GHOST: Guided Healing and Optimization Search Technique for Healing Large-Scale Embedded Systems

Steve Nordstrom, Abhishek Dubey, Turker Keskinpala, Sandeep Neema, Ted Bapty
Institute for Software Integrated Systems
Vanderbilt University, USA
{steve.nordstrom, abhishek.dubey, turker.keskinpala, sandeep.k.neema, theodore.bapty}@vanderbilt.edu

Abstract

Reflex and Healing architectures have been shown to provide adequate user-defined initial failure mitigation behaviors in the presence of system faults. What is lacking, however, is a user-guided means of healing the system after the initial reflexes have been enacted. This process should be autonomic in the sense that new system configurations can be achieved by defining a priori only a small set of criteria to which the healed system should conform. What follows is an explanation of this technique for guided healing which allows system designers to direct the healing process from a higher level in such a way that the resulting system configurations satisfy their particular needs. A brief example outlining the application of this approach is given.

1. Introduction

Large-scale embedded systems (systems which typically consist of thousands of specialized processing nodes) continue to prove themselves to be difficult to design, build, and manage. System developers and integrators increasingly desire the inclusion of more autonomic behavior [8] into their systems which can serve to alleviate design- and run-time problems.

High-performance computational systems used to process data from large-scale high energy physics (HEP) experiments [9] are expensive and prone to failure. Long experiment lifetimes and increased usage of commodity components lead to higher probabilities of component and subsystem failure during the operational lifetime of the system.

Crucial to the operability of these large-scale computation systems is their ability to adapt to changing environmental conditions in a manner which is deemed appropriate by *both the system architects and the users*.

Previous work has been done which leverages the concepts of Model Integrated Computing (MIC) [4] to define modeling languages and computer architectures which are capable of fast reactive behavior when failures occur within the system [5]. The purpose of using domain-specific modeling languages is to allow the designer a means to express the system using concepts in which she is familiar and can understand. Other efforts have included model based system adaptation techniques to fix problems due failure in running systems [6, 7, 13, 14]. Efforts for providing autonomic behavior in other scientific applications have also been explored [10, 11].

What is also needed, however, is an autonomic means for large-scale real-time systems to adapt to changing environmental conditions over longer periods of time which is guided not only by the system designer but also by the user of the system. Just as domain-specific modeling languages are tailored for the application domain, so should the means by which these systems heal themselves. The work described in this paper is motivated by the need to allow the users of the system to guide the long term manner in which the system will recover from more systemic failures which occur over the course of the system lifetime.

2. Reflex and Healing management structures

Management structures typically used to regulate large numbers of online processes or tasks are often hierarchical in nature and can be likewise be visualized as a tree structure. Our current Reflex and Healing (RH) architecture [5, 15] is an example of one such management architecture which uses a tree-like

Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)
0-7695-2544-X/06 $20.00 © 2006 IEEE

structure to define the relationship of manager processes throughout a large-scale embedded system and is shown in Figure 1. Some details regarding the management of processes in an RH network are as follows:

- Managers are only responsible for those subordinates which are directly beneath them
- Direct communication between peer managers is strictly disallowed
- Reflexes are defined as behaviors in which inter-managerial interactions or interactions between managers take place
- Any further optimization which requires more global knowledge or an interaction across a managerial boundary is done under the hospice of *healing*.

RH structures have two attributes which help to characterize their structure:

- Order o: The number of levels in the tree; this is sometimes referred to as the "height" or "depth" of the tree structure, inclusive of the highest or global level manager.
- Branching factor β: The number of children of a given parent node in the tree structure.

Any RH tree structure can be classified as having a homogeneous or heterogeneous branching factor. For the purposes of this paper we will consider trees which have uniform or homogeneous branching factors.

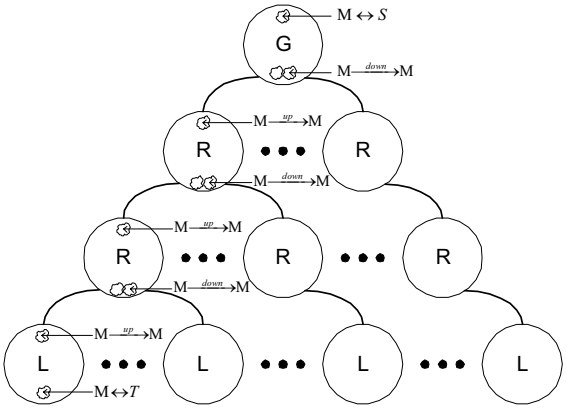

Figure 1. A typical hierarchical layout of RH management entities involves global (G), regional (R) and local (L) management entities.

