Deliverable D2.1 - Report on Pervasive Supervision

State of the Art, Basic Algorithms and Approaches, and Basic Supervision Architecture

Status and Version:	Final Version	
Date of issue:	2007/04/10	
Distribution:	Public	
Author(s):	Name	Partner
	Peter H. Deussen	FOKUS
	Matthias Baumgarten	UU
	Rosario Alfano	TI
	Luciano Baresi	DEI
	Marco Plebani	DEI
Checked by:	Peter H. Deussen Franco Zambonelli Luciano Baresi Mathias Baumgarten	FOKUS Unimore DEI UU

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document reports the key results and current state of the work in the CASCADAS work package 2 (Pervasive Supervision). It provides

- an introduction to the current approach taken by WP 2
- an extensive summary of the current State of the Art
- an illustrative application example
- a collection of requirements for pervasive supervision
- a formal framework for model based supervision (in the companion document [34])
- a framework for the detection of and reaction to concept drifts
- a software architecture for supervision pervasions.

1.2 Reference Material

1.2.1 Reference Documents

[1] Canfora, G., M. Di Penta, R. Esposito, and M. L. Villani, A Lightweight Approach for QoS-Aware Service Composition, forum paper at ICSOC 2004, IBM Technical Report RA221 (W0411-084).
[2] Dietterich, T.; Ensemble Methods in Machine Learning, , In Proceedings of the 1st International Workshop on Multiple Classifier Systems. (pp. 1-10). LNCS, Vol 1857, Springer-Verlag, 2000
[3] Park, J., P.Chandramohan, Static vs. Dynamic Recovery Models for Survivable Distributed Systems. HICSS'04
[4] Patterson, D., Recovery Oriented Computing (ROC): Motivation, Definitions, Techniques, and Case Studies. Technical report.
[5] Aamodt, A., and E. Plaza, Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, Artificial Intelligence Communications 7 (1994): 1, 39-52
[6] Agrawal, R., and R. Srikant: Fast Algorithms for Mining Association Rules, Proc. Of the 20th VLDB Conference, Santiago, Chile, 1994
[7] Agrawal, R., and R. Srikant; Mining Sequential Patterns; Proc. Of the Int'l Conference on Data Engineering (ICDE); Taipei, Taiwan, March 1995. Expanded version available as IBM Research Report RJ9910, October 1994

[8] Agrawal, R., T. Imielinski, A. Swami; Mining Associations between Sets of Items in Massive Databases; Int'l Conference on Management of Data; Proc. Of the ACM-SIGMOD; Washington D.C., May 1993, 207-216.

[9] Aha, D. W., D. Kibler, M. K. Albert, Instance-based learning algorithms, Machine Learning, 6(1), 1991, pp 37-66.

[10] Ahmed, T. and A. Tripathi: 2003, 'StaticVerification of Security Requirements in Role Based CSCW Systems'. In: In 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003). pp. 196–203.

[11] Alur, R., D. L. Dill, A Theory of Timed Automata. Theoretical Computer Science, Col. 126, 1994, pp. 183 - 235

[12] An architectural blueprint for autonomic computing, Autonomic Computing White Paper, IBM, 2004

[13] Andrews, T. et al. Business Process Execution Language for Web Services, v1.1, http://www-106.ibm.com/developerworks/library/ws-bpel

[14] Baresi, L., and S.Guinea, Towards Dynamic Monitoring of WS-BPEL Processes. ICSOC 2005, 3rd International Conference On Service Oriented Computing. Amsterdam, The Netherlands, December 2005

[15] Baresi, L., C. Ghezzi, S. Guinea, Smart Monitors for Composed Services", Second International Conference on Service Oriented Computing, ICSOC04, 2004.

[16] Barkley, J. F., A. V. Cincotta, D. F. Ferraiolo, S. Gavrila, and D. R. Kuhn: 1997, 'Role Based Access Control for the World Wide Web'. In: Proc. 20th NIST-NCSC National Information Systems Security Conference. pp. 331–340.

[17] Bauer, E. and R. Kohavi, An empirical comparison of voting classification algorithms: Bagging, Boosting, and Variants. Machine Learning, 36, 105-142, Kluwer, 1999

[18] Beer, S. , The Brain of the Firm, 2nd ed., John Wiley & Sons, 1995

[19] Beer, S. , The Heart of the Enterprise, John Wiley & Sons, 1994

[20] Bettini, C., X. Wang, J. Lin, S. Jajodia, Discovering Frequent Event Patterns With Multiple Granularities in Time Sequences. IEEE Transactions on Knowledge and Data Engineering. 10 (2), 1998.

[21] Breiman, L. Bias, variance and arcing classifiers, Technical Report 460, University of California at Berkeley, (996.

[22] Breiman, L.; Bagging Predictors. Machine Learning, 24:123-140,1996.

[23] Breiman, L.; Stacked Regressions. Machine Learning, 24, 49-64, 1996.

[24] Brin, S., R. Motwani, J.D. Ullman, S. Trur, Dynamic Itemset Counting and Implication Rules for Market Basket Data. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 255-264, Tuscon, Arizona, May 13-15 1997

[25] Broadwell, P.,Sastry, N., and Traupman, J., FIG: A Prototype Tool for Online Verification of Recovery Mechanisms, In ICS SHAMAN Workshop 2002

[26] Brown, A., A Recovery-Oriented Approach to Dependable Services: Repairing Past Errors with System-Wide Undo, Technical Report. University of California.

[27] Brown, G. and J. Wyatt, J., The use of the Ambiguity Decomposition in Neural Network Ensemble learning methods. In Proceedings of the 20th International conference on Machine learning, 2003

[28] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, Design and Evaluation of a Wide-Area Event Notification Service, ACM Transactions on Computer Systems, 19(3):332-383, Aug 2001.

[29] CASCADAS project homepage, http://cascadas-project.org/

[30] Cholvy, L. and F. Cuppens: 1997, 'Analyzing Consistency of Security Policies'. In: RSP: 18th IEEE Computer Society Symposium on Research in Security and Privacy. pp. 103–112.

[31] Chomicki, J., Efficient Checking of Temporal Integrity Constraints Us- ing Bounded History Encoding. ACM Transactions on Database Systems, 20(2):149-186, June 1995

[32] Dan, A., D. Davis, R. Kearney, R. King, A. Keller, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web Services on Demand: WSLA-driven Automated Management. IBM System Journal, Special Issue on Utility Computing, volume 43, Number 1, pages 126-158, IBM Corporation.

[33] Deussen, P. H., "Supervision of Autonomic Systems", International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'2006), Erfurt, Germany, Sept. 20, 2006.

[34] Deussen, P. H., "Towards a Mathematical Framework for Pervasive Supervision", Part of the CASCADAS Milestone Deliverable D2.1, available on SVN repository as "D2.1 – Mathematical Framework.pdf"

[35] Deussen, P. H., G. Din, I. Schieferdecker: A TTCN-3 Based Online Test and Validation Platform for Internet Services. ISADS 2003: 177-184

[36] Deussen, P. H., G. Valetto, G. Din, T. Kivimaki, S. Heikkinen, and A. Rocha, "Continuous On-Line Validation for Optimized Service Management" in EURESCOM Summit 2002.

[37] Deussen, P. H., I. Schieferdecker, H. Kamoda: A Methodology for Policy Conflict Detection Using Model Checking Techniques, FORTE 2004, 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems, September 2004, Madrid, Spain.

[38] Deussen, P. H., L. Baresi, M. Baumgarten, M. Mulvenna, C. Nugent, K. Curran; Towards Pervasive Supervision for Autonomic Systems; IEEE 2006 Workshop on Distributed Intelligent Systems; Prague, Czech Republic, June 2006.

[39] Deussen, P.H., G. Valetto, G. Din, T. Kivimaki, S. Heikkinen, and A. Rocha, Continuous On-Line Validation for Optimized Service Management, in Proceedings of EURESCOM Summit 2002, Heidelberg, Germany, October 21-24, 2002.

[40] Din, G., H. Akihiro, I. Schieferdecker, P. H. Deussen: An Auditing System for QoS-Enabled Networks. IEEE 3rd Intern. Workshop on Distributed Auto-adaptive and Reconfigurable Systems, Providence, Rhode Island, USA, IEEE Press, May 2003.

[41] El-Hajj, A., and O. R. Zaïane, COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation, in Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM 2003, Melbourne, Florida, USA, 19 November, 2003

[42] eTOM Overview, http://www.tmforum.org/browse.asp?catID=1648

[43] ETSI European Standard (ES) 201 873-1 V2.2.1 (2003-02 Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language

[44] Eurescom, P1108 Workflow-based On-line Validation of Complex Component Based Internet Services - Project Conclusion and Exploitation Opportunity, 2002

[45] Eurescom, P1108 Workflow-based On-line Validation of Complex Component Based Internet Services - Terminology, Basic concepts and Notation, 2001

[46] Eurescom, P1108 Workflow-based On-line Validation of Complex Component Based Internet Services - Decentralized Work Flow Management System and Other Enabling Technologies, 2001

[47] Eurescom, P1108 Workflow-based On-line Validation of Complex Component Based Internet Services - BT6.3 Evaluation Report - D6 Final, 2002

[48] Exclusive Ore Inc., Association and Sequencing, 1998 – 2000

[49] Ganek A. G. Autonomic computing: implementing the vision Keynote presentation at the autonomic computing workshop, (AMS 2003), Seattle, WA, 25th June 2003.

[50] Garlan, D., Increasing System Dependability through Architecture-based Self-repair

[51] Garlan, D., R.T.Monroe, and D.Wile, Acme: Architectural Description of Component-Based Systems. Foundations of Component-Based Systems. Leavens, G.T., and Sitaraman, M. (eds). Cambridge University Press, 2000 pp. 47-68.

[52] 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.

[53] Hall, R.S., D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. An Architecture for Post-DevelopmentConfiguration Management in a Wide-Area Network. In Proceedings of the 1997 International Conference on Distributed Computing Systems, pages 269–278. IEEE Computer Soci-ety, May 1997.

[54] Hall, R.S., D.M. Heimbigner, and A.L. Wolf. A Cooperative Approach to Support Software Deployment Using the Software Dock. In Proceedings of the 1999 International Conference on Software Engineering, pages 174–183. Association for Computer Machinery, May 1999.

[55] Hall, R.S., D.M. Heimbigner, and A.L. Wolf. Evaluating Software Deployment Languages and Schema. In Proceedings of the 1998 International Conference on Software Maintenance, pages 177– 185. IEEE Computer Society, November 1998.

[56] Han, J., and J. Pei; Mining Frequent Patterns by Pattern-Growth: Methodology and Implications; ACM SIGKDD, Dec. 2000.

[57] Han, J., J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu; FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. Int. Conf. Knowledge Discovery and Data Mining (KDD2000), Boston, 2000, pp 355 - 259

[58] Han, J., J. Pei, Y. Yin. Mining Frequent Patterns without Candidate Generation, Proc. 2000 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000

[59] Hanemann, A, D. Schmitz, Service-Oriented Event Correlation—Workflow and Information Modeling Approached, Munich Network Management Team, Leibniz Supercomputing Center, 2004

[60] Heimbigner, D., N.Arshad, and A.L.Wolf. A Planning Based Approach to Failure Recovery in Distributed Systems. WOSS 2004

[61] Heimbigner, D., N.Arshad, and A.L.Wolf. Dealing with Failures during Failure Recovery of Distributed Systems. DEAS 2005

[62] Horn, P. Autonomic computing: IBM perspective on the state of information technology, IBM T.J. Watson Labs, NY, 15th October 2001. Presented at AGENDA 2001, Scottsdale, AR (available at http://www.research.ibm.com/autonomic/); 2001.

[63] IBM, BEA, Microsoft, SAP, Sonic Software, and VeriSign, 2004. Web Services Policy Framework (WS-Policy), September 2004 (www6.software.ibm.com/software/developer/library/ws-policy.pdf)

[64] ITIL & SERVICE MANAGEMENT PORTAL, http://www.itil-service-management-shop.com/index.htm

[65] Jakobson, G. and M. Weissman. Real-Time Telecommunication Network Management: Extending Event Correlation with Temporal Constraints. Integrated Network Management IV, IEEE Press, 1995.

[66] Jakobson, G., J. Buford, L. Lewis, Towards an Architecture for Reasoning about Complex Event-Based Dynamic Situations, Technical Report, Altusys Corp, Southern New Hampshire University, USA, 2004

[67] Joshi, N., J. Pilgrim, B. Subramanian, B. Topol, Use autonomic computing for problem determination Perform root-cause analysis with the Autonomic Management Engine and ABLE components, http://www-128.ibm.com/developerworks/autonomic/library/ac-able/

[68] Kaiser, G., J. Parekh, P. Gross, G. Valetto, "Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems." Autonomic Computing Workshop -- IEEE Fifth Annual International Active Middleware Workshop, Seattle, USA, June 2003.

[69] Kaiser, G., J. Parekh, P. Gross, G. Valetto, "Retrofitting Autonomic Capabilities onto Legacy Systems", Journal of Cluster Computing, 2005 (in press)

[70] Kephart J, Chess D. The vision of autonomic computing. IEEE Comput 2003;36:41–50.

[71] Klinkenberg, R., Learning Drifting Concepts: Example Selection vs. Example Weighting. In Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift, Vol. 8, No. 3, pp 281-300, 2004.

[72] Knight, J. C., D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbu, M. Gertz, The Willow architecture: comprehensive survivability for large-scale distributed applications, Intrusion Tolerance Workshop, The International Conference on Dependable Systems and Networks, Washington, DC, June 2002

[73] Knight, J. C., K. Sullivan, M. Elder, C. Wang. "Survivability Architectures: Issues and Approaches" In Proceedings: DARPA Information Survivability Conference and Exposition. IEEE Computer Society Press. Los Alamitos, CA, January 2000, pp. 157-171.

[74] Knight, J., D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbum, "The Willow Survivability Architecture", Proc. 4th Information Survivability Workshop (ISW-2001), Vancouver, B.C., pp. 18 – 20, 2002.

[75] Konstantinou, A.V., Y. Yemini, "Programming Systems for Autonomy", in Proc. IEEE Autonomic Computing Workshop, Active Middleware Services (AMS 2003), Seattle, Wa., USA, pp. 186-196, 2003.

[76] Konstantinou, A.V., Y. Yemini, and D. Florissi, "Towards Self-Configuring Networks", in Proc. DARPA Active Networks Conference and Exposition, San Francisco, Ca., USA, pp. 143 – 156, 2002.

[77] Kubat M., and G. Widmer, Adapting to drift in continuous domains, Tech. Report ÖFAI-TR-94-27, Austrian Research Institute for Artificial Intelligence, Vienna, 1994

[78] LaPadula, L. J. and D. E. Bell: 1973, 'Secure Computer Systems: A Mathematical Model'. Technical Report 2547, Vol. II, MITRE.

[79] Lazovik, A., M. Aiello, M. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution, Second International Conference on Service Oriented Computing, ICSOC04.

[80] Lin, C., C. Yun, M. Chen, Utilizing Slice Scan and Selective Hash for Episode Mining; 7th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining (KDD01), August 26, 2001, San Francisco, CA, USA

[81] Ludwig, H., A. Dan, and R. Kearney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. Second International Conference on Service Oriented Computing, ICSOC 2004

[82] Lupu, E. and M. Sloman: 1999, 'Conflicts in Policy-based Distributed Systems Management'. IEEE Transactions on Software Engineering — Special Issue on Inconsistency Management 25(6), 852–869.

[83] Mahbub, K., and G. Spanoudakis. "A framework for Requirements Monitoring of Service Based Systems", Proceedings of the 2nd International Conference on Service Oriented Computing - ICSOC 2004

[84] Mandell, D.J., and S.A.McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in Proceedings of ISWC 2003, Lecture Notes In Computer Science, Springer-Verlag, pp. 227-241.

[85] Mandell, D.J., and S.A.McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in Proceedings of ISWC 2003, Lecture Notes In Computer Science, Springer-Verlag, pp. 227-241.

[86] Mannila, H., and H. Toivonen: Discovering generalized episodes using minimal occurrences. 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), Portland, Oregon, August 1996. AAAI Press, p. 146-151.

[87] Mannila, H., H. Toivonen, A. I. Verkamo: Discovering Frequent Episodes in Sequences. First International Conference on Knowledge Discovery and Data Mining (KDD'95), Montreal, Canada, August 1995. AAAI Press, pp 210 – 215.

[88] Mannila, H., H. Toivonen, A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences, Data Mining and Knowledge Discovery, 1:259 289, 1997.

[89] Martin-Flatin, J. P., Distributed Event Correlation and Self-Managed Systems, CERN, IT Dept.

[90] Martin-Flatin, J.-P., Distributed Event Correlation and Self-Managed Systems, CERN, IT Dept.

[91] Masseglia, F., F. Cathala, P.Poncelet; The PSP Approach for Mining Sequential Patterns. 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD98), France, 1998, pp 176 – 184

[92] Mever, M., The features and facets of the Agent Building and Learning Environment (ABLE), http://www-128.ibm.com/developerworks/autonomic/library/ac-able1/

[93] Miseldine, P., A. Taleb-Bendiab, "Rainbow: An Approach to Facilitate Restorative Functionality within Distributed Autonomic Systems", PGNet 2005, Liverpool, 2005

[94] Moffett, J. D.: 1998, 'Control Principles and Role Hierarchies'. In: ACM Workshop on Role-Based Access Control. pp. 63–69.

