

Towards Pervasive Supervision for Autonomic Systems[†]

Luciano Baresi

*Politecnico di Milano -
Dipartimento di Elettronica e
Informazione, Milano, Italy*

baresi@elet.polimi.it

Matthias Baumgarten,
Maurice Mulvenna,

Chris Nugent, Kevin Curran
*University of Ulster,
Belfast, United Kingdom*

*{m.baumgarten;
md.mulvenna; ch.nugent;
kj.curran}@ulster.ac.uk*

Peter H. Deussen

*Fraunhofer Institute for
Open Communication Sys-
tems, Berlin, Germany*

deussen@fokus.fraunhofer.de

Abstract

One of the key motivations for the provisioning of autonomic communication features in next generation services is to reduce the need of human inference for management tasks. This however means essentially that system control is delegated to the system itself, i.e., the system operator gives up control to a certain extent. The consequence is that autonomic systems might deviate from intended behavior, may show inconsistent or unwanted states and behavior. We propose an approach to develop control structures complementary to distributed, heterogeneous services. We concentrate on necessary properties of those control structures, and furthermore on issues like self-applicability and self-evolution.

1. Introduction

One of the key motivations for the provisioning of autonomic communication features in next generation services is to reduce the need of human inference for management tasks. This however means essentially that system control is delegated to the system itself, i.e. the system operator gives up control to a certain extent. On the other hand, the approach to replace the human system operator by some kind of “intelligent mediator” that controls the whole system from a central point clearly fails not only because of complexity considerations, but also because a common (unified) foundation, theory, and terminology of “intelligent system management” currently does not exist.

In **autonomic communication** [28] an alternative approach is chosen. Instead of having an automated central control, intelligence is pushed into the system configuration itself, which means that meaningful behavior emerges from the dynamics of large configurations of elements that exhibit only a limited intelligence. But in effect this signifies that the exact behavior of such autonomic configurations is not *a priori* determined but evolves during operation. Nobody knows (and can know) what autonomic communication systems really do!

Let us talk about pathology. The biological system metaphor for autonomic systems provides us several scenarios of illness comprising various kinds of malfunctions starting at the level of physical links and devices up to service provisioning platforms, performance bottlenecks, selfish behavior and general fairness violations on various levels, as well as security concerns like virus attacks, intrusion, denial of service attacks, etc. We therefore propose a complementation of the autonomic communication paradigm by the concept of the analog of an **immune system** that continuously senses a system and reacts and infers if pathological behavior exhibits. This concept is called **pervasive supervision**.

This paper addresses the foreseen developments in the area of supervision that will be carried out in the CASCADAS¹ project [4]. CASCADAS focuses on the definition and development of a self-organizing component-based service infrastructure. A ground concept is the so-called *Autonomic Communication Element* (ACE). An ACE is a building block for autonomic services which can be seen as embracing all essential

[†] This work has been supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

¹ Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services

characteristics that are required by autonomic services within a ubiquitous networked environment. All ACEs are equal but some are more equal than others – in other words, a multitude of ACEs will exist with each type providing varying types of service and capabilities. In the CASCADAS project, the ACE concept build a software abstraction of all components which will be developed in the project and thus will be a common terminological and technological foundation of the issues addressed; particularly self-organization and self-aggregation of services, security issues for autonomic services, knowledge representation and distribution by means of common overlay network, and – last but not least – supervision as a generic hierarchical control paradigm.

This paper is organized as follows. Section 2 addresses a generic architecture for supervision systems which will be extended and altered in the course of the paper to fit the anticipated requirements of future service provisioning systems. Section 3 summarizes the current state of the art. In Section 4, a recent approach of supervision for WEB Service is selected to analyze properties of supervision system for autonomic services. These requirements are analyzed in depth in Section 5 with respect to structural issues and in Section 6 with respect to long-term self-adaptation (self-evolution) of supervision systems. The final Section 7 draws conclusions and summarizes the paper.

2. A Meta-Architecture for Supervision

Following the CASCADAS approach, consider an environment comprising a number of ACEs providing a catalog of *functions* or *atomic services*. Linking of functions into a certain control structure results in *composed services*. By *supervision* we mean

1. the continuous monitoring of ACE configurations and the interpretation of monitored data according to certain requirements (safety, functional correctness, consistency, performance, reliability,
2. etc.); enforcement of corrective measures if a violation of these requirements is detected.

