Self-Healing and Recovery Methods and their Classification

Onn Shehory, Josu Martinez, Artur Andrzejak,
Cinzia Cappiello, Wlodzimierz Funika, Derrick Kondo,
Leonardo Mariani, Benjamin Satzger, Markus Schmid

Abstract

This document summarizes the results of the Working Group 1 - "Self-Healing and Recovery" - within the Dagstuhl Seminar 09201 "Self-Healing and Self-Adaptive Systems" (organized by A. Andrzejak, K. Geihs, O. Shehory and J. Wilkes). The seminar was held from May 10th 2009 to May 15th 2009 in Schloss Dagstuhl – Leibniz Center for Informatics.

1 Introduction

The self-healing and recovery work group has undertaken to examine a sample of techniques and technologies in the field and devise a taxonomy according to which these can be classified. Specifically, it was suggested that, at a high level, self-healing and recovery methods and architectures can be classified according to the following:

- Recovery action suggested by the method. Several such actions are listed in proceeding sections. At a high level, these can be classified into failure removal, alleviation, learning for future cases, etc.
- Preconditions and requirements subject to which the method is to be applied. For instance, some methods require specific instrumentation of the target system.
- The stage of system lifecycle in which the method is applicable. For instance, many methods apply to systems in production environments, whereas others may apply to coding and testing stages, or design time.
- Technology and language constraints. Some methods are applicable only to one language or only to one system architecture (e.g, Java and SOA), whereas others are more generic in scope.
- The type of failure addressed by the method. Some methods address functional failures whereas others address performance failures. Some are very specific, whereas others are very generic, etc.

Dagstuhl Seminar Proceedings 09201
Combinatorial Scientific Computing
http://drops.dagstuhl.de/opus/volltexte/2009/2108

- The effect of the method on the healed system and its environment. For example, some methods change the system's code thus affecting, e.g., actions scheduling.
- The ease of deployment and use of the method. For example, some methods may require expert knowledge to work with, whereas others may be accessible to novices. Some methods may work automatically or semi-automatically, while other may be operated manually.

We view the above classification as a rough starting point aimed at facilitating discussion on a self-healing taxonomy. It is by no means complete, and can surely be improved.

2 Self-Healing and Recovery Approaches

In what follows we provide descriptions of both generic approaches and specific methods for self-healing and recovery.

2.1 Computational Reflection and Control Loops

To be self-healing, a system must have some reflective capabilities [1]. First, it must be able to perform *introspection*. That is, it must constantly monitor the running state of the system to identify any anomalous behaviour during its operation. Second, whenever any failure is detected, it must execute some *intercession* actions, i.e., carry out a certain procedure to recover from the failure and repair faults without interrupting any of the functional services it provides, so that it returns back to a stable state. This generic approach to software self-healing is also widely in the field of control theory

In similarity to reflection, a large body of research on control theory [2, 3] has suggested that self-* properties, and in particular self-healing, can be modelled, and then implemented, as a control loop. Such a control loop includes a monitoring component, an analysis component, a decision making component, and an actuation component. Self-healing thus includes monitoring of the healed system, analysis of the monitored data to identify problems, decision upon the healing action to be performed, and performance of this action. The results of the action will feed into the monitoring component, thus closing the control loop, allowing feedback and improvement of the healing.

Examples of such loops can be found in multiple specific self-healing solution as well as a few generic architectures. For example, see the Panacea architecture in [4].

3 Existing Self-Healing Approaches

The contents of this section are a part of the doctoral thesis of one of the authors (Josu Martinez) and have been published in [5].