[95] Ohsie, D., A. Mayer, S. Kliger, and S. Yemini, "Event Modeling with the MODEL Language : A Tutorial Introduction," SMARTS (System Management Arts), 14 Mamaroneck Ave., White Plains, New York, 10601, White Plains.

[96] OLIVES Home Page: www.eurescom.de/public/projects/P1100-series/P1108.

[97] OMG, "Object Constraint Language Specification," Object Management Group (OMG) ad/97-08-08 (version 1.1), September 1, 1997 1997.

[98] Oppenheimer, D., J. Albrecht, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. 14th IEEE Symposium on High Performance Distributed Computing (HPDC-14), July 2005.

[99] Papazoglou, M., M. Aiello, M. Pistore, and J. Yang. XSRL: A Request Language for Web Services (www.webservices.org).

[100] Park, J.-S., M.-S. Chen, P. S. Yu, Using a Hash-Based Method with Transaction Trimming for Mining Association Rules, IEEE Trans. On Knowledge and Data Engineering, Vol. 9, No. 5, October 1997, pp. 813-825.

[101] Pautasso, C., and G. Alonso. Visual Composition of Web Services. Proceedings of the 2003 IEEE Symposia on Human Centric Computinf Languages and Environments (HCC 2003), Auckland, New Zealand, October 2003.

[102] Pautasso, C., JOpera: An Agile Environment for Web Service Composition with Visual Unit Testing and Refactoring. VL/HCC 2005: 311-313

[103] Pei, J., J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proceedings of the 2001 International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.

[104] Perazolo, M., Symptoms deep dive, Part 1: The autonomic computing symptoms format Know thy symptoms, heal thyself, http://www-128.ibm.com/developerworks/autonomic/library/ac-symptom1/

[105] Perazolo, M., Symptoms deep dive, Part 2: Cool things you can do with symptoms Use common scenarios and patterns for increased autonomic computing, http://www-128.ibm.com/developerworks/autonomic/library/ac-symptom2/

[106] Pratt, K. B., and G. Tschapek: Visualizing concept drift; Int. Conf. on Knowledge Discovery and Data Mining KDD-2003; pp 735-740

[107] Ribeiro, C., A. Zuquete, P. Ferreira, and P. Guedes: 2000, 'Security Policy Consistency'. In: Workshop on Rule-Based Constraint Reasoning and Programming.

[108] Roach, A. B., Session Initiation Protocol (SIP)-Specific Event Notification, RFC 3265, June 2002 http://www.ietf.org/rfc/rfc3265.txt

[109] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002 http://www.ietf.org/rfc/rfc3261.txt

[110] Salganicoff, M., Tolerating concept and sampling shift in lazy learning using prediction error context switching, AI Review, Special Issue on Lazy Learning, 11 (1-5), 1997, 133-155

[111] Schlimmer, J.C., and R. H. Granger, Incremental learning from noisy data, Machine Learning, 1(3), 1986, pp 317-354.

[112] Schmerl, B., Aldrich J., Garlan D., Kazman R., and Yan H., Discovering Architectures from Running Systems. In IEEE Transaction on Software Engineering, Vol.32(7), July 2006.

[113] Sen, S., A. Vardhan, G. Agha, and G. Rosu. Efficient Decentralized Monitoring of Safety in Distributed Systems, Proceedings of the 26th International Conference on Software Engineering, 23-28 May 2004, Edinburgh, Scotland, pages 418-427.

[114] Shanahan, M., The event calculus explained, In Artificial Intelligence Today, LNCS: 1600, 409-430, Springer.
[115] Smirnov, M., "Autonomic Communication – Research Agenda for a New Communication Paradigm", Whitepaper, Fraunhofer Institute for Open Communication Systems (FOKUS), 2004, available at http://www.fokus.gmd.de/web-dokumente/Flyer_engl/Autonomic-Communicatin.pdf.
[116] SNMP http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.htm
[117] Spanoudakis, G., and K. Mahbub. "Web-service requirements monitoring: Towards a framework based on Event Calculus", Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE 2004), pp. 379-384
[118] Srikant, R., R. Agrawal; Mining Sequential Patterns: Generalizations and Performance Improvements, 5th Int. Conf. on Extending Database Technology (EDBT), France, 1996. Expanded version available as IBM Research Report RJ 9994, 1995.
[119] Stanley, K.O., Learning concept drift with a committee of decision trees, Tech. Report UTAI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA, 2003
[120] System Management Arts, "MODEL Language Reference Manual," White Plains, NY 1996.
[121] Toivonen, H., Sampling large databases for association rules; 22th International Conference on Very Large Databases (VLDB'96), 134 – 145, Mumbay, India, September 1996. Morgan Kaufmann.
[122] Tosic, V., W. Ma, B. Pagurek, B. Esfandiari, Web Services Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services. In Proc. of NOMS (IEEE/IFIP Network Operations and Management Symposium) 2004, Seoul, South Korea, April 19-23, 2004, IEEE, 2004, pp. 817-830
[123] Tsymbal, A., The Problem of Concept Drift: Definitions and Related Work, www.cs.tcd.ie/ publications/tech-reports/reports.04/TCD-CS-2004-15.pdf; 2004
[124] Valetto, G., G. Kaiser, "Using Process Technology to Control and Coordinate Software Adaptation", in Proc. Int. Conf. on Software Engineering (ICSE 2003), pp. 262 – 272, 2003.
[125] Wang H., W. Fan, P. S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining KDD-2003, ACM Press, 2003, 226-235
[126] Widmer, G., and M. Kubat, Learning in the presence of concept drift and hidden contexts; Machine Learning 23 (1), 1996, pp 69-101
[127] Wile, D.S., Patterns of Self-Management, WOSS'04
[128] WS-Agreement specification http://www.gridforum.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf
[129] WSDM http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm
[130] Zaki, M. J., Scalable Algorithms for Association Mining; IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, May/June 2000, pp 372-390.
[131] Zaki, M. J., SPADE: An Efficient Algorithm for Mining Frequent Sequences; in Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), Vol. 42 Nos. 1/2, Jan/Feb 2001, pp 31-60.
[132] Zeng, L., B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality driven web service composition. Proceedings of WWW 2003. pages 411-421
[133] Zeng, L., B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. QoS-Aware Middleware for Web Services Composition. IEEE Transaction of Software Engineering 30(5), pages 311-327

1.2.2 Acronyms

ABLE	Agent Building and Learning Environment
ACE	Autonomic Communication Element
ADL	Architecture Description Language
API	Application Programmer Interface
APRM	Agreement Protocol Role Management
ASRM	Agreement Service Role Management
BPEL	Business Process Execution Language
BPEL4WS	BPEL for Web Services

CBR	Case Based Reasoning
CDL	Constraint Definition Language (Nestor)
COLV	Continuous On-line Validation
Cremona	Creation and Monitoring of Agreements
CSM AP	Customer Service Management Access Point
DAP	Directory Access Protocol
DB	Data Base
Dynamo	DYNAmic Monitor
EC	Event Correlation
eTOM	Enhanced Telecom Operations Map
GPS	Global Positioning System
GUI	Graphical User Interface
IDL	Interface Definition Language
IT	Information Technology
ITIL	IT Infrastructure Library
JVCL	JOpera Visual Composition Language
KX	Kinesthetics eXtreme
LWF	Local Weighted Forgetting
MAPE	Monitor—Analyze—Plan—Execute
MIB	Management Information Base
OCL	Object Constraint Language
Olives	Online Validation Enactment System (Eurescom project P1108)
PDA	Personal Digital Assistant
PDL	Policy Definition Language (Nestor)
PECS	Prediction Error Context Switching
PMB	Policy Based Management
PMML	Predicative Model Markup Language
PS	Presence Server
PT-DTL	Past Time Distributed Temporal Logic
PT-LTL	Past Time Linear Temporal Logic
QoS	Quality of Service
RDL	Resource Definition Language (Nestor)
RDS	Resource Directory Server
ROC	Recovery Oriented Programming
SC	System Controller (Rainbow)
SIP	Session Invitation Protocol
SLA	Service Level Agreement
SLM	Sublevel Manager (self-managed systems)

SMS	Short Message Service; also Self-Managed System
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
TLM	Top Level Manager (self-managed systems)
TMF	Telemanagement Forum, also Time-Windowing Forgetting
TTCN-3	Testing and Test Control Notation (3rd Edition)
VSM	Viable System Model
WF Manager	Workflow Manager
WP	Workpackage
WS	Web Service
WSDL	Web Service Definition Language
WSDM	Web Services Distributed Management
WSLA	Web Service Level Agreement
WSOL	Web Service Offerings Language
XML	eXtendeble Markup Language
XSAL	XML Service Association Language
XSRL	XML Service Request Language

1.3 Document History

Version	Date	Authors	Comment
1.0	18/12/2006	P. H. Deussen, M. Baumgarten, L. Baresi, M. Plebani, R. Alfano	Compiled document, harmonization of contributions already done.
1.1	18/01/2007	P. H. Deussen, M. Baumgarten, L. Baresi, M. Plebani, R. Alfano	Reviewed version based on comments from F.Zambonelli, M. Baumgarten, L. Baresi
1.2	10/04/2007	P. H. Deussen, M. Baumgarten, L. Baresi, M. Plebani, R. Alfano	Final adjustments according to review comments

1.4 Document Overview

The course of developments done in WP2 and their relationships can be visualized by means of the following Figure 1.

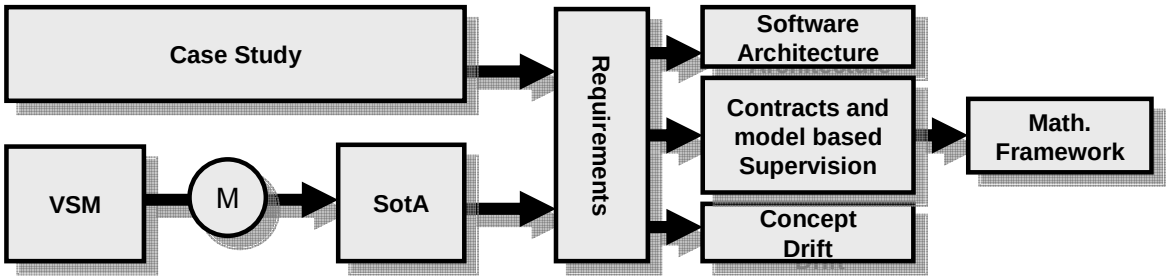


Figure 1 - Progress in WP2

The conceptual set-up of the notion of supervision and supervision pervasions has been started with the employment of the Viable System Model (VSM) [18][19] (discussed in Section 2.3) as a reference model for supervision services for hierarchical organized systems. It provides a description of the functional blocks of such a service, their interrelationships, and their relation to the system that is under supervision.

The VSM have been found its first application as a methodological reference to compare and to evaluate several general approaches for currently existing supervision systems – this work has been published in the Month 4 Milestone document on State of the Art (SotA). In parallel, a small case study has been performed to evaluate the basic ideas developed. From both activities, a number of requirements have been derived that provide cornerstones for the further activities in WP2. Currently, WP2 follows three main lines of work:

1. **Contract and model based supervision** refers to the idea that supervision is essentially a service that is provided to ACEs or ACE configurations. The specific properties or invariants to be supervised form a supervision contract that is committed between the system configuration under supervision and the supervision system. The contract is expressed in form of an operational model of the functions and abilities of the system under supervision.

   In order to express these ideas in a clear, unambiguous way, a formalization of a number of notions is required. In particular, it has to be justified what is meant by the terms like "systems", "operational models", "states", "behaviours", and so on. We therefore have developed a **mathematical framework** for the formal work with these concepts that bases on (not necessarily finite) distributed transition systems. *Partially ordered mulitsets* (i.e. sets of actions equipped with a notion of causality and operational independence) are used as operational semantics of those transition systems. We have further defined a notion of system refinement, and of system composition, that leads to a theory of system models that are hierarchically organized according to their levels of abstraction. Operations have been defined to change between local abstraction levels, i.e. to "zoom" into a model to increase its degree of detail only for a given constitutive component.

   On the basis of these ideas, two supervision algorithms have been developed, a basic one to work with "flat" models (i.e. models that comprise just a single level of abstraction), and a hierarchical one.

2. The second work line deals with long-term changes in the supervised system or its environment. The need for such a more long-term oriented supervision approach is based on the fact that the real world model of individual services or the underlying data thereof are of a 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 in order to achieve a stable state around some predefined boundaries. In order to adapt to such changes effectively a supervision mechanism needs to incorporate a computational model of the real world problem they were originally designed for. Simplified, a concept of interest reflects the underlying model of a given service or application in a machine readable format. Due to the fact that a concept of interest may depend on a hidden or very complex context it is often extremely difficult to design and implement them, let alone the modelling of the system that is intended to supervise it.

Approaches dealing with concept drifts resemble the basic supervision cycle, i.e. the continuous observation of the system under supervision and its environment, the analysis of these observations, and the enforcement of reactions if their necessity is stated. In order to allow a system to evolve over time but at the same time assure the correctness of the underlying logic, advanced forecasting and prediction methods are required which allow the system to:

- forecast the "direction" of a supervised system;
- predict individual attributes based on past behaviour or on other attributes;
- and finally, detect critical states before they actually occur.

These issues obviously are also strongly related to the notion of Knowledge Networks (WP5) which provides the information necessary to perceive and to analyze the state of the environment of a system.

3. Finally, a software architecture for supervision pervasions have been developed that is dedicated to realize the functions and relationships defined in the Viable System Model. It comprises of the following software components:

   - **Sensors** capture the data from ACEs, and the communications among them. They also send *monitored data* to the other components of the supervision system.
   - **Correlators** analyze the monitored data to construct a coherent picture of the supervised system. This component has a ***repository*** of collected information and a ***reasoner*** to extract important information from collected data.
   - **Assessors** create an abstract model of the system under supervision based on monitored data and correlation analysis. It is also able to detect if the status of single or composed elements is "suitable" or "admissible". As soon as a problem is detected, the Assessor declares it. This means that it can detect both the status of a single element under supervision and problems with the environment of the system under supervision.
   - **Planners** elaborate the set of actions that must be executed on the supervised system when the Assessor declares a problem. It uses the data received from both the Correlator and the Assessor.
   - **Effectors** translate planned recovery actions into executable actions and messages that are then sent to the supervised components.
   - **Predictors** retrieve information from the Sensors, Correlators, and Planners to predict the likely effect of planned recovery actions.

The document is structured as follows: Section 2 explains the general vision of contract based supervision. Section 3 provides an overview of the current State of the Art. The material of this section matches that on the M4 report of WP2, its presentation however is improved. Section 4 deals with an application example for pervasive supervision, namely a simplified version of the Behavioural Advertisement Example. Requirements are discussed in Section 5. Section 6 is basically a placeholder for the accompanied document [34] which describes the mathematical framework for model based supervision, and discusses also basic supervision algorithms for non-hierarchical and central-

ized as well as for hierarchical and distributed supervision. The notion of concept drift is introduced in Section 7, and basic mechanisms for its detection are discussed. A software architecture for ACE based supervision is described in Section 8. The final Section 9 provides a summary and gives an outlook on integration activities with other work packages.

2 Pervasive Supervision

Consider a distributed environment comprising a number of components providing a catalogue of *functions* or *atomic services*. Linking of functions into a certain control structure or behavioural rule system results in *composed services*, relating the components by certain communication and coordination patterns yields a *system configuration* (or a *system*, for short). By *supervision* we mean

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

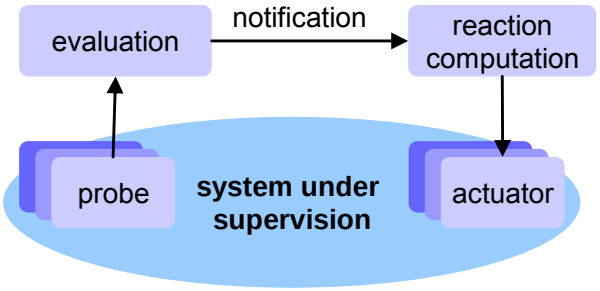


Figure 2. Basic supervision architecture

Let us start with a discussion of a paradigmatic architecture of a supervision system as shown in Figure 2. The basic organization of a supervision system is that of a closed control loop. Monitoring components gather information about the supervised system. Efficiency considerations require that probes perform event filtering and pre-analysis task of locally observed behaviour, since flooding the communication infrastructure with monitored raw data will result in an unacceptable communication overhead. Evaluation components perform global analysis and correlation tasks. If a problem in the behaviour of the supervised system configuration is detected, an appropriate reaction is computed and enforced in the supervised system using special actuator components.

In the literature two principle approaches to define a supervision system can be found:

The *intrinsic approach* defines supervision functions as a part of the supervised system itself, e.g. as special features of a middleware. This approach has the advantage that there is no technological distinction between the system under supervision and the supervised system, thus no additional management overhead occurs. Additionally, performance burdens caused by the supervision activities remain in reasonable limits. On the other hand, intrinsic solutions are often proprietary to specific problems and applications and provide no framework to define supervision tasks in a generic way.

In opposite to that, the *extrinsic approach* defines the supervision system as a unit which is technological and conceptual separated from the system under supervision. The disadvantage of this approach is that it requires the operation and maintenance of two different systems resulting in increased management efforts and performance overheads. It allows however for the generic definition of arbitrary supervision tasks and appears—under this perspective—much more appropriate for systems which react autonomously and are able perform self-adaptation.

