How to Prevent Failure of the Healing Mechanism in Self-Healing Systems

Janne Metso
Department of Computer Science
University of Helsinki, Finland
Email: Janne.Metso@cs.Helsinki.FI

***Abstract*— Self-healing systems tackle the issue of ever increasing complexity of software and its management. Self-healing systems provide solutions for the software systems to manage themselves and recover from errors. There are multiple approaches to self-healing systems ranging from agent based solutions to frameworks. The self-healing process is usually enclosed into a separate layer or components in the system. The separate layer does system diagnosis and repair actions on the system. This layer can fail just as the application layer.**

In this paper I take a look into the problem of self healing layer failures. I will present two systems and discuss their differences and weak points. There is a couple of possibilities to improve the reliability of the healing layer. These are computational verification and common fault tolerance techniques. To minimise the possibility of failures on self-healing systems, either of these techniques can be used.

I. Introduction

Companies are more and more reliant on different types of software systems on their day-to-day activities. At the same time the systems used by these companies are more and more complex. This means that the companies need to have more and more staff to deal with possible problems in the systems. Self-healing systems try to answer the issue by providing software systems that are able to self diagnose the problem and correct it without the need of human assistance.

While self-healing systems are developed to ensure that the software keeps working, increasing attention must be directed towards ensuring that the healing functions are reliable and do not malfunction. The challenge with self-healing functionality is that it is usually complex and needs sophisticated services in order to work effectively. Companies dependability on software requires that the self-healing parts of the software works flawlessly. Performing a wrong or faulty healing measure could have critical consequences and render the system useless.

Self-healing systems are usually divided in two layers, an application layer and a healing layer. On the application layer are the software components that provide services such as web server et cetera. On the healing layer are the software components that are responsible for the steps during the healing process. The self-aware systems use sensors on the application layer to keep track of how well they are functioning. The healing layer gathers the sensor information and infers what actions are needed to heal the system if any.

Quite common approach to self-healing systems is to have a multi step procedure to detect any possible problems and decide what to do about it. One example of this kind of step based approach is presented in [6]. The steps include monitoring, translation, analysis, diagnosis, and feedback. Monitoring follows the state of the system and reports if anything is out of the ordinary. Translation is used to create an understanding of the situation and after that it is analyzed to see what really is wrong. After diagnosing the fault, it is fixed and feedback is required to determine if the action was correct or not. For the purposes of this paper, any of these steps is equally important and a failure in any of them can be just as disastrous. For example, a failure in monitoring will cause the self-healing system to fail to notice problems while incorrect analysis of the situation may result in incorrect healing attempt.

This paper is organised as follows. First we will introduce two example approaches to self-healing systems. In section III we briefly compare the systems to each other and analyse the weaknesses in their self-healing mechanisms. Then we discuss how the robustness of the self-healing layer could be improved. The issues are discussed in the light of previously presented systems. Finally there are conclusions.

II. Two approaches to Self-Healing

There are multiple available approaches to building and designing a self-healing system. Two of them are presented here. One of the approaches is to use multi-agent systems and the other approach is a framework based approach. These approaches serve as examples and provide basis for further analysis and discussion in the following sections.

Multi-agent systems are built from a number of software components which have a clear operational goal. An integral part of the multi-agent systems is a rewarding system that is based on achieving goals. This provides an important feedback loop that can also be used in a self-healing environment. Agents are capable of communicating with each other and they are used in for example distributed decision making.

A framework is a set of generic support services which support a certain type of activities. Self-healing framework describes a set of services to help building of self-healing systems. The support services are generic in the sense of being able to support any self-healing system.

The multi-agent and framework approaches are different from each other in how they are designed and how they support the self-healing process. The multi-agent system is designed to be a separate system that can easily be added to

