Open issues in self-inspection and self-decision mechanisms for supporting complex and heterogeneous information systems

Michele Colajanni
University of Modena
Modena, Italy
colajanni@unimo.it

Mauro Andreolini
University of Roma Tor Vergata
Roma, Italy
andreolini@ing.uniroma2.it

Riccardo Lancellotti
University of Modena
Modena, Italy
lancellotti.riccardo@unimo.it

Abstract

Self-* properties seem an inevitable mean to manage the increasing complexity of networked information systems. The implementation of these properties imply sophisticated software and decision supports. Most research results have focused on the former aspects with many proposals of passing from traditional to reflective middleware. In this paper we focus instead on the supports to the run-time decisions that any self-* software should take, independently of the underlying software used to achieve some self-properties. We evidence the problems of self-inspection and self-decision models and mechanisms that have to operate in real-time and in extremely heterogeneous environments. Without an adequate solution to these inspection and decision problems, self-* systems have no chance of real applicability to complex and heterogeneous information systems.

1 Introduction

Self-* systems seem the inevitable answer to the continuously increasing complexity of networked information systems. Let us define a complex and heterogeneous information system (CHIS) as a system with multiple application classes and multiple Service Level Objectives, such as performance, availability, security, energy saving, costs. These SLOs may be contradictory and, even worse, difficult or impossible to quantify with the same measurable metrics. Many researchers have addressed the issues of designing and implementing software that coordinates interactive networked applications. For example, CORBA [1], J2EE [3], .NET [2] hide from the programmer many complicated details of the underlying software and hardware platforms, thereby increasing portability and facilitating maintenance. They provide an abstract interface that masks to the application low-level details of the operating system and network layer, and guarantees interoperability among application components through standard interfaces. On the other hand, these infrastructures lack the necessary support for the dynamic aspects of todays' new computational needs. Hiding the underlying details has many advantages, but a certain degree of awareness is necessary for scalability, QoS, and adaptability to context and conditions of highly dynamic environments. For these reasons, a desirable middleware for implementing complex policies related to the above mentioned aspects should provide an adequate mix of transparency and control depending on the applications. Many researchers think that to provide the software with adequate flexibility requires the passage from conventional middleware to some forms of adaptive middleware. For example, the reflective middleware model (e.g., DynamicTAO [4], OpenORB [9]) is implemented as a collection of concurrent objects that can react dynamically to changes in the underlying platform and to external requirements through migration, enabling of dynamic interaction patterns, reconfigurations, insertion and removal of components. In this way, it is possible to select protocols, algorithms, policies and any other mechanism to optimize system performance for unpredictable context and situations. However, it is important to remark that self-* properties and reflective middleware are not synonymous. Actually, some partially self-* systems have been implemented without recurring to reflective middleware. As an important example of self-oriented software that does not seem based on reflective middleware, we can cite the *IBM WebSphere Application Server*, that in its present version is considered *predictive*, that is, at the third level in the IBM scale of Autonomic Computing, ranging from ”basic” (level 1) to ”autonomic” (level 5). Independently of the software supports to achieve self-* properties, there is an underlying (and less investigated) level that must help the middleware to trigger or not the adapting actions at run-time.

Self-inspection Self-inspection refers to the ability of automatically capture all information about the internal state and also adapting the monitoring system to internal and external conditions. The support to self-adaptive applications for a CHIS must be well developed in the following parts: monitoring, measurement, comparison, information retrieval from other sources, including resource utilization monitors.

Self-decision This is the capacity of taking autonomous decisions according to some SLO rules and to a measure of the internal system state that is obtained from the previously described self-inspection component.

We think that *self-inspection* and *self-decision* properties are among the most important issues that should be considered to implement really operative self-* systems. Self-* capabilities for inspection, decision (and, possibly, adaptation) are desirable in a middleware system, but they must be disciplined much better than supports to conventional middleware, because dynamic and autonomous modifications can result in unpredictable system behavior and possible breakdown. These risks limit the applicability of self-*systems as the basis of a complex and heterogeneous information system.

2 Self-inspection

Operating any distributed information system without accurate statistics is not desirable. In general, it is not easy to find the right combination between data sources providing low volume, coarse-grained, non-application specific data, and data sources providing high volume, fine-grained, and application specific information. These issues are even more complex in self-* systems that are governed by the imposed SLOs. Their policies should use system-wide and component status information to take the appropriate actions and to react to events, but this kind of valuable information is not directly available. Distributed monitors usually yield raw, OS-level data (e.g., CPU and disk utilizations, network throughput) or application-level data (e.g., request throughput) that has to be aggregated to infer conclusions about the specific subsystem. Even worse, self-* systems should be able to weigh different measurements into an homogeneous indicator that is used to quickly estimate the status of one or more components and to take actions accordingly. Similarly to a neural network that has to make assumptions about which weights to increase/decrease after an error signal, a distributed system relies on a number of interactive sub-parts that together result in a global phenomenon. In this context, one interesting challenge is how to quickly transform heterogeneous and distributed measurements into an homogeneous indicator of performance or status. These indicators should help to take run-time decisions that allow the system to perform sufficiently well. Due to the complexity and heterogeneity of the models, we cannot expect optimality and we should not search for it. It is much more important to escape from worst cases and to respect SLOs. The literature helps only partially. Conversion from multi-objective to single objective is often done by computing a weighted sum of the different metrics, as shown in [10, 6, 7]. In these works, either a simple weighted arithmetic mean or a simple weighted geometric mean are used to aggregate individual ratings of system features. The logical relationships among features and the distinction of mandatory, desirable, and optional selection criteria are not incorporated in these early models. Even more sophisticated hierarchical models [8] do not aim to capture and combine the dynamics of transient phenomena fairly accurately. The large majority of statistical models provide off-line data analyses.