Catering the requirements of today's and anticipated future, 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 (e.g. reliability of financial transactions in opposite to technological restrictions of the security infrastructure). Therefore, neither the intrinsic approach is sufficient because of its lack of generality, not the extrinsic approach is because of its strong separation between the two system parts. Particularly, since the supervision system itself is vulnerable against malfunctions, the basic architecture described in Figure 2 does not provide a robust and self-healing solution.

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.* We refer to this unified approach to as **pervasive supervision**.

2.1 Limitations of Current Approaches – The MAPE Example

With the Autonomic Computing initiative, IBM introduced MAPE (monitor—analyze—plan—execute) as a control paradigm for autonomic systems; MAPE boils down to a feedback control loop that continuously perceives the state of a system and interacts with it. In this section, we are using the MAPE approach as a prototypical example to discuss limitations of current approaches.

MAPE leaves open a number of questions. Consider for instance **autonomicity**. If MAPE is used as an architectural paradigm for the design of a system, and is the element that "adds" autonomicity to the very system, then the data to be gathered during monitoring, the analysis and planning algorithms, and the control functions to be executed can be defined in the development phase of the system. But if the controlled system itself is autonomic, then it is pretty unclear how to define a control loop that deals with a system that is essentially designed to operate without external control.

Consider **self-organization**. Having a system that composes itself in an automatic way from a set of available (but not necessarily pre-defined) components (which might be itself complex, self-organized ensembles), it is by no means clear which data are relevant for control purposes, how to evaluate (or even to define) their state, and how to interact with systems which are dynamically changes their inner composition. The crucial point here is that there is no *a priory* knowledge available to effectively define control purposes and tasks. As functions and structures of self-organized systems *emerge* rather than follow a pre-defined "architecture", associated control functions (and structures) have to *co-emerge*.

We conclude that—despite of the impact of the IBM's initiative on the current research addressing autonomic systems (resulting the equation "*autonomicity* = *MAPE*")—that the closed control loop paradigm is not sufficient to exemplify autonomic systems.´

This report addresses supervision approaches that go beyond the MAPE paradigm. The basic methodology is first to analyze the relationship between the supervision system and the supervised system. For this, the notion of a supervision contract is defined and requirements concerning the capability of a system of being supervised are derived from this notion. We then employ a reference model which describes the structure of a supervision system and its structural relationship to the supervised system.

2.2 Contract Based Supervision

The CASCADAS project focuses (among other things) 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 characteristics that are required by autonomic services within a ubiquitous networked environment. A multitude of ACEs will exist with each type providing varying services and capabilities. The ACE concept builds 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 such as self-organization and self-aggregation of services, security issues for autonomic services, knowledge representation and distribution by means of common overlay network, and supervision as a generic hierarchical control paradigm. Thus, if we talk about a "system under supervision", "supervision system" or "supervision pervasion", we always mean a configuration or aggregation of ACEs (or – in the most elementary case – a single ACE).

2.2.1 Supervision Contracts

Supervision is basically a service, i.e. an activity which is performed by some entity (namely: the supervision system) by executing a certain activity (e.g. improved fault tolerance, SLA validation) on a supervised system (the *object* of the activity; hence: the supervision system is the *subject* of the activity) for some other entity (the *owner* of the activity) which has a certain benefit from employing (and might pay for) this supervision service. Supervision is always performed to ensure the effectiveness and quality of some other service—called the *target service*. There are several classes of supervision contracts.

Class 1: The *service user* (which might be an automated entity) of the target services is interested e.g. to validate the quality of this service, and to enforce adaptive measures if the provided quality is insufficient. Thus the "contract" is between service user and service provider and includes—to certain extends—the permission to supervise the service execution.

Class 2: The owner of a service might be aware that its implementation can be mal-functional, malicious, non-optimal, etc. (probably because the service makes use of 3rd party components), and a supervision service is required to ensure operation and the advertised service level. Now the contract is between the service owner and the supervision system.

Class 2 contracts may be viewed as enrichments or refinements of Class 1 contracts by service specific information. The Class 1 part defines the requirements for the usage of the target service, while the Class 2 part explains how the service is realized and thus how the validation of the user requirements has to be performed.

Class 3: Consider a service composed out of available resources and functions which are—or are provided by—autonomic entities. In order to supervise such a composite service, the supervision system has to reference monitoring and actuation functions of the sub-services of the composition. We now have an interesting situation: The owner of the composed service appears to be the user of its sub-services, thus we have a contract of Class 1 between these entities. It might be however also the case that the contract is not between the owner of the composed service and the sub-services, but—assuming that there is a Class 2 contract between the several sub-services and associated supervision sub-systems—between the supervision system for the composed service and the supervision sub-services contracted by the sub-services.

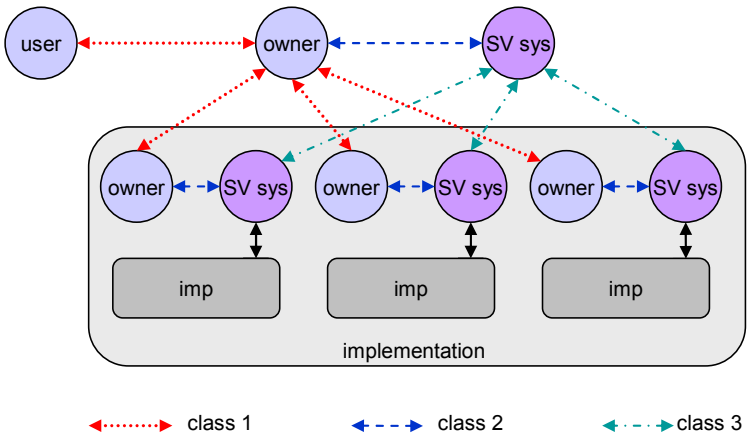


Figure 3. Types of contracts

Class 3 contracts may therefore be viewed as delegations of Class 1 contracts; Class 1 contracts may be composites of Class 3 contracts. This relationship reflects the idea to define supervision pervasions as recursive structures aligned with the (probably emerging) architecture of the service providing systems under supervision. It defines a transitive relationship between supervision components on different levels based on abstraction and concretization. The reflexive version of this relationship leads to self-application of supervision as a Class 1 contract between supervision systems.

2.2.2 What Makes A Supervision Contract?

The ability of being supervised (i.e. being perceivable and controllable) is an integral property of a system that is able to commit supervision contracts, an ability which has to be provided by all ACEs to a certain extend. A non-supervisable ACE may be regarded as non-trustable! An adaptable service configuration may refrain from invoking a non-supervisable ACE because its service level cannot be validated. But of course the extend to which an ACE is willing (allowed, designed) to disclose internals and permit external control depends on the ACE itself, its purpose, service model, security policies, etc.

The generic nature (explained above) of the supervision pervasions to be developed requires novel mechanisms for system perception and actuation; in particular, there is no definite list of attributes that need to be perceivable and modifiable defining the contents of supervision contracts because of the autonomicity and self-organization abilities of supervised service configurations.

Therefore, let us think in a different direction. In principle, service configuration (represented/symbolized by ACEs) need to be open in the sense that behaviour aspects are perceivable and controllable[1]. On the other hand, complete openness is not what is really needed as information on the level of concrete executions and concrete system states (variable values) are far too fine-grained for effective supervision. Thus the image that a supervision system maintains about the supervised service configuration is an abstraction, an *operational model* of it. Such a model describes the usage protocol (order of operations/messages, exceptions, states, etc.), the exchanged data and data types, constraints, etc. Concerning this, we may state:

> *A supervision contract is about the validation of the system's state and behaviour against its operational model* ***and*** *on the enforcement of consistency with this model.*

Note that in using this definition we do not assume that a model of a system actually is a correct description of the system—in fact, the need to have an additional supervision service is motivated by the opposite: models might be incorrect; systems may be selfish (and thus try to benefit form an idealized picture of themselves), they might get hacked, or might simply be faulty.

The relationship between the several types of contracts already has been discussed in the previous section. Let us now apply the equation "contracts = models" to make these relations more concrete:

2.2.3 What to Do With Supervision Contracts?

We elaborate the idea of a model by looking at the usage of models for supervision:

Monitoring: We assume that each ACE provides methods to query its operational model, further provides a mechanism to query the actual state in which is (e.g. by "pulling"), and events that occurs (by "pushing", callbacks, or similar). Since it is not always clear that the ACE is aware of its own computational model (it might be faulty), it is further necessary to enhance dynamic information (state and events) by semantic information, i.e. data structures (e.g. as abstract data types) and data constraints for states, and information on the state transformation function of events.

[1] We are fully aware of the security issues raised with this requirement, we however have to insist that this does not concern the WP 1 — WP 2 relationship, but has to be discussed in the WP 1 — WP 2 — WP 4 triangle.

Simulation: Models are not only used to put monitored data into a certain context, but also to pro-actively estimate the effect of the execution of some action by simulating the effect of this action within the model. This basically means that models have to be executable (at least in an abstract sense).

Event Control: Further ACEs have to provide a "controlled mode" in which the execution of events (1) is impossible without explicit external permission, and (2) can be forced by external intervention. This controlled mode is activated by the supervision system if an ACE shows an unintended behaviour or enters a bad state. We cannot assume that all events are controllable by the supervision system. Timeouts, interrupts, (human) user or operator actions, etc. are examples of events which remain uncontrollable.

Adaptation: We anticipate the need for mechanisms for the adaptation of the operational model of ACEs and ACE configurations:

1. Depending on the self-organization mechanisms we foresee the necessity of directly influencing the operational model of an ACE if the "emergent behaviour" is not the intended one.
2. Further, since supervision is intended to be "pervasive", the respective supervision components appear to be an integral part of the supervised ACE configuration, thus have to be part of the operational model of this configuration.
3. ACEs may cheat! Or may be non-informed! Or simply may be broken! We cannot rely in the model which is provided by a particular ACE. In fact, it has continuously validated against actual observations, and—in necessary—repaired.

2.3 Reference Model

In this report, we investigate the problem of finding a unified pervasive approach for supervision on a conceptual level. To do so, we employ a methodology which bases on a reference model for autonomic systems that is used as a conceptual framework which allows defining the borders of the supervision system and the supervised system, and the principal relationship of these systems. Furthermore, to solve the problem of self-supervision it is necessary to understand supervision systems itself as autonomic systems. The identified relationship "supervised system vs. supervisions system" then can be applied to the reflexive case.

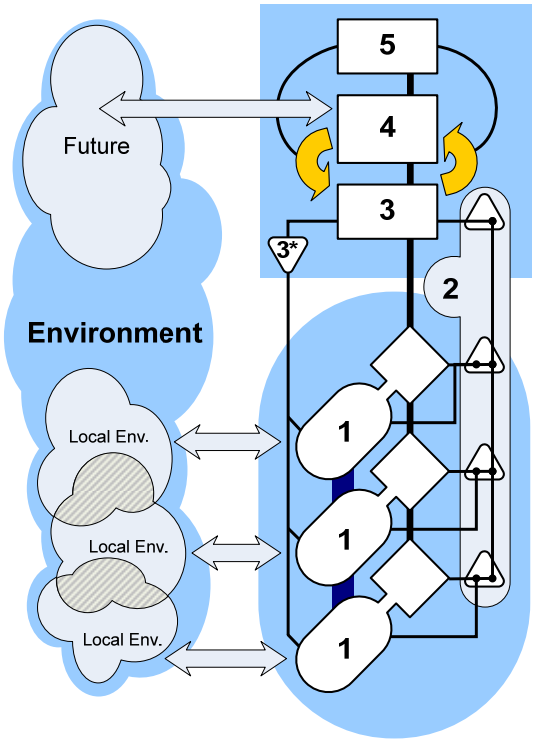


Figure 4. Viable System Model

The Viable System Model (VSM) has been introduced by Stefford Beer [18][19] as a model for the management of human organizations like enterprises, facilitating the inspiration of the (human) nerve system.

The rationale to use the VSM as a reference model for supervision pervasions is given by the fact that the VSM already exhibits a number of aspects that are important in this context, namely:

- **Continuous perception and reactive influencing** of the state and behaviour of the supervised system;
- **Pro-activity**, i.e. the extrapolation of future events and the planning of suitable

measures to support or prohibit such events;

- **Purpose-orientation**, i.e. an overall instance mapping policies into concrete actions
- **Algedonic** stimuli play a paramount role in the sense that a management entity only influences if a certain condition is meet. The VSM is not based on a continuous in-depth observation of the managed entities, but uses a restricted set of signals and commands that provide an abstracted view of the current state of the managed system.
    - *Algedonic signals* are used if the managed system enters a non-desirable state (e.g. a failure state) or is in danger to enter such a state. Thus algedonic signals define a notification of the fact that external management activities are required, because the managed entity is not able to resolve the problem situation by itself.
    - If an algedonic signal is received, the management system has issue a number of commands that lead the managed system back into a desirable state. Those commands may not only addressed to the subsystem where the problem has occurred (or has been detected), but may also involve other parts of the system.

  The VSM therefore leaves room for autonomy as long as no direct measures are required. If the managed system (or system part) turns out to be unable to deal with a problem by itself, its autonomy is restricted until the problem is resolved by a management system acting from global system perspective.
- The model is recursive and therefore allows a **pervasive organization**: Each VSM is constructed from various other VSMs. Thus it pervades the managed (or supervised, in this context) system with the management (supervision) system on various organizational levels.

  It should be noted that pervasiveness has also another, technological dimension that is not present in the theoretical VSM: In order to obtain an unified view on the couple supervised/supervision subsystem it is necessary that both subsystems uses the same technological basis and thus are manageable by the same technical means. Since the main purpose of autonomic communication (as well as of autonomic computing) is to *integrate* the complexity of a system into a comprehensive and coherent control and management approach, alleviating the technological complicity of a system with yet another (probably even more) complicated supervision system is like fighting fire with fire.

The VSM defines the structural parts and their relationships as necessary components of a *homeostatic* system, i.e. a system which is able to maintain its operation and stability when embedded into a continuously changing environment. It identifies three principal system parts (compare Figure 4): *Operational system(s)* (ellipsoid), a *meta-system* (rectangle) which controls the operational parts, and the *environment* (cloud) of the system. These system parts (with the exception of the environment) are numbered as Systems One to Five (S1 to S5, for brevity, where the numbers 1 to 5 in Figure 4 refer to these subsystems).

S1: Operation. This subsystem(s) provides the functions of the system, e.g. its productive units, computation routines, resources, etc. A VSM might comprise a large number of S1's.

The systems S2 to S5 compose the meta-system of a viable system.

S2: Stability. This system is responsible for the overall stability and coordination of the activities of the S1 units. The original VSM concentrates on the resolution of conflicts as the primary objective of S2 as the basic problems for self-organization of entities driven by humans. In the context of self-organized computer systems it is however appropriate to view S2 in general as a subsystem supporting the emergence of sustainable structures in large configurations.

While the systems S1 and S2 work mainly on the basis of a local perspective relative to a single component or the interaction of a small set of components, the systems S3 and S4 maintain a global, system-wide view.

S3: Regulation. The purpose of this system is to regulate the system's activities on a global level, to identify system wide problems, but also to utilize possible synergies and to perform optimizations. The perception of the state of the S1 subunits is done by the generation of so-called *algedonic signals*, i.e. notifications about specific events like the occurrence of problems, unexpected situations, etc. Thus algedonics provide a conceptual way not to flood the S3 subsystem with information that is not needed. S3 furthermore comprises a "command channel" that allows restricting the autonomy of the S1 subsystems if a situation occurs that cannot be handled appropriate on the S1/S2 level but requires a system-wide solution.

System S3 performs in tight cooperation with S4:

S4: Adaptation. The subsystems S2 and S3 are concerned with the current internal structure and processes of a system. The system however maintains an ongoing interaction with its environment. In order to react appropriately on developments and changes in the system's environment, the current state of the surroundings of a system has to be perceived, and additionally, an extrapolation of these future changes is necessary.

S5: Policy. Systems have a purpose (e.g. to provide a set of services). In current computer systems, this purpose is "hard-wired" in the execution code of the system. Future systems are however anticipated to have the ability to choose autonomously between several alternative (and probably contradicting) goals to achieve. Thus a system function is required to assign a value representing the "desirability" of a certain goal to enable the system to do this selection.

One might have noticed that the S1 units shown in Figure 4 are drawn by an ellipsoid and a rectangle. The VSM is *recursive* in the sense that each of the S1 units is considered to be a viable system by itself comprising operational sub-units and a meta-system, and in particular maintaining their own interactions with their environments. Concerning the recursive nature of the VSM, it is useful to establish the notion of a system-in-focus, i.e. a level of recursion being managed by a single meta-system (systems S2 – S5 and the associated S1 units viewed as atomic subsystems).

Aligning the supervision architecture shown in Figure 2 with the VSM, the most obvious way is to identify the subsystem S3 with the supervision system.

- S3 performs an ongoing monitoring of the S1 units and their interaction (S2) by the generation, perception, and interpretation of algedonic signals, thus represents the "observation" part" (probes, evaluation) of the supervision architecture.
- The "command channel" of S3 which is activated in problem situations that cannot be resolved on the S1/S2 level corresponds to the "reactive" part of the architecture (reaction computation and enforcement).