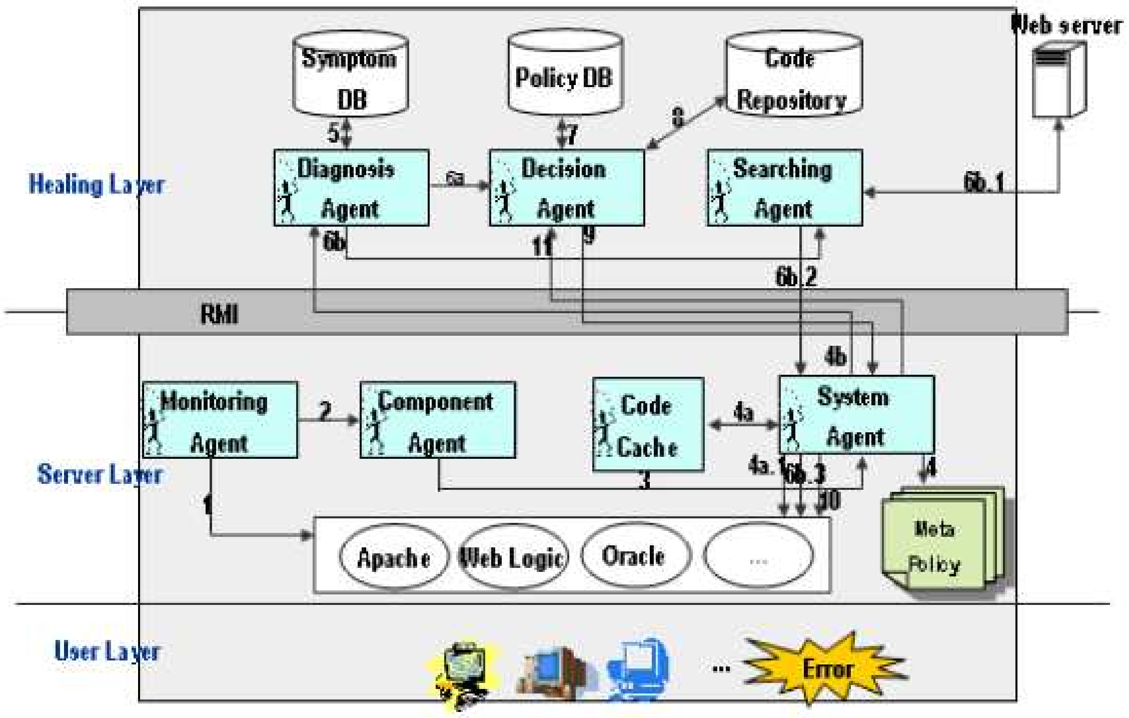

Fig. 1. System architecture for the multi-agent based self-healing system [6].

run alongside existing system. The multi-agent solution does not need any special modifications to the existing system. The framework based approach is designed to be used in event based systems, where the system is built from set of components. The components should be easily detachable from each other in order for the framework to do self-healing on the system.

A. Multi-agent based context aware self-healing system

The multi-agent based system [6], [7] is a proposal to build a self-healing system based on scanning log output of necessary components, such as web servers. The goal is to determine possible problems from the log output. As the name suggests the system is built from multiple agents which cooperate to achieve self-healing of the system. This kind of approach should result in a low demand for system resources.

The architecture is divided into two layers, as show in Fig. 1. The layers are the server layer and the healing layer. The server layer is responsible of monitoring the behaviour of the servers and the healing layer is responsible for decision making. On the healing layer are the following agent components: Diagnosis Agent, Decision Agent, and Searching Agent. Various databases and repositories provide supporting data and functionalities for the previously mentioned agents. Monitoring Agent, Component Agent, and System Agent are located on the server layer. In addition to these agents there is a Code Cache which is used in emergency healing situations. Code repository provides available healing methods. Symptom and policy databases provide information about nature of server failures and how they should be dealt with.

The self-healing process of the multi-agent is based on a sequence of steps [6]. The healing process includes the following steps: monitoring, filtering and translation, analysis, diagnosis, and decision and feedback. Monitoring Agent is responsible for following the logs that are generated by services (*Step 1* in Fig. 1). If the size of any of the logs change the Monitoring Agent notifies the Component Agent (*Step 2*). In cases where the monitored piece of software does not generate log entries system event viewer is monitored for relevant events. After receiving notification from the Monitoring Agent the Component Agent extracts required logs. The logs are processed for certain keywords that indicate problems in the software component. These keywords include *not*, *error*, *reject*, and *fail*. If any of these keywords are found the events are transformed into Common Base Event (CBE) format.