We are studying models that distinguish two main classes of reaction to external forces: resources degrading *gracefully* (e.g., CPU) and resources degrading *suddenly* (e.g., thread pools, memory, process descriptors, number of connections). Resources degrading gracefully cause a smooth deterioration of system performance, while those degrading suddenly may have tragic consequences on the SLOs and even on the availability of the whole system. To avoid abrupt performance degradation, a possible solution is that of adjusting the weights according to the availability of suddenly degrading resources. In this way, resource scarcity is gradually signaled by an increasing performance indicator. This could lead to novel self-inspection supports that combine heterogeneous sources of information by working on the distance from the maximum capacity of each critical resource and by focusing on *performance trends* more than on instantaneous values, possibly combined with some past measures. The goals of these and other models for self-inspection run-time support should be clear: to extract from many heterogeneous and raw data ”the information” that is really valuable to the self-decision support of CHIS in order to activate the right component.

3 Self-Decision

There is on open debate whether a CHIS with multiple SLOs may be characterized by steady state behaviors or just by an aperiodic behavior. In the latter case, it appears dangerous to predict future conditions by crunching current information, and almost impossible to take any valuable adaptive decision. However, drawing a pessimistic view, such as that CHIS is characterized by unstable aperiodic behavior resembling chaos, only because any human product is not (and for a long time will not be) helped by other natural features typical of chaotic systems, is premature. We think that to associate self-* properties to nervous systems or nature-like behaviors is a dangerous hype. Human products cannot embed all known (and unknown) forces that govern natural beings, such as selection, evolution, and the time scale is completely different. Maybe one day in the future, this will be the reality. But we need to support CHIS tomorrow, not in an unforeseen future. Hence, we have to consider what it is possible to do now.

On the negative hand, we undoubtedly have to forget about linearity assumptions at the basis of many previous models. We also have to exclude all optimization models and algorithms that do not provide an answer in reasonable or real-time. On the positive hand, self-decision run-time supports for a CHIS can confide in at least two important facts. First, any CHIS consists of layers, components, subsystems and hierarchies, hence traditional divide-et-impera approaches remain the most valuable source of solutions. Last, and even more important, a CHIS must not tend to optimization but to something we will call ”good enough quality”.

Lloyd suggests that the combination of the dynamical systems theory and information theory could be used together to formulate a solution for the control of complex adaptive systems. However, control of complex, nonlinear systems requires insight and intuition [5]. But, what happens if the decision algorithm has not enough time to learn? The time to learn, the time to reach another stable state, the time necessary for optimization, is often neglected by previous theories. Is another theory necessary? Can existing theories be extended or combined? If we try to adopt previous theories that tend to optimization, that assume to have enough time to reach a steady state, that have many try-and-drop possibilities, that have *natural* selection, we do not have many hopes to build a sufficiently reactive self-*system .Fortunately, in most instances, a CHIS does not require optimization methods that are interested in finding the best solution possible. A self-* system supporting a CHIS can be fully satisfied by an acceptable state that escapes a critical situation. This goal is not so difficult to meet, since most real systems are largely over-provisioned. However, the full requirements and implications of *good enough quality* remain to be explored.

4 Conclusions

Integrating self-* properties into current systems is one of the future challenges of distributed computing. The claim of this position paper is that the real applicability of self-* properties to complex and heterogeneous information systems requires not only sophisticated software supports (such as reflective middleware), but also new insights and models for self-inspection and self-decision that can support real-time adaptation. We presented some open issues that have to be addressed. It is yet unclear how to aggregate several heterogeneous measurements into an homogeneous indicator of performance or status. Given the highly dynamic nature of CHIS, fast and ”good enough quality” decisions are often preferred to slow and optimal solutions. However, the full requirements and implications of ”good enough quality” in most contexts remain yet to be explored.

References

[1] The Object Management Group. `http://www.omg.org/`.

[2] Microsoft .NET Information. `http://www.microsoft.com/net/`.

[3] Java 2 Platform, Enterprise Edition (J2EE). `http://java.sun.com/j2ee/`.

[4] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. C. Magalhaes, and R. H. Campbell. Monitoring, Security, and Dynamic Confi guration with the dynamic TAO Reflective ORB. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, Apr. 2000.

[5] Learning How to Control Complex Systems. `http://www.santafe.edu/sfi/publications/Bulletins/bulletin-spr95/10cont%rol.html`.

[6] J. R. Miller. *Professional Decision-Making: a procedure for evaluating complex alternatives*. Praeger, New York, NY, 1970.

[7] J. D. Sable. System evaluation methodology. Technical Report AUER-1834-TR-2, INFROM Data Management System Study, Auerbach, 1970.

[8] S. Y. W. Su, J. Dujmovic, D. S. Batory, S. B. Navathe, and R. Elnicki. A cost-benefi t decision model: analysis, comparison and selection of data management. *IEEE Trans. on Database Systems (TODS)*, 12(3):472–520, Sept. 1987.

[9] The Community OpenORB project. `http://openorb.sourceforge.net/`.

[10] D. R. J. White, D. L. Scott, and R. N. Schulz. POED – A method of Evaluating System performance. *IEEE Trans. Eng. Manage.*, pages 177–182, Dec. 1963.