This identification is however only partial. Autonomic systems are supposed to be reactive to situational developments and to be adaptive to changes in their environments. The supervision system has to be aware of those adaptations to

- perceive the current situation to evaluate the appropriateness of the reactions of the system under supervision to it;
- align its own structure and functions to the according the structural and functional adaptations of the supervised system;
- anticipate future developments and problem situation for pro-active execution of supervision activities; pro-activity is in particular important if real-time reaction computation is not possible and has to be replaced by the deployment of pre-defined reaction patterns.

The conclusion is that the supervision system has to define (at least from the point of view of prohibiting problem situations) parts of the S4 subsystem.

The anticipated ability of autonomic systems to determine an appropriate course of action in response to perceived situations and the associated necessity to assign "values of desirability" to those plans (or their outcomes) have been discussed as the motivation to have a policy subsystem S5. But to distinguish e.g. desired from undesired states from the perspective of the system under

supervision, the accompanied supervision system need to adopt the policies of the supervised system.

The recursiveness of the VSM and the associated abstraction of concerns and task has to be has to be reflected by the supervision system. Each S1 unit therefore has its own supervision subsystem (comprising its local S3/S4/S5 subsystems). On the global level, these supervision subsystems appear as generators of algedonics and as management interfaces (command channel) which allow the supervision system to interact with the operational units. Thus it is possible to perform supervision on an appropriate level of abstraction facilitating for more concrete, but more local views on the deeper level of the recursive system structure, and for a more abstract, but more global perspective at the higher levels.

The alignment of the structure of the supervision system with the recursive structure of the system under supervision justifies the use of the term "supervision pervasion". On a particular level of this recursion, the two systems appear to be separated and integrated by defined interfaces (algedonic signals and command channels). The supervision system for this recursive level is thus able to operate "in terms of this level", i.e. at the same level of abstraction and conceptualization. From the perspective of the next higher recursive level, the system pair appears as "atomic" S1 unit, i.e. from this perspective, the supervision set-up is intrinsic.

Within the currently defined relationship between supervision system and supervised system, the supervision system—although it "pervades" the supervised system—is conceptually still a separated entity. In particular, the problem of self-supervision exists still on conceptual level when the supervision system does not follow the described VSM structure. Thus the question arises whether supervision pervasions do exhibit the basic block of viable systems.

S1: The operational units of a supervision pervasion are given by the basic supervision components – probes, evaluators, reasoning engines to compute reactions, actuators, but also the S1 units of the supervised system which are generators of algedonics and receivers of commands sent using the command channel. Moreover, the Stability System (S2) of the system under supervision appears from the perspective of the supervision system as a S1 unit.

S2 is given by the basic principles for composing the S1 units to form closed control loops. Note that the system S3 of the supervised system appears to be the part of the system S2 of the supervision system, the current system-in-focus.

This "shift of the conceptual level" indicates that the supervision system operates on a higher level than the system under supervision, i.e. justifies its role as a part of a meta-system.

S3: The purpose of the Regulation System is to manage the distribution of control loops and the relationship of these loops. An interesting issue is the nature of the algedonic signals that the S3 subsystem perceives. Consider a control loop that gets activated because a certain problem situation occurs, and tries to resolve this situation by means of the enforcement of a corrective measure. It might be successful or not. In either case, it sends a signal to the system S3 which allows evaluating the effect of the enforced reaction leading to the notion of the *competence* of the supervision system. Competence may be viewed from different angles. For instance:

- *Effectiveness* refers to the ability of a supervision system to enforce countermeasures at all. Possible reactions to the lack of effectiveness are the deployment of additional actuator components or the re-deployment of existing ones, the isolation and deactivation of system parts which cannot be affected at all, etc.
- *Timeliness* is the ability to react in time. Decrease of resolution can be an appropriate remedy, i.e. the use of faster, but probably less precise evaluation and reaction computation algorithm. There seems to be a general trade-off between Timeliness and Appropriateness:
- *Appropriateness* means that the system has the capabilities to determine a sequence of actions that leads to the desired results. Improvements are possible be increasing the resolution, but also the replacement of algorithms.

Note that on the level of self-evaluation (i.e. by means of a meta supervision system) the frequency those reconfigurations performed by the S3 subsystem yields another possible measure for the competence of the supervision system.

The responsibilities of the systems S4 and S5 and their relationships the other VSM subsystems are as described in above. Their rules within the "supervision system as meta-system" and the "supervision system as viable system" view coincide. Concerning S4, the notions related to the system's competence elaborated for S3 can be applied to measure the effectiveness, timeliness, and appropriateness of environment perceptions and anticipations of future situations which provide the motivations and triggers for adaptations performed by S3.

2.4 Questions

The conceptions outline in the previous sections raise a number of questions:

1. How to perform contract based supervision is by no means clear from the explanations given above. In particular, the following notions need to be justified:
    a. What is a model? We do not aim for a specific model formalism, but on a notion of model which is compatible with a large range of different formalisms. Using this approach, conflicts with the model formalisms used in other WPs (in particular that one defined in WP1) are avoided.
    The most basic notion of a model is that of a structure comprising a set of system states, a set of transitions between states defined by system actions, and—since we focus on distributed systems—a notion of parallel or concurrent execution of system actions.
    b. How to express suitable of system states and behaviour, i.e. how to define triggers for supervision activities? At the current stage, we use a somewhat simplified approach. We assign values of suitability only to system states (i.e. we do not look into the assessment of system behaviour and in particular not into pro-active supervision).
    c. Having metrics for the self-assessment of competence of the supervision system has to be considered as a first step towards a self-supervising system. So the question is how to define these metrics in terms of models and execution sequences of models.
    d. The notions of contracts of several classes and the idea of transitions between contract classes boil down to the questions of appropriate notions of system model refinement and composition. In particular, to understand the relationship between Class 2 and Class 3 contracts, the notion of a hierarchical model organized by levels of abstractions as well as a mechanism to change between abstraction levels is necessary.
2. On the basis of models and model contracts, how can a supervision algorithm be defined that takes into account the hierarchical structure of the VSM (and in particular the hierarchical structure of the supervision model)?
3. The VSM is not a software architecture. Such an architecture needs to be defined in terms of ACEs and compositions of ACEs and need to be harmonized in particular with the notion of Knowledge Networks developed in WP5.

3 State of the Art

This chapter intends to provide a State of the Art report on approaches, theories, and technologies which are important to the envisioned supervision approach. The compilation of a comprehensive report however is an almost impossible task:

- Restricting to approaches that aims on the definition of a supervision platform in the style of Figure 2. Basic supervision architecture would leave us with a large variety of approaches which hides similar concepts behind different terminologies, architectural variations, and application domain specifics. Apart form the problem of systematization such an approach is also likely to blind us against approaches from different research areas which are likely to give us useful additional methods and inspirations.
- On the other hand, a comprehensive description of all relevant research areas (and the relevant approaches herein) is certainly impossible because of the amount and variety of relevant research areas which has to be considered, ranging from control theory to semantic modeling, from monitoring approaches in various types of systems and networks to artificial intelligence, from agent technologies to test theory, from formal methods to (software) system engineering methodologies, from Web Services to management approaches for telecommunication systems, just to mention a view topics which *actually are* considered in this chapter.

We thus have to restrict ourselves to a selection of topics. The selection is motivated as follows:

- Firstly, it should define a starting point and a source or requirements for the research to be done in Work Package 2 not by listing all relevant approaches but those which are—from the experience of the WP 2 partners—are useful in the context of pervasive supervision.
- Secondly, it should fairly reflect the interests as well as the expertise of the WP 2 partners.

The chapter is structured as follows: Section 3.1 addresses general approaches. A number of existing supervision system architectures is discussed. The VSM is used as a reference to the comparison and evaluation of these approaches. The following Sections 3.2 to 3.5 address more specific aspects of system supervision. Section 3.2 deals with approaches for system monitoring; Section 3.3 is concerned with the evaluation of monitored data, and in particular with the detection of problems. Section 3.4 addresses approaches for system repair, and the determination of corrective measures. With reference to the VSM, Sections 3.2 to 3.4 are thus concerned with the subsystem S3. The consideration and interpretation of observations of the system's environment, and in particular the prediction of future (problem) situations, and the pro-active evolutionary adaptation of a system to those situations, is obviously a difficult task. Section 3.5 therefore addresses approaches to identify so-called "concepts of interests" and to detect and predict changes of those underlying concepts. In terms of the VSM, this section addresses the VSM subsystem S4. Finally, Section 3.6 provides a summary and conclusions.

3.1 General Approaches

In this section we are going to summarize a number of approaches related to supervision techniques which are envisioned to be elaborated in the CASCADAS project. As we will see, all these approaches depart from the basic idea to employ feedback control loops that base on the continuous observation of a system and the selection and execution of appropriate actions. Concerning this general picture, in order to compare the several approaches we employ the Viable System Model (VSM) as a reference model that incorporates all the elements, structures, and functions that we consider to be necessary for a supervision system and explains various the relationship between the supervision system and the supervised system and the environment of both subsystems.

3.1.1 Comparison Summary

The following Table 1 summarizes the comparison of the approaches described in detail in the following sections. Whenever it is not clear form the literature whether a certain property holds (Y) or not (N), we use a more precise quantification.

Table 1 Comparison of the approaches with respect to the VSM

		MAPE	**Rainbow**	**Willow**	**COLV**	**Nestor**
Integration		Extrinsic	Extrinsic / partially pervasive	Extrinsic	Extrinsic	Intrinsic
VSM Subsystems	*System 3*	Y	Y	Y	Y	Y
	System 4	Y	Y	Y	Y	Y
	System 4 anticipates future	N	Potentially	Potentially	N	N
	System 4 is adaptive	N	N	N	Considered in some instantiations	Y
	System 5	N	N	Partially	N	N
Potentially self-aware		Y	Y	Y	Y	Y
Potentially situation-aware		N	N	N	N	Y
Algedonic		N	Y	Y	Y	N
Self-applicable		N	N	N	N	N

Note that the approaches often deal with architectures for supervision systems, not with concrete instantiations those systems. Thus the question whether a certain realization of such an architecture has some property or not depends in may cases on the techniques and components specific to this realization.

3.1.2 MAPE

IBM has launched the *Autonomic Computing* initiative in 2001 as a general control paradigm. Autonomic Computing, as originally presented [49][62], is based on the idea to replace explicit system management by human operators by a number of feedback loops that altogether implement the analogy of an autonomic nervous system (in the biological sense). Although the idea is to mimic "unconscious reflexes" of living beings, the set-up of this basic reflex mechanism is quite heavy-weighted. MAPE [12][70] (monitor—analyze—plan—execute) provides a reference architecture of an autonomic manager component which is responsible for the implementation of a control loop. The architecture dissects the loop into four parts that share knowledge:

- The monitor part provides the mechanisms that collect, aggregate, filter, manage and report details (metrics and topologies) collected from an element.

- The analyze part provides the mechanisms to correlate and model complex situations (time-series forecasting and queuing models, for example). These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The plan part provides the mechanisms to structure the action needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.

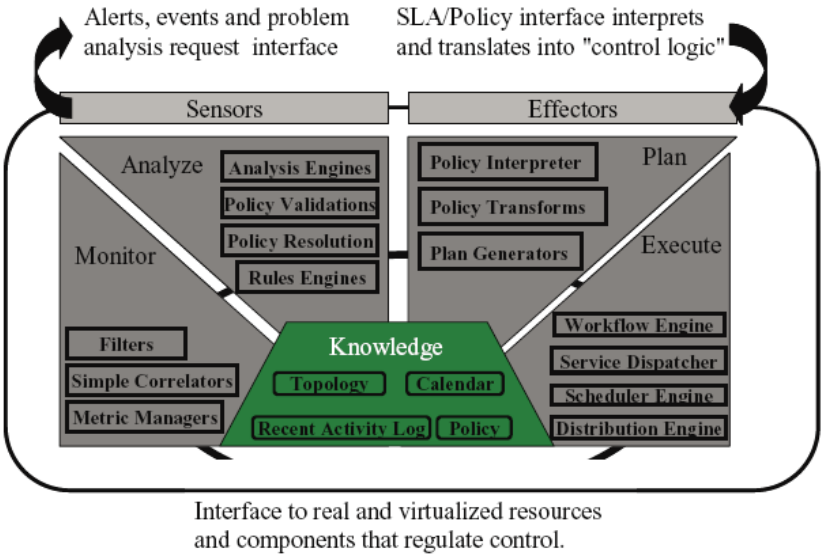


Figure 5. MAPE Architecture [12]

- The execute part provides the mechanisms that control the execution of a plan with considerations for on-the-fly updates.

The MAPE approach combines the VSM subsystem 3 and partially subsystem 4. (The managed element is of course the VSM Subsystem 1 [Subsystem 2 is not explicitly mentioned].) Although self-awareness is enabled, situation awareness is not in the focus of MAPE because sensing and interaction is mainly directed towards the controlled element itself and not toward the environment of the system (in fact, an explicit notion of an environment is not present in the MAPE reference architecture). Situation dependent adaptation of the System 4 is not considered.

Knowledge representation is by a set of deductive rules (including uncertainty and fuzzy logics). Decision making is driven by policies which are also considered as being part of the knowledge base, i.e. represented by deductive. Thus policies necessarily are related to technical control parameters (as there is no distinction between technical and non-technical rules); the step from non-technical requirements and goals to their technical consequences is missing (or at least left implicit). We therefore tend to the conclusion that the VSM subsystem 5 is not represented in the MAPE approach.

MAPE is designed as a general exclusive control paradigm. A combination with other approaches is not intended, thus for instance emergent self-organization of systems is not compatible with MAPE.

3.1.3 Rainbow

The Rainbow architecture [52][93] aims on the storage and restoration of configuration data in distributed autonomic systems. The basic idea is to use configuration snapshots to restore the configuration of a system that have been proved usefully within a given usage context of the system.

Rainbow bases on two ingredients: A so-called autonomic configuration language named Neptune and the implementation framework Cloud:

The Neptune Scripting Language enables axioms, norms, and governance rules to be symbolized within an introspective object framework, such that the individual constructs that comprise a logical statement or assignment can be both inspected and modified without recompilation at runtime In addition, Neptune scripts include an abstraction based on the separation of process flow and their underlying logical model. By linking forks expressed within the process flow with the logic

determining the path through the model, the context of a decision can be provided by way of data association.

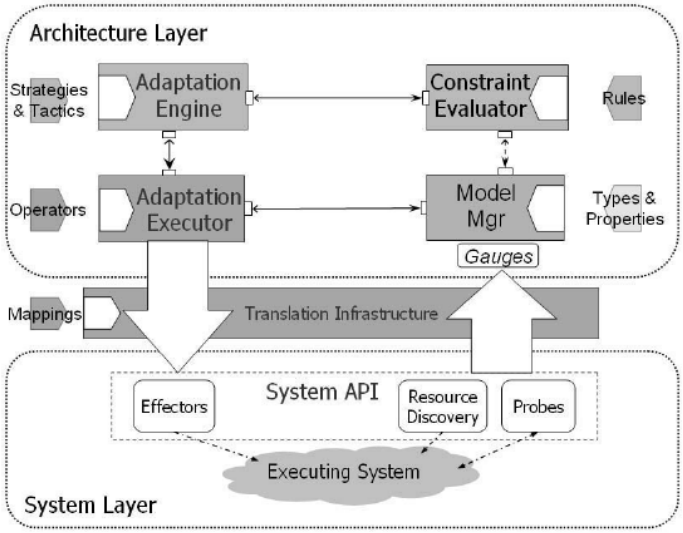


Figure 6. Rainbow Architecture [47]

The notion of a Cloud is a system with loose boundaries which can interact and merge with other such systems. A Cloud can be thought of as a federation of services and resources controlled by a system controller and discovered through a system space. In a distributed system, oftentimes services and dependencies can overlap with different configurations on different systems. Systems based on the Cloud framework can interact with each other, sharing and pooling resources for greater efficiency over a large deployment such as an enterprise. Neptune objects are executed on demand through an event model exposed by the Clouds architecture yielding a powerful extensible platform that is both wholly configurable at runtime, and that can be modelled at runtime.

Clouds interact by means of a shared distributed data storage that acts as a dashboard to publish and to access information. At the centre of a Cloud is the System Controller (SC), a distributed service that controls access to and from the individual services and resources that are within the cloud. The SC brokers requests to services based on the system status and governance rules defined in Neptune Objects.

The Rainbow architecture comprises the following elements:

System-layer infrastructure. At the system layer, the necessary system access interfaces are defined. A system measurement mechanism, realized as probes, observes and measures various states of the system. The system information may be published by or queried from the probes. A resource discovery mechanism can be queried for new resources based on resource type and other criteria. An effector mechanism carries out the actual system modification.

Architecture-layer infrastructure. At the architecture layer, gauges aggregate information from the probes and update the appropriate properties in the model. A model manager manages and provides access to the architectural model of the system. A constraint evaluator checks the model periodically and triggers adaptation if a constraint violation occurs. An adaptation engine determines the course of action and carries out the necessary adaptation.

Rainbow provides an extrinsic additional control layer responsible for monitoring and context dependent restoration of system configurations. An explicit system model is used to interpret and to classify monitored information on the current system state and its configuration. These context should not be confused with situational information, as in the previous approaches, an explicit notion of the environment of a system is not given. Thus Rainbow realized the VSM subsystems 3 (system-layer) and partially also 4 (architecture-layer) as reflection of the inner system structure is a basic

element of the architecture, but does not provide situation awareness. Model (i.e. System 4) adaptation is not considered. As Rainbow is purposed to configuration restoration it does not implement any anticipation of future situations. As being an approach having a special purpose, no System 5 is present, but since the system organization is orthogonal to the restoration system, Rainbow acts algedonically.