System Agent is responsible for receiving (*Step 3* in Fig. 1) and analysing the CBE formatted events. When it receives an event it gathers relevant system resource information such as CPU, memory, process and job schedule information. A specific module inside the System Agent, Adaptation module, is responsible for determining which is the best healing method. The decision is based on threshold values, which represent policies (*Step 4*). Under emergency circumstances, which are dictated by the policies, System agent executes the selected healing method right after the adaptation module has decided on it (*Step 4a*). Otherwise the available data is sent to Diagnosis agent (*Step 4b*). Diagnosis agent is responsible for diagnosing the problem with the help of Symptoms DB (*Step 5*). To achieve this it analyzes the CBE log, resource information, and the dependencies between components. Decision

Agent determines which healing method is correct (*Step 6a*) based on the information analyzed by the Diagnosis Agent. The Decision agent also uses a Policy DB (*Step 7*) and Code Repository (*Step 8*) to make the decision. Finally the selected healing method is executed by System Agent (*Step 10*). After executing the method System Agent sends feedback information to Diagnosis Agent.

When the system cannot heal itself Searching Agent is used to find suitable information (*Step 6b* in Fig. 1) which can be used to resolve the problem. The Searching Agent uses common search engines such as Google[1] (*Step 6b.1*) to find this information. Code Repository holds the available self-healing mechanisms. The Code Cache in Fig. 1 is used by the System Agent to achieve rapid healing in emergency situations [6]. The healing is quicker because multiple analysing steps are skipped during the healing process.

Park et al [6], [7] have created an implementation of the previously described system. The implementation of multi-agent system was compared with Adaptive Services Framework by CISCO[2] and IBM[3] [2]. During the comparison process it was found that the multi-agent based approach uses significantly less system resources [6], [7]. System performance of the multi-agent system was also determined very good and the time required to self-healing is less than the competitor takes. During the measurements it was also found that, since the system only has one monitoring agent which monitors all logs and thus only one process in memory for monitoring purposes, the system was very scalable in terms of followed software components [7]. Memory usage of the multi-agent system stays on a nearly constant level no matter how many server level components are simultaneously monitored.

B. Framework based self-healing

The framework based approach proposed by Dashofy et al [4] is focused on describing the architecture and planned repairs in it. The framework extensively uses architecture descriptions to present the current situation and how it should be changed in order to be repaired. The main environment of the framework are event driven systems which are composed of multiple independent components. The components are connected to each other using connectors to reduce interdependencies between the components. The connectors also mean that the components themselves are not concerned where their communication counterparts are and thus other components can be relatively easily changed from the other end.

The framework uses architecture descriptions as an integral part of the system. The descriptions describe which components are available and how the components are connected to each other. Connectors enclose the details required for low level communication and therefore the components need not be concerned about the communication technology. This type of architectures increase the flexibility of the system.

[1]http://www.google.com
[2]http://www.cisco.com
[3]http://www.ibm.com

The architecture descriptions are deployed along the software system, as a basis when the system and its components are instantiated. Essentially the system is a reflective system where the architecture description is used both to describe the system and to make changes into the system. Changes to an architecture are done by creating an architecture difference description. In the context of self-healing systems the changes to the architecture are done in order to repair the system. Other kind of changes can also be made. An architecture can be changed by replacing, removing, and adding components. The architecture difference description is much like the output of diff command in Unix which is used to compare two text files together [4]. For example a replaced component is expressed by removing the old component and adding a new, similar, component.

The tools and documents used in the framework are depicted in Fig. 2. The framework uses xADL, an extensible, XML based architecture description language to express components and their relationship in the system. xADL language is also used to describe differences in the architecture. The xADL descriptions are stored into a repository which is capable of noticing changes in the descriptions and notifying the AEM component. AEM component is used to maintain mapping between the xADL description and the runtime system. AEM is also responsible for initiating and managing the self-healing subsystem. ArchMerge is used to merge architecture differences to current architecture. The result of the merge is a new architecture which is deployed by the AEM component.

Design Critics in Fig. 2 are used to perform a what-if analysis on a new architecture. The purpose of the analysis is to find out if the new architecture is compliant with the specification of the system. This means that for example all required connections between components exist and that all components that are needed by fully functional system are available. The analysis must also determine that, if a component is changed, the new component is able to produce the required functionality. This is essential in cases where an old component is replaced with a newer version of it. Another important issues to consider when an architecture is changed are the possible special needs of an architecture. It may not be possible to take these special needs into account by using a generic analysis tool. Therefore multiple analysis tools are needed to take all possible requirements into account [4].

