The A&A metamodel in the context of Autonomic Computing

Fabiano Dalpiaz

April 22, 2007

1 Extended abstract

Autonomic Computing (Ganek and Corbi [2003], Kephart and Chess [2003], Murch [2004]) is a research line introduced by IBM, whose objective is the creation of software systems with self-maintenance capabilities (often called "self-* properties"). There have been several architectural proposals in this context, and many of them are based on the construction of a multi agent system (MAS), such as Tesauro et al. [2004] and Tianfield [2003]. Agents are seen by many researchers as a key requirement of autonomic systems, because of their autonomous enacting capabilities are essential to buold self-managing software systems. The Agents and Artifact (A&A) Metamodel has been implicitly introduced in various works (such as Omicini et al. [2006], Ricci et al. [2005]) as a model to define multi-agent systems on the basis of the concepts of agent and artifact. The research proposal I describe consists of using the A&A Metamodel to design an autonomic system.

The first step is to define the general architecture behind the proposed autonomic system: it will be based on software agents, and implement four different self-* properties, namely self-configuration, self-healing, self-protection, and self-optimization. Artifacts will have a crucial role in the system, because they will be the "tools" used by the agents to act and to change the environment, which consist of everything agents cannot access and change directly, even if they are situated in it.

In order to make a more effective analysis and have a real context where the system can be developed and deployed, I propose a case study. The scenario consists of a server system, where a number of server applications are deployed: a web server to provide dynamic pages support, a database to store data, a printer server to manage a remote printer. There are also some other support applications, such as a software firewall to block unauthorized access to the server, and an antivirus software to protect from worms and viruses. Each of these applications is provided with an administrative interface, where the system administrator can tune their behavior setting particular values to a set of parameters. The current server system does not provide any autonomic computing property, and the goal of the research proposal is the study of how

they can be added and their actual development. The advantages should be enhanced efficiency and reduction of administrative interventions. The development of a complete autonomic system of this kind is not trivial, and will require a lot of research and experiments, but current technology and the multi agent systems can give strong contribution in fulfilling this task.

One of the first questions to be addressed is if there is the need of agents to control the various software systems installed on the server. They would not have direct access to these systems, because they are part of the environment, but should access through an artifact, which works as an interface between the MAS and the environment. From a first analysis such kind of agents looks not to be necessary, because the self-management properties can be ensured by general purpose agents providing self-* properties. The answer can be the opposite (they are needed) if they can choose to enact or not the suggestions given by the agents providing self-* properties, but in that case it would be more difficult to ensure the fulfillment of the autonomic properties (the agent accessing the web server can reject the suggestion to change the file upload maximum size). This topic is absolutely nontrivial, and a deep analysis is required to make the correct choice.

Despite the fact of having agents to autonomously control the software applications, some artifacts to interface the applications have to be defined. The described scenario already contains these artifacts, however, because they are the administrative interfaces to the software systems, which allow a complete control, or at least an extended way to set the tunable parameters of the controlled applications. Each software system can have more than one controlling artifact, such as command-line utilities, a set of application program interfaces (API), or other types of controlling mechanisms. For instance, databases have some command-line administrative tools such as a database optimizer, the commands to relocate databases on other disks, the backup tools; web servers usually have configuration files which can be manipulated and change some functionalities after restarting the service; some software firewalls have a client to define the security policies and the open communication ports; remote printers are controlled by a device driver, which may be interfaced using specific calls using APIs. Moreover, some log files can be considered as artifacts in the considered MAS, because they can give additional information to the agents in order to better provide the self-maintenance properties.

As mentioned before, the environment consists of everything is not directly controlled by the multi-agent system, namely the software system installed and providing services to the users connecting to the server, but also the other software systems. These applications may however influence the behavior and the performance of the systems that can be controlled through artifacts, but it is very difficult defining the relationship between non-controllable software systems and the controlled ones, and my choice is not to consider their role. One possible comprehensive environmental artifact could be represented by the system monitoring tools, which can give information about the availability of memory, CPU, and so on.

It is now time to discuss the role of the agents providing the key self-* properties, and verify what kind of properties they have:

- *self-configuration*: each server software task can be performed by more than one software system (e.g., database functionalities can be provided by MySQL, PostgreSql, Ingres, and so on), and a software agent should control the configuration of the server system in terms of the running software systems. The agents will use the artifacts to act on the environment and modify the current configuration of the system.

- *self-healing*: the agents in charge of providing self-healing should make large use of the logging artifacts, because they can give extensive information about the correct working of the various systems; a software system going down twice a day can be a symptom of an "illness" - a bug or an incompatibility. This kind of agents have to download available patches to repair known bugs, and report detected problematic situations to the application producer.

- *self-protection*: this class of agents will have the goal of defending the server system from attacks. They should hence manage the security-related software systems, such as the firewall and the antivirus through the respective controlling artifacts. Moreover, they should use the system monitoring artifact to get information about anomalies (such as 100% CPU usage with no active clients) that could be due to security issues. All the security patches should be managed by this class of agents, using of an artifact to verify the availability of updates.

