Adaptive, Intelligent Control of Embedded Applications

Paul Robertson and Robert Laddaga
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Email: {paulr, rladdaga}@ai.mit.edu

Abstract

The paper describes an adaptive intelligent controller architecture (GRAVA) for an image understanding system. The controller architecture is based on the concept of self-adaptive software, an approach that has seen significant effort in the past few years. Self-adaptive software involves synthesizing code that monitors, diagnoses and repairs behavioral deviations from explicitly stored and managed goal and method information. There have been several self-adaptive systems produced in recent years, but none of them has yet seriously addressed the issue of behavioral guarantees (including GRAVA). In addition to describing GRAVA, this paper proposes a new mechanism for providing assurance of stability in self-adaptive architectures. That mechanism is a process reflection layer to reason about the behavior of the program correction parts of the self-adaptive system. This mechanism will limit the scope in time and severity of deviations from nominal behavior, reduce or eliminate unstable corrections, and provide design time assurance of these behavioral characteristics.

Key Words
Intelligent Control, Adaptive Control, Self-adaptive Software, Stability

Introduction

Embedded software systems, by which we mean any system that utilizes software to monitor or control real world processes, are both common and extremely important. Examples include automobiles, medical devices such as CAT scanners, airplanes, and consumer electronic products. The bulk of computer chips produced are for embedded systems (rather than familiar personal, departmental and mainframe computers). As the tasks that we set for these embedded systems become more complex, and the degree to which we deal with the true dependencies and complexities of the real world increases, the difficulty of producing the software for these embedded systems rises dramatically. It is therefore quite clear that new approaches to building such software are needed.

The dominant approach to such software has been hand coding, instruction counting, and very little use of frameworks. More recently we’ve seen adoption of real-time operating systems, and even a few other frameworks, such as Time-Triggered Architecture (Kopetz 1997), or Rate Monotonic Scheduling (Lehoczky 1989). In addition, there have been uses of analysis and assurance technology such as theorem proving (Rushby 2000), model checking (Alur 1990), and code generation from provably correct transformations (Smith 1999). While we strongly support all of the mentioned improvements, as well as others, we also strongly believe that they are not enough in themselves to overcome the difficulties of producing modern embedded software.

Instead, we rely on an intelligent control approach to software for embedded systems. Rather than simply proving properties of program correctness and behavior, we synthesize code that monitors, diagnoses and repairs behavioral deviations from explicitly stored and managed goal and method information. This self-adaptive, dynamic, intelligent control approach to assurance takes advantage of the singular nature of the case at hand at any given point of program execution, rather than the full generality of possible cases that can be foreseen at program development time.

Self-adaptive software is an attempt to build such computations as systems that apply a corrective force by making changes to the program code. There have been several self-adaptive systems produced in recent years (Robertson 2000), but none of them has yet seriously addressed the issue of behavioral guarantees. The proposed approach to this problem relies on three mechanisms in addition to the self-adaptive mechanism itself. These are the decision theoretic mechanisms, incorporation of a theorem prover to prove assertions needed for the decision theoretic mechanisms, and incorporation of a process reflection layer (Robertson 1992) to reason about the behavior of the program correction parts of the self-adaptive system.

Recently we have developed such a self-adaptive architecture for aerial surveillance (Robertson 1999a,b), and we would like to briefly describe that work, since it will serve as the foundation for the proposed mechanisms.
The Grava Architecture

We present a system called GRAVA (for Grounded Reflective Adaptive Vision Architecture) that segments and labels aerial images in a way that attempts to mimic the competence of a human expert.

![Figure 1: Logical Components](Image)

**Figure 1: Logical Components**

Figure 1 shows the logical components of the system along with the supporting relationships between the parts. We now sketch the roles of these components.

To produce an image interpretation, a variety of tools need to be brought into play. First, the image is processed by various tools in order to extract texture or feature information. The selection of the right tools determines ultimately how good the resulting interpretation will be. Next, a segmentation algorithm is employed in order to produce regions with outlines whose contents are homogeneous with respect to content as determined by the chosen texture and feature tools. The segmentation algorithm also depends upon tools that select seed points that initialize the segmentation. The choice of tools to initiate the segmentation determines what kind of segmentation will be produced.