There exist the following four types of management interaction behaviors for RH managers, each of which is based on the participating parties of the interaction:

1) $S \leftrightarrow M$: Interaction between an operator/system interface and a manager
2) $M \xrightarrow{up} M$: Interaction between a manager and a governing manager
3) $M \xrightarrow{down} M$: Interaction between a manager and subordinate managers
4) $M \leftrightarrow T$: Interaction between a manager and a managed task

A Global Manager G is defined as a manager which is capable of both $S \leftrightarrow M$ and $M \xrightarrow{down} M$ behavior. A Regional Manager R is defined as any manager which is capable of both $M \xrightarrow{up} M$ and $M \xrightarrow{down} M$ behaviors. Finally, local managers L are managers which are only capable of $M \xrightarrow{up} M$ and $M \leftrightarrow T$ behaviors. It is important to note that only local managers are capable of $M \leftrightarrow T$ interactions, and only global managers are capable of $S \leftrightarrow M$ interactions.

3. Healing RH structures

We choose to categorize conditions which require a healing action to be performed within an RH structure as being of two varieties, 1) the reduction of resources caused by hardware faults in the system or 2) a reduction of tasks caused by faulty software. The term *resource* can be used to describe what a software process may need to execute successfully (processors used to perform computation, disk-based storage facilities, etc.). Similarly, a *task* is any piece of software that may be executed. An example of a reduced resource condition might be the overheating of a machine in the data processing portion of the network, while an example of a reduced task condition would occur when a persistently unruly group of software processes are disabled by a reflex behavior.

System healing transformations can therefore belong to one of two classes: reduced-resource (RR) or reduced-task (RT). In RR, static or persistent faults occur which diminish the availability of resources needed to complete the given task(s). In this case, the RR transformation should reorganize the software structure such that the new set of tasks can complete on the diminished resources. For RT cases, however, the RT transformation must first expand the task model then remap this new task set onto the existing resources.

3.1 GHOST

The Guided Healing and Optimization Search Technique (GHOST) provides a method by which an appropriately healed system model can be achieved whenever a healing action is triggered by the management system. The technique is defined as a new transformation from either the old task model to a new resource model, or from a new task model to the old resource model.

In the ideal case, a model of an embedded system is realized by mapping elements of a task model T_M to elements in the resource model R_M using a given transformation m.

$$T_M \xrightarrow{m} R_M$$

To perform healing, GHOST will reduce or expand the task model in an attempt to map it onto the reduced or otherwise unchanged resource model. We will first consider a case where a reduced resource condition is healed.

When a resource fault f_r occurs, a reductive change occurs to the available resources and accordingly to the resource model.

$$R_M \xrightarrow{f_r} R'_M$$

GHOST is triggered by this reduced resource condition and applies a reductive transformation to the old task model, given the set of transformation criteria $\{c_k\}_{k=1}^{n}$ which are defined in the following section.

$$reduce\left(T_M, \{c_k\}_{k=1}^{n}\right)$$

The result is a new task model which has been derived through this reductive process.

$$T_M \xrightarrow{reduce} T'_M$$

The newly optimized system is therefore derived by mapping the reduced task model onto the currently reduced resource model using the known transformation m,

$$T'_M \xrightarrow{m} R'_M$$

thus healing the system.