3.1.4 COLV — KX, Olives, and Relatives

A major challenge on the current Internet Service Providers and Telecommunication Operators infrastructure is related to the capacity to organize and orchestrate distributed and heterogeneous software components by a multitude of actors resident over various kinds of networks.

Continuous On-line validation (COLV) involves the **monitoring and adaptive control** of already deployed services during their operation, with the purposes of inspecting their impact on the network facilities and operation, investigating their interoperability, and ensuring their intended quality and performance levels at all times during operation. In practice continual validation will allow continual, run-time validation of critical system properties.

COLV aims also at intervening on the run-time service, by exerting reactive and proactive control measures that keep the running service in check with respect to its nominal functionality. The aforementioned measures thus implement active validation policies, by dynamically adapting the running service, e.g. by modifying its configuration, or its operating parameters, or by installing functional patches on the fly, etc. The range of dynamic adaptation policies that can be applied to a service may be seen as a derivation of service specifications and each dynamic adaptation action that is taken is in itself a subject of on-line validation from that moment on.

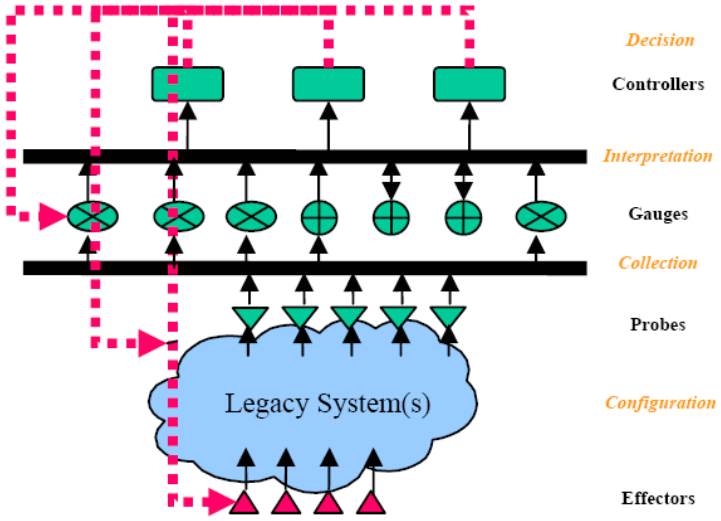


Figure 7. COLV Architecture [68]

The control loop variant of the COLV approach is shown in Figure 7.

- *Probes* are generally small, constrained, non-invasive pieces of code which get installed in or around the target application system—they may inject source code, modify byte codes or binaries, replace DLLs or other dynamic libraries, inspect network traffic, and/or perform other related tasks to collect this information;
- *Gauges* are responsible for interpreting data from these probes, and generate *semantic* events about the behaviour of the application—often operating in an effective hierarchy where higher-level gauges interpret aggregate events from lower-level gauges;

- *Controllers* receive analysis results from the gauges, and decide if and when to coordinate one or more effectors to attempt a repair.
- *Effectors* apply reconfiguration or repair, usually tuning or replacing an individual component, or spinning up a new component, as per the task(s) defined by relevant controllers.

The COLV concept as such brings not too much new concepts with respect to the VSM based view, i.e. not to many Y's in the table at the beginning of this section. It has however elaborated in various projects and works that add interesting specific elements to the proposed supervision approach.

Kinesthetics eXtreme. What is of special interest on the Kinesthetics eXtreme (KX) approach [68][69][124] (among other things) is the employment of a very specific effector concept that bases on mobile code, called Workflakes:

A Workflakes process unfolds according to a task decomposition strategy, which in the end generates, configures, activates groups of effectors, and coordinates them towards actuating the desired side effects onto the running controlled system. Worklets are code carrying agents that Workflakes selects as effectors, configures and dispatches onto the target system, as a side effect of process steps. Each Worklet carries Java mobile code snippets, and deposits them onto one or more target components, according to a programmable trajectory. Once deposited, the execution of Worklet code is governed by constructs that specify conditional execution, repetition, timing, priority, etc. The agent transport facilities and the code execution environment are provided by a Worklet Virtual Machines residing at all "stops" in a Worklet trajectory.

Thus the KX approach provides us with a way to define intelligent, asynchronously acting actuators that—equipped with a general purpose (which might be encoded in the "payload" code)—may adapt to situation and technology specific circumstances and thus allow distributed global activities.

Auditing. The KX approach has been continued in the Olives projects [96] in various case studies [37], and further developed in [35] and [39] under the main keyword "system auditing". What is novel in this work is the employment of testing technology—namely the test system definition language TTCN-3 [43] (which has been developed in the context of protocol performance testing) for the interpretation of monitored data and the classification of fault states. TTCN-3 combined the classical notion of decision trees with interaction triggered branching with procedural elements for computations and data analysis. In particular, it defines means for the set-up distributed dynamic configurations for data correlation systems and thus provides means for self-adaptation of the auditing system.

The Auditing approach to employ modern technologies for distributed testing thus equips us with means for the definition of adaptive procedural internal system representations (the VSM System 4) which are somewhere in between formal models as e.g. in Willow (Section 3.1.5), rule based approach as proposed e.g. by MAPE (Section 3.1.2), and the encryption of the image of the system (and environment) structure into code.

3.1.5 Willow

The Willow architecture [72][74] provides a comprehensive architectural approach to the provision of survivability in *critical information networks*, i.e. networks with large numbers of heterogeneous nodes that are distributed over wide geographic areas and that employ commodity hardware, and COTS and legacy software, and providing information distribution and access services for a variety of application level systems such as telecommunication systems, banking systems, etc. Damage to the information system will in many cases lead quickly to the loss of at least a large part of the service provided by the infrastructure application. *Survivability* refers to the ability of a system to continue to provide service (possibly degraded) when various faults occur in the system or operating environment.

Willow is an example of a survivability architecture in the sense of [73]. The class of faults which are in the focus of a survivability architecture are characterized as non-local (i.e. effecting a probably

large number of nodes) and non-maskable (i.e. there is no [local] mechanism that hides the effect of the fault from the perspective of the global system).

The basic architectural paradigm of survivability is that of a closed control loop that senses the controlled system and manipulates it on the basis of the perceived data. Furthermore, the following characteristics are elaborated:

- decentralized, i.e. parts of the control system act autonomously on parts of the controlled system,
- adaptive i.e. the control system provides its service in the face of changes to the controlled system and also to the control system
- hierarchical, i.e. control actions are determined various levels in a hierarchical system, with low-level control system elements influencing and being influenced by higher levels of control.

Knight et. all. [73] do not only collect requirements on the survivability control system but also analyze the necessary design characteristics of the controlled system:

- Support for application reconfiguration, i.e. the controlled system has to provide means for dynamic run-time configuration and reconfiguration. This includes:
  - Start, suspend, resume, terminate, and delay.
  - Change process priority.
  - Report prescribed status information.
  - Establish recovery point, and discard recovery point.
  - Effect local forward recovery by manipulation of local state information (e.g., reset the state).
  - Switch to an alternate application function as specified by a parameter.
  - Database management services such as synchronizing copies, creating copies, withdrawing transactions, and restoring a default state.
- Design flexibility, i.e. the controlled system must exhibit a certain degree of functional and structural flexibility in order to be able to continue its execution in the presence of global faults.
- Securing survivability mechanisms, i.e. the application of those mechanisms must not cause additional faults, corruption, system instabilities, etc. In particular, the survivability architecture has to be secured against misuse and attacks.

The Willow architecture is designed to enhance the survivability of critical networked information systems by: (a) ensuring that the correct configuration is in place and remains in place during normal operation; (b) facilitating the reconfiguration of such systems in response to anticipated threats before they occur (including security threats); and (c) recovering from damage after it occurs (including security attacks).

Reconfiguration is understood in a very broad sense as to be applied to any situation that is outside of normal, "steady-state" operation. Thus, for example, initial system deployment is included intentionally in this definition as are system modifications, posturing and so on. The system reconfigurations supported by Willow are:

- Initial application system deployment.
- Periodic application and operating system updates including component replacement and re-parameterization.
- Planned posture changes in response to anticipated threats.

- Planned fault tolerance in response to anticipated component failures.
- Systematic best efforts to deal with unanticipated failures.

Figure 8 illustrates the basic Willow architecture and its components. Sensors include reports from applications, application heartbeat monitors, intrusion detection alarms, or any other means of measuring actual network properties. From sensing events, independent diagnosis and synthesis components build models of network state and determine required network state changes. RAPTOR [72] is a formal-specification driven diagnosis and synthesis system that receives sensor events to analyze and respond with desired network changes, automatically and in bounded time. RAPTOR bases on sets of finite state machines that are used to keep track of the states of the controlled system and are driven by events emitted from the system sensors. The Administrative Workbench is an interactive application allowing system administrators to remotely monitor application conditions and adjust application properties. Additional diagnosis and synthesis components can be added by modification of the Willow specification input. Synthesis components issue their intended network changes as workflow requests. Nephele is a large-scale network workflow execution environment. It oversees coordination and arbitrates resource usage between independently synthesized work requests. Different workflows with differing intentions from different diagnosis and synthesis components might conflict, and Nephele maintains ordering of their operation to best meet the survivability goals of the application domain. When workflows are allowed to activate, workflow events are received by the Software Dock and result in local system state changes. The Software Dock infrastructure provides a single interface for universal actuation at application nodes across enterprise level networks [53][54][55]. Actuation completes the control loop cycle. All of the components of the Willow architecture interact via the Siena publish-subscribe communication system [28]. This allows efficient, scalable event-driven communication to Willow components throughout large-scale networks. In turn, the components of Willow provide efficient, scalable, well-defined, proactive and reactive network change capabilities. This enhances network application survivability, security, and manageability.

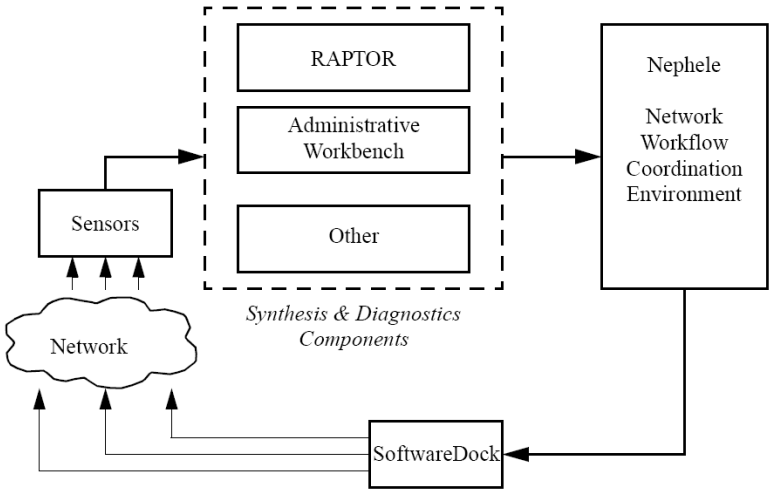


Figure 8. Willow Architecture [74]

The discussion of the Willow Survivability Architecture approach concludes with a mapping into the VSM:

The integration model—although described as hierarchical—based on a clear technological separation of the supervision (i.e. the control) system and the supervised system. The VSM System 3 is presented, System 4 is realized by the RAPTOR set of finites state machines that are used to interpret system events and to classify system states. It is not clear from the available sources whether these state machine models are used to make prediction on future system states (e.g. by simulation), but pro-activity is at least the potentially enabled. Adaptation of the System 4 is not considered. Thus we conclude that Willow provides self-awareness, but situation awareness is not maintained (or at least not discussed in the available literature). It is interesting that Willow has at least a rudimentary System 5: by employing a so-called resource manager/priority enforcer Willow is able to select between several high-level policies such as alleviating a security attack versus

masking a minor fault effecting only a few system components. These decision models are implemented by (asynchronous) workflows which define a partial order on the reconfiguration tasks to be performed by the survivability system. Finally, Self-application is not considered.

3.1.6 Nestor

The Nestor architecture [76] aims on the automation of dynamic configuration tasks in distributed network structures like Active Networks. Nestor is not strictly a supervision system in the sense of this report, we nevertheless add a description since the approach concentrates on internal system state and structure representation and thus builds a complementation to the more process oriented approaches described so far.

Nestor is concerned with several technical challenges:

- How to unify access to heterogeneous configuration databases and repositories so that configuration management tasks can be programmed and executed by software rather than manually.
- How to code knowledge of configuration consistency rules in a composable form, and enforce these rules through configuration changes,
- How to support rollback and/or recovery of operational configuration states,
- How to detect and handle emergent inconsistencies between configuration states and states controlled by underlying built-in procedures.

Data and semantic modelling play and important role in the Nestor architecture. System models are described using a variety of languages[2]:

The **Resource Definition Language** (RDL) is a subset of MODEL [95][120], a language for modelling network systems for event correlation. MODEL extends the CORBA Interface Definition Language (IDL) with support for instrumented and computed attributes, declaration of problems (events), and association relationships for modelling event propagation. Instrumented attributes are bound to values stored in the managed element, whereas computed attributes are bound to an expression that is evaluated dynamically.

The **Constraint Definition Language** (CDL) is a declarative expression language for stating assertions over the valid values of objects in RDL. As an inherent language feature, statements in CDL cannot modify any attributes or relationships in the model and do not cause side effects. Constraints may be composed from restrictions on the configuration of component devices or services. CDL is based on the Object Constraint Language (OCL) [97].

The **Policy Definition Language** (PDL) is used to assign values to configuration model objects based on the configuration of related objects and thus is used to define interrelationships and dependencies of model data.

Figure 9 describes the Nestor architecture. In the top layer, **self-configuring Applications** access a unified semantic configuration model to discover the configuration of their environment and to export their own configuration state, operational constraints, and change propagation rules. NESTOR applications access the repository using the *Directory Access Protocol* (DAP), a remote interface permitting applications to execute either locally or remotely. NESTOR uses protocol proxies to interface with legacy dynamic configuration protocols. Existing configuration servers are wrapped by NESTOR protocol proxies. Clients connecting to the proxy server continue to receive the same service with the difference that changes are effected through the NESTOR repository.

[2] Nestor provides also a plug-in mechanism using a programming language called JSpoon (a JAVA derivative). The authors of [75] claim that this language provides constructs for the programming of Autonomic Systems.

The **Self-Configuration Management** layer consists of a constraint and change propagation manager responsible for authorizing changes in the model, maintaining consistency through change propagation, and assuring that the composition of the change propagation rules does not lead to cyclical changes. The constraint and propagation manager subscribes for changes in the model and has the right to abort configuration transactions, or to effect additional changes. Its actions are controlled by CDL constraints and PDL rules.

The **Configuration Modelling** layer is responsible for maintaining the model and supporting the advanced model operations. The Resource Directory Server (RDS) maintains an object repository that stores and controls access to model object instances. Repository objects reflect configuration settings at the real network elements plus meta-information that is supplied or inferred from multiple sources.

The **Protocol Adapter** layer provides instrumentation for network elements that are not NESTOR-enabled. Adapters are responsible for propagating information, forward and backward, between the RDS repository and the managed element or service. Use of protocol adapters separates the task of mapping the unified model attributes to the real element attributes, from the protocols realizing that mapping.

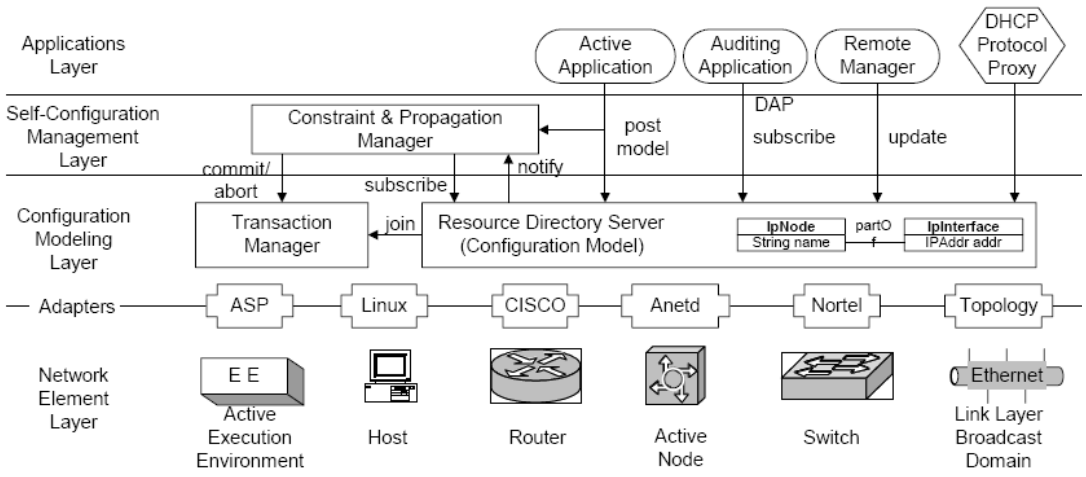


Figure 9. Nestor Architecture [76]

As said above, Nestor is not a supervision system in the sense of this report but nevertheless, and identification of the VSM equivalents of the several components of the architecture is possible.