Labeling the regions depends upon two processes. The first tries to determine possible designations for the regions by analyzing the pixels within the regions. The second is a statistical parser that attempts to parse the image using a 2D grammar. Our application currently doesn’t make use of the parse; but it could be used as the basis for further image interpretation. An important side effect for our application is that contextual information mobilized by the parse process enables good labels to be chosen for regions when there may be several ambiguous possibilities if one only looks at the pixels within the region.

At any point, a bad choice of tool—for initial feature extraction, seed point identification, region identification, or for contextual constraints — can lead to a poor image interpretation. The earlier the error occurs, the worse the resulting interpretation is likely to be.

The problem of interpreting the real world is inherently ambiguous. A speech or vision program must select the most likely interpretation from the ambiguous candidates. Selecting the most likely interpretation is equivalent to selecting the interpretation with the minimum description length (MDL). We developed apparently for the first time an agent architecture based on the MDL principle, and supporting a conjecture of Leclerc (Leclerc1989) that MDL can apply to higher-level semantics.

![Figure 2: Segmented Image](Image)

**Figure 2: Segmented Image**

The region competition algorithm of Zhu & Yuille (Zhu 1996) is probably the leading approach currently to segmentation, it produces reasonably good segmentations based on a purely low level approach; but since there is no provision for combining evidence from higher level semantics, the approach performs poorly when there is poor low level discrimination between regions. By developing an MDL algorithm similar to the region competition algorithm, using the MDL agent architecture, interaction between agents at differing semantic levels allows evidence from higher-level semantics to influence the segmentation. The use of MDL in a segmentation algorithm is not unique (Zhu1996, Leclerc1989); but our use of MDL as a coordination device using high-level semantics in segmentation is new.

The algorithm produces good results. Figure 2 shows an image that has been segmented without any high level semantics. Seed points were selected and the base segmenter was allowed to proceed without any image semantics. The results are as good as those achievable using the Zhu & Yuille (Zhu 1996) segmenter.
When semantics are introduced, difficult segmentations in which the boundaries are not evident at all, such as the Marr example of overlapping leaves shown in Figure 3, are possible. Although the leaves can easily be segmented by human sight, analysis of the pixels along the overlapped leaf region shows that there are no intensity changes from which the edge could be found using low level techniques. The edge that we perceive is a kind of subjective contour.

Two corpora were developed using aerial images imaged from: a satellite, and from a plane. Annotations were made to the images of the corpora by an expert, and the image grammar is induced from the corpus. The image parser attempts to make structural sense out of the segmentation produced by the semantic segmentation algorithm. This results in the propagation of contextual information, assignment of labels to the segments indicating region contents, and interaction with the semantic segmentation algorithm to refine the segmentation based on evidence from the parse.

The parser is a structure-generating application that interprets the image contents. Both the parse structure and the synthesized program structure for the application are generated by a common structure generating theorem prover. The proof that has the shortest description length is judged to be the best proof. In this kind of proof, the theorem is a valid solution. In the case of a parse it is a syntactically correct structure for the set of lexical items (a lexical item in the case of a patchwork parse is a region). In the case of a program it is a legal program that addresses the details of the program's design; in the case of a plan, the theorem would be a legal plan that addresses the details of the goal structure for which the plan seeks to find a solution. While the parse, program, or plan may be legal; none is guaranteed to be correct. The execution of the program may fail at some point. A plan may not succeed. Checkpoints are added to the generated structure whenever a component has a probability of success less than 1. These checkpoints provide a mechanism whereby self-adaptation can be initiated.

Two corpora were developed using aerial images imaged from: a satellite, and from a plane. Annotations were made to the images of the corpora by an expert, and the image grammar is induced from the corpus. The image parser attempts to make structural sense out of the segmentation produced by the semantic segmentation algorithm. This results in the propagation of contextual information, assignment of labels to the segments indicating region contents, and interaction with the semantic segmentation algorithm to refine the segmentation based on evidence from the parse.

