \left[\frac{1}{N} \sum_{t=1}^N \Phi(t)\Phi^T(t) \right]^{-1} \frac{1}{N} \sum_{t=1}^N \Phi(t)Y^T(t) \quad (3)$$

The matrix $\hat{\Theta}_N^{LS}$ represents the approximation for the coefficient matrices A , B , C , and D in Eq. 1. They form the state model capturing the dynamics of the system under consideration. The state model is then used to predict and control the behavior of the system.

3. THE SMART FRAMEWORK

The SMART Framework is based on the use of control theory to manage the adaptation issues when resource constraints and multiple design choices are available. A linear approximation of the system resources and the application is expected to capture their dominant behavior. This approximation has been proven to be general enough for such representation [4]. After a model is specified, feedback control can be applied to predict and regulate (select alternative design choices) the behavior of the system.

3.1 Objectives

The overall goal of the SMART Framework is to provide an environment for the design of dynamic adaptive software which allows the use of multiple solutions based on system resources and constraints. The components used to build the alternate solutions may be available when the system is launched or included while the system is running. In either case, the swap from one set of components to another is done at run time. The assumption here is that the selection of a set of components is made that better uses the available system resources of interest (e.g., CPU, memory, bandwidth, etc) and therefore improves the overall quality of the system.

To accomplish this goal, the SMART Framework integrates two areas of leading edge research in software engineering. The first is use of feedback control theory to support the continuous monitoring and control of the system in a dynamically changing environment. Although feedback control theory is well-known in a variety of engineering areas, its application to software engineering is currently being investigated [7, 10]. The second area is the effective specification, matching, and selection of off-the-shelf (OTS) components. SMART provides a knowledge-based repository of components. The functional and non-functional behavior

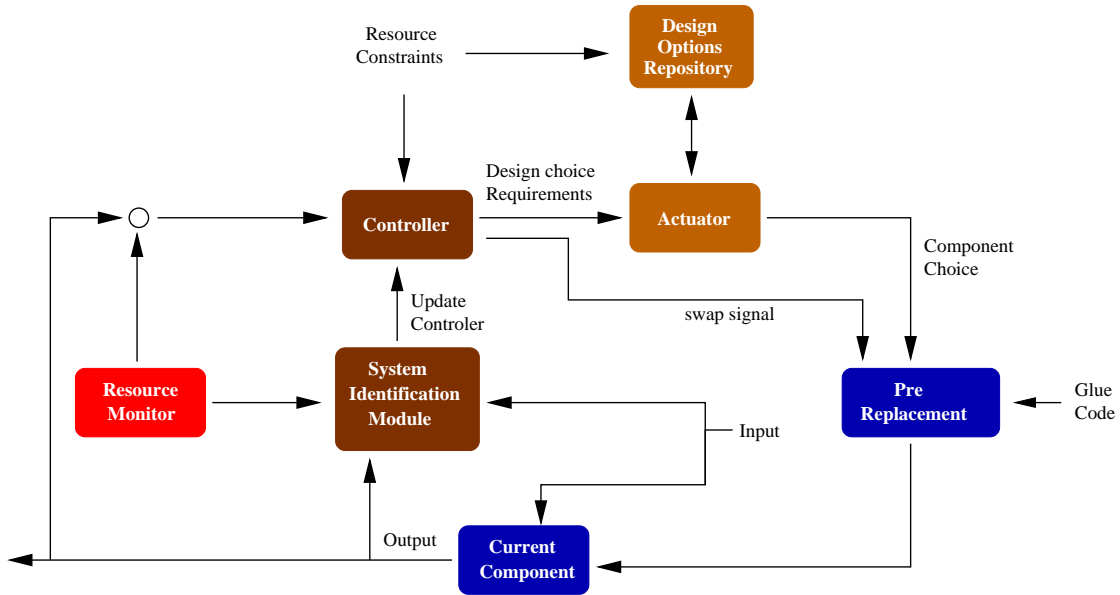


Figure 1: Overview of the SMART Framework architecture.

of the components are rigorously specified; the system can run queries that identify components that better utilize a resource. Once identified the components can be swapped into the system to better meet the needs of the users. If these components are not yet integrated/loaded into the system, then the SMART Framework needs to make use of dynamic programming languages such as Java or Dylan. The integration of these two areas of research are reflected in the architecture of the SMART Framework.