Now let us consider a reduced task condition where a persistent software failure has triggered GHOST to perform healing. A task fault f_t occurs, which causes a reductive change to the number of tasks currently running in the system. This can be considered a new task model where the faulty task is missing.

$$T_M \xrightarrow{f_t} T'_M$$

In this case, GHOST is triggered by the detection of a faulty task and attempts to apply an expansive transformation to the new task model, given the same set of transformation criteria $\{c_k\}_{k=1}^{n}$

$$expand\left(T'_M, \{c_k\}_{k=1}^{n}\right)$$

to find a new task model T''_M.

$$T'_M \xrightarrow{expand} T''_M$$

The newly optimized system is therefore derived by mapping the expanded task model onto the original resource model using the known transformation m.

$$T''_M \xrightarrow{m} R_M$$

3.2 Appropriate healing actions

In order to avoid a veritable explosion of candidate task models only a limited set of operations must be allowed for reducing or expanding the task model. These operations are limited to the following:

- Addition or deletion of a manager from a level of the tree
- Addition or deletion of an application from a local manager's managed process group (for the purposes of this paper a single application will be managed by a local manager on the same resource)
- Reconfiguration of a manger's $M \xrightarrow{up} M$ interaction to specify a new governing manager
- Promotion or demotion of a manager via removal or activation of one or more management behaviors (this operation may also incorporate a composition of the previous operations)

Overall, the concerns of both the systems designers (experts in large-scale embedded systems) and the users (experts of the application domain) must *both* be satisfied by the healing operation. Therefore, it is desirable to construct a procedure for guiding the

healing process of the RH structure which takes in to account the (sometimes conflicting) desires of both groups.

To do this a set of performance criteria are defined which are used to evaluate the system model at any given stage of the healing process. In order to fully capture the nature of the healing process we must not only define these criteria but also a means of prioritizing them. The following rules are in place for creating priorities within the performance criteria:

1) Each criterion in the set $\{c_k\}_{k=1}^{n}$ is assigned a real-valued weight on the closed interval $[0,1]$.
2) The criteria will be ordered highest to lowest such that $c_1 \geq c_2 \geq \cdots \geq c_n$.
3) Each criteria must have a set of actions which, when applied to the RH structure, are known to increase of decrease the evaluation of the criteria (e.g. defining a criteria for "managerial latency" must also specify that the addition of behaviors to a manager is a known means to increase the reaction-time latency for any behavior of that manager, and reducing the number of behaviors will correspondingly decrease managerial latency).

The GHOST process is therefore responsible for finding an appropriate system model which satisfies the performance criteria with respect to their assigned weights.

4. Examples of system healing

To demonstrate the process of healing RH networks we will define a simple RH structure, and show how GHOST can heal the structure in the presence of faults, given a set of weighted criteria for evaluating the candidate structure.

4.1. Resource faults

As an example, let us consider a simple, fully populated, uniform RH structure of order 3 and with a branching factor of 2. This structure can be expressed as $RH\{o(3),\beta(2)\}$ and is illustrated in Figure 2. Also consider that a group of seven machines can be used to deploy the RH system. For this simple example we can treat each machine as a resource, numbered one through seven.

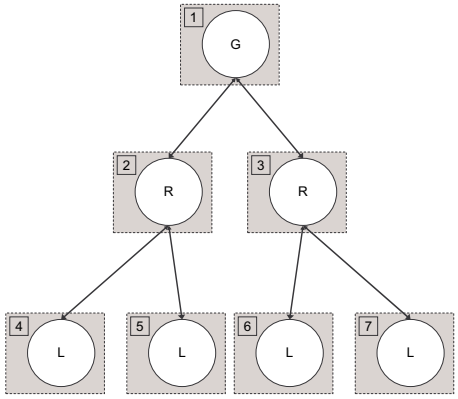

Figure 2. A healthy RH structure with order 3 and a branching factor of 2, with all tasks mapped to a unique resource.

In order to heal this structure when resource failures occur, we must also define our set of weighted performance criteria $\{c_k\}_{k=1}^{n}$ by which we can evaluate candidate healed structures. The following criteria will be used to guide the healing and optimization process:

c1: *Throughput* – the quantity of processing time allocated to the managed application (each local manager is assumed to manage a single application)

c2: *Robustness* – the resulting structure's ability to withstand future faults

c3: *Similarity* – a differential relationship to the previously planned system model (this measure is used to ensure that the time and resources taken to enact the optimization will be minimized)

For the sake of brevity and to keep the example simple, only three criteria are considered. Other examples of performance criteria might include *management latency* which relates the latency or response time for a manager to the number of subordinate managers for which is it responsible, *symmetry*, which ensures that the branching factors between levels of the hierarchy are uniform, and *system utilization*, as it may be possible to rearrange the hierarchy such that machines with more computational power are allocated to managed applications and not higher level management.

