Model Based Diagnosis and Contexts in Self Adaptive Software

Paul Robertson
Dynamic Object Language Labs
9 Bartlet Street, Suite 334
Andover, MA 01810
email:paulr@doll.com

Robert Laddaga
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
email: rladdaga@ai.mit.edu

Abstract— **Self Adaptive Software monitors its own operation and attempts to correct deviations from required behavior. In the self adaptive architectures we are developing, it accomplishes this by diagnosing the sources of deviant behavior, whether internal program problems, or contextual changes in an embedded program's environment. The software then responds by reconfiguring itself, to use alternate procedures that either correct the malfunction, or perform better in the current context. The diagnosis and reconfiguration are in part accomplished by storing models of the goals, structure, design and requirements of a program such that they are available to the program at run time. Since most of our self adaptive systems are also embedded systems, models of the physical plant which the software controls or effects are also stored and referenced at run time. In this paper, we describe our architecture, its use of model based diagnosis, and give some examples of applications of the technology.**

I. Introduction

Software design consists in large part in analyzing the cases the software will be presented with, and ensuring that requirements are met for those cases. It is always difficult to get good coverage of cases, and impossible to assure that coverage is complete. If program behaviors are determined in advance, the exact runtime inputs and conditions are not used in deciding what the software will do. The state of the art in software development is to adapt to new conditions via off-line maintenance, and the required human intervention delays change. The premise of self adaptive software is that the need for change should be detected, and the required change effected, while the program is running (at run-time).

The goal of self adaptive software is the creation of technology to enable programs to understand, monitor and modify themselves. Self adaptive software understands: what it does; how it does it; how to evaluate its own performance; and thus how to respond to changing conditions. We believe that self adaptive software will identify, promote and evaluate new models of code design and run-time support. These new models will allow software to modify its own behavior in order to adapt, at runtime, when exact conditions and inputs are known, to discovered changes in requirements, inputs, and internal and external conditions.

Ideally, Self Adaptive Software should be able to diagnose problems by referring to models of the software itself, as well as models of processes in the world with which the program interacts. In this paper we describe the overall approach to such diagnosis, as well as providing an example from our own work in Self Adaptive architectures for computer vision.

II. Self Adaptive Software

A definition of self adaptive software was provided in a DARPA Broad Agency Announcement on Self-adaptive Software [11]:

> Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.
>
> This implies that the software has multiple ways of accomplishing its purpose, and has enough knowledge of its construction to make effective changes at runtime. Such software should include functionality for evaluating its behavior and performance, as well as the ability to replan and reconfigure its operations in order to improve its operation. Self adaptive software should also include a set of components for each major function, along with descriptions of the components, so that components of systems can be selected and scheduled at runtime, in response to the evaluators. It also requires the ability to impedance match input/output of sequenced components, and the ability to generate some of this code from specifications. In addition, DARPA seek this new basis of adaptation to be applied at runtime, as opposed to development/design time, or as a maintenance activity.

Self adaptive software constantly evaluates its own performance, and when that performance is below criteria, changes its behavior. To accomplish this, the runtime code includes the following things not currently included in shipped software:

1) descriptions of software intentions (i.e. goals and designs)
2) descriptions of program structure;
3) descriptions of the environment that the program is running in, both computational and (for embedded software) physical;

4) a collection of alternative implementations and algorithms (sometimes called a reuse asset base).

We will return to further discussion of the first three items above, in a later section: Model Based Diagnosis. Three metaphors have been useful to early researchers on self adaptive software: coding an application as a dynamic planning system, or coding an application as a control system, or coding a self aware system, [12]. The first two are operational metaphors, and the third deals with the information content and operational data of the program.

In programming as planning, the application doesn't simply execute specific algorithms, but instead plans its actions. That plan is available for inspection, evaluation, and modification. Replanning occurs at runtime in response to a negative evaluation of the effectiveness of the plan, or its execution. The plan treats computational resources such as hardware, communication capacity, and code objects (components) as resources that the plan can schedule and configure. See [2], [9], [16], and [20].

In program as control system, the runtime software behaves like a factory, with inputs and outputs, and a monitoring and control unit that manages the factory. Evaluation, measurement and control systems are layered on top of the application, and manage reconfiguration of the system. Explicit models of the operation, purpose and structure of the application regulate the system's behavior. This approach is more complex than most control systems, because the effects of small changes are highly variable, and because complex filtering and diagnosis of results is required, before they can serve as feedback or feed-forward mechanisms. Despite the difficulties of applying control theory to such highly non-linear systems, there are valuable insights to be drawn from control theory, and also hybrid control theory, including for example the concept of stability. See [9], [10], and [14].