Let us start with a discussion of a paradigmatic architecture of a supervision system as shown in Figure 1. The basic organization of a supervision system is that of a closed control loop. Monitoring components gather information about the supervised system. Effi-

ciency considerations require that monitors perform event filtering and per-analysis task of locally observed behavior. Evaluation components perform global analysis and correlation tasks. If a problem in the behavior of the supervised network is detected, an appropriate reaction is computed and enforced in the supervised system using special actuator components. Additional knowledge in the network provide information about the current context (sensed situation). Task specific analysis and reaction algorithms are made available to the supervision system by another repository component. Finally, an information transport system (coordination bus) is used to synchronize the distributed components of the supervision system.

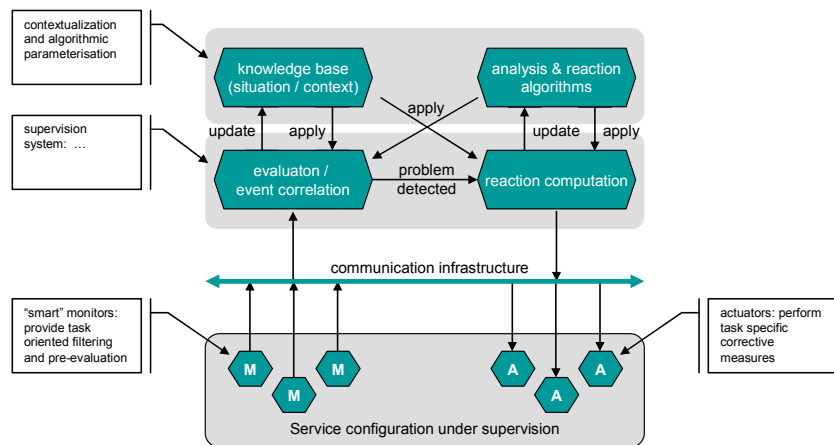


Figure 1. Basic supervision architecture

3. State of the Art

Supervision systems have been recently investigated, particularly in the context of Autonomic Computing [10] efforts, and focus mainly on automated management of Information Systems and their provisioning infrastructure. Most of them, building on the conceptual framework termed MAPE-K [16], remain topologically external to and are superimposed upon the supervision subject. Very few examples of control loops applied to communication scenarios exist. The OLIVES project [35] has experimented with a general-purpose software infrastructure, derived from work carried out within the KX project [15] (a variation of the MAPE-K conceptual framework), which remains external to the system under supervised. OLIVES has been applied to and validated in a number of application domains [7][33].

Another extrinsic approach is promoted by Rainbow [11], a framework for architecture-driven adaptation of software systems and services [23]. Rainbow has been recently used for the supervision of communi-

cation services, such as videoconferencing [11]. The Willow survivability architecture [17] aims on adopting a distributed communication and computing infrastructures for survivability, in particular to thwart attacks and intrusions. While extrinsic supervision systems have been primarily researched in the realm of computation-intensive systems, for communication-intensive services it is more common to embed supervision capabilities intrinsically within the very infrastructure that carries those systems. An example, in the field of active networking, is NESTOR [18] which inserts a self-configuration management layer within the active network stacks, to model, enact and modify the network configuration. The JSpoon language [19] complements NESTOR, making service components aware of and accessible by the NESTOR self-configuration management layer. The main argument against intrinsic approaches is that they tend to be rather inflexible and built ad hoc with respect to the supervised system.

Considering the execution monitoring of service-oriented applications, nowadays we must consider several different standards and proposals in particular from the WEB service domain. Cremona [13] proposes a architecture for the semi-automatic creation and monitoring of Web Service Agreements [12]. A different perspective is taken by Robinson, who extends the requirements monitoring techniques described in [9][8] to deal with monitoring of service-based systems focusing on functional requirements of concurrent transactions [25][26]. These requirements are acquired following the goal-driven requirement acquisition process of the KAOS framework [6]. The authors of [21][30] develop a framework for monitoring requirements for service-based systems including behavioral properties of the co-ordination process of the service based system.