For each criteria detailed, it is necessary to list the means by which GHOST will heal the system in an attempt to increase or decrease the evaluation of the system with respect to the given weighted criterion. A representative set of goals for each criterion are provided as follows:

g1: Distribute tasks across all available resources, until all tasks have been assigned. If after all tasks are assigned resources are left unallocated, expand the task model to add local managers and managed applications in order to fully utilize the remaining resources.

g2: Minimize single points of failure to ensure that a level of redundancy is available at each management level. Avoid RH structure adaptations which leave only one remaining manager at any given level.

g3: Maximize the similarity between successive models. This criteria is subject to various interpretations, as the term "similarity" is not formally defined with respect to RH tree structures, although similarity measures for other tree structures have been explored in [1, 2, 3]. (It is our intention to use the similarity of tree structures not to measure the difference in shape or structure of candidate trees, but instead to measure the cost of changing from one tree structure to another.)

Clearly, these criteria can be conflicting. When also considering that some criteria are defined by the system experts and others are specified by the user it becomes clear that satisfying all of these constraints may not be feasible. In the most extreme cases the criteria can be in direct opposition to each other. Therefore it becomes necessary to utilize the weights of the criteria by which GHOST uses to heal the system. The effects of this prioritization can be profound; the weights of each criterion should therefore not be chosen lightly. It can be shown even in the simple examples given just how these weights can affect the outcome of the GHOST calculation.

4.2. Optimizing for throughput

Consider again the example of the fully populated and uniform RH structure shown in Figure 2. Also assume the three performance criteria which have been defined are assigned the following weights,

$$\{c_1(0.6), c_2(0.3), c_3(0.1)\}$$

maximizing system throughput.

A fault f_{r1} occurring on resource 7 is detected as shown in Figure 3 (rendering that hardware resource useless) and a healing action is triggered. The resulting candidate model is shown in Figure 4 which maximizes c_1 at the expense of c_2 and c_3 by demoting a regional manager to a local manager while allowing the missing managed application in the system to be replaced. However, while the resources are more fully utilized, there now exists a single regional manager, constituting a single point of failure. The model can also be considered dissimilar to the original (with respect to uniformity of branching).

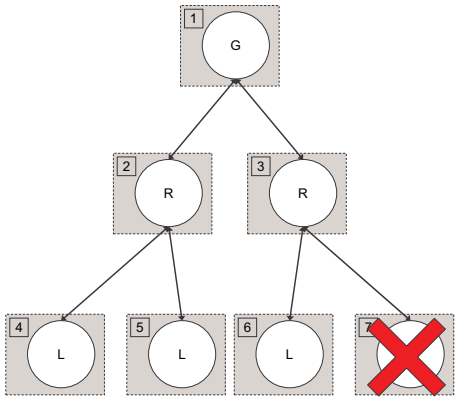

Figure 3. A fault occurs on node 7, affecting the local manager and all tasks assigned to that resource.

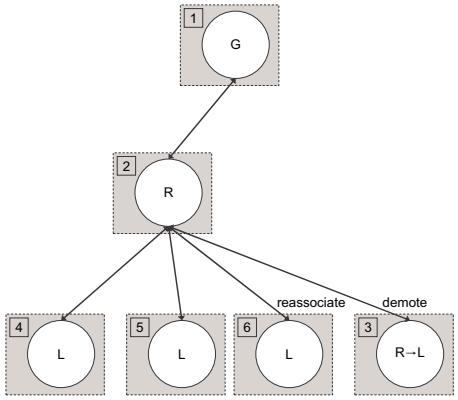

Figure 4. A healing action takes place which maximizes the throughput criteria in response to a failure on resource 7.

Consider the same initial RH structure and criteria in which a fault f_{r2} occurs on resource 3 where the regional manager resides (shown in Figure 5) The resulting candidate model is shown in Figure 6 which again maximizes c_1 at the expense of c_2 and c_3.