The key factor in a self aware program is having a self-modeling approach. Evaluation, revision and reconfiguration are driven by models of the operation of the software that are themselves contained in the running software. Essentially, the applications are built to contain knowledge of their operation, and they use that knowledge to evaluate performance, to reconfigure and to adapt to changing circumstances (see [12], [20], and [15]). The representation and meta-operation issues make this approach to software engineering also intriguing as an approach to creation of artificial intelligence.

III. Model Based Diagnosis

We said before that self adaptive software will include descriptions of:

1) software intentions (i.e. goals and designs)
2) program structure;
3) the environment that the program is running in, both computational and (for embedded software) physical;

Each of these descriptions will generally be in the form of a model. That is, the descriptions must involve a significant functional abstraction, so as to support operations on the descriptions that can in turn affect the functional behaviors of the things described. So for example, we must be able to recomputed subgoals of a goal in the light of new contextual information. Also, we want to use the models of program structure to diagnose problems and support reconfiguration of the program. Finally, models of the physical environment can be used to:

1) diagnose program failures and performance problems,
2) provide contextual basis for subgoaling and reconfiguring
3) provide a basis for choosing new strategies for the computation.

We also said earlier that the chief engineering issue for self adaptive software was evaluation of program performance. It is of course possible to do evaluation without actually diagnosing a problem, even when one is determined to respond to the evaluation. For example, given a poor evaluation (which may itself provide no diagnostic information) we might simply respond by randomly picking a different algorithm or implementation. Although this fits a broad definition of self adaptive software, our goals are much higher.

Instead, the kind of evaluation we envision is one that includes and partially depends on a diagnosis of at least the proximate cause, and where possible the root cause, of the failure or performance problem. In this sense, the entire self adaptive apparatus in the program can be thought of as a model based diagnosis system, in support of the program's main goals. The program, its goals, and the environment that it runs in are all modeled in the running system, and diagnostic reasoning is employed to evaluate program performance. Thus model based diagnosis realizes the self aware metaphor for self adaptive software.

In the next sections, we introduce a computer vision system that uses diagnosis changes in the current context, in order to adapt to those changes.

IV. Vision and Context

Image understanding programs have tended to be very brittle and perform poorly in situations where the environment cannot be carefully constrained. Natural vision systems in humans and other animals are remarkably robust. We believe that recognition of (and adaptation to) changes in the environment is what allows natural vision to be so much more robust than computer vision. At any instant a program is operating within a *context* in which the complexity of the real world is bounded. In AI we have been fairly successful at building systems that perform robustly within environments of restricted complexity. If we can divide the complexity of the world up into a collection of contexts, each of a bounded and manageable size, we can in principle consider the hard problem of making a robust embedded system as an easier problem formulated as the composition of a collection of manageable parts.

A premise of the self-adaptive approach is that it should be possible, at runtime, to synthesize context specific systems, to determine the need to change context and to self-adapt the program so that the program's context matches the state

of the environment and operates robustly because each of its components is operating well within their optimal range.

The idea of self-adaptation is to adapt the program to a particular "context". In order to achieve this adaptation we build structural descriptions that facilitate dividing the model space into contexts and provide a mechanism for determining when a context is a good fit to the environment, and diagnosing when we have a poor context fit.

V. Complexity and Context in GRAVA

The first application of the GRAVA architecture [19], [18] was to the interpretation of satellite aerial images. In GRAVA (for Grounded Reflective Adaptive Vision Architecture), satellite images were segmented into regions of homogeneous content and the regions were parsed, much as words are in a sentence to form a structural understanding of the image. Different image types are comprised of different kinds of regions, different colors and textures, and different parse rules. Rather than making one huge grammar that includes all textures and region types, it is better to have grammars, and optical models tailored to the context because tailored contexts provide greater accuracy and constraint. In that program the contexts as well as the grammars and region content models were learned from a corpus of images annotated by a human photo interpreter.

The use of corpora in building trainable intelligent systems has been a growing trend in A.I. in recent years especially in natural language [5], [4], speech understanding [8], and computer vision [17]. These problems are far too difficult to tackle in their full generality, and require techniques for managing that complexity. One common and beneficial use of corpora is in managing complexity by learning contexts. Contexts introduce constraints that bound the choices to be made (or learned) about attributes and their values.