The parser is a structure-generating application that interprets the image contents. Both the parse structure and the synthesized program structure for the application are generated by a common structure generating theorem prover. The proof that has the shortest description length is judged to be the best proof. In this kind of proof, the theorem is a valid solution. In the case of a parse it is a syntactically correct structure for the set of lexical items (a lexical item in the case of a patchwork parse is a region). In the case of a program it is a legal program that addresses the details of the program’s design; in the case of a plan, the theorem would be a legal plan that addresses the details of the goal structure for which the plan seeks to find a solution. While the parse, program, or plan may be legal; none is guaranteed to be correct. The execution of the program may fail at some point. A plan may not succeed. Checkpoints are added to the generated structure whenever a component has a probability of success less than 1. These checkpoints provide a mechanism whereby self-adaptation can be initiated.

In our test case, what we have been calling semantics in the foregoing is a meaningful parse of the image. The patchwork parser is a statistical 2D parser. The parser takes as input a segmented image with content descriptors assigned to the segments. The parser builds a structural description of the image. Difficulties in producing a good parse can cause self-adaptation, which results in the segmentation program producing different segmentations that may parse better.
Figure 5: Description for River Using Red and Blue

Figure 5 shows data points for river using the red and blue channels. The MDL representation for these data points involves dividing the data into two separate models as indicated by the coloring of the data points.

**GRAVA’s Self Adaptive Architecture**

The goal of the architecture is to support self-adaptation. When the self-assessment determines that the program is doing poorly, the program should seek some way of adjusting its structure so as to do better. The self-adaptive architecture is a collection of supporting capabilities that permits this simple approach to self-adaptation to work. The supporting components are as follows:

1. **Self-assessment**—the ability of a computational agent to evaluate how well it is doing at its current task. The GRAVA architecture provides a protocol for supplying self-assessment functions.

2. **Structure building**—the mechanism that constructs a program from a collection of computational agents. This structure building apparatus is invoked whenever self-assessment indicates poor performance; the system tries to improve by re-synthesizing its program code, using the statistical theorem prover.

3. **Reflection**—the support for self-understanding within the system. By inspecting the state of the embedded semantic account, the system can reason about what the system is doing in terms of a goal that its actions are intended to achieve.

**Results of Initial GRAVA Experiments**

Grava was based on the idea that if vision programs knew what they were doing, and knew when they were doing poorly, then they could adjust their assumptions and thereby do a better job. It seemed that the idea of reflection, developed in AI, offered a way to do this. That approach lead us to utilize reflection in a novel way—to support self-adaptation.

We were able to get positive results from all phases of the research on GRAVA, including the final step of making the program self-adapt in order to produce an acceptable interpretation of images. The problem domain involved segmenting, labeling, and parsing aerial images. We had no consumer for the image parse and so it is difficult to access the success or usefulness of the parse for any particular use. The parses that were generated seemed plausible when inspected by hand. The parse process was certainly useful as a mechanism for propagating non-local context in arriving at plausible region labeling. We did not collect enough data to quantify the benefit to labeling that the parser provided but anecdotally there are many cases where otherwise ambiguous regions were correctly labeled.

We are encouraged by the success of the architecture within the limitations discussed above. We have succeeded in building a segmentation algorithm that is capable of using higher-level semantics in order to produce appropriate segmentations (Robertson 1999a). We have produced an image parser that produces structural descriptions of the image in terms of labeled regions, and we have shown how models can be induced from an annotated corpus and used as the basis for the segmentation, labeling, and parsing of an aerial image.

**The Need for Additional Mechanisms**

There are several areas in which GRAVA can be improved, and we mention three such areas, and discuss a solution for one of them.

**Resource Contention, Real-Time Constraints, and Utility**

The compilation of interpreters in GRAVA collects all relevant agents for a chosen context and puts them into the interpreter. This means that all agents that are valid within the context and which may contribute to the computation will be available for the interpretation task. We have not taken into consideration the possibility that there may not be sufficient computational resources to run them all. We have concentrated on finding an approximation to the best interpretation at any cost.