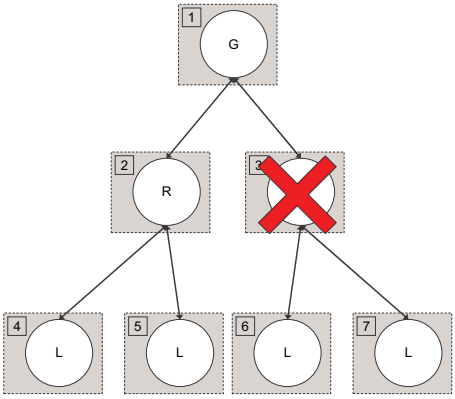

Figure 5. A fault occurs on resource 3 removing the regional manager.

An interesting note to these healing actions is the similarity of solutions provided by the technique given differing faults throughout the system. This hints to the importance of weighing heavily the similarity criterion (which in this example is of a low priority). Doing so would also prevent a rapid divergence from the original design toward a small set of (non-similarity criteria maximizing) solutions which may not in any way resemble the originally designed system.

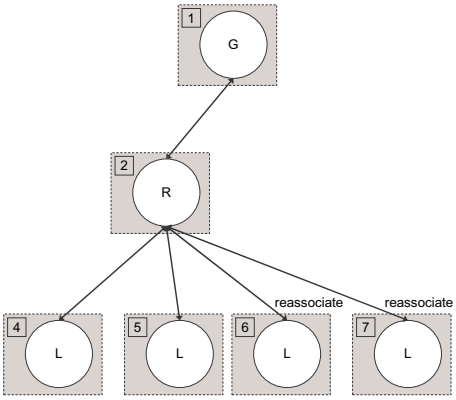

Figure 6. A healing action optimized for throughput results in a single regional manager associated with four local managers.

4.3 Optimizing for Similarity

Let us now see what results by rearranging the weights of the criteria in the following manner, which maximize the weight of the similarity criterion.

$$\{c_1(0.1), c_2(0.3), c_3(0.6)\}$$

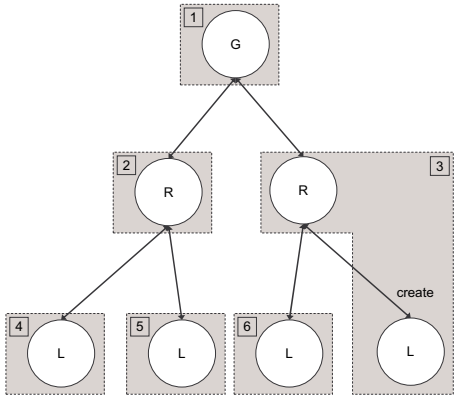

Figure 7. The original fault on resource 7 healed according to the similarity criterion.

Just as before, when fault f_{r1} occurs on resource 7 the result is a loss of a local manager and corresponding managed application. In this case, however, the resulting candidate provided by GHOST (shown in Figure 7) is determined to be most similar to the original model. This case is rather interesting as the course of action taken is to simply reassign the local manager to an existing resource. Clearly in this case the throughput suffers as a local manager, a managed application, and a regional manager are now all competing for resource 3.

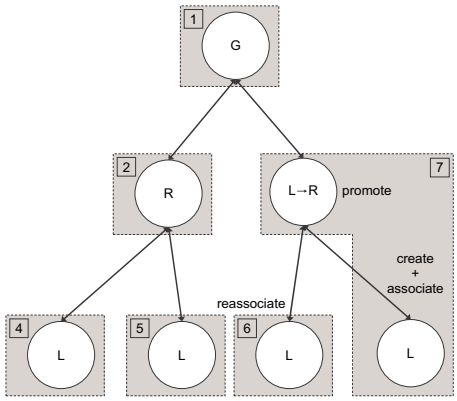

Figure 8. The original fault on resource 3 is healed in a similar fashion according to the similarity criterion.

Again if we consider fault f_{r2} occurring on resource 3 an interesting facet of the GHOST technique can be seen (refer to Figure 8). In this case an orphaned local manager is promoted to a regional manager and the other local manager in the sub-tree is placed under its supervision. In this case GHOST

provides a solution similar to that of a failed local manager, effectively rearranging the available resources. Clearly in this case throughput is also not maximized.