Since Nestor is intended as a (generic) network node architecture, it is an example of the intrinsic integration of a control system with the system under control. The VSM System 3 is somewhat hidden behind various sensor and configuration functions provided both by the Application layer and the Network Element layer, and the continuous reflection of these data in the Configuration Modelling layer. System 4 is presented by the Configuration Modelling layer and the Self-configuration management layer. Both layers provide a dynamically adapting model of the system which enables for self-awareness. Since the architecture is located on specific network elements which act as autonomic entities, the environment of these elements (namely, the rest of the network) is also represented in the modelling and self-configuration management layer.

3.2 Monitoring

This section surveys the main approaches to monitor the execution of software components. Many proposals deal with Web services, but we also address other types of software elements. Illustrated approaches are divided into two different classes: languages for monitoring contracts, and existing monitoring architectures.

3.2.1 Languages for the Description of Monitoring Contracts

WSLA is a language defined by IBM; it stands for Web Service Level Agreement. WSLA is part of the *on-demand* [32] focus. WSLAs define agreements between a service provider and a service consumer. These agreements are represented as *obligations* for each party. Obligations for a service provider are to perform its service within *parameters* that represent, for example, availability, throughput, response time, etc. A *parameter* is an aggregation of *metrics* which detail what should be measured in the system and how (i.e. the algorithm). WSLAs also define the *actions* that should be executed in case of an *obligation* failure. This standard implies the monitoring of information like availability, throughput, response time, but is not limited to that and allows for constructing new monitoring information. The language offers mathematical functions to express new computations to help transform measurement into *parameters.* WSLA abstracts the measurement level and propose to express the computation of a *parameter* on top of a *measurement directive*, which is retrieve as a value from a web service. From the user perspective the WSLA language enables the declaration of composite metrics that allow constructing complex functions and the computation of non-trivial *parameters*. The user expresses the functions (that are defined as a library) to be applied on measurement directives and then produces a result. The declaration of obligations allows expressing the comparison of *parameters* value to threshold values that represent the failure.

WS-Agreement [128] is a standardization effort being conducted in the Global Grid Forum. It is an XML language for specifying agreements between service providers and consumers, and also defines a protocol for the creation of agreements using agreement templates. WS-Agreement is very generic in the way that it can use any term of service descriptions, any condition specification language, and can be composed with various negotiation models. An agreement is composed of the definition of the parties (i.e. the *context*) and the *terms* of the agreement. The main part of an agreement – the *terms* - is divided into the *service description terms* and the *guarantee terms*. The *service description terms* define the functionality that will be delivered under an agreement. The *service description terms* are generic and their content itself is dependent on the particular domain. *Guarantee terms* define the assurance on service quality, associated with the service described by the *service definition terms*. WS-Agreement defines a two layers architecture which is composed of the service layer that represent the application-specific (i.e. the business service being provided) and the agreement layer that provides a Web service-based interface that can be used to represent and monitor agreements with respect to provisioning of services implemented in the service layer. From the user perspective the WS-Agreement specification defines a very generic framework for the description of agreement and guarantee. It relies on the definition of domain specific descriptions.

In this context, we must also mention WS-Policy [63], which does not support monitoring directly, but it allows the designer to specify policies, which in turn might address monitoring.

3.2.2 Monitoring architectures

Cremona [81] proposes an architecture for the semi-automatic creation and monitoring of WS-Agreements. It provides implementations of the WS-Agreement interfaces, provides management functionality for agreement templates and instances, and defines abstractions of service-providing systems that can be implemented in a domain-specific environment. In the Cremona Architecture (Creation and Monitoring of Agreements), two different layers of agreement management help in the runtime monitoring of the agreements. The Agreement Protocol Role Management (APRM) provides functions for agreement protocol roles. It is a middleware layer for creating agreements and for accessing agreement state at runtime. In particular, it provides the functions for discovering the state of service description and guarantee terms. On the other hand, the Agreement Service Role Management (ASRM) builds on the APRM and relates it to the service implementation or to the service-consuming system. In particular, for the service provider, the ASRM includes monitors for mapping the state of a service to a status for a guarantee term.

AgFlow[3] [132][133] is one of the new middleware platforms motivated by service-oriented applications for the dynamic composition of services and guaranteeing the QoS parameters set or negotiated during the composition. AgFlow moves the problems of dynamic and QoS-aware composition from the application to the supporting infrastructure. The middleware platform considers five basic generic quality criteria: execution price, execution duration, reputation, successful execution rate, and availability, but also allows the designer to add special-purpose dimensions. The composition can be based on either *local optimization* or *global planning*. Local optimization can cause sub-optimal solutions if we consider aggregated services. This is why the second strategy (global planning) exploits a novel integer programming approach to select the best (global) set of operations for the given composition. Re-planning must be performed every time the platform signals a discrepancy between the negotiated QoS and the provided one, and also when a given service does not answer anymore. Since the middleware platform is in charge of supplying the agreed QoS, these quality dimensions must be monitored. The approach collects QoS information from service data published by the providers, execution monitoring, and requesters' feedback. Moreover, the approach distinguishes between deterministic and non-deterministic QoS criteria. In the former case, the value is known and certain when the service is invoked (e.g., the execution price); in the latter case, values are not available when services are invoked. Non-deterministic QoS variables have a random value that follows a normal distribution with a certain mean and standard deviation calculated from the history of past executions. Both classes of criteria need to be suitably monitored to ensure that there is no disagreement between agreed and measured values.

CA WSDM tool [129] monitors Web Services traffic for a variety of metrics. Transaction driven metrics include Response Time, Request Size, Reply Size, Transaction Volume and Transaction Rate. Message driven metrics include SOAP Fault monitoring and a configurable Content Monitor which enables user defined observation and alerting based on specific message content. Real time alerts are generated upon violation of any monitor threshold. All of these metrics can be monitored and deliver reports and alerts if this is deployed in a host based installation, inside the corporate firewall and enforcement points. However a majority of these metrics can be used to monitor web services that are consumed outside the firewall in a remote manner. This permits the users to monitor significant performance and availability metrics on web services that would normally be considered out of the user control. The metrics are gathered by observing the SOAP message stream. As such CA WSDM can be used by either the publisher, consumer or by a third party offering a service monitoring web services. CA set up an example of such a service called CA performance index, monitoring popular public web services. Monitoring results can be seen on the CA web site www.ca.com

JOpera [101] is, first of all, a toolkit for visual composition of services. It defines a proprietary Visual Composition Language (JVCL). This visual abstraction permits every service to be seen as a black box that takes a certain number of parameters in input and provides an output. The composition is achieved by mapping outputs to the inputs of subsequent services. JOpera can be used to compose web services, but it also can be used to compose components produced with other technologies. For example, mismatch between outputs and inputs of black-box operations are resolved by calling upon a piece of Java code. Monitoring in JOpera consists of a runtime overview of the execution of the composed process. A visual tool is provided. With this tool the single operations of the process are coloured accordingly to their state (*running*, *completed*, etc.). Throughout the execution it is possible to obtain the values of the inputs and outputs of the operations by simply looking at a sidebar that contains the operation's run-time properties. A part from the values these inputs and outputs contain, it is possible to see, in the case of Web Services, the contents of the SOAP message that was sent to and received from a certain operation. Monitoring allows, in ultimate analysis, to keep an eye on the execution of a process. So, if a process is blocked by some erroneous service, it is possible to immediately discover which of the services is responsible. Furthermore, the contents of the input and output messages sent to it can be analyzed to try and understand where the problem lies.

[3] Notice that the new system is now called self-serv. (It is unclear which new system is meant)

Spanoudakis and **Mahbub** propose a framework for monitoring requirements for service-based systems [83][117]. Their framework assumes service based systems which incorporate a central process that co-ordinates the individual services deployed by them and which is expressed in BPEL [13]. In this framework, the requirements to be monitored include: (a) *behavioural properties* of the co-ordination process of the service based system, and (ii) *assumptions* about the atomic or joint behaviour of the services deployed by the system. Both these types of requirements are expressed in a requirements specification language which is based on *event-calculus* [114]. The behavioural properties to be monitored at run-time are extracted automatically from the specification of the co-ordination process of a service-based system in BPEL while the assumptions to be monitored must be specified by the providers of the system. These assumptions must be specified in terms of: (a) events that can be observed at run-time and correspond to either operation invocation and response messages or the assignment of values to global variables used by the co-ordination process of the system, and (b) conditions over the state of the co-ordination process of the system and/or the individual services deployed by it. These restrictions ensure that requirements monitoring can be based solely on events, which are generated by virtue of the normal operation of the system without the need for instrumenting the individual services deployed by it. The requirements specification language that is used by this framework is a first-order logic language that incorporates special predicates to signify assertions about time and, to this end, it provides a very expressive framework for specifying functional requirements, which may include temporal characteristics. However, the language used by this framework does not support the specification of a full range of quality-of-service requirements including, for example, requirements expressed in terms of aggregate measures of system functionality. At run-time, the framework deploys an *event interceptor* that catches events, which are exchanged by the different services and the co-ordination process of the system, and stores them in an event database. A requirements monitor that can detect different types of violations of requirements accesses this database. These types are: (i) violations of assumptions caused by the recorded run-time behaviour of the system, (ii) violations of behavioural properties of the co-ordination process of the system or assumptions made for specific groups of services deployed by it that would have occurred if the system was functioning according to the entire set of assumptions specified for it, and (iii) unjustified actions which the system has taken by wrongly assuming that certain pre-conditions associated with the undertaken actions were satisfied at run-time. The detection of these types of violations is fully automatic and is based on an algorithm that has been developed as a variant of algorithms for integrity constraint checking in temporal deductive databases [31]. The detection of requirement violations in this framework can happen only after a violation can occur. Thus violations cannot be prevented and need to be handled after they have occurred (reactive approach). It should, however, be noted that, in its current state of development, this framework does not support the handling of these violations.

Dynamo (DYNAmic MOnitor) [14] proposes an assertion-based approach for monitoring BPEL process. A *Visual Tool* allows the process provider to define some assertions and associate them with the invocation of services. These assertions are written in WSCoL, a first order language part of the approach. The original BPEL process and defined assertions are passed through an instrumentation tool, called BPEL^2 that creates an executable and monitored version of the process: this is still pure BPEL and it is executable on any BPEL engine. The instrumented version contains additional BPEL code inserted to call the Monitoring Manager, which conversely is in charge of interacting with external Web services that act as monitors. During process execution, Dynamo (the monitoring manager) works as a proxy; it checks the data exchanged between the process and called services against the desired behaviour described in WSCoL. If an assertion is violated, Dynamo communicates the problem to the BPEL engine, which can continue or throw an exception that signals that something has gone wrong. The exception is caught by a BPEL exception handler, which in turn might perform a graceful halt on the process, communicate the error to the client of the process and exit.

University of Trento proposes a framework for associating business rules and client requests to business processes. It is capable of planning and executing a process that is compliant with the specified rules and requests. In this approach assertions (business rules) are classified along two dimensions: operational assertions and actor assertions. Assertions are classified along the

operational dimension on the basis of the operational context and complexity of the assertion. They are classified along the actor dimension on the basis of the ownership of the assertion. An operational assertion can be a simple assertion, a preservation assertion or a business entity assertion. The first is an assertion that must be true in a certain state in order to reach the next. The second is a condition maintained throughout all states reached during the execution of a process. The third is a property that applies to the evolution sequence of a process variable. An actor assertion can be a business process level assertion, a role level assertion or a provider level assertion. The first is applied to an entire business process. The second is valid for all the providers playing a certain role. The third is tied to a precise service provider. The framework is based on two languages: one for the definition of business assertions (XML Service Association Language, XSAL) [79] and one for the definition of client requests (XML Service Request Language, XSRL) [99]. Both predicate using terms from standard business processes provided by the market maker (domain maker). The system plans sequences of actions by reasoning on the combination of business level assertions, assertions provided by the client, and the business domain. This step can be iterated if a valid plan cannot be formulated and re-planning is necessary. Once execution commences the system can monitor the assertions. If violation occurs or extra information is acquired by the system, re-planning can also be undertaken.

DIANA [113] proposes an algorithm for monitoring a distributed program's execution for violations of safety properties. The monitoring is based on formulae written in PT-DTL (Past Time Distributed Temporal Logic), a proprietary 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. In this approach the, monitoring is performed locally at an actor's site. In order to achieve this, the remote values necessary to the monitoring code are passed using a so-called Knowledge Vector (a set of remote values necessary for monitoring) that is constantly piggybacked on the messages flowing throughout the system. The Knowledge Vector is added to the message by whoever sends it. Each process keeps track of one Knowledge Vector. The size of this Knowledge Vector does not depend on the number of processes in the distributed system, but on the number of remote expressions and formulae. In fact, each entry in the vector contains the values stored within a certain process and the sequence number of the last event seen at that process site. The sequence number is useful for solving problems due to asynchronous messages arriving out of order. It is the case, in fact, that older values for expressions could overwrite the new ones if the sequence number is not taken into account. With DIANA, a distributed systems application development framework is provided. To use DIANA, a user must provide a distributed program and the safety properties he/she wishes to monitor. DIANA is capable of synthesizing the monitoring code and weaving the appropriate instrumentation code into the distributed system.

Canfora et al. propose [1] another proxy services to perform monitoring. To effectively monitor QoS attributes and trigger re-planning, some information needs to be collected. The data on the actual QoS of each service are used to get more accurate QoS estimations for future executions of that service, possibly in the context of compositions, whereas the information on the workflow instances is used to predict the likelihood of each branch, and the number of iterations of each loop of the workflow, from which the overall QoS is computed. Similarly to what proposed by Mandel [85], their proxy services receive invocations from composite services, as well as from any service-oriented system, and forward the request to the invoked service. At the same time, the proxy service has the responsibility of monitoring the service QoS:

- by reading declared QoS attributes from the QoS description (expressed in any of the available languages, e.g. WSLA or WSOL - Web Service Offerings Language [122]) hyperlinked to the service WSDL;

- by directly performing some measures (e.g. checking availability, measuring response time or throughput using the Simple Network Management Protocol (SNMP) [116] possibly following the measurement specification indicated in the service QoS specification[4].

To monitor workflow executions, the BPEL description is probed with partner links to a service that collects structural information (such as paths followed), but also keeps track of the overall values of the QoS attributes of the composite service. In particular, service probes are placed on each loop and branch of the workflow, and after each service invocation.

This permits service re-planning whenever the paths followed or the QoS values measured indicate that the workflow may not be able to meet the SLA. Several scenarios may happen:

Once information related to which branch is executed in a conditional statement is known, the overall QoS can be re-estimated accordingly (the previous estimated was a weighted average of all branches QoS), and this new estimate could suggest that a re-planning is needed;

Similarly, the monitoring may indicate that the estimated number of iterations of a while statement deviates from the actual value;

Measured QoS attributes for a service can grossly deviate from the declared/estimated values; or

A service may be not available.

Although the proposed approach requires instrumenting a BPEL process description, there is no need for any intervention on the workflow engine, since measures are collected by the proxy services properly invoked by the probes.

TILab has developed a set of facilities for service management for the new Telecom Italia service platform based on the Session Invitation Protocol (SIP [109] The monitoring feature, based on the standard SIP Event Framework [108], provides the capability to collect information about and inspect the state of entities within the network as they evolve over time The monitoring feature has been designed to support the collection of different kinds of data, tailored to different network elements and services residing on them. Monitoring information is XML-based to allow for extensibility, so that any kind of component can report its specific state information. The architecture envisions that a single network element may incorporate many monitoring components, each of them publishing different information, using different XML namespaces. In that context, basic node information common to all nodes of a certain type can be provided by a core component in that node. The subscription mechanism, furthermore, allows filtering the events in various ways, constructing multiple monitoring views that respond to different needs and can be directed to diverse recipients, ranging from human readable reports, to management consoles, to software agents, to decision support systems. To detect the unavailability of network or service entities, keep-alive mechanisms are used that require network elements to frequently refresh their information, so that the monitoring server could notify watchers in near real-time about unavailability, whenever a refresh information is not received in time.

3.2.3 Conclusions

All the described infrastructures and languages assume the presence of a contract between a provider and a consumer. In the CASCADAS environment, it is difficult to identify such roles, simply because the provider role can be fulfilled by a large number of elements that can change dynamically Furthermore, in the CASCADAS infrastructure, the supervision system must oversee the behaviour of a (probably large) set of distributed elements and, to the best of our knowledge, Diana is the only proposal that considers that the elements to oversee are distributed without any centralized controller.

[4] For example, WSLA provides information to indicate how some measures can be performed: *Measurement directives* define how the measurement is conducted and which information is needed for this purpose depends strongly on the particular system to which measurement is applied.

3.3 Evaluation, Event Correlation, and Problem Detection

A key feature for a self-managed system is to detect problems by itself, reducing the resolution time, rapidly identifying the root cause of a fault, in order to ensure properly service level agreement.

Event correlation can be defined as the conceptual interpretation procedure of assigning a new meaning to a set of events that happen within a defined time interval [65]. Typically, in traditional communication networks, the analysis of events is characterized by a static environment with well defined messages with limited syntax. The strong causal relations between events and type of faults make the analysis the application of a sequence of predefined rules.

In an autonomic communication environment we have to face new challenges due to the high dynamicity of the environment, the unpredictability of the events and their syntax, the heterogeneity of the information and of the event sources.