Most of the already existing self-healing solutions differ in the intercession mechanisms they use. Gosh et al. [6] surveyed in 2006 some of the strategies used by other researchers to accomplish failure recovery. However, some other techniques of interest were not taken into account in their study. The following list summarizes four generic categories of failure recovery identified so far and briefly exposes some of the reviewed techniques:

3.1 *Redundancy Techniques*

Nagpal et al. [7] suggest a self-assembly mechanism based on an agent entity that replicates components to replace dead neighbours and enables recomposition of entire structures. Another strategy inspired from biology is providing the system with the ability of replicating cells in excess to combat external intrusions [8]. Finally, one of the techniques used in Recovery-Oriented Computing (ROC) [9] consists on isolating faulty components and replacing them with redundant ones.

3.2 *Architecture Models and Policies*

Some component-based frameworks support interchangeable architectural styles to suit performance deviations. Two examples are Rainbow [10] and Madam [11] Rainbow statically associates a set of action rules for each of the pre-identified failure causes. Madam uses some utility functions to select the most suitable architectural variant to repair the fault.

Dashofy et al. [12] have developed an infrastructure that supports dynamic reconfiguration of connector links. They also provide a tool that enables the system to merge the architecture description of the running system and the description of the architectural changes to be executed to effect the repair plan.

Huang [13] uses Java-like recipes to specify the different components and connections between them. Then, a synthesizer component analyses these recipes to decide which component and connections suit the environmental circumstances of the system.

Georgiadis et al. [14] posit that it is possible to dynamically bind components while fulfilling architectural constraints (akin to architectural styles) defined by human administrators. Each component has a manager part that automatically binds its required methods to remote interfaces of methods provided by other components at runtime.

De Lemos and Fiadeiro [15] suggest a framework that performs some dynamic reconfiguration of the system through atomic operations. This approach consists on isolating the failing component and substituting it with another that offers alternative services, even if in a downgraded mode, and adapting the connector among them so that the services provided by the new component satisfies the expectations of its consumers. A configuration layer defines rules for controlling the evolution of the system.

Some other approaches [16, 17] replace services or components by equivalent ones. In these systems failing components are substituted by others that provide

equivalent functionality. Some decision policies determine which alternative component replaces the original one.

Another type of replacement is hot-swapping of components [18], accomplished by inter-positioning of code or by replacement of code. Components are dynamically replaced depending on certain attribute values described by system administrators (e.g., size of accessed files) by optimized or newer ones. A mediator object swaps different implementations depending on the circumstances of the environment. This mediator is actually interposed between the two implementations. Interposition wrappers enable components to extend or modify their interfaces without requiring to rewrite any code.

Shin and Hoon [19] propose that each component of the system communicates with others using self-healing connectors similar to component ports. If one of the objects of any of these two entities fails a reconfiguration plan is constructed, all the objects associated with the anomalous object are blocked and the affected components notified so that they also generate a repair plan if needed. Once the paralysed objects are fixed, communications with other components are re-established.

More recent studies [20, 21] adopt Case-Based Reasoning (CBR) to provide failure recovery. The system collects symptoms of failures and stores them in a problem experience repository as case-solution pairs. The hypothesis of this technique claims that similar problems may be resolved by applying the same type of solution. Similarly to the previous approach, Littman et al. [22] use a reinforcement learning algorithm that enhances the efficiency in selecting the appropriate remedy actions that may heal the system from failures in its operation.

Fuad and Oudshoorn [23] have implemented a priority-based fault-action algorithm. In this approach all the faults are described by pre-conditions, remedial actions and post-conditions. Every remedial action has a priority level associated to it. This work also offers a way of dynamically providing self-healing capabilities to already built applications.

Kephart and Chess [24] suggest the use of negotiation theory. Violations of the Service Level Agreement (SLA) reached between two components are considered operational failures. Remedies to resolve this type of issue are various: asses a penalty, renegotiate the agreement, take technical measures or terminate the agreement.

3.3 *Component Micro-Rebooting*

Faulty modules are micro-rebooted independently and automatically to avoid fault propagation whenever they are suspected of not functioning properly [25, 26]. The efficiency of this technique resides on the fact that re-starting single components takes less time than rebooting the whole system. A hierarchy of different restart groups of components is created. This strategy allows the system to tolerate successive re-starts at multiple levels. Thus, rebooting a high level component in the hierarchy may take a longer time than re-starting a lower level component, but the recovery of the former is of a higher confidence.