5. Conclusions and Future Work

It can be seen that GHOST is a viable means for giving expert embedded system designers and non-expert users input as to how *RH* architectures should be healed when they fail. This technique places considerable emphasis on the weights of the criteria defined by which candidate *RH* structures are evaluated.

Techniques for healing both real-time and time-unbounded systems are needed to ensure reliable operation throughout a system's lifetime, and this method shows promise in both regards. However, these techniques should be flexible enough to allow criterion specification by individuals whose expertise lies outside the areas of real-time or large-scale autonomic and embedded systems.

Further study into the effects of the weights of a given criterion under identical fault scenarios is in order. There is also a need to more rigorously define the similarity criterion for *RH* tree structures. This will involve future work to provide a modeling and simulation environment with which to test candidate solutions provided by an automated GHOST algorithm. The author is confident that more issues regarding the inability to satisfy conflicting criteria will arise, and that a more refined means of finding "best approximation" candidate solutions can be achieved.

This work is supported by NSF under the ITR grant ACI-0121658. The authors also acknowledge the contribution of other RTES collaboration team members at Fermi Lab, UIUC, Pittsburg, and Syracuse Universities.

6. References

[1] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," Proceedings of the 23rd VLDB, August 1997.

[2] S.A. Nene, S.K. Nayar, "A Simple Algorithm for Nearest Neighbor Search in High Dimensions", IEEE Trans. Pattern Anal. Machine Intell., vol. 19, 989-1003, 1997.

[3] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin, "Searching in metric spaces", *ACM Computing Survey*, Vol 33, No. 3, pp 273-321, ISSN-0360-0300, ACM Press, New York, NY, USA, 2001.

[4] J. Sztipanovits, G. Karsai, "Model-Integrated Computing", *IEEE Computer*, pp. 110-112, April, 1997.

[5] S. Neema, Ted Bapty, S. Shetty, S. Nordstrom, "Developing Autonomic Fault Mitigation Systems", *Journal of Engineering Applications of Artificial Intelligence Special Issue on Autonomic Computing and Grids*, Elsevier, 2004.

[6] P. Broadwell, N. Sastry and J. Traupman., "FIG: A Prototype Tool for Online Verification of Recovery Mechanisms", Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), New York, NY, June 2002.

[7] D. Garlan, B. Schmerl, "Model-Based Adaptation for Self-Healing Systems", Workshop on Self-healing systems (WOSS), *Proceedings of the first workshop on Self-healing Systems*, Charleston, South Carolina, 2002.

[8] R. Sterritt, "Autonomic Computing", *Innovations in Systems and Software Engineering, A NASA Journal*, vol. 1, No.1, ISSN-1614-5046, Springer, April 2005.

[9] J. Gutleber, et. al, "Clustered Data Acquisition for the CMS experiment", International Conference on Computing in High Energy and Nuclear Physics (CHEP 2001), Beijing, China, September 3-7, 2001.

[10] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and S. Hariri. Automate: Enabling autonomic grid applications. Cluster Computing: The Journal of Networks, Software Tools, and Applications, Special Issue on Autonomic Computing, 9(1), 2006.

[11] H. Liu and M. Parashar. Rule-based monitoring and steering of distributed scientific applications. *International Journal of High Performance Computing and Networking*, 2005.

[12] M. Agarwal, V. Bhat, Z. Li, H. Liu, B. Khargharia, V. Matossian, V. Putty, C. Schmidt, G. Zhang, S. Hariri, and M. Parashar. "Automate: Enabling autonomic applications on the grid". *Proceedings of the Autonomic Computing Workshop, 5th Annual International Active Middleware Services Workshop* (AMS2003), pages 48–57, June 2003.

[13] B. Williams, M. Ingham, S. Chung, P. Elliott, M. Hofbaur, and G.T. Sullivan. "Model-based programming of faultaware systems". *AI Magazine*, 24(4):61–75, 2004.

[14] B. Williams and R. Ragno. "Conflict-directed a* and its role in model-based embedded systems". *Journal of Discrete Applied Math, Special Issue on Theory and Applications of Satisfiability Testing*, January 2003.

[15] S. Nordstrom, S. Shetty, S.K. Neema, and T.A. Bapty, "Modeling Reflex-Healing Autonomy for Large Scale Embedded Systems" *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Autonomic Computing*, to be published 2006.