The changes in the architecture follow a four step process. First, the components and connectors which are to be removed from the system are allowed to run cleanup code and to send data about their state to another component. The state information is used by the replacement component. Second, components and connectors that are on the edge of the area where the failure happened are suspended. This prevents the components from sending further messages (events) to the affected area. During the third step, the components and connectors are removed from the system and new, replacing components are added to the system. Finally, the suspended components on the edge of the affected area are resumed and the systems continues its operation.

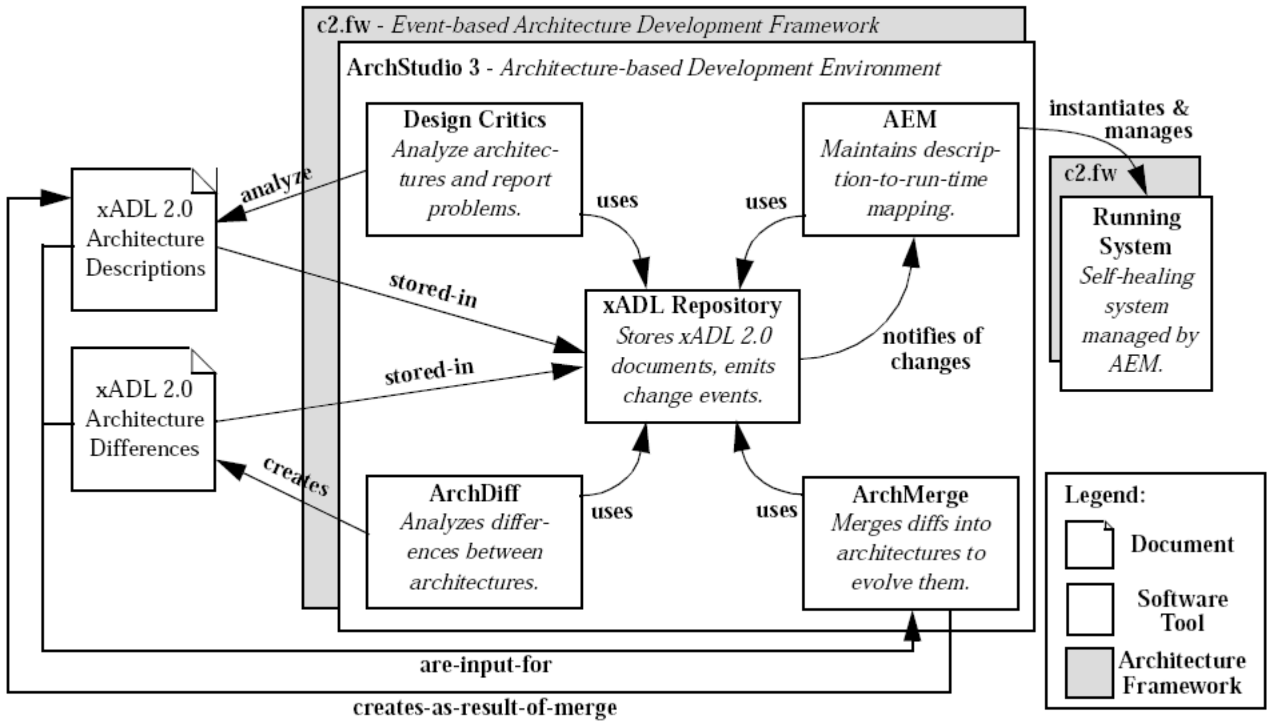
c2.fw - Event-based Architecture Development Framework
ArchStudio 3 - Architecture-based Development Environment
Design Critics
Analyze architectures and report problems.
AEM
Maintains description-to-run-time mapping.
instantiates & manages
c2.fw
Running System
Self-healing system managed by AEM.
xADL 2.0 Architecture Descriptions
analyze
uses
uses
stored-in
xADL Repository
Stores xADL 2.0 documents, emits change events.
notifies of changes
xADL 2.0 Architecture Differences
stored-in
creates
ArchDiff
Analyzes differences between architectures.
uses
uses
ArchMerge
Merges diffs into architectures to evolve them.
are-input-for
creates-as-result-of-merge
Legend:
Document
Software Tool
Architecture Framework

Fig. 2. Tools and documents used in the architecture based self-healing system [4].

Dashofy et al [4] have implemented all other parts of the outlined system except fault detection. The group has approached the subject from reflective self-managing software perspective and have built the services that are needed to allow reflection first. The analysis component is expected to be built in future work.