Baresi et al. [1] propose an assertion-based approach to monitoring: By inserting commented annotations into a BPEL process it is possible to add assertions to be monitored at run-time by an external web service called a monitor. The University of Trento proposes assertions (business rules) classified along two dimensions: operational assertions and actor assertions. Assertions are classified on the basis of the operational context and complexity of the assertion. Two languages are provided by the framework: XSAL [20] (definition of business assertions), and XSRL [24] (definition of client requests). Both predicate using terms from standard business processes provided by the market maker (domain maker).

DIANA [27] proposes an algorithm for monitoring a distributed program's execution for violations of safety properties based on formulae written in PT-DTL,

a variant of PT-LTL (Past Time Linear Temporal Logic) capable of predicating on remote expressions and remote formulae without the use of global or shared variables. Finally, similarly to what proposed by Mandel [22], Canfora et al. [3] propose to adopt proxy services to perform monitoring. A similar approach was taken in [14] for proxy based on-line testing of Corba Components [5].

4. An Illustrative Example

This section introduces a simple example to explain the problem addressed in the paper, the *Pizza Company* [2]. Following the process definition of Figure 2, we can informally state the requirements of the application.

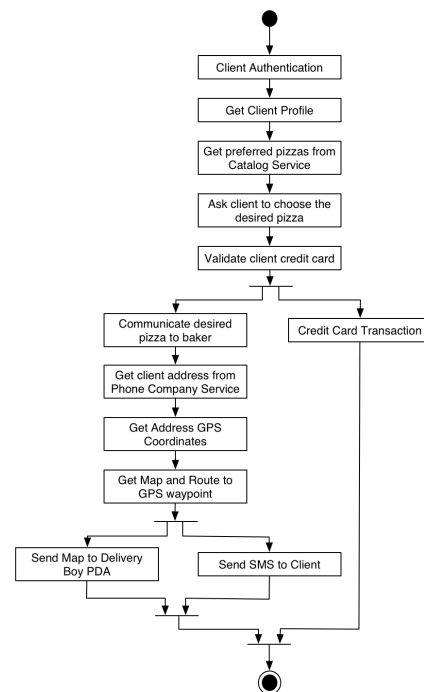


Figure 2: Example process

Suppose that a client orders a pizza. The client dials the *Pizza Company* with a browser-enabled mobile phone and, after suitable identification (*Authenticate Service*), his/her profile (*Profile Service*) determines which kind of pizza the client likes. The *Pizza Catalog Service* then offers the client four kinds of pizza; after selecting the favorite one (Double Cheese), the client provides his/her credit card number (included in the client's profile) which is validated by the *Credit Card Validation Service*. If everything is okay, the client's account is debited and the pizza company's account is credited. Meanwhile, the pizza baker is alerted to the order, because after the selection the pizza appears in

his browser, which is integrated with his cooking equipment.

At this point, the *Phone Company Service* is used to obtain the address of the client using his/her telephone number. The *GPS Service* is then called to get the coordinates of the delivery point. These coordinates are passed onto a *Map Service*, which sends a map with the exact route to the pizza delivery boy on his PDA. In the mean time the client receives an SMS announcing the delivery of the pizza within 20 minutes.

If we think of the supervision features required by this example application, we can easily consider that:

- Since in the end, clients want their pizzas and they also want to be sure that their credit cards are handled in the right way, the scenario must exploit a reliable infrastructure.
- The whole application must run on a secure infrastructure. This means that the interactions among the different services/components involved in the application must assume a secure layer.
- Besides the supervision that must be conceived for the supporting infrastructure, the application itself needs to be monitored and supervised. For example, the map sent to the delivery boy must have a given quality and it must also be cheap. We can also imagine that the delay with which the client receives the SMS must not exceed a given threshold.