3.2 The SMART Framework Architecture

As shown in Figure 1, the architecture of the SMART Framework is composed of the design option repository, actuator, pre-replacement, current component, system identification module, resource monitor, and controller. The architectural components and their relationships are described below.

Design Option Repository: this repository contains the components which are available to provide alternative design solutions. For example, if a system needs to multiply large sparse matrices, then two component choices may be available: (i) a fast but high memory usage approach, and (ii) a slow but low memory usage approach.

Each component is formally specified to describe its functional properties, non-functional properties (e.g., CPU usage, memory usage, interface requirements, etc.), its composition (a component may be composed of one or more components), and the location of the code in the system. When the actuator queries the repository, the algorithm to determine the best possible matches also uses the resource constraints specified for the system.

The formal notation used to define the components is the Extensible Markup Language (XML) [11]. The advantages of using this notation include it is straightforward to learn

and tool support such as parsers, editors, and browsers is available.

Each alternative solution (i.e., a collection of one or more components) must be smoothly incorporated to the system and therefore each component must be designed such that it can be integrated with the whole system without recompilation and relinking. Techniques such as structural conformance, delegation, and wrapping may be used to achieve this goal [12].

System Identification Module: in order to use control theory [8] one needs a mathematical model (i.e., a system of equations) that relates the inputs and outputs of the system. Since the components and their interactions effect the systems resources and, as a result, a specific model is unlikely to accurately capture the dominant behavior of all the components. However, system identification techniques [9] allow for the dynamic determination of tailored models for a specific component/system.

This part of the SMART Framework is similar to the work done for autonomous agents [13]. By observing the output from the Software Product (component) and the systems resources provided by the **Resource Monitor**, the **System Identification Module** relates them to the input and creates a state model (**Controller**) that captures the dominant aspects of this relationship. The precision of the model increases as more data (input, resources, and output) become available and the **Controller** is updated.

Controller: the controller predicts when a bottleneck constraint is expected to occur and determines the latest possible binding time to correct the problem. This prediction allows for the minimization of the overhead costs associated with component switch. The controller uses the *resource constraints* and the current state of the system to make this prediction.

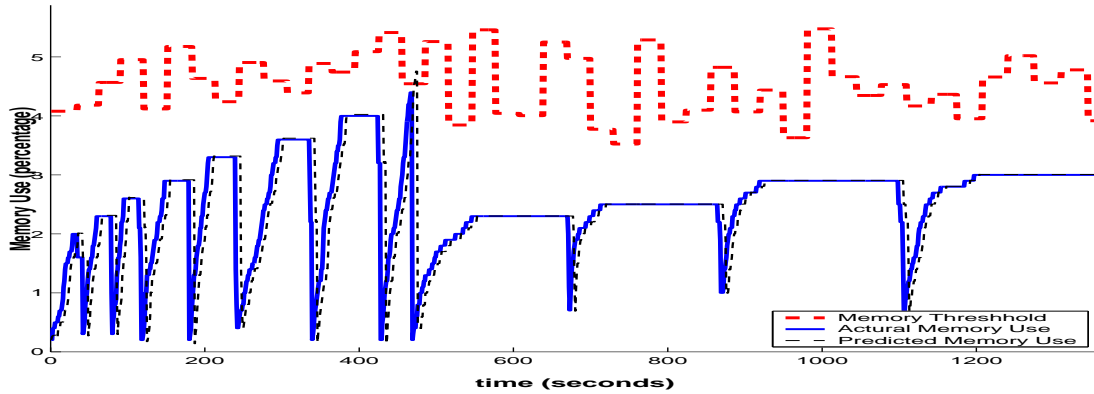


Figure 2: Scenario I. Memory Usage vs. CPU Time for C++ Components

Resource Monitor: the resource monitor represents an auxiliary tool that monitors the resources of the environment and sends this information to the **Controller** and the **System Identification Module**.

Current Component: this block represents the current design choice that is running. It receives the input and passes information (output) to the **Controller** and the **System Identification Module**.

Actuator: the actuator selects one of the design options provided by the design option repository. The selection algorithm uses the desired system requirements from the controller that should be met to achieve the system's integrity, performance, and consistency. The notation used to query the **Design Component Repository** is XML SQL [14].