III. Analysis

In this section we discuss the two systems and briefly compare them to each other. There are a number of similarities and some differences between the systems. In deep down both of the systems are fairly similar. We will also discuss the weaknesses of both systems especially on the healing subsystems. The weaknesses in self-healing can cause big issues. For example the systems should not try and repair a fully functional system or replace wrong components inside the system when a failure occurs.

Despite their different background and approaches to self-healing systems the solutions are somewhat similar. Both of the systems share the view of differentiating the self-healing subsystem from the actual application layer. Both approaches recognise the need for sensors in the application layer to monitor the applications. Underneath both of the approaches also have similar step-based algorithms to achieve the self-healing itself.

The multi-agent system uses a simple method of monitoring the system healthiness. It employs a log monitoring agent which constantly checks if log file sizes change. When they change a separate agent is responsible for scanning the log file for certain keywords. The weakness in this is the list of the keywords. If this list is incomplete, the agent does not find all error situations. Another weakness is the translation of found error messages into CBE. The logs of software components are not standardized and therefore it is not straightforward to translate the messages. If the translation fails, the system cannot recover from the failure.

The difficulty of detecting real failure situations is shown in the fact that there is no component responsible for that in the framework based approach. A monitoring scheme can be implemented by requiring that each component and connector can detect their own error situations. This solution is not optimal because it would pose a significant demand on the components themselves. Similar approach that is used in the multi-agent system is not possible because it would be impractical that every component and connector would have their own log. Depending on the level of modularisation and the amount of components and connectors there could be hundreds of logs.

Both of the presented systems also rely on pre-existing knowledge about the system. The framework uses explicit descriptions about the architecture which are then modified to replace components. In the multi-agent system there is a specific agent that is responsible for analysing the components and their dependencies. The approach used in the framework is stronger because explicit information can be verified by human users. After verification human users can try and modify the information so that the system has a correct and complete picture of the system. In the multi-agent system the agent must infer the system composition from secondary clues such as process ids. This approach is weaker because the agent can easily overlook unexpected connections between software components. This can lead to poorly isolated faults in the system. From explicit descriptions the connections are easily detected provided that the description is kept up to date. To make sure that the description is up to date the framework uses a dedicated service.

The multi-agent system uses secondary information to infer the situation of the system and determine the correct healing measure. This information (see section II-A) gives a rather complete picture of the state of the process but is it enough? If a process uses unusually large amounts of processing capacity it could be busy looping in a live lock situation. A deadlock situation can be detected by the fact that the process does not respond to requests in any way. Software components that leak memory can be detected by ever increasing memory usage of that component. The a fore mentioned information does not provide enough clues to situations where a subsystem of a bigger component is malfunctioning unless the subsystem has its own thread or process. When detecting a live lock it is not clear how long the high processor use needs to continue in order to be sure that the component is indeed suffering from a live lock. The same problem is with the deadlock situation.

The selection of proper healing method is dependent on the correct outcome of the problem analysis. The multi-agent approach uses databases to select the correct healing method once it has analysed the situation. This is an advantage for this system because the databases can be populated by human users. The databases will help in the general problems and the system is capable of searching new information which informs how a repair should be done. In the end the system invokes human administrators and informs them with its data. This is also an advantage for the system. When the system cannot heal itself it can always request help automatically which will result in faster correction.

Lack of diagnosis functionality is the main drawback of the framework. When the monitoring and diagnosis facilities are included in the framework the main challenge will be translating the available information to correct architecture difference descriptions. The framework seems to have a strong bidirectional reflection mechanisms which makes implementing planned changes to the architecture straightforward. The main question will be how to produce correct plans. Another advantage of the framework are the what-if checks. The checks are done before the changes are actually implemented. While they decrease the responsiveness of the framework when responding to failures, it means that the framework is more likely to take corrective action. Nevertheless it should not be forgotten that the analysis is difficult to make.

IV. Preventing Failures in Healing Mechanism

The failure of self-healing mechanisms can be prevented using computational verification or replication of key healing components. These approaches can also be combined. Computational verification has some advantages but the drawback is that it cannot be applied everywhere. The use of existing fault tolerance solutions in key healing mechanisms is mainly limited by system resource usage. Use of fault tolerance solutions consumes more system resources and increases software complexity.