3.4 *SOA-based Process Reorganization*

Service-Oriented Architectures (SOA) is a flexible coordination paradigm that enables components to export and discover services over the network [27]. Similarly to the approach presented in this document, the main purpose of SOA is to provide service publication, discovery, selection and binding [28]. As an example of a service binding mechanism, Baresi et al. [29] describe business processes using directed graphs, and propose a technique to apply transformation rules to autonomously modify the topology of the graphs at runtime. These rules transform a single node into a sequence of various nodes, into a parallel composition of two nodes, or into a branch. There are also rules to transform sequences, parallel compositions and branches into single nodes. To make it feasible, all the services have to be formally described using pre- and post-conditions.

4 Concluding Remarks

The self-healing and recovery workgroup has studied several methods. One of the results of this study was the recognition that the terminology is not well defined. Specifically, describing a method or comparing it to another is not a simple task. The time and effort afforded during the Dagstuhl seminar could not suffice to overcome this difficulty. However, by recognizing it we have set a research challenge to the reference community. We have also provided an initial, rough set of properties to be examined when classifying and comparing self-healing methods. We have further listed a sample set of technologies to be examined. We leave for future work the systematic classification of these methods and the development of a richer and finer self-healing taxonomy.

5 Future Plans

The self-healing and recovery workgroup has established the following goals for future activities: collect information on available self-healing solutions, case studies and benchmarks; classify these according to the taxonomy; write a survey paper using the collected and classified data.

To achieve these goals, the following actions are planned: set up a wiki hosting service, linked to the Dagstuhl seminar page; refine and extend the taxonomy dimensions and individual categories; add solutions and case studies as found and needed. Finally, these will serve as input for the planned survey paper.

References

[1] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H.C. Cheng, "Composing adaptive software", *IEEE Computer*, vol. 37, no. 7, pp. 56–64, July 2004.

[2] Yixin Diao, Joseph L. Hellerstein, Sujay S. Parekh, Rean Griffith, Gail E. Kaiser, and Dan B. Phung, "Self-managing systems: A control theory foundation", *ECBS*, pp. 441–448, 2005.

[3] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury, *Feedback Control of Computing Systems*, John Wiley & Sons, 2004.

[4] D. Breitgand, M. Goldstein, E. Henis, O. Shehory, and Y. Weinsberg, "PANACEA – Towards a Self-healing Development Framework", in *Proceedings of the 10th IFIP/IEEE international symposium on Integrated Network Management*, Munich, Germany, 2007.

[5] Josu Martinez, "Functionality recomposition-based self-healing", Ph.d. proposal document, University College Dublin, Ireland, November 2008.

[6] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya, "Self-healing systems - survey and synthesis", *Decission Support Systems*, vol. 42, pp. 2164–2185, 2007.

[7] Radhika Nagpal, Attila Kondacs, and Catherine Chang, "Programming methodology for biologically-inspired self-assembling systems", in *AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality*, March 2003.

[8] Selvin George, David Evans, and Steven Marchette, "A biological programming model for self-healing", in *SSRS'03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*, New York, NY, USA, 2003, pp. 72–81, ACM.

[9] Berkeley/Stanford, "Recovery-Oriented Computing (ROC)", http://roc.cs.berkeley.edu, 2008.

[10] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste, "Rainbow: Architecture-based self adaptation with reusable infrastructure", *IEEE Computer*, vol. 37, no. 10, pp. 46–54, October 2004.

[11] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven, "Using architecture models for runtime adaptability", *IEEE Software*, vol. 23, no. 2, pp. 62–70, 2006.

[12] Eric M. Dashofy, Andre van der Hoek, and Richard N. Taylor, "Towards architecture-based self-healing systems", in *WOSS'02: Proceedings of the first workshop on Self-healing systems*, New York, NY, USA, 2002, pp. 21–26, ACM.