Pre-Replacement: this pre-replacement module prepares the component selected by the actuator to replace the current one. It inserts any necessary glue code and links it to the component. The **Pre-Replacement** then waits for a signal from the controller determining the exactly time when the swap should occur. This technique improves the performance by preparing the component in advance.

3.3 Application and Evaluation of the SMART Framework

Based on the description of the SMART Framework in Section 3.2 one could think that its application is restricted to systems with severe resource constraints such as embedded systems. The following example demonstrates that the need for an adaptive framework like SMART exists whenever there are constraints associated with design choices.

In the example, a system needs to perform a sequence of square matrices multiplication; the matrices are highly sparse (i.e., 5%). There are two components available to perform the multiplication of the matrices:

- C_1 : this component is designed to have high performance (i.e., it is fast) but demands high memory usage. It is implemented by allocating a two dimensional array for each matrix.

- C_2 : this component is designed for low usage of memory and therefore does not have good performance under certain scenarios. It is implemented by allocating a one dimensional array of pointers for each matrix; these pointers reference a linked list of non-zero elements for each row in the matrix.

A tradeoff between memory usage and performance can be inferred from the description of the components C_1 and C_2 .

A sequence of experiments have been conducted as a first step in the validation of the SMART Framework. Scenarios with a variety of sequential and concurrent launches of components C_1 and C_2 have been analyzed [15]. Due to space constraints, only the results of the scenarios described below are presented here. These scenarios are designed to measure the memory, CPU time, accuracy of the framework, and the impact, if any, the implementation language (C++, Java) has on the behavior of the components.

The expectation is that the use of C++ makes the trade-off between memory use and performance more distinct while still allowing runtime adaptation. That is, the loaded components can be used for runtime adaptation but the insertion of a new component requires recompiling and linking the application.

The alternative use of Java as the programming language makes it possible to insert a new component at runtime through its dynamic loading properties. However, the exact impact of the Java garbage collector on the accuracy of the framework is unknown. The expectation is that some delay in releasing the unused memory results as memory usage increases for component C_2 .

Experimental Scenario I (C++)

In the first scenario, the components are implemented in C++ and are executed sequentially. The example system, launched as a single process, initially executes component C_1 by default. The system performs 20 iterations of the following calculations: the system multiplies a sequence of square matrix multiplications of order n , where n ranges

from $n = 400, 430, 460, \dots, 670, 700$; this sequence of calculations is repeated three times.

The use of system identification techniques in the framework to produce a linear state space model for changing environments allows the prediction of the future behavior of the application. A horizon of seven steps ahead is used to predict the behavior of the components. The constraint on the system that needs to be monitored and controlled is the memory utilization. Here, the constraint is that the memory use must not exceed a specified threshold of memory usage.

As the system is executing, the framework’s prediction capabilities are used to make the change to execute component C_2 . The change from one component to another may occur at any time. If a prediction that the threshold constraint will be violated, the current multiplications are aborted for component C_1 and start over using component C_2 . It should be clear that robustness and not performance is the major goal for this scenario.

Additionally, to make the experiment more realistic, the memory threshold changes randomly at runtime ranging from 3.5% to 5.5% of overall memory use. This changing threshold emulates an environment in which processes are being continuously created and terminated.

Figure 2 show the results for the first scenario where it can be seen that the threshold is changing dynamically (this is the dashed red line near the top of the Figure). The framework’s predictions of memory usage are in the dashed black line; the actual memory used is in the solid blue line.

An interesting result demonstrating the prediction of a memory constraint violation, swapping from component C_1 to C_2 (which uses less memory), and the resulting aversion of the constraint violation near the time $t = 470$ seconds is shown in Figure 2. We zoom in on this region of time and illustrate it in Figure 3.

At time $t=470$, the actual memory use is 4.3%; the prediction made is that the system is going to (in seven steps) violate the constraint of crossing the threshold of 4.5% of memory usage. At this point, C_1 is swapped out for component C_2 . As a result, the threshold is not exceeded and the system complies with its constraint. An increase in the robustness of the application can be inferred from these results.