A. Prevention Using Formal Verification

Montanegro et al [5] propose a method for self-healing that is based on formal logic. The formal logic is built around the concept of events. The event model is useful when distributed systems need to be verified. The authors introduce a new operator (Δ) which is used to note events. The logical formulations in their approach is based on states of the components. Certain states in the components trigger events which are transferred to another components further triggering state changes in those components. In the paper the authors demonstrate their approach by modeling the establishment of a token ring network. Using formal modeling of the steps that are taken while a token ring is established the authors are able to prove that the coordination will work in any given situation.

The use of formal logic allows computational verification of the behaviour of the self-healing system. For the verification purpose a system is modeled using the formal logic and the resulting description is computationally verified. Through verification we can be sure that the system behaves correctly in all circumstances. If the self-healing mechanism of a given system is verified and found correct it can be trusted to work correctly on logical level. However, verification does not solve all problems. The main problem of verification is that it is computationally expensive and cannot be applied to large and complex systems. Verification cannot be used in large systems because state machines, which are used to describe system behaviour and processing, would become too large to verify in any given time. This problem is referred to as state explosion. Because the verification is computationally expensive it also has an increasing effect on development time.

Verification is suitable for systems where single nodes are small and the cost of each node is relatively high. In these kind of situations the verification is computationally viable and the cost of verification is paid back in small number of node failures. Verification is not needed when single nodes need not be self-healing and when they are inexpensive. In these circumstances failures are expected and dealt with using large number of nodes. One example of such a case is wireless sensor networks [1] and their nodes. Within large and complex systems it is possible to use verification for small subsystems. In these situations only the most crucial elements of the software are verified. This will increase the reliability of the software. In many cases it can even be impossible to have access to all code of all components to verify them. Especially in self-healing architectures only the self-healing parts of the system need to be verified. One example of such a system is the multi-agent based solution presented in section II-A where you would only need to verify the agents responsible for the healing process.

Formal verification is also used to ensure communication protocol behaviour. The communication protocols are very important to distributed applications. For example in wireless sensor networks it would be beneficial to use verification to validate the communication protocols and routing protocols. In wireless sensor networks any node can be rendered useless anytime [1]. This is even expected to happen. At the same time nodes rely on each other to be able to communicate with the base station. In this kind of scenario it is important that the communication link will survive as long as possible.

B. Improving Fault Tolerance

Existing fault tolerance mechanisms can be used on the healing layer. This means that the mechanisms are applied to the components on the healing layer. Classic fault tolerance methods include replication, redundancy, and diversity. All of these methods can be used to improve self-healing systems. Improving fault tolerance is especially important when human life is dependent of the system or the system is otherwise critical like power supplies in naval vessels [3].

Replication can be used in self-healing systems in situation analysis and selection of healing method. Replication can be used in these key decision points to use multiple software components that independently of others arrive to a decision of what is the problem. Then the multiple components need to agree on or vote on what or where the problem is. In voting situations majority wins. To ensure that there is a majority we should use odd number of decision makers. Once agreement on the problem is reached, another set of components must independently arrive to a conclusion of what action to take to heal the problem situation. The use of multiple decision makers and the requirement of agreement between the decision makers does not help unless we can distinguish if any of the decision makers is failing or not. If we assume that all decision makers are deterministic and produce similar results we can determine if one decision maker is faulty by using a voting system. A voting system will also be tolerant of small number of malfunctioning decision makers. In the context of multi-agent system replication can be used to at least System Agent, Diagnosis Agent and Decision Agent. These functions would benefit from increased reliability and fault tolerance.

Increasing redundancy in the context of self-healing systems would mean that entire self-healing subsystem would be replicated. In this case the system should somehow monitor how the self-healing subsystem is functioning. Monitoring is needed to determine if the subsystem is faulty or not. To achieve this, self-healing systems could be used on top of them selves to create yet another layer of self-healing mechanisms. To prevent a failure on healing layer A we would need a healing layer B. To prevent a failure on healing layer B we would need a healing layer C and so on. Using multiple layers of self-healing systems to monitor other self-healing systems is not very practical anymore.

Increasing diversity in self-healing means that the system uses multiple different algorithms and components to self-healing. For example in a diverse self-healing system both of the presented approaches could be present at the same time. Using this kind of solution will also need joint decision making on what action to take.