Event correlation in such a complex and heterogeneous environment should be able to correlate events in order to identify *symptoms*. We can define a symptom as:

> *"Perception resulting from an interpreted observation that indicates a possible problem or situation in the environment. Basically, symptoms are indications of the existence of more general or serious problems"*

From this definition we may use Symptoms as a way to identify possible problems from observations in order to undertake the right actions.

In the following sections we describe different techniques for event correlations which consider the key aspects described above.

3.3.1 Distributed Event Correlation and Self-Management System

In [90], a distributed event correlation technique based on the hierarchical model and self-management features is described. The paper mainly describes the impacts of introducing autonomous systems capable of taking independent corrective actions. In fact, the paper points out how this implies change in the organizational model, which in management architecture deals with the way different entities interacts and share the management work load.

The need for distribution in event-correlation arises from the limits of the manager/agent paradigm applied to the management of systems supporting high available services. Such environment are typically characterized by managed object (application or device) which is serviced by an agent, which is responsible for updating information related to the managed object in a database known as the Management Information Base (MIB). The manager communicates with the agent to store data in the MIB (such as new configuration settings for the managed object) or to retrieve information from the MIB (such as current managed object status). Additionally, the agent can asynchronously signal the manager when an interesting event occurs. Centralized managers and static information format modelled by MIBs make this environment unsuitable for an extensible and unpredictable autonomic environment.

The self-management feature is introduced in the system by adding a component, named SMS (self-managed system) into the management model. In highly distributed environments, centralized event correlation is not applicable, hence distribution techniques has to be adopted. In the proposed technique the distribution of the event correlation task takes place at the SMS level. Basically, the SMS tries to run the event correlation tasks as local as possible, limiting the needs to propagate events to higher level components (Top-Level-Manager TLM and Sub-Level-Manager SLM).

The following figure shows the management model:

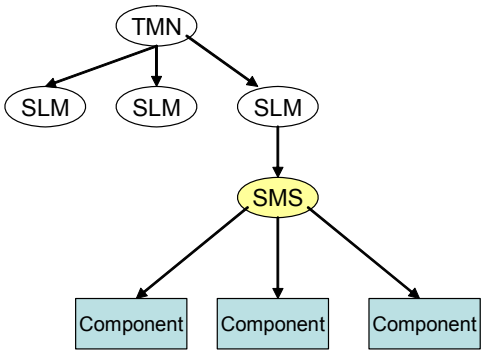


Figure 10: management model

3.3.2 Service-Oriented Event Correlation

The authors of [59] describe the impacts on event correlation due to the shift from device-oriented to service-oriented management.

An overview of the main IT process framework (ITIL [64] and eTOM [42]) is done in order to highlight how such frameworks cope with IT service management. Usually, such frameworks describe only high level process steps regarding fault management, problem detection, and other related tasks, not taking into account details on how to implement such processes.

For example in eTOM three processes are defined for fault management:

Problem handling deals with receive trouble from customer and solve them using Problem Management.

Service Problem Management deals with customer-effecting service failure. Using the information provided by the problem handling process tries to find the root causes and the problem solution. It contains a subtask named "Diagnose Problem" which tries to find the root causes performing tests but no event correlation is explicitly mentioned.

Resource Trouble Management performs the resource failure event analysis, alarm correlation and filtering.

A service model is used allowing the definition of the dependences among the services described.

The technique is based on a workflow which defines the steps needed to carry out event correlation taking into account service failures other then resource failures only. It is reached correlating the different kind of failure in order to avoid SLA violations and to identify the fault in a very short time.

The workflow defined is the following:

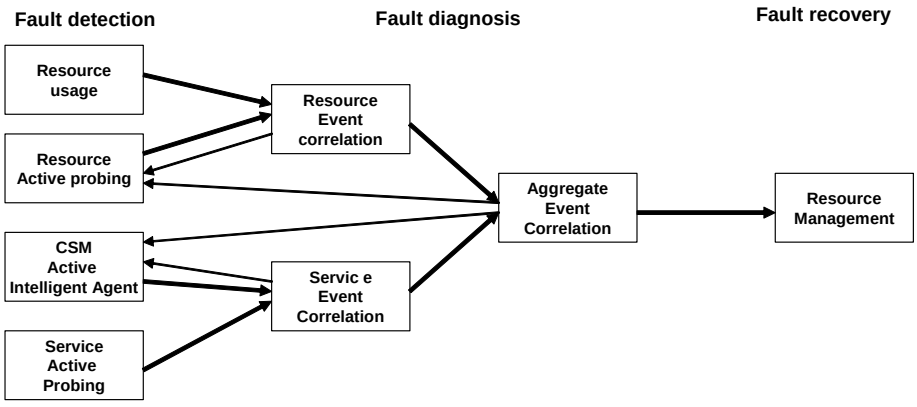


Figure 11. Event correlation workflow

The key elements shown by the figure are the following:

Customer Service Management Access Point (CSM AP) Intelligent Agent: The purpose of this element is to reduce the effort for the provider's first level support, structuring the customer's information about a service trouble. It basically uses a decision tree in order to dynamically adapt tests to customer answers.

Resource/Service Active Probing: The aim of these activities is to react to problems before a customer notices them. Such activity may be periodically or triggered by the event correlators to test important services and resources in order to achieve more information if the current correlation result needs to be improved. A specific aspect which differentiates Active Probing respect to management tools and customers is that it notifies also positive events in order to have a wider view of the resources/services behaviour.

Resource/Service Event Correlator: The most important aspects regard the dependencies between the services and resources usage. The result of this activity is a list of services/subservices which could contain a failure in the resource. The event correlation is carried out separately at resource level, where only relations between the resources are considered (e.g. caused by network topology), and at the service level, which considers relations among services and between services and resources. Information elaborated separately at these two levels is aggregated in the Aggregate Event Correlation.

The figure shows some back arrows starting from event correlation boxes to the Resource Active Probing and the CSM: those arrows should improve the correlation results; in fact, going back to the fault detection phase it is possible to start the active probing to get additional events helpful to confirm a correlation result.

3.3.3 Reasoning About Complex Dynamic Situations

The document [66] describes an architecture that implements dynamic event-based situation analysis for application areas which involve awareness of complex unfolding scenarios such as homeland security. Such applications involve dynamic objects which change their state and are involved in complex relations with other objects. In order to manage such application it is important to understand the situation in which these objects are involved in order to undertake protective actions when a threat is recognized. Situation may be defined as states which have assigned time value. That definition emphasizes the time as a critical aspect of the event-correlation for situation analysis.

The aim of this architecture is to fulfil the requirements of an environment with complex event-based dynamic situations. Such environments differ considerably with respect to traditional event management systems where usually (i) the topology of the operational environment is known, (ii) the environment is mainly static, and (iii) events are well defined and we have strong causal relations due to the propagation of faults through interconnected network components.

On the contrary, the target environment for the proposed architecture is characterized by complex temporal/spatial relations which are formed by multiple complex ontologies, the operational environment is highly dynamic, events are very diverse in nature and contents, involving signal, textual, visual and other types of information; moreover, causal relations are typically weaker.

The architecture is based on the combination of two approaches: Real-Time event correlation (EC) [65] and Case-Based Reasoning (CBR) [5]. CBR refers to a style of designing a system so that thought and action in a situation are guided by a single distinctive prior case (precedent, prototype, exemplar, or episode). It is interesting for an autonomic environment because it tries to address the following issues:

- CBR does not require an explicit domain model and so elicitation becomes a task of gathering case histories,

- implementation is reduced to identifying significant features that describe a case, an easier task than creating an explicit model,
- CBR systems can learn by acquiring new knowledge as cases thus making maintenance easier.

The architecture proposed tries to make the CBR dynamic exploiting the EC capability. Therefore, in order to manage the dynamics of the situation changes the proposed architecture uses cases for describing situations and correlated events for determining situation transitions. An example of situation could be:

- Spatial relations: s1 LOCATED_AT s2, s1 ABOVE s2, s1 CO-LOCATED s2, s1 NEAR s2
- Administrative relations: s1 SUBORDINATE_TO s2, s1 DIRECTS s2
- Structural relations: s1 PART_OF s2, s1 CONNECTED_TO s2

The system recognizes dynamic situation using CBR. A case is a sort of template for generic situation. When a new event is generated in the system, it tries to compare the event with the available templates in order to recognize the most similar known situation. The integration with EC is realized in order to use significant events to select the case then raw events. In fact the events used to select the right situation template are events generated by the event correlator. Moreover, in the reverse direction, the case has the opportunity to suggest further information to the event correlator in order to strengthen some hypothesis.

3.3.4 Root-cause Analysis

The document [67] describes a framework with reasoning capability in order to perform root cases analysis in a self-managed environment.

The framework is built by an Autonomic Management Engine which is basically used to filter and collect raw events and to make some basic correlation. ABLE [92], the reasoning module, is used to develop a rule set implementing a decision tree which supports the root case analysis. The module support both *machine learning* and reasoning, different kind of data reading and writing and inference mechanisms.

Other interesting features of the ABLE module, interesting for an autonomic environment but not used in the proposed architecture are:

1. A reflective property between the basic components which built the architecture which enables a box within a box scenario. Those mechanisms may be exploited in order to achieve dynamic adaptation by means of composition.
2. The reasoning module implements different kinds of rule engines, allowing the adoption of the most suitable engine depending on the specific problems to tackle with.
3. In order to avoid inflexibility due to the exclusive adoption of rule-based reasoning, learning capability is provided which embody neural networks. This feature provides support to build adaptive application. As far as event correlation is concerned, for example, learning may be used in order to recognize unpredictable symptoms.

3.3.5 Symptoms Deep Dive

In the definition of the autonomic elements specifying the architectural view for autonomic application developed by IBM [12] the monitor function collects the details from the managed resources, and organizes them into *symptoms* that need to be analyzed. The details can include topology information, metrics, configuration property settings and so on. This data includes information about managed resource configuration, status, offered capacity and throughput. Some of the data is static or changes slowly, whereas other data is dynamic, changing continuously through time. The monitor function aggregates, correlates and filters these details until it determines a symptom that needs to be analyzed.

The article [104] discusses a new autonomic computing symptoms format, which is an evolution of the current symptoms format [12] widely available in the autonomic computing toolkit under the Log and Trace Analyzer component. This article briefly addresses the following points:

- How knowledge in the autonomic computing context is defined;
- How symptoms are a form of knowledge;
- The various components of a symptom;
- The roles of a symptom in the autonomic computing architecture;

The article relates symptom adoption not only to self-healing but considers it as suitable to deal with other kinds of problems such as self-protecting, self-optimizing, and self-configuring.

A symptom is composed by metadata (generic part information), a schema (specific part information), and a definition, i.e. a piece of logic which is used to recognize a symptom. Practically the symptom definition specifies how events must be arranged in order to a symptom to be recognized.

What is missing in order to make a symptom-based environment completely autonomic and adaptable is the capability to build new symptom definitions autonomously, i.e. to autonomously infer a new significant sequence of events that represents a new symptom definition.

Some common IT scenarios that show how both users and IT personnel would benefit from a symptoms-based autonomic computing architecture are described in [105]. You'll also see how generic situations can be associated with relevant canonical symptoms to enable a higher degree of automation in IT processes. The scenarios include situations involving: security, service support, service availability, service continuity, business logic.

For example for service availability the following symptoms are defined:

Table 2. Example of service availability symptoms

Symptom name	Description
Resource capacity met	A given resource or set of resources is fully loaded and reached their maximum capacity
Resource unavailable	A given resource or set of resources is installed but not available
Resource degraded	A given resource or set of resources had its service level degraded
Resource unreachable	A given resource or set of resources cannot be reached
Repeated availability problem	A given resource or set of resources fails multiple times within a specific time period

3.4 Repair and Corrective Measures

This section surveys the main approaches to design recoverable applications and some research proposals in recovery oriented computing. We survey existing planning-based approaches, some architecture oriented towards dynamic changes, and a new vision in recovery computing. The last approach changes the viewpoint: a failure is considered a fact and a new goal consists in decreasing the time to repair it.

3.4.1 Planning-based approaches

Mainly adapted for Artificial Intelligence problems, the **Planning Domains and Definition Language** (PDDL) is used to describe deterministic planning domains and problems. In traditional AI planning, there are three artefacts to define a plan: a domain, an initial state, and a goal state. In order to standardize the planning terminology and to exchange results, the AI planning community has developed the PDDL language to define these three artefacts. Moreover, the AI community has constructed a number of planners that use different heuristics to compute a plan.

A PDDL planning *domain* is described by hierarchically organized types, global objects (instances of types), variables, predicates, functions, and actions. *Objects* and *variables* are called terms and each term has a type. *Predicates* and *functions* permit the definition of respectively Boolean and numeric state variables; *actions* are used to define state transitions. A planning problem consists of a set of state variables V, a set of actions A, an initial state S_0, a goal condition identifying a set of goal states, and an optimization metric that is typically a function of numeric state variables evaluated over a goal state. A state is simply an assignment of values to the set of state variables. In PDDL, a planning problem is always associated with a domain definition, and the definition of a planning problem includes a declaration of a set of problem-specific objects. The state variable for the planning problem are obtained from objects O, constraints C, predicates P, and functions F as type-consistent applications of predicates or functions to objects (including domain constants). The set A of actions is obtained similarly as type-consistent applications of action to objects.

This language can be used as starting point for an ad-hoc language to describe recovery actions; in fact PDDL is useful to describe system states and rules that must be executed to reach a desired state. Heimbigner et al. [60] propose a planning based approach to failure recovery that used PDDL. The methodology follows a classic monitoring, analyzing, planning and executing mechanism that they call Sense-Plan-Act. An Architectural Description Language (ADL) describes the architecture of the system; the plan is defined by using a pseudo version of PDDL. Their approach automates recovery by capturing the state after a failure using the ADL description, defining an acceptable recovered state as goal, and applying a PDDL planning to get from the initial state (i.e. the state in which the system is right after the occurrence of the failure) to the goal state.

Furthermore, in [61] Heimbigner et al. propose a methodology to analyze failures during recovery actions; they use the dynamic recovery model defined by Park and Chandramohan [3]. The approach is based on a three-phase model: Sense, Plan, and Execute. They assume that if a component has a fail-stop behaviour it does not start to work again; further, the recovery process is perfect and does not cause other failures. A dependency model, that is, a graph where nodes are components, describes the system and/or resources (Application Server, Databases) and arcs are dependencies between the components. They propose two kinds of dependency: *hard* and *soft*; a hard dependency represents actual functional dependency between components without which the dependent component cannot provide any real functionality; instead a soft dependency represents a use relationship between components. Furthermore, the state of a single node can be classified as: working, with no functionality, and with reduced functionality. The authors also show patterns of failure during recovery actions and propose some solutions to handle these failures.

This research proposes interesting approaches to deal with failures in distributed systems, but there are no available prototypes that can be used to evaluate the approach.

3.4.2 Architectures repair-oriented

Wile [127] introduces a meaningful set of architectural patterns for self-management to design a self-repairing system or to adapt a system not previously capable of managing itself. This kind of patterns is useful to describe how a system can be structured to deal with recovery situations.

A pattern called *resource reallocation* assumes that some probes watch the consumption of resources and gauges determine average usage and threshold violation. Then some decision logic determines how to reallocate resources, either by adding new resources to one process or removing resources already allocated to others.

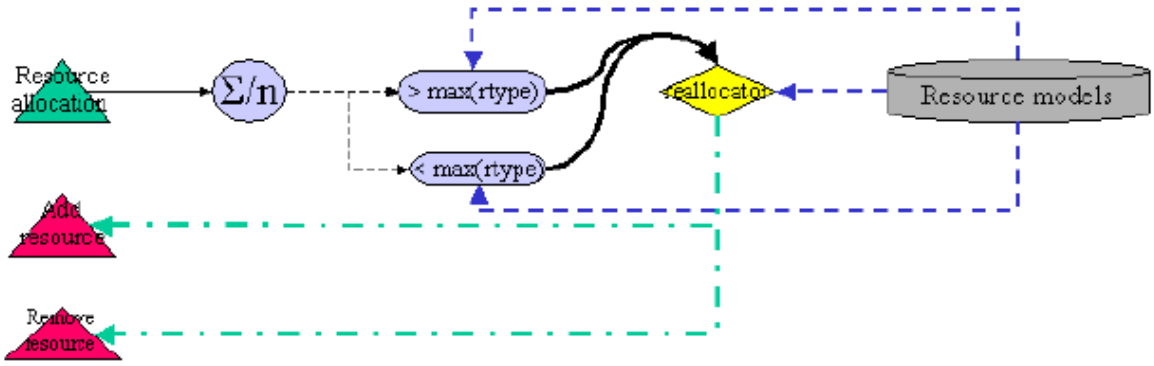


Figure 12: Resource reallocation pattern

The main goal of the pattern *corruption resiliency* is to bring the system back on track after discovering tampered resources.