As the 20 iterations of the matrix multiplication sequence are executed, an average of 9 switches between components C_1 and C_2 is measured. The change to an alternative component has prevented memory overflow according to the threshold constraint in 86% of the cases. The cases that are not successful may be due to a delay in the prediction, most likely caused by the overhead of running the Matlab environment and the monitoring process. This problem is expected to be solved when a more efficient monitoring algorithm is implemented. Currently, the interface between the MatLab scripts and the monitoring process is implemented with input files, an inefficient solution. The overhead caused by Matlab can also be significantly reduced by generating C code using the embedded C compiler in Matlab. These issues

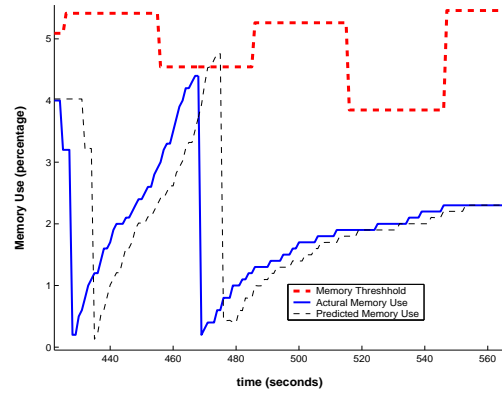


Figure 3: Scenario I. Time Window Demonstrating Prediction, Swapping, and Aversion of Memory Constraint Violation

are currently being investigated and a complete evaluation of the overhead incurred by the control and the monitoring processes are deferred to future work. Despite of the delay observed in the prediction process, the 86% of success is a good indication of the potential of the framework proposed here.

Experimental Scenario II (Java)

The purpose of this scenario is to measure memory usage and CPU time for the components and allow a comparison of the behavior of these components, written in Java, with the components written in C++.

In the second scenario, the components C_1 and C_2 are implemented in Java and are sequentially executed. The following calculations are performed: the system multiplies a sequence of square matrix multiplications of order n , where n ranges from $n = 300, 430, 460, \dots, 580, 610$; this sequence of calculations is repeated four times. The range is reduced in comparison to the previous scenario, in anticipation of a large amount of CPU time expected to execute the matrix multiplications.

The memory use of the Java components yielded interesting results. In Figure 4(b), the memory use associated with C_2 oscillates according to the order of the matrices. In contrast, the memory use tends to stabilize at a certain level when it is associated with component C_1 .

In addition, when the components are implemented in Java instead of C++, the memory used by C_1 does not drop even when repeating the cycle by re-starting with matrices of order 300. The Java garbage-collector, even when explicitly invoked, does not appear to free all the unused memory for component C_1 . The same is not true for component C_2 where the garbage collector seems to be more effective, perhaps due to the dynamic allocation of memory.

The performance of the JAVA components, presented in Figure 4(a), demonstrate the Java version of C_1 uses approximately 16 seconds to calculate a matrix multiplication of

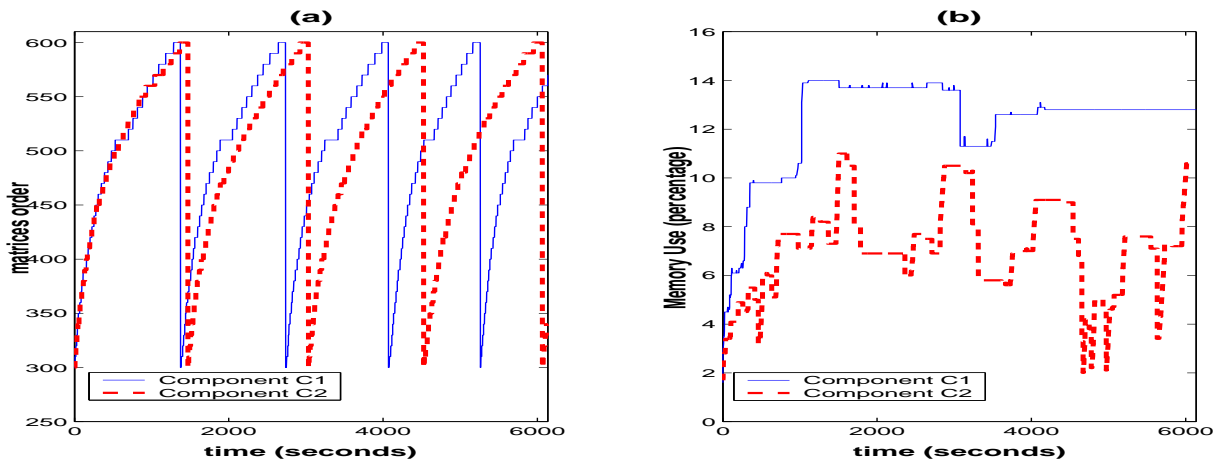


Figure 4: Scenario II. Matrices order, memory use percentage vs. CPU time (Java)