Contexts occur for a variety of reasons, at different levels of processing, and in different parts of the corpus. Given a set of images it is generally not possible to divide the images into separate piles with each pile representing a different context. Contexts for different aspects of the problem can be composed in a variety of ways. The explosion of possible combinations of contexts is one reason why the self-adaptive approach is attractive. That is, rather than generating all possible combinations of contexts in advance—and then having a "big switch" to choose which to use—it is better to generate the particular combination of contexts on demand.

A. An Overview of the GRAVA Architecture

The purpose of the reflective architecture is to allow the image interpretation program to be aware of its own computational state and to make changes to it as necessary in order to achieve its goal. The steps below provide a schematic introduction to the GRAVA architecture.

1) The desired *behavior* is specified in the form of statistical models by constructing a corpus.
2) The behavior, which covers several different imaging scenarios, is broken down into contexts. Contexts exist for different levels of the interpretation problem. Each context defines an expectation for the computational stage that it covers. Contexts are like frames and schemas; but because the contexts are gathered from the data automatically it is not necessary to define them by hand.
3) Given a context a program to interpret the image can be generated from that context. This is done by *compiling* the context into a program by selecting the appropriate agents.
4) The program that results from compiling a context can easily know the following things:
   a) What part of the specification gave rise to its components.
   b) Which agents were involved in the creation of its components.
   c) Which models were applied by those agents in creating its components.
   d) How well suited the current program is to dealing with the current input.
5) The division of knowledge into agents that perform basic image interpretation tasks and agents that construct programs from specifications is represented by different reflective levels.

B. Context in Aerial Image Interpretation

To better understand the idea of contexts, consider the case of optical model contexts and language model contexts.

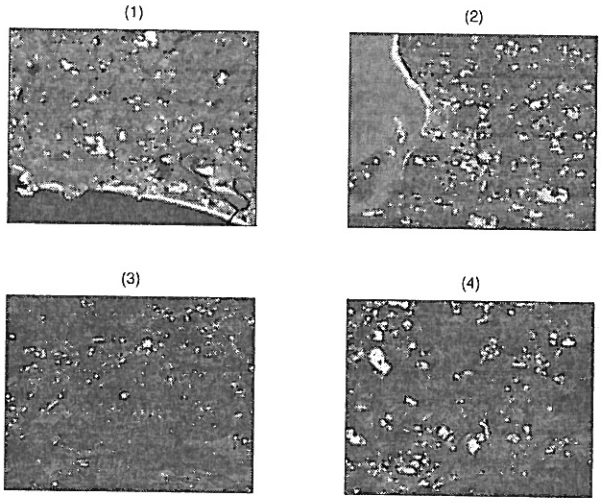

Fig. 1. Image Contexts

Figure 1 shows four multi-spectral color SPOT images from the color corpus that demonstrate different contexts. Images (1) and (2) are similar in content (mostly farmland and small towns) but the colors and textures of the regions are very different. In fact, the images are taken under different imaging conditions. In the case of these two images, the major difference is with the optical models, since, grammatically, the two are rather similar. In images (3) and (4) the nature of

the terrain is very different. Image (3) shows part of a major city whereas image (4) shows a rural setting with only small villages. The grammar that is suitable for parsing images 3 and 4 is quite different. Attempting to interpret any of these images with the wrong collection of optical or grammatical models may be expected to produce a poor result especially since knowledge weak segmentation algorithms often give poor results. In this case, the reason for the differences between image (1) and image (2) were changes in the SPOT technology used to image them.

C. Context and Face Recognition

We next consider an example from the application of computer vision to face recognition. Most face recognition systems work by measuring a small number of facial features given a canonical pose and matching them against a database of known faces. Frequently however, in practical applications, few frames show a full frontal face. By building a face recognizer that can seamlessly switch between different contexts such as pose and lighting we can construct a recognizer that is robust to normal changes in the natural environment. This permits a much wider application of face recognition technology. (See [22] for a more complete description of the face recognition system).

Our application involves recognizing people as they move about an intelligent space [3] in an unconstrained way. An intelligent(or smart) space is a room or collection of rooms and corridors that have an abundance of sensors so that computer monitoring can track and understand activities in the space and provide intelligent support for the activities of the participants. Although the use of a space may in general be very complex, most spaces have a number of frequently repeated uses. By modeling the relationship between contexts the system can predict activities that occur in sequences within a space. For example the sequence of "assembling", "meeting", and "adjourning" can be learned as a hidden markov model (HMM) [23], [1].

