Perceptually Enabled Software

A White Paper for the Workshop on New Visions for Software Design and Productivity

By

Paul Robertson and Howie Shrobe

MIT AI Lab
NE43-804
200 Technology Square
Cambridge, MA 02139

617 253 xxxx

paulr@ai.mit.edu
hes@ai.mit.edu

Our vision of ubiquitous computing is that our environment will be heavily populated by devices under computer control, and that these devices will be interconnected and interfaced with each other and with us. Many of the devices under computer control will be sensors that provide a window from the world of interconnected computation into the physical world around us. Our view of ubiquitous computing makes it clear that moving ahead in the sensor, effector arena is not sufficient, we also need our computers to be able to dialogue, and understand both command and context. We understand our world, and communicate with each other via seeing, hearing, speaking and gesturing. If we are not to become enslaved by our ubiquitous computing environments, they will need to communicate with us in the same manner, and understand the context in which such dialogues occur. We focus on what is required of software development tools and methods for building the comprehensive vision systems of the embedded computing future. In particular, we present an example of a self-adaptive software architecture for vision systems, developed by the first author while at Oxford.

We are swiftly moving to a world of ubiquitous computing. Each year we produce more than one processor chip for each person on the planet, and the rate of growth of chip production exceeds the population growth rate. So, by some definition of ubiquitous computing it surely is upon us. For some, though, ubiquitous computing means that we all carry devices that enable us to connect to the Internet wherever we happen to be. Although it does seem that this will happen, it is not what we mean by ubiquitous computing.

Our vision of ubiquitous computing is that our environment will be heavily populated by devices under computer control, and that these devices will be interconnected and interfaced with each other and with us. Many of the devices under computer control will be sensors that provide a window from the world of interconnected computation into the physical world around us. Today, cars are run by microcontrollers that can sense acceleration, engine performance, breaking action, etc. Tomorrow's cars will have even more processors, connected with each other, sensing ambient temperature, weather conditions, other cars, roadside facilities, and communicating with other cars, traffic controllers, roadside facilities, and with us.

The major importance of ubiquitous, pervasive, connected computing will be the ability for computationally empowered devices to sense the world around us, and to respond by changing that world. There will be a tremendous need for these devices to take direction, explain their behavior, and modify their behavior in accord with our directions. They won't need to communicate with us all the time, but they will always need to be able to communicate in those instances when they are unsure or we wish to override. Needless to say, it won't work (for a variety of reasons) to communicate with these devices via a keyboard on our person, or dangling from the device. Our machines will need to use vision and auditory capability to see and hear what is going on in the background, in order to establish context, and will need to understand and generate speech to engage in dialogue with humans.

Vision for Natural Perceptual Context

Our view of ubiquitous computing makes it clear that moving ahead in the sensor, effector arena is not sufficient, we also need our computers to be able to dialogue, and understand both command and context. We understand our world, and communicate with each other via seeing, hearing, speaking and gesturing. If we are not to become enslaved by our ubiquitous computing environments, they will need to communicate with us in the same manner, and understand the context in which such dialogues occur. We focus on what is required of software development tools and methods for building the comprehensive vision systems of the embedded computing future.

The state of the art for vision research is that we can recognize some gestures and faces in real time, but general visual context recognition is still elusive. It is instructive to

consider in more detail the case of computer vision research, to see where we have come to and how we got there.

There has always been a desire to simplify problems. In the early days this was dealt with by building toy world problems. Toy world problems fell out of favor, and rightly so, when it became clear that solutions to toy world problems wouldn't scale up to solving real problems. For example the early work on understanding images (Waltz 1975, Clowes 1971, Huffman 1971) attempted to make sense of images that consisted of blocks. Rules for line junction interpretation allowed the lines in the images to be understood as three-dimensional blocks. Unfortunately the lines could not always be found in the images unless they were specially enhanced. The notion that the prerequisite lines could be found turned out not to be the case. Not all boundary lines are visible and of course not all objects in the world are blocks. As analytically interesting as these pieces of work were and as influential as they were in helping to understand issues such as propagation of constraints they would not directly lead to a solution to the computer vision problem.

Since then decades of research focused on low-level vision has yielded many important algorithms for solving a wide range of important parts of the computer vision problem that can be applied to non-toy problem areas. Examples include algorithms for computing stereo disparities (Marr 1979), algorithms for extracting structure from motion (Hildreth 1984), and algorithms that enable us to analyze and represent the rich textures found in the real world (Geman 1984).