order 400, whereas the C++ version uses less than 2 seconds. The slower performance of the Java components in comparison to the C++ components is an expected result. Also, in another scenario where the Java versions of C_1 and C_2 were executed concurrently, C_1 was five times faster than C_2 but was only 10% better when the two components were launched sequentially as Scenario II [15].

As expected, the behavior of equivalent components implemented using different programming languages impacts the performance and consequently the control of an application.

If we assume a deterministic behavior for the scenarios described above, then their observed results could lead to the specification of a static controller for the two components. However, such controller would, most likely, not be useful when considering other scenarios such as the simultaneous execution of $N = 3, \dots, 10$ processes using one of the two design choices. Another situation to consider would be the addition of a third design choice. In any of these cases the controller would need to be tailored to fit the distinct scenarios. A dynamic self-tuning mechanism for the controller would avoid its re-definition in both cases.

Furthermore, without a complete knowledge of all the elements executing in the system at the same time, we cannot make the assumption of a deterministic behavior for the components and therefore, a scenario may produce different results at different times. Again, a run-time self-tuning mechanism for the controller appears to be the solution to handle such dynamically changing environment.

The examples presented in this section are useful in identifying some of the issues to be addressed by the SMART Framework. It also serves to show the applicability of the framework. More experiments are being designed to provide additional evidence of the applicability of the SMART framework.

4. RELATED WORK

The SMART Framework is an adaptive approach based on the selection of components available in a repository. There-

fore, the work most related to SMART, adaptive systems and component based architecture, are described next.

4.1 Adaptive Systems

Self-adaptive systems has been proposed for specific domains such as QoS [3, 4], ad-hoc network management, traffic control, and signal processing [15]. The world wide web has also been the focus of the use of adaptive techniques at “interface” level as well as at a lower level [15]. Approaches that are not domain specific have also been proposed [6]. Many of successful approaches are based on the sound theory of feedback control [4, 3, 15]. Decision theory is another technique used in adaptive systems. Uncertainty has common place when dealing with adaptive systems and Decision Theory provides mathematical tools, such as game theory and subjective probability, to address decision making problems under the presence of uncertainty.

Next, a brief description of two distinct adaptive techniques is presented. They are selected as representatives, among a multitude of other approaches [15], due to their relation to the SMART Framework. *Containment Units* approach has a similar goal of designing/specifying adaptable components while the QoS framework uses a similar, control theory technique that is applied in SMART.

Containment Units: a *Containment Unit* is basically a module that implements some functionality. The basic difference between a *Containment Unit* and a regular module is that the first is also defined according to nonfunctional requirements, such as time, memory, and sensors [6]. The interface of a *Containment Unit* is defined by a tuple (F, R, CP, FC) . The functional requirements are represented by F and the nonfunctional by R . The expected input and output, including internal faults is represented by CP (Communication Protocol). External faults due to operational components but that are handled inside the *Containment Unit* are specified in FC [6]. The architecture of a *Containment Unit* is composed of the “Operational Component”, the “evaluator”, the “change agent”, the “Adapter”, and the “Implementation”. A *Top* component initializes the unit and manages the communication protocol. A set of opera-

tional components provides the functionality of the *Containment Unit*. The *Evaluator* monitors the performance of the operational components in order to ensure that the specification of the *Containment Unit* interface is satisfied. When the *Evaluator* determines that the unit is not performing according to the specification, the *Change Agent* is in charge of selecting a different operational component that better copes with the current resources. Another alternative is the re-allocation of resources to match the current operational component [6].