To better understand contexts for face recognition consider the face "pose" contexts:

Fig. 2. Four Pose Contexts

Figure 2 shows four pose contexts: "profile", "oblique", "off-center", and "frontal". The profile view, for example, is supported by agents that measure points along the profile of the face, the corner of the eye, and the lips. The oblique view, on the other hand, is supported with measurements of the ear and measurements of the position of the ear, eye, and nose. The triangle formed by the eye, ear, and nose help to determine the angle of the face to the camera which allows measurements to be normalized before recognition. The different contexts control, among other things, what models can be used for matching, what features can be detected and what transformations must be made to normalize the measurements prior to matching. This example shows contexts for pose but there are also contexts for lighting, race, gender, and age.

The recognizer supports a collection of face candidate finders, face models, feature finders, and normalization algorithms implemented as agents. Face candidate finder agents look for face like shapes in the image and generate evidence that supports the selection of a set of contexts based on the shape and shading of the face candidate. Agents appropriate to the context are selected to make a special purpose face recognizer. If the recognizer doesn't succeed in finding appropriate features where they are expected to be the system self-adapts by using available evidence to select a more appropriate context, constructing a new recognizer, and trying again.

D. Separation of Contexts

Separating contexts in this way suggests (for example) that grammar learned using one sensor implementation may be usable even when the sensor technology is changed, so long as new optical models are available for the new sensor. One of our original goals was to be able to build vision systems that continue to work well despite changes in the sensor and image preprocessing environment. For that reason, the separation of imaging contexts from language contexts is an appealing idea.

Even without changing the sensor technology, different optical contexts may be called for, for example, variations in optical characteristics due to changes in weather conditions or season. When the seasons change, the optical characteristics of fields and trees vary dramatically but the language of the terrain changes little grammatically because it is defined by the man-made structures and natural constraints of the terrain. San Francisco is not a city one day and a rural region the next.

AI has long understood the importance of contexts. In 1975 Minsky introduced the notion of *frames* [13] which was essentially an approach to contexts. Frames have been used extensively in AI research, especially for natural language. Riseman's Schemas [6] was a similar idea specifically for Computer Vision.

VI. Principle Component Decomposition

The architecture discussed above can adapt to a changing environment, given a decomposition of the problem space into contexts. We now describe how we derive our set of context models from a corpus, automatically.

A corpus provides multiple positive examples of a structure that we wish to model. The structures in question have one or more dimensions, and the corpus provides examples of the structure that enable us to model the location within the appropriate multidimensional space. One way of doing this is to model the structures as a probability distribution function (PDF). The natures of the structures may be very different but

the essential nature of a corpus is the same: positive examples of structures in a multi-dimensional space.

Principle component decomposition is the interpretation of a set of data points into the component collections (five in this example) by analyzing the principle components of the interpretation space. After the space has been divided into separate clusters conventional PCA may be applied to produce the models.

The algorithm builds upon two earlier works. The first is a classification program developed by Wallace [24], and the second is the practice of using principle component analysis [7] to reduce the dimensionality of high dimensional problems to model separate populations. Our algorithm applies principle component analysis recursively in order to separate the collection into successively smaller clusters. At each point the criterion for separating a population is that it reduces the global description length of the original population. A more complete description can be found in [21].

A. A Statistical Model for Clusters

Given an n-dimensional space S_n containing m points. we can interpret the points in this space as being unrelated points, all members of a single cluster, or grouped into a number of clusters.

A premise of the GRAVA architecture is that knowledge of the world in the form of models can be used to produce better descriptions of an image. A good description of the world in the GRAVA architecture is one that has a minimum description length. A model allows a shorter description length if the model reduces the amount of uncertainty about the values of features in the image.

The entropy of the collection data points in the corpus is given by:

$$H = -\sum_{d \in S_n} P(d) log_2 P(d) \tag{1}$$

The lower bound MDL of a description that represents all of the points in the S_n is given by:

$$DL = -\sum_{d \in S_n} log_2 P(d) \tag{2}$$

In order to compute this theoretical description length, it is necessary to know the PDF for points in S_n. A corpus doesn't specify every possible point in the space. A corpus provides a collection of *representative* points in the space. The job of interpreting the corpus involves modeling the PDF. There are many choices for modeling a PDF. One model that is simple, predictive, and which often pertains to naturally occurring distributions is the Gaussian.