5. Pervasive Supervision

The example discussed in the previous section clarifies a number of properties of supervision systems:

- Supervision has to take place on a large variety of levels (infrastructure, service execution, accounting, client's profile) and in a heterogeneous technological environment, but also has to take into account various conceptual levels (quality of the delivery boy in opposite to reliability of e.g. financial transactions). The conclusion is that neither the intrinsic nor the extrinsic approach for supervision is sufficient.
- Particularly, since the supervision system itself is vulnerable against malfunctions, this architecture does not provide a robust and self-healing solution.
- Autonomic systems are supposed to be highly dynamic in configuration and behavior (suppose local availability of different communication and location technologies, alternative "pizza service" providers with different service models, etc.). Thus a supervision system constructed using the principle architecture is able to react accordingly to new

requirements only if it exhibits self-adaptation features by itself.

From this discussion it becomes clear that a supervision system must be understood as an integral part of the supervised system that cannot be separated architecturally, organizationally, or technologically from it. *Supervision appears itself to be a composed, structural integrated service unifying the intrinsic and extrinsic paradigm.* Because of these observations, we call our approach **pervasive supervision**. Instead of talking about a single supervision system, we refer to a **supervision pervasion**.

As we assume that infrastructures have the capability to perform run-time configurable task (as intended in the CASCADAS project), we deploy not a single supervision configuration but a skeleton configuration on each or at least a sufficiently large number of network locations. By a skeleton configuration we mean a rudimentary supervision system that is capable to accept and to perform various supervision tasks. Thus we end up with a distributed capability for supervision. Robustness requirements now can be fulfilled by deploying several parallel instances of the same supervision task on various locations (synchronized by some distributed coordination protocol), ending up in system configuration that is robust against malfunctions. If one of these supervision subsystem system senses that one of the service providing components is not longer active, it coordinates with all other components in order to negotiate that it is in charge to solve the problem. Then it delegates the functions provided by the lost component on another component (assuming such an element is available), and re-configures the associated services. Note that there is no difference to the reaction on the loss of an atomic service provided for functional purposes and the loss of an atomic supervision service.

In an environment that provides dynamic service creation and composition, supervision tasks are itself formulated in terms of (composed) services. Thus there is no technological and organizational difference between supervision system and the supervised network.

Pervasive supervision is "holistic" in the sense that supervision functions can be delegated on arbitrary locations to provide a sufficient degree of redundancy to make the supervision pervasion robust against internal problems. This of course requires resource overprovisioning.

6. Self-Evolution

Supervision methodologies and systems as discussed so far define closed control loop approach. Current analytics of such systems are often based on static

rule- or policy-based methods that react on individual changes. While static rule based methods, as currently used, are sufficient for traditional applications working in non-distributed environments, future autonomic systems will require more dynamic, highly intelligent and fully automated services that are able to operate in distributed context aware environments and as such are able to not only adapt the system but, more importantly, the supervising system itself.

The need for advanced supervision, independent of the environment they are applied to, is based on the fact that the underlying concept of individual services is of volatile nature and as such is likely to change constantly over time. Thus, continuously opening a gap between the actual model and the real world concept they were designed for. This problem, referred to as *concept drift*, implies the constant adaptation of intelligent services and their underlying models that goes beyond traditional self-learning approaches.

The concept of interest for any real world service often depends on a hidden or very complex context [32] which makes it extremely difficult to design and implement them let alone the modeling of the system that is intended to supervise it. Typical examples include almost any type of forecasting where depending on individual models have to be build/adapted constantly, e.g., seasonal or geographical specifics.

Another general problem within this area is the handling of noisy data as a supervising system is initially doomed to react on any type of data. This problem can be particular hazardous as it may cause oversensitivity or insensitivity for the supervision systems with respect to their adaptability for changing conditions by erroneously interpreting noise as a type of concept drift [34].

For most systems two types of concept drifts are relevant, firstly *continues concept drifts* which may be further divided into slow and moderate drifts depending on the speed of change they follow [31] and secondly, *sudden concept drifts* where abrupt and immediate visible changes occur. The literature [34] also differentiates between two different concepts of “concept drifts”, namely *virtual* and *real* concept drift. While virtual concept drifts only depend on changing operational data distributions, real concept drifts may also depend on any change of the underlying context. Whatever the case either one requires the update of the reflecting model.