In certain practical situations, it may be necessary to further limit the agents available in an interpreter. To do this we would need:

1. Some representation of a resource budget,
2. Some representation of the cost of running an agent,
3. Some way of estimating the extent to which agents overlap in their coverage of the interpretation space, and
4. Some way of distributing the resource budget over the entire program.

**Unsupervised Learning**

We have seen several places where it is possible to learn a new model without the need for an expert annotation. Typically this happens when smoothing has occurred. Consider the case of the parser. Since not all parse rules are likely to be encountered in the corpus we have a way of generating rules that allow the parse to succeed.
Similarly we may have a region for which we don’t have a content model that would allow us to label the region so we pick the most likely region given the available parse rules that may consume it.

In the present system these ‘smoothed’ models are not retained, and are given low probabilities so that they are not preferred over rules that do occur in the corpus. If the smoothed models were retained as if they had been encountered in the corpus they could participate in future interpretations. Each time the rule is used, whether it is an original model from the corpus or a model learned from a prior smoothing, its frequency of use could be tracked as the program encounters real data. Over time, the system would converge on a set of rules that match what the ‘real world’ suggests it needs. Much work would be needed to realize this potential.

Assurance of Adaptive Stability

One important concern that is not so easily dispensed with is unstable behavior. In one example, the system self-adapted twice before it was able to interpret the image presented to it. Each adaptation brought the system closer to what was required in order to interpret the image. One adaptation was to adjust the optical context and one was for the labeling context. The system converged until it was able to interpret the image. In an unstable system the program would either get no closer to its set point or would diverge from the set point resulting in an infinite series of self-adaptations.

Design methodologies have been developed for linear control systems and for nearly-linear control systems. Intuitively it would be nice if these methods could be adapted for self-adaptive software systems. Unfortunately, the techniques cannot be applied straightforwardly because software is highly non-linear. It most cases, it is impossible even to decompose the problem into piecewise approximations to linear.

The design of how world knowledge is updated during an adaptation appears to be the key to controlling whether a self-adaptive program implemented in GRAVA will be stable. What is needed is an approach that allows us to verify that the resulting program cannot get into an unstable cycle of self-adaptation.

Self-adaptive software in GRAVA takes the form of a control loop. Deviations from desired behavior detected within the reflective framework are used to re-synthesize the program code. The resulting program code may still differ from desired behavior requiring further self-adaptation. In a stable system this process will eventually converge towards the set point. In an unstable system the self-adaptation process will repeat indefinitely and will either not improve or even degrade (become further from the set point). Since the synthesized code is produced by automated formal methods the correctness of the synthesized programs is not at issue---but stability is.

We plan to augment the existing GRAVA architecture with design tools and methodologies that allow stability to be guaranteed by:

1. Incorporating tests for instability into the reflective framework that enables a system to automatically detect its own instability.
2. Building mechanisms to ‘break’ unstable behavior at runtime.
3. Develop design methodologies for self-adaptive software that either prohibit instability or ensure that adequate detection and recovery is available where instability can’t be prohibited.

Conclusions

The self-adaptive software approach to building reliable embedded systems is exciting because it provides a way of dealing with highly complex environments using an approach that depends upon automated formal methods. Unlike other approaches this allows us to develop a high level of confidence about the resulting system performance. The GRAVA architecture introduced above has the following beneficial attributes:

1. The architecture is structured in the form of a software control system in which process performance is continually compared against the system specification. When deviations are detected, because some assumptions about the environment are not providing a good match with the actual environment, the program structure of the system is modified to reduce the deviation and bring the system back towards its set point defined by the system specification.

2. When the system code is regenerated it is done by generating a provably correct program from the system specification and the environment assumptions. At any point in time there is a single program running that has well understood characteristics.

While existing research has concentrated on the practical issues of building self-adaptive systems that work, it is now time to pave the way towards the confident deployment of complex embedded systems in the near future. We need experimental implementations of self-adaptive systems that provide performance guarantees including resource management, stability, and adequate domain coverage.
**Acknowledgements**

Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-98-0056. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

**References**