In spite of the wealth of contributions in these areas and in low-level vision in general, advances in high level visual interpretation has been slow to develop. Success stories in computer vision are few and far between. Computer vision that works is pretty much restricted to problem domains that have been carefully constrained. The lure of the toy world or the artificially constrained world is that by carefully controlling the complexity of the environment which our programs must operate in we can construct complex algorithms that can perform reasonably well. This is true not just of computer vision but also for speech recognition and even robot planning. Dealing with the complexity of the real world is unavoidable in the embedded computing domain. The 21st Century will bring a proliferation of devices with embedded cameras. Some of these cameras will be embedded in robots, some in buildings, and some in vehicles. These cameras will often be deployed in environments that we cannot control. The challenge will be to take what we have learned about low-level vision and to architect visual interpretation systems that can reason about the visual interpretation task at hand and about the world that it is operating in in order to deliver robust visual interpretations. The new systems will have to *respond* to the world not just react to it. Model induction will surpass hand tailored models of the world as the unrestricted world represents a large challenge compared to the restricted domains in which we have largely operated to date. Intelligent data-fusion will grow in importance as devices proliferate that support multiple sensory modalities.

GRAVA

The goal of the architecture is to support self-adaptation. The idea behind self-adaptation is simple. The program needs to be able to continually access how well it is doing at its task rather than running an algorithm blind as is usually the case in non-adaptive programs. When the self-assessment determines that it is doing poorly the program seeks a 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:

Self-assessment---the ability of a computational agent to evaluate how well it is doing at its current task. Self-assessment is not so much a component as it is a protocol. The GRAVA architecture provides a protocol for supplying self-assessment functions.

Structure building---the mechanism that constructs a program from a collection of computational agents that implements the overall objective of the program. This structure building apparatus is invoked whenever self-adaptation becomes necessary. When self-assessment indicates poor performance the system tries to improve by re-synthesizing its program code.

Reflection---the support for self-understanding within the system. Reflective systems are systems that contain an embedded semantic account of their computational processes to some level of detail allowing introspection of the programs state and also semantic modification by mutating the semantic account. The embedded semantic account is not just a static representation but is intimately involved with the operation of the system. By inspecting the state of the semantic account the system can *understand* why it is doing what it is doing.

To date, reflection has existed as a means whereby a programmer can gain access to the computational state of the program. Sometimes this access is purely introspective. It is easy to implement a debugger on top of a reflective system. In other cases, the semantics may be extended or modified by the programmer.

The role of reflection in the GRAVA architecture is to allow the system to modify itself. The idea in this case is that if the program can know why it is doing what it is doing and also that it is not doing very well in some aspect of its computation, then the system could in principle adjust itself to do what it is doing in a different way and perhaps be more successful. If the system knows why it is doing the thing that it is not doing very well at, it may be able to find other ways of achieving what it was trying to do. If no other ways can be found, it will fail at a meta-level, causing a meta-level reorganization. The way that these goals are achieved in GRAVA is by having the meta-level goal of the program be described in some form of specification. Agents are provided that interpret that specification and produce a design for a program that would satisfy it. Agents that are built to interpret those parts of the design are then used to interpret the design in the form of a program. The number of levels of *meta* that lay between the meta-goal of the system at the top and the program code at the bottom are arbitrary. When the ultimate program code is run it interprets the image in order to produce its description. This

arrangement generates a tower of interpreters. At each point in the decomposition from meta-goals to image interpretation the components at one level are linked to those at a higher level that governed their semantics.

The GRAVA architecture described above has been successfully used to implement a satellite image interpretation system that self-adapts to changes in the scenes that it is interpreting.

Conclusions

Computer Vision is useful as an example in this paper because it provides a glimpse of the kind of computing environment that will likely be prevalent in the future. In the future *most* computation will involve interactions with a complex environment for which current software development practices are wholly inadequate. GRAVA represents the first attempt at finding a new paradigm for software capable of providing robust performance in complex embedded applications. The future of computing will be less about building large monolithic software systems that try to cover all eventualities and more about developing points of capability and specifications for behavior and allowing the capabilities to be composed automatically and recomposed dynamically in response to changing conditions. Much more research is required to understand how to build systems in this way and to develop tools and methodologies for their construction.

References

Clowes M. (1971) On Seeing Things, Artificial Intelligence (2) pp 79-116.

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Trans. Pattern Analysis and Machine Intelligence (6) pp 721-741.

Hildreth, E. C. (1984) Computations Underlying the Measurement of Visual Motion, Artificial Intelligence (23#3) pp 309-354.

Huffman D. (1971) Impossible Objects as Nonsense Sentences, Machine Intelligence (6) pp 295-323.

Karsai, G. and Sztipanovits, J. (1999) A Model-Based Approach to Self-Adaptive Software, IEEE Intelligent Systems (14#3 May/June) pp 46-53

Marr, D. and Poggio, T. (1979) A computational Theory of Human Stereo Disparity, Science (194#4262) pp 283-287.

Waltz D.L (1975) Understanding Line Drawings of Scenes with Shadows in P. H. Winston ed. Psychology of Computer Vision pp 19-91, McGraw-Hill New York.