Figure 3 illustrates the basic supervision cycle described earlier and extends the architecture to accommodate for concept drifts. While similar to the original concept it incorporates the *concept of interest* in order to analyze changing behavior. In essence, individual

context specific instances are identified / observed before distinct weights are allocated modeling the current concept of interest. Next, current model(s) are compared to past concepts and experiences to identify differences before a reaction is triggered that may adapt the current concept of interest and as such any adjacent modules of the service.

While the problem of concept drift is only one possible method to allow for self evolution, the *concept of interest* and the real world problem they reflect are key to enable autonomic services to self evolve in context aware environments. More research is needed to better model not only real world problems but also the environment they are operating in. In particular, scale free and fully autonomic services have to be built in order to implement advanced evolutionary techniques into future supervision systems.

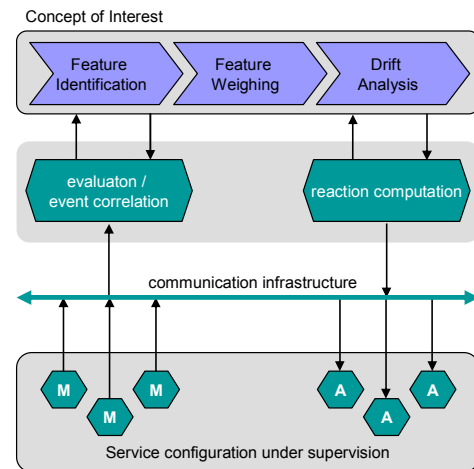


Figure 3: Extended Architecture

7. Outlook

It is clear that autonomic communications can in principle offer novel paradigms that offer self-management capabilities for the next generation of networks and higher level services. This paper seeks to explore the main opportunities for research if we draw upon models from pathology, for example, and explore how such paradigms can be introduced realistically into systems. The pervasive supervision approach we propose goes beyond the limitations of both extrinsic as well as intrinsic supervision systems, and automated monitoring approaches, by extending monitoring approaches to a complete evaluation and reaction cycle; by keeping the desirable generality and separation of concerns offered by extrinsic approaches, while avoiding its overhead, and additional architectural complication and heterogeneity; and by addressing long term adaptability and

evolutionary strategies for system intrinsic self-optimization.