Any combination of previously mentioned fault tolerance measures can be used at the same time. This will increase the fault tolerance of the system. However, the number of different components grows bigger. This means that more and more resources are consumed to keep the system running. How many of the proposed solutions is practical to use is defined by a cost-benefit analysis. If the the system is critical, more fault tolerance and other precautions should be used, while a non critical system can be allowed to fail.

V. Conclusion

Companies rely more and more on their software systems during day-to-day activities. At the same time the software systems are more and more complex. Self-healing systems tackle the growing issue of software management and repairs by proposing solutions that are able to heal themselves. In this setting it is crucial that the self-healing systems manage to keep the software running and therefore allow companies to do business. Self-healing systems rely on self-diagnosis and selecting correct repairing functions to heal the system. It is even more important that the self-healing functionality itself functions correctly. Surprisingly many approaches do not take into account the possibility of failures on the self-healing layer.

In this paper I have presented two different approaches to self-healing systems. Multi-agent based approach can be added to any available system to allow some degree of self-healing functionality. The multi-agent approach uses log output of software components to track and repair the problems. The system relies on existing methods that can be executed to achieve self-healing behaviour. The method is selected as a result of a multi-step diagnosis and analysis process. Each of the steps in the process are handled separately in agents distributed to both application and healing layer. The framework approach relies heavily on architecture descriptions and planning ahead on the repair actions. The framework has a number of tools to assist in the architecture description and change planning activities. The pre-planning of the repair actions allows the framework to speculate on the outcome of the repair action and thus detect if the corrective action is compliant with the architecture or not.

We have discussed the weaknesses of presented solutions and proposed a couple of solutions to prevent failures on the healing layer. The main weaknesses of both presented systems are the analysis of the problem and the execution of correct healing measure based on the analysis. If either of these steps fail, the other is of no use. The analysis step is challenging in complex and large systems. The problem needs to be isolated so that the healing can happen correctly.

To improve the reliability of self-healing mechanisms I proposed formal verification as a pre-production measure and fault tolerance techniques as the runtime solution. Verification increases the time needed to bring a product to the market but can be very essential in certain applications. Fault tolerance measures increase the resource demands during runtime operation. These can also be justified in correct circumstances.

All solutions to increase the reliability of the self-healing mechanisms require either more time and preparation in advance or more system resources during runtime. Before any of the solutions is adapted to a given solution a cost-benefit analysis must be done. If for example replicating diagnosis components does not provide a substantial increase in reliability of the system it should not be done. In other systems things are expected to fail and single nodes are not accounted for. Where human life is dependent of self-healing systems bigger resource requirements will not pose a problem.

REFERENCES

[1] ASSUNCO, H. P., RUIZ, L. B., AND LOUREIRO, A. A. A service management approach for self-healing wireless sensor networks. In *Autonomic Networking 2006* (2006), vol. LNCS 4195/2006, Springer, pp. 215–228.

[2] BAEKELMANS, J., BRITTENHAM, P., DECKERS, T., DELAET, C., MERENDA, E., MILLER, B., OGLE, D., RAJARAMAN, B., SINCLAIR, K., AND SWEITZER, J. Adaptive services framework, Oct. 2003. CISCO White Paper, `http://www-03.ibm.com/autonomic/pdfs/Cisco_IBM_ASF_100.pdf`.

[3] BUTLER-PURRY, K. L. Multi-agent technology for self-healing shipboard power systems. In *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems, 2005* (Nov. 2005), IEEE, pp. 207–211.

[4] DASHOFY, E. M., VAN DER HOEK, A., AND TAYLOR, R. N. Towards architecture-based self-healing systems. In *Proceedings of the first workshop on Self-healing systems* (2002), ACM Press, pp. 21–26.

[5] MONTANGERO, C., SEMINI, L., AND SEMPRINI, S. Logic based coordination for event-driven self-healing distributed systems. In *Coordination Models and Languages* (2004), vol. LNCS 2949/2004, Springer, pp. 248–263.

[6] PARK, J., YOUN, H., AND LEE, E. A multi-agent based context aware self-healing system. In *Intelligent Data Engineering and Automated Learning - IDEAL 2005* (June 2005), vol. LNCS 3578/2005, Springer, pp. 515–523.

[7] PARK1, J., YOO1, G., JEONG1, C., AND LEE, E. Self-management system based on self-healing mechanism. In *Management of Convergence Networks and Services* (Sept. 2006), vol. LNCS 4238/2006, Springer, pp. 372–382.