The description of a Gaussian model consists of a mean and variance of the distribution $< \mu, \sigma^2 >$. For a set of points the Gaussian model can be fitted simply by computing the mean μ and the variance σ^2. Given this characterization, for any point d we can compute the probability $P(d)$ as follows:

$$\begin{aligned} P(d) = erf\left(\frac{(pos(d) - \mu_n + \epsilon/2)}{\sigma_n}\right) \\ -erf\left(\frac{(pos(d) - \mu_n - \epsilon/2)}{\sigma_n}\right) \end{aligned} \tag{3}$$

where $pos(d)$ is the position of the point d, ϵ is the position resolution, μ_n is the n-dimensional mean, σ_n^2 is the n-dimensional variance, and erf() is the error function.

The choice of whether to consider the points in the corpus as (1) unrelated individual points, (2) all members of the same model, or (3) divided into groups each of which is modeled, is to select the choice that yields the minimum description length.

The interpretation task can therefore be characterized as dividing the data points in S_n into n proper subsets $C_{i,n}$ such that:

$$S_n = \bigcup_{i=1}^{n} C_i \tag{4}$$

The MDL is

$$\begin{aligned} arg\min_{C_{1,n}} \sum_{i=1}^{n} \Bigg\{ \left(\sum_{d \in C_i} -log_2 P(d|C_i) - log_2 P(d \in C_i) \right) \\ +ddl(C_i) \Bigg\} \end{aligned} \tag{5}$$

where $ddl(C_i)$ is the description length of the distribution used to model C_i. The description of a point is divided into two parts. The first part identifies its position in the space ($-log_2 P(d|C_i)$) and the second part identifies to which collection it belongs ($-log_2 P(d \in C_i)$).

The statistical models chosen for C_i determine the size of the point descriptions. In order to specify the position of a point we choose a resolution ϵ to be used uniformly since otherwise a point can have an arbitrary precision and its representation would be arbitrarily large.

If the representation of a collection includes its mean position μ, the positions of the points in the collection can be described as distances δ from the mean. Figure 3 shows the representation of a point within a collection C_i as an n-dimensional mean ($n = 2$ in this example) and an n-dimensional displacement. So any point d can be described as:

$$\mu + \delta - \frac{\epsilon}{2} \leq d \leq \mu + \delta + \frac{\epsilon}{2} \tag{6}$$

Given the original set of points it is possible to reconstruct the statistical model that was used to represent it. So to communicate the collections, all that is required is the mean position represented to an accuracy of ϵ. The points are represented as a description of which collection they belong to and the offset from the mean: $< C_i, \delta >$.

As the data points in a corpus are divided up into smaller collections the description length of the individual points is

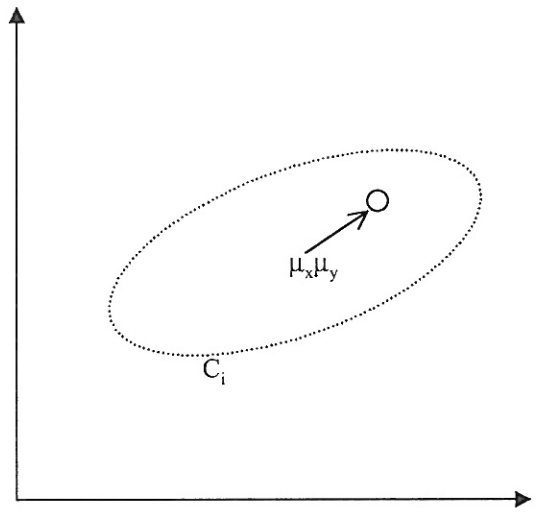

Fig. 3. Representation of a Point in a Collection

reduced if the distribution that characterizes the collection is more predictive about the position of its component points than the distribution for the entire corpus was. Any suitable statistical distribution can be chosen for a collection.

VII. Conclusion

The GRAVA system effectively uses models based diagnosis of context models to adapt to changing environments. This adaptation has been demonstrated in aerial image interpretation as well as in face recognition applications.