References

- [1] Baresi, L., C. Ghezzi, S. Guinea, Smart Monitors for Composed Services", 2nd Int. Conf. on Service Oriented Computing, ICSOC04, 2004.
- [2] Baresi, L., C. Ghezzi, S. Guinea, "Towards Self-healing Compositions of Services", Proc. PRISE'04, 1st Conf. on Principles of Software Engineering, pp. 11 – 20, 2004.
- [3] Canfora, G., M. Di Penta, R. Esposito, M. Villani, "A Lightweight Approach for QoS-Aware Service Composition", forum paper at ICSOC, IBM Technical Report RA221 (W0411-084), 2004.
- [4] CASCADAS project homepage, <http://cascadas-project.org/>
- [5] Corba Components, version 3; OMG specification, www.omg.org
- [6] Dardenne, A., A. Lamsweerde, S. Fickas, "Goal Directed Requirements Acquisition", Science of Computer Programming, Vol. 20, 1993, 3-50, 1993
- [7] Deussen, P.H., G. Valetto, G. Din, T. Kivimaki, S. Heikkinen, A. Rocha, Continuous On-Line Validation for Optimized Service Management, in Proc. EURESOM Summit 2002.
- [8] Feather M.S., Fickas S., 1995. "Requirements Monitoring in Dynamic Environments", Proc. IEEE Int. Conf. on Requirements Engineering, 1995
- [9] Feather, M.S., 1997. "FLEA: Formal Language for Expressing Assumptions – Language Description", June 25, 1997.
- [10] Ganek, A.G., T.A. Corbi, The Dawning of the Autonomic Computing Era, IBM Systems Journal, 42(1): 5-18, 2003.
- [11] Garlan, D., S. Cheng, A. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure, IEEE Computer, 37(10):46-54, Oct. 2004.
- [12] Global Grid Forum, 2004. Web Services Agreement Specification, Version 1.1 Draft 18, 2004.
- [13] H. Ludwig, A. Dan, R. Kearney, "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements", 2nd Int. Conf. on Service Oriented Computing, ICSOC04.
- [14] Hoffmann, A., H. Batteram, W. Hellenthal, W. Romijn, A. Rennoch, A. Vouffo: "Implementation of an Open Source Toolset for CCM Components and Systems Testing", Testcom 2004, Oxford (UK), March 17-19, 2004.
- [15] Kaiser, G., J. Parekh, P. Gross, G. Valetto, Retrofitting Autonomic Capabilities onto Legacy Systems, Journal of Cluster Computing, 2005 (in press)
- [16] Kephart, J., D. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1), 2003.
- [17] Knight, J., D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbum, The Willow Survivability Architecture, in Proc. 4th Information Survivability Workshop (ISW-2001), Vancouver, B.C., 18-20 March 2002.
- [18] Konstantinou, A.V., Y. Yemini, and D. Florissi, Towards Self-Configuring Networks, in Proceedings of the DARPA Active Networks Conference and Exposition, San Francisco, Ca., USA, May 2002.
- [19] Konstantinou, A.V., Y. Yemini, Programming Systems for Autonomy, in Proc. IEEE Autonomic Computing Workshop, Active Middleware Services (AMS 2003), Seattle, Wa., USA, June 2003.
- [20] Lazovik, A., M. Aiello, M. Papazoglou, "Associating Assertions with Business Processes and Monitoring their Execution", 2nd Int. Conf. on Service Oriented Computing (ICSOC04), 2004.
- [21] Mahbub, K., G. Spanoudakis, "A framework for Requirements Monitoring of Service Based Systems", Proc. 2nd Int. Conf. Service Oriented Computing (ICSOC04), 2004
- [22] Mandell, D., S. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation", in Proc. of ISWC 2003, LNCS, Springer, pp. 227-241, 2003
- [23] Oriezy, P., et al., An Architecture-based Approach to Self-adaptive Software, IEEE Intelligent Systems, 14(3):54-62, 1999.
- [24] Papazoglou, M., M. Aiello, M. Pistore, J. Yang, "XURL: A Request Language for Web Services (www.webservices.org)", 2003
- [25] Robinson, W., "Monitoring Web Service Interactions", Proc. Workshop On Requirements Engineering and Open Systems (REOS), Monterey CA, 2003
- [26] Robinson, W., "Monitoring Web Service Requirements", Proc. 12th Int. Conf. on Requirements Engineering, 2003
- [27] Sen, S., A. Vardhan, G. Agha, G. Rosu, "Efficient Decentralized Monitoring of Safety in Distributed Systems", Proc. 26th Int. Conf. on Software Engineering, 23-28 May 2004, Edinburgh, Scotland, pages 418-427, 2004.
- [28] Smirnov, M., Autonomic Communication – Research Agenda for a New Communication Paradigm, Whitepaper, 2004, avail. as http://www.fokus.gmd.de/web-dokumente/Flyer_engl/Autonomic-Communication.pdf.
- [29] Spanoudakis, G., K. Mahbub, "Web-service requirements monitoring: Towards a framework based on Event Calculus", Proc. 19th IEEE Int. Conf. on Automated Software Engineering (ASE 2004), pp. 379-384, 2004.
- [30] Spanoudakis, G., K. Mahbub, "Web-service requirements monitoring: Towards a framework based on Event Calculus", Proc. 19th IEEE Int. Conf. on Automated Software Engineering (ASE 2004), pp. 379-384, 2004
- [31] Stanley, O.K.; "Learning concept drift with a committee of decision trees"; Report UT-AI-TR-03-302; Dep. of Computer Sciences University of Texas; 2003
- [32] Tsymbal; "The Problem of Concept Drift: Definitions and Related Work"; www.cs.tcd.ie/publications/tech-reports/reports.04/TCD-CS-2004-15.pdf; 2004
- [33] Valetto, G., G. Kaiser, Using Process Technology to Control and Coordinate Software Adaptation, in Proc. Int. Conf. on Software Engineering (ICSE 2003), May 2003.
- [34] Widmer, G., M. Kubat; "Learning in the presence of concept drift and hidden contexts"; Machine Learning 23 (1); pp 69-101; 1996
- [35] OLIVES Home Page: www.eurescom.de/public/projects/P1100-series/P1108.