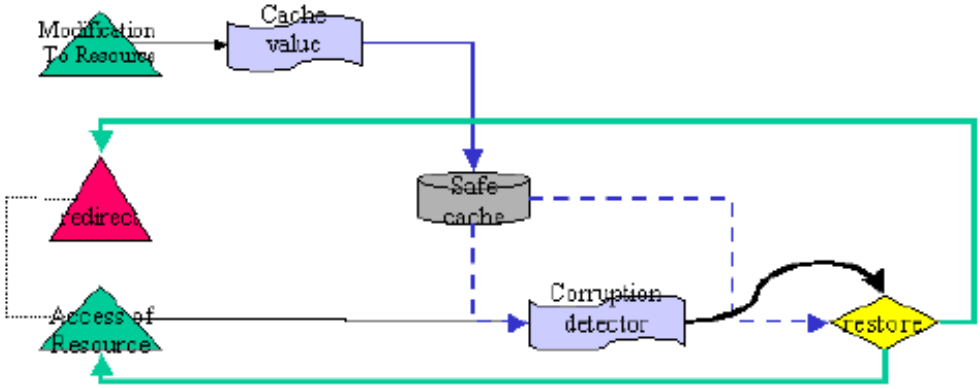


Figure 13: Corruption resiliency pattern

The *authorization* pattern uses gauges to determinate that a particular action is being attempted; a threat modem decides if this action should be prevented or is allowed to proceed. If the decision cannot be made automatically, it is delegated to the user.

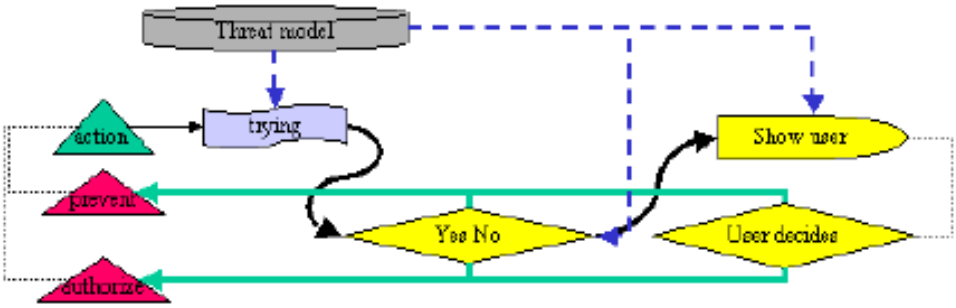


Figure 14: Authorization pattern

The pattern *model comparator* requires the construction of two models of the system: the *actual* and *simulated* model. Environmental events are executed in both models and the system compares the results.

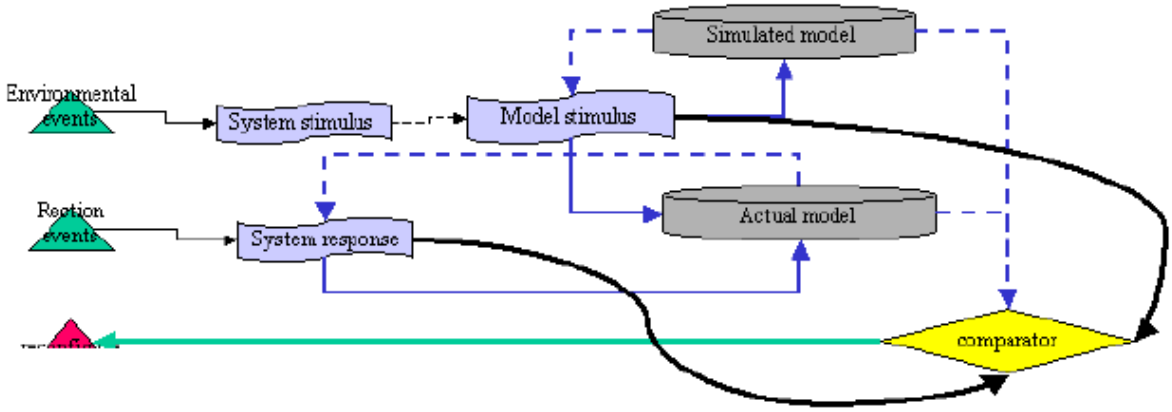


Figure 15: Model comparator pattern

The last pattern, *progress measurement* reports on the percentage the job that has been accomplished: the quantity of task to be made is "announced" by a module that can count elementary steps.

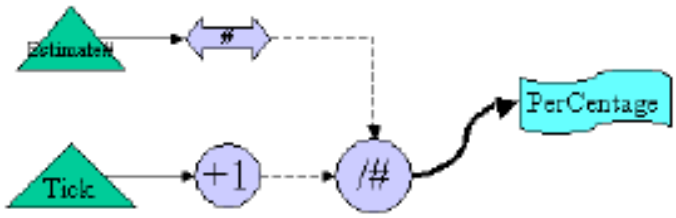


Figure 16: Progress measurement pattern

Another theoretical approach to design recoverable systems is proposed by Park and Chandramohan [3]. Their goal is to develop applications that can recovery during attacks, components failures, or accidents. They describe three different kinds of survivability models for distributed systems based on off-the-shelf components. The *static recovery model* is based on redundant servers located on the same machine or on different machines. Existing dynamic recovery actions can be simply associated with this model. In this model, possible recovery actions are based on two different modes: *Restart mode* through which client data are not stored and *Continue mode* where client states are stored when they reach a coherent state. In the *dynamic recovery model*, the components are replicated on the fly when a failure occurs. It is assumed that a component can be duplicated and deployed on any machine inside the network. In this way, there are not duplicated components. The last approach described is the hybrid survivability model that combines features of static and dynamic models. At the beginning, there is a set of redundant servers as in the static model; when a server fails, a new server is instantiated on the fly as in dynamic model.

David Garlan et al. [52] propose another approach for failure recovery in distributed systems. Garlan's approach reflects the *Model Comparator pattern* described above: it uses an ADL (Architecture Description Language) model of the system and compares it with its monitored

behaviour. The execution system is composed of many distributed machines and it is monitored to observe runtime behaviour. Monitored values are compared against an architectural model of the system: changing properties of the architectural model trigger architectural analysis to determine whether the system is operating correctly. Unacceptable operation causes repairs, which adapt the architecture, and architectural changes are propagated to the running system. The style used in this work to represent a system is composed of a system of types and a set of rules and constraints. The types are defined in ACME [51] a generic ADL language, instead rules and constraints are defined in Armani, a first order predicate logic, augmented with a set of architectural functions. To create a working prototype, the group has adapted a number of their existing software tools: *AcmeStudio*, *xAcme*, and *AcmeLib*[5] are used to define architectures in the Acme language, *AcmeStudio* is a graphical design environment, *xAcme* is a set of XML schemas to translate architectures in the Acme language into XML files, and *AcmeLib* offers two different implementations (Java and C++) of a library to manipulate Acme architectures. The *Gauge Infrastructure* is a library that allows for the definition of gauges, gauge managers, and gauge consumers. Gauges are monitoring entities that are attached to a high-level model (in most cases, to an architecture); gauge managers control the lifecycle of gauges; gauge consumers listen to gauge values, and can display or analyze these values. Moreover, to monitor the runtime behaviour of architectures described in the Armani and Acme languages, there is DiscoTect [112], a run-time system capable of constructing an architectural view of the system by observing its behaviour at runtime. DiscoTect works in sequential steps; monitored events are filtered by a Trace Engine to select out the subset of system behaviour. The resulting stream of events is then fed to a State Engine. The heart of the recognition engine is a state machine designed to recognize interleaved patterns of runtime events and to output a set of architectural operations. Those operations are then fed to an Architectural Builder that incrementally creates the architecture, which can then be displayed to a user by architecture analysis tools.

Building on previous experience gathered in the development of JOpera, Pautasso et al. [102] provide an autonomic reconfiguration component for their distributed service composition and deployment platform. It allows the system to automatically reconfigure its deployment strategy in the wake of QoS problems that can arise due to excessive workload. Recovery strategies are chosen by the autonomic controller according to a number of possible goals, such as minimizing resource allocation or response time. In practice, they consist of a modification of the number of nodes among which the deployment infrastructure is distributed. JOpera offers an open and flexible platform for service composition. It maintains a library of re-usable components, which can be dragged, dropped, and connected into data and control-flow diagrams. JOpera makes very few assumptions on the properties of these components, so that the user can freely choose to compose the most appropriate kinds of services in terms of performance, reliability, security, and convenience. Such components may represent the invocation of basic, remote services, but also, for example, job submissions to external resource management and scheduling systems, or the execution of local applications under a variety of operating systems. Additionally, composite services can be re-used in two different ways. On the one hand, computations can be decomposed hierarchically into different modules, which can be invoked and re-used independently. On the other hand, composite Grid services can define re-usable patterns of interaction between services, which can then be customized and tailored to specific applications. To do so, service interfaces are bound to actual sites providing compatible services at deployment or invocation time.

3.4.3 Recovery Oriented Computing

Recovery Oriented Computing (ROC) has a different point of view in recovery methodologies. In their manifesto [4], they consider faults, errors and bugs as facts. Their goal is to reduce the Mean Time to Repair rather than the Mean Time to Failure. The idea is to try to reduce the recovery time and thus to offer high availability.

There are three projects developed by the group:

[5] This software and libraries are available at http://www.cs.cmu.edu/~acme/

SWORD[6] [98] is a scalable resource discovery tool for wide-area distributed systems. The SWORD project aims to explore techniques for specifying and evaluating resource discovery queries in wide-area distributed systems. SWORD discovers nodes on which to deploy a service. Unfortunately, the code of the SWORD prototype is not available. The group provides a running application of SWORD on PlanetLab. Even if the code is not available, the authors provide three different interfaces to their prototype: two interfaces are web based and the last is based on a command-line client. SWORD collects reports about available resources on nodes, and answers queries from users requesting nodes matching user-defined criteria. These criteria may be per-node (e.g., load, free memory, or free disk space) or inter-node (e.g., inter-node latency). The nodes about which SWORD collects reports do not have to be the same nodes as those that are running SWORD, but for SWORD on PlanetLab, they are the same set of nodes.

UNDO[7] [26]. One of the key tenets of the ROC philosophy is that systems should provide undo functionality for their operators and administrators, to allow them to recover from human errors, as well as to recover from failed operations like software upgrades, installs, and configuration updates. Undo for System Administrators and Operators is a tool developed to achieve this goal. This approach explores system-wide undo via a framework based on the concept of *spheres of undo*, bubbles of state and time that provide scope to the state recoverable by undo and serve as a structuring tool for implementing undo on standalone services, hierarchically-composed systems, and distributed interacting services.

The traditional undo tools usually lost information and data during the undo steps; e.g. consider a user that is writing a text and he wants to recuperate a paragraph he deleted in the past; through the traditional undo system, the user loses all the changes from the deletion to the last version of the document. The aim of the Undo research is to avoid these situations.

Exploiting the concept of spheres of undo, this undo methodology works in three phases. In the *Rewind* phase, all state within the spheres of undo is rolled back in its entirety to a prior version, as recorded in the history. In the *Repair* phase, the system operator can optionally make any desired changes to the system. In the *Replay* phase: all rolled-back end-user interactions with the inner sphere of undo are re-executed against the repaired system.

Through the classic concept of Undo operation, this kind of recovery strategy permits to recover a subset of the previous operation in the history of a tool without loosing any other operation made.

One of the last studies considers Undo operation for distributed services. The source code of the UNDO prototype is available.

FIG[8] [25] is a lightweight extendible prototype for testing the recoverability of software packages against a variety of external errors. FIG is a tool for injecting errors and logging errors at the application/library boundary with minimal configuration and run-time overhead. FIG runs on UNIX-like operating systems and operates by interposing a library between the application and other function libraries that intercepts calls from the application to the system. When a call has been intercepted, FIG then choose, based on testing directives from a control file, whether to allow the call to complete normally or to return an error that simulates a failure of the operating environment. In order to facilitate the dynamically adaptation to different application to test, FIG toolkit provides an automatic stub generator. The source code of FIG is available.

3.4.4 Conclusions

The approaches illustrated in the previous sections to recover faulty systems address two different supervision tasks in CASCADAS: the planning phase and the recovery phase. In fact, planning-

[6] http://sword.ucsd.edu/

[7] http://roc.cs.berkeley.edu/projects/undo/index.html

[8] http://roc.cs.berkeley.edu/projects/fig/

based approaches face the problem by using a composition of possible corrective measures provided to the supervision system by the supervised system. They can be used during the planning phase of the supervision task. Following the architecture of self-repairing systems, supervised elements can easily change their structure.

None of presented approaches considers the fact that the overseen system can be a black box element. All the approaches know the internal structure of the supervised system; so that the recovery strategies can change the intrinsic nature of the elements. In CASCADAS, the overseen group of elements can also be composed by black box components and, in this case, the only actions allowed to the supervision system are the tuning of some parameters. Partially, Heimbigner et al. consider the problem of the distribution of elements, but their approach consider as repair measures only the connections among elements; furthermore, in CASCADAS we also need to change the internal behaviour of the ACEs.

3.5 Evolutionary Strategies

Supervision methodologies and systems as discussed so far usually follow a closed control loop approach. Current analytics of such systems is often based on static rule or policy-based methods that react on individual changes. While static rule-based methods 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 that is under supervision but, more importantly, the supervising system itself. For the above vision to be realized the rationale of the system to be supervised as well as the rationale of the supervising system need to be modelled and observed over time. While flexible and intelligent methods are required to analyze the resulting behavioural patterns, the success for future pervasive supervision systems will depend mainly on the ability to model underlying real world problems, scenarios and policies in a flexible yet fully comprehensive manner.

We call any such reflections of real world processes a 'concept of interest', which will play a central role for future supervision mechanism enabling individual systems to evolve within the boundaries of the environment they operate in. Simplified, continuous observation of the system together with long term trend analysis allows for adaptations that can improve service and system performance on different levels of granularity that go beyond traditional introspective based learning approaches.

3.5.1 The Concept of Interest

The need for advanced long term supervisions, independent of the environment they are applied to, is based on the volatile nature of the underlying model of individual services, which are likely to change constantly over time. In order to adapt to such changes effectively any supervision mechanism needs to incorporate a computational model of the real world problem they were originally designed for. This so-called *concept of interest* and the real world problem it reflects are therefore of paramount importance to enable autonomic services to self evolve in context aware environments [38]. Simplified, a concept of interest reflects the underlying model of a given service or application in a machine readable format. Therefore, a concept of interest provides the "boundaries" a system can evolve in without violating its general purpose. Consequently, the principal task for a learning system is to incrementally learn about changing contexts without being explicitly informed about them [77].

The concept of interest for any real world service or scenario often depends on a hidden or very complex context [123] which makes it extremely difficult to design and implement let alone model the concept of interest that is intended to supervise it. Typical examples include almost any type of forecasting scenarios where individual models have to be built or adapted constantly to adapt to changing conditions, e.g., seasonal or geographical specifics.

Currently, there are no standards to model individual concepts of interests and no specific language has yet been developed that enables the description of real world scenarios in a domain

independent manner. Nevertheless, established XML based mark-up languages such as PMML (see http://www.dmg.org/pmml-v3-0.html) do provide features that allow the modelling of specific models stemming from the data mining arena. Another general problem in this area is the handling of the often large amount of properties that are required to reflect even relatively small concepts of interests and, furthermore, the implementation, adaptation and constant supervision of relationships among individual properties can be particular difficult.

3.5.2 Concept Drifts

As mentioned, advanced long-term supervision is based on the specific nature of the underlying concept of interest of individual services and their environments, which are likely to change constantly over time. Continuously changing conditions can open a gap between an actual concept of interest, implemented for a given service, 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.

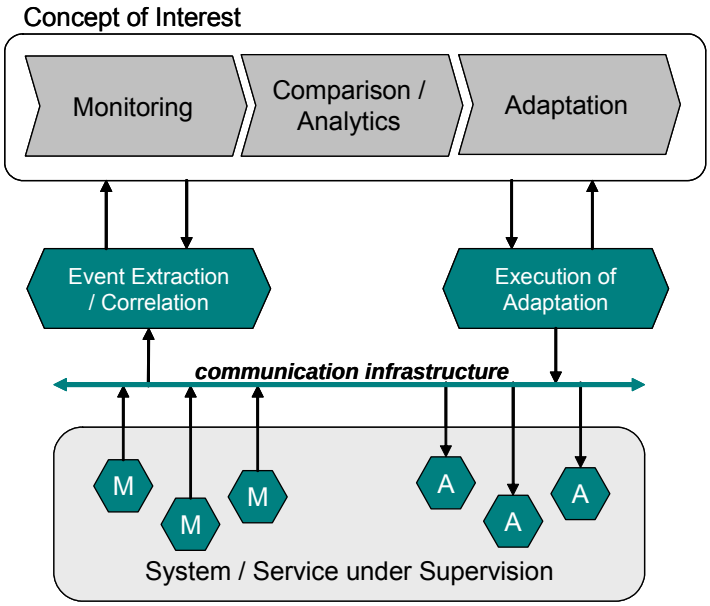


Figure 17: Concept Drift Supervision Architecture (schematic)

Figure 17 illustrates the basic supervision cycle to accommodate for the analysis of concept drifts. Similar to more general supervision architectures it incorporates the concept of interest in order to analyze changing behaviour within the boundaries set out through the concept of interest. In essence, individual context specific attributes are observed and compared to past concepts and experiences in order to identify differences between them. Based on those differences specific reactions may be triggered to adjust the system towards its original concept of interest.

Individual changes of a system or the underlying concept are often concealed by the complexity of the context they are used in, which makes it difficult to predict the impact of any changes made to the target concept. One of the important features of a concept drift based supervision system is therefore formed by its ability to track and react on individual changes as quickly as possible without being explicitly informed of them [126]. Another central problem for the observation of concept drifts 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 (see [126]).

For most scenarios only two types of concept drifts are relevant; continues concept drifts which may be further divided into slow and moderate drifts depending on the speed of change they follow [119]

and sudden concept drifts where abrupt and immediate visible changes occur. The literature [126] 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.