Adaptive Control Framework for QoS: the work done on adaptive systems for QoS guarantees [4] is based on the application of control theory aspects with an automatic adjustment of the controller by means of system identification techniques. A pole placement adaptive control technique is used to overcome the restrictions of a static designed controller. The goal is to maintain a specified target hit ratio performance by adjusting storage allocation. The *Automatic Model Estimator* uses the system input and output to update the controller. The *Controller* monitors the system and signalizes the *Actuator*, when necessary, to re-allocate resources.

The SMART Framework has a similar architecture to the one presented by Lu and Abdelzaher [4]. Both approaches are based on control theory and on automatic tuning of the controller. SMART differs from the QoS framework by adjusting the system not based on resource allocation but on swapping to a component that provides the same functionality but better copes with the environment status. Also, as described in Section 5, SMART is a more generic approach by allowing the specification of the system constraints and their relation to the alternative choices.

4.2 Component Based Architecture

The issues involved with developing a complex system with OTS components are extensive; they impact almost every aspect of software development including the requirements specification, architecture, implementation, testing, maintenance, OTS selection and evolution, project management, cost estimates, licensing, etc. Specific solutions have been proposed to address how to evaluate, specify, select, and compose OTS components. A survey of work in these areas is available in [15].

Repository based approaches, which address a collection of issues including the evaluation, specification, selection, and composition of OTS components, have also been proposed [16, 17]. As our research is most closely related to work focused on the design of component based systems using a repository, we present some of the proposed solutions in this area.

On-line Repository for Embedded Systems: a component based approach to embedded software development called ORES (Online Repository for Embedded Software) [17] provides a repository of components and tools for component specification, composition, and analysis at design time. In this semi-automated approach, the developer interacts with the tools to create a component based system. The ORES approach uses ontology based repository browsing. The components are stored in the repository using a hi-

erarchy of domain, sub-domains, and packages. A document type definition (DTD) is used for information attributes definition and the extended markup language (XML) is used to specify the actual measurement data for each component. Due to the hierarchical nature of the repository, a child node inherits the attributes of the parent and we can add or delete attributes to form its own set. The node information contains general information (identification, type, keyword list with weight, short description of the node), information pointers (pointers to file names for modules of code or documentation), and properties (reliability, portability, resource requirements, etc.)

K-BACEE: the Knowledge-Based Automated Component Ensemble Evaluation (K-BACEE) approach [16] is a partially automated, knowledge-based approach to evaluating ensembles of components within the context of a system requirements specification (SRS). K-BACEE emphasizes the identification and selection of ensembles, or groups, of components to satisfy the requirements of a system. The main activities include the iterative development of an SRS, identifying individual components that meet the SRS using queries, identifying ensembles of components that are compatible using integration rules, defining functional and non-functional attributes of COTS components, and defining the integration rules (i.e., rules that determine if components are compatible). The SRS defines the functional and non-functional requirements of the system. The attributes of the COTS components define its functional behavior and the non-functional characteristics that impact compatibility with other components, for example, the protocols supported or the implementation language of the component. The integration rules define how the values of component attributes affect their integration. The SRS, components, queries, and integration rules are stored in the knowledge base. A Java prototype of K-BACEE tool support has been developed, the requirements specification and components are defined in the extensible markup language (XML); queries are defined in the extensible query language (XQL). The integration rules are defined using an object-oriented rule-based programming language, JRules. Once the functionality and constraints of the system are defined, the SRS is manually converted into a series of queries on the component repository. Components that match the requirements specified in the SRS are grouped into ensembles by the tool support; their compatibility is evaluated based on the value of attributes and a repository of software engineering integration rules. The integration rules are executed to rank the ensembles; the ensembles are presented to the K-BACEE user for selection.

5. CONCLUSIONS AND FUTURE WORK

The objectives, description, and advantages of a new adaptive framework named SMART have been presented here. The overall goal of this framework is to provide an environment for the design of adaptive software which allows the dynamic use of multiple solutions based on system resources, component constraints, and user requirements that is based on a strong mathematical foundation.

Significant advantages of the SMART Framework include that it is domain independent, and may be applied to a variety of problems, and it supports the prediction of future

behavior, which supports swapping components very efficiently. Based on its capabilities, the SMART Framework appears to be a more complete alternative for the designers of adaptable systems. SMART impacts not only on the way adaptive systems are conceived but also on the mechanisms used to control them. In addition to the advantages of many adaptive system approaches (self-adaptation, robustness, better resource usage, etc.), the SMART Framework provides two distinguishing features.