So, in addition to knowledge of content (faces or aerial images), that the GRAVA agents use to interpret images, GRAVA brings to bear two other kinds of knowledge. One of these is knowledge of the contexts that can be presented by the environment (contextual awareness), and the other is self-awareness, or meta-knowledge about the state of the program and the agents. What is unique about the recognizer outlined above is that it has multiple ways of interpreting images. For example, in face recognition, it divides up a complex space of lighting, age, race, sex, and pose into contexts that can be composed in a huge number of ways and self-adapts the recognizer at runtime.

In addition, we have developed a novel algorithm for the decomposition of complex models into collections of simpler models. This forms a backbone mechanism for automatically interpreting corpora, and automatically building both contexts and code for diagnosing context changes.

Since contexts are not random but are structurally related, transitions between contexts can be modeled as hidden Markov models (HMM). We are currently extending the architecture described in the paper to use HMM reasoning to optimize the context switching mechanism.

VIII. Acknowledgements

Effort sponsored in part by MIT Project Oxygen, and the Project Oxygen partners: Acer, Delta, Hewlett-Packard, Nokia, NTT and Philips.

References

[1] L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process. *Inequalities*, 3:1–8, 1972.

[2] I. Ben-Shaul, H. Gazit, O. Holder, and B. Lavva. Dynamic self adaptation in distributed systems. In R. Laddaga P. Robertson and H. Shrobe, editors, *Self-Adaptive Software*, pages 134–142. Springer-Verlag, 2000.

[3] R. A. Brooks. The intelligent room project. In *Proceedings of the Second International Cognitive Technology Conference (CT'97), Aizu, Japan*, 1997.

[4] E. Charniak. *Statistical Language Learning*. MIT Press, 1993.

[5] E. Charniak. Statistical techniques for natural language parsing. pages 33–43, 1997.

[6] B. Draper, R. Collins, J. Brolio, A. Hansen, and E. Riseman. The schema system. Technical Report COINS TR88-76, Computer and Information Science, Univ. Massachusetts at Amherst, 1988.

[7] J.E. Jackson. *A user's guide to Principal Components*. John Wiley and Sons, New York, 1991.

[8] F. Jelinek, J.D. Lafferty, and R.L. Mercer. Basic methods of probabilistic context-free grammars. In Pietro Laface and Renato De Mori, editors, *Speech recognition and understanding. Recent advances, trends, and applications, volume F75*. NATO ASI Series. Berlin: Springer Verlag, 1992.

[9] G. Karsai and J. Sztipanovits. A model-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):46–53, May/June 1999.

[10] M.M. Kokar, K. Baclawski, and Y.A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14(3):37–45, May/June 1999.

[11] R. Laddaga. Self-adaptive software sol baa 98-12. 1998.

[12] R. Laddaga. Creating robust software through self-adaptation. *IEEE Intelligent Systems*, 14(3):26–29, 1999.

[13] M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[14] D.J. Musliner, R.P. Goldman, M.J. Pelican, and K.D. Krebsbach. Self-adaptive software for hard real-time environments. *IEEE Intelligent Systems*, 14(4):23–29, July/August 1999.

[15] G. Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi. "meta-modeling rapid design and evolution of domainspecific modeling environments". In *Proceedings of the IEEE Conference and Workshop on Engineering of Computer Based Systems*, 1999.

[16] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[17] P. Robertson. A corpus based approach to the interpretation of aerial images. In *Proceedings IEE IPA99*. IEE, 1999. Manchester.

[18] P. Robertson. An architecture for self-adaptation and its application to aerial image understanding. In R. Laddaga P. Robertson and H. Shrobe, editors, *Self-Adaptive Software*, pages 199–223. Springer-Verlag, 2000.

[19] P. Robertson. *A Self-Adaptive Architecture for Image Understanding*. PhD thesis, University of Oxford, 2001.

[20] P. Robertson and J.M. Brady. Adaptive image analysis for aerial surveillance. *IEEE Intelligent Systems*, 14(3):30–36, May/June 1999.

[21] P. Robertson and R. Laddaga. Principle component decomposition for automatic context induction. In *Proceedings Artificial and Computational Intelligence 2002, Tokyo, Japan*, 2002.

[22] P. Robertson and R. Laddaga. A self-adaptive architecture and its application to robust face identification. In *Pacific Rim Conference on Artificial Intelligence 2002*. Springer-Verlag, 2002.

[23] A.J. Viterbi. Error bounds for convolution codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[24] C.S. Wallace. Classification by minimum-message-length inference. In G. Goos and J. Hartmanis, editors, *Advances in Computing and Information–ICCI'90*, pages 72–81. Springer-Verlag, 1990.