- *self-optimization*: the optimal working of the software systems depends on a correct parameters tuning. These agents should start from logging files and other monitoring artifacts to evaluate the performance of the systems, and try to modify the parameters in order to get the maximal performance. This can involve the use of learning algorithms, which can identify good solutions and isolate bad tunings.

After this sketch of a possible MAS-based architecture for autonomic systems, follows the examination of the properties the agents should have. Every agent in the system is *autonomous* and *situated* by the definition of agent in the A&A metamodel: an agent without autonomy is an artifact, and each agent makes sense only when situated inside an environment. The *reactiveness* is also required, because the agents sense the changes in the environment they are placed in, modify their internal state depending on that, and act when they believe an action is required. For instance, the agents enacting self-optimization monitor the context using some artifacts such as the system performance monitor, modify their internal state when sensing a high load of the CPU, and try to change the parameters of the applications to reduce the load. The *sociality* property is also ensured, because all the proposed agents do not live alone, but in a cooperative context in which each agent collaborates to enact self-management. *Interactivity* is another key property: when the self-protection system identifies a problem in a software system, but does not find any solution to it, the best choice is to contact the self-configuration agent and ask her to

modify the configuration by disabling the risky application. Even if the definition of the A&A metamodel does not require it, agents providing autonomic properties need to be *goal-oriented*: they have to ensure a particular property, namely that is the goal to be pursued. For instance, the self-protection agent has the goal "Ensure protection to the server system". *Intelligence* is also a requirement for the agents in such a kind of system: without that property they cannot make "rational" reasonings and pursue their goals at best, and hence they would not be able to improve the quality of the system. The agents providing autonomic properties do not need to be *mobile*, because they are aimed to work in a specific environment (on a particular server system); if we allow some agents to control interface artifacts we may need the mobility property, in order to let them enter other systems having the same controlled interface, but this requirement is not definitory at all. Some agents need *learning* properties, such as the self-optimization ones, which can learn from the past (or from suggestions of other agents) which parameters should not be used because proven to be ineffective.

The same analysis (with different properties) can be proposed with respect to the artifacts of the system: they are *inspectable*, since the operations, interface, and functions need to be available in order to let the agents use them. For instance, we can provide an XML-based description of the artifacts, that can be inspected by the agents to know how to use the software system interfaced through the artifact. A property that does not hold is *controllability*, because we cannot trace the internal behavior of the artifacts. *Malleability* also does not hold: we use an extended vision of artifacts, where they are not created to act on the environment within the MAS, but already exist. For sure they need to be *predictable*, because we want them to perform our requests: if the self-optimization agent uses an artifact to change the maximum size of uploads, we expect this option to be enacted. *Formalisability* is not an easy point to be discussed, and depends on what and how much we know about the artifacts behavior, but since we do not know the internal behavior, but only the expected result, this property is not always ensured. *Linkability* and *Distribution* are not required properties, even if linkability can allow the use of a common interface artifact to control more software systems part of the environment.

2 Future perspectives

The whole scenario we described and partly analyzed is quite complex, and cannot be faced by a single research project. The technology is mature to face such a problem, but in order to choose the best techniques we need to proceed step by step, and face the various problems independently, linking them in a second moment. We then need a research plan, divided in various dimensions and with different temporal terms:

- *Systems*: in the first phase we should face only one software system as a single entity, namely having only one artifact (or a little number), in order to understand the problems that can arise. In a second phase the various

systems can be managed together, and the configuration agents will play a crucial role in verifying the interaction among them.

- *Autonomic properties*: the idea is starting from the analysis of one property as a time, and make a deep examination of what we intend and what we require from the self-* properties, and start to join them and examine the relationships only after having a clear view of each of them.

- *Intelligence*: the layer of intelligence provided with the agents should be incremented, but we should start with a very little level of intelligence and see unexpected negative behaviors the agents choose and perform.

In order to get a complete, robust, and well founded solution for the context, we will need a couple years, but in one year we can have a clear view of the problem and a first implementation following a specified design for such a MAS.

References

A.G. Ganek and T.A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

R. Murch. *Autonomic Computing*. Prentice Hall PTR, 2004.

A. Omicini, A. Ricci, and M. Viroli. *gens Faber*: Toward a theory of artefacts for mas. *Electr. Notes Theor. Comput. Sci.*, 150(3):21–36, 2006.

A. Ricci, M. Viroli, and A. Omicini. Programming mas with artifacts. In *PROMAS*, pages 206–221, 2005.

G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, and S.R. White. A multi-agent systems approach to autonomic computing. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 464–471, 2004.

H. Tianfield. Multi-agent autonomic architecture and its application in e-medicine. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 601–604, 13-16 Oct. 2003.