[13] An-Cheng Huang, *Building self-configuring services using service-specific knowledge*, PhD thesis, Pittsburgh, PA, USA, 2004.

[14] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer, "Self-organising software architectures for distributed systems", in *WOSS'02: Proceedings of the first workshop on Self-healing systems*, New York, NY, USA, 2002, pp. 33–38, ACM.

[15] Rogério de Lemos and José Luiz Fiadeiro, "An architectural support for self-adaptive software for treating faults", in *WOSS'02: Proceedings of the first workshop on Self-healing systems*, New York, NY, USA, 2002, pp. 39–42, ACM.

[16] Nathan Combs and Jeff Vagle, "Adaptive mirroring of system of systems architectures", in *WOSS'02: Proceedings of the first workshop on Self-healing systems*, New York, NY, USA, 2002, pp. 96–98, ACM.

[17] Ada Diaconescu and John Murphy, "Automating the performance management of component-based enterprise systems through the use of redundancy", in *ASE'05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, New York, NY, USA, 2005, pp. 44–53, ACM.

[18] Jonathan Appavoo, Kevin Hui, Craig A. N. Soules, Robert W. Wisniewski, Dilma M. Da Silva, Orran Krieger, David J. Edelsohn, Marc A. Auslander, Ben Gamsa, Gregory R. Ganger, Paul McKenney, Michal Ostrowski, Bryan Rosenburg, Michael Stumm, and Jimi Xenidis, "Enabling autonomic behavior in systems software with hot-swapping", *IBM Systems Journal*, vol. 42, no. 1, 2003.

[19] Michael E. Shin and Jung Hoon An, "Self-reconfiguration in self-healing systems", in *EASE'06: Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems*, Washington, DC, USA, 2006, pp. 89–98, IEEE Computer Society.

[20] Stefania Montani and Cosimo Anglano, "Achieving self-healing in service delivery software systems by means of case-based reasoning", *Applied Intelligence*, vol. 28, no. 2, pp. 139–152, 2008.

[21] M.J. Khan, M.M. Awais, and S. Shamail, "Enabling self-configuration in autonomic systems using case-based reasoning with improved efficiency", March 2008.

[22] M.L. Littman, N. Ravi, E. Fenson, and R. Howard, "Reinforcement learning for autonomic network repair", *ICAC'04: Proceedings of the International Conference on Autonomic Computing*, pp. 284–285, May 2004.

[23] M. Muztaba Fuad and Michael J. Oudshoorn, "Transformation of existing programs into autonomic and self-healing entities", in *ECBS'07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Washington, DC, USA, 2007, pp. 133–144, IEEE Computer Society.

[24] J.O. Kephart and D.M. Chess, "The vision of autonomic computing", *IEEE Computer*, vol. 36, pp. 41–50, January 2003.

[25] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kiciman, Matthew Merzbacher, David Oppenheimer, Naveen Sastry, William Tetzlaff, Jonathan Traupman, and Noah Treuhaft, "Recovery oriented computing (roc): Motivation, definition, techniques and case studies", Tech. Rep., Berkeley, CA, USA, 2002.

[26] Jeffrey O. Kephart, "Research challenges of autonomic computing", in *ICSE'05: Proceedings of the 27th international conference on Software engineering*, St. Louis, MO, USA, 2005, pp. 15–22, ACM.

[27] Francesco Nachira, "Digital business ecosystems", http://www.digital-ecosystems.org/book/de-book2007.html, 2007.

[28] M. P. Papazoglou and D. Georgakopoulos, "Service-oriented computing", *Communications of the ACM*, vol. 46, no. 10, pp. 46–54, October 2003.

[29] Luciano Baresi, Carlo Ghezzi, and Sam Guinea, "Towards self-healing service compositions", in *PriSE'04: First Conference on the Principles of Software Engineering*, 2004, vol. 42, pp. 27–46.