Flexibility: the majority of the adaptive techniques are domain specific. This is due to the fact that partial or complete knowledge about the relationship of the inputs and systems resources have to be known a-priori. The System Identification Module in the SMART Framework dynamically identify these relationships and therefore allows its application to a variety of different domains.

Predictability: the availability of a state model capturing the dominant behavior of the system allows the prediction of future behavior. The time when changes are necessary can be predicted and the system preparation to swap to a new component can be done in advance resulting in an overall increase in the systems performance and robustness.

Though SMART is applied here to a general problem, the framework appears to be powerful enough to handle applications running in a pre-specified, but very constrained host such as most of the environments of embedded systems.

In the experimental evaluation, SMART successfully predicted and prevented memory overflow in 86% of the cases. This is a good indication of the viability and accuracy of the framework. Possible techniques that may be used to improve the accuracy of the framework include creating a compiled version of the Matlab scripts in order to reduce processing delays. This feature is supported in the Matlab tool and is going to be investigated in the next step of the work.

Additional verification of the SMART Framework is needed. In the next step of the work, SMART is going to be evaluated to determine how effective it is for applications that are distributed over many heterogeneous platforms such many web systems. Additional experiments are planned to rigorously evaluate the SMART framework that include monitoring and controlling multiple, concurrent processes for a variety of example systems.

6. REFERENCES

- [1] A. Ledeczki, G. Karsai, and T. Bapty, "Synthesis of self-adaptive software," in *Proceeding of the IEEE Aerospace Conference*, March 2000.
- [2] P. Norvig and D. Cohn, "Adaptive software." PC AI Magazine, January 1997.
- [3] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Operating Systems Design and Implementation*, pp. 145–158, 1999.
- [4] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao, "An adaptive control framework for qos guarantees and its application to differential caching services," in *10th IEEE International Workshop on Quality of Service*, pp. 23–32, 2002.
- [5] Z. T. Kalbarcyk, R. K. Iyer, S. Bagchi, and K. Whisnant, "Chameleon: A software infrastructure for adaptive fault tolerance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 560–579, June 1999.
- [6] J. M. Cobleigh, L. J. Osterweil, A. Wise, and B. S. Lerner, "Containment units: A hierarchically composable architecture for adaptive systems," in *Proceedings of the 10th International Symposium on the Foundations of Software Engineering (FSE 10)*, (Charleston, SC), pp. 159–165, November 2002.
- [7] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "A formal model for the software test process," *IEEE Transaction on Software Engineering*, vol. 28, pp. 782–796, August 2002.
- [8] G. C. Goodwin, S. F. Graebe, and M. E. Salgado., *Control system design*. Upper Saddle River, New Jersey: Prentice Hall, 2001.
- [9] L. Ljung, *System identification: Theory for the user*. Englewood Cliffs, New Jersey: Prentice-Hall, 1987.
- [10] M. M. Lehman, "Feedback in the software process," in *SEA Workshop: Research Directions in Software Engineering*, (London), Imperial College, 1997.
- [11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, *Extensible Markup Language (XML) 1.0*. W3C Recommendation 6, second edition ed., October 2000.
- [12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [13] J. Liu and Y. Zhao, "On adaptive agentlets for distributed divide-and-conquer: A dynamical system approach," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, pp. 214–227, March 2002.
- [14] J. Melton, "Xml-related specifications (sql/xml)," August 2002. ISO-ANSI Working Draft.
- [15] J. W. Cangussu and K. Cooper, "A new approach for the design and control of adaptive systems," Tech. Rep. UTDCS-21-03, University of Texas at Dallas, Richardson-TX, USA, May 2003.
- [16] R. Seacord, D. Mundie, and S. Boonsiri, "K-bacee: Knowledge-based automated component ensemble evaluation," in *Proceedings of the 2001 Workshop on Component-Based Software Engineering*, 2001.
- [17] I. Yen, J. Goluguri, F. Bastani, L. Khan, and J. Linn, "A component-based approach for embedded software development," in *Proceedings of the 5th International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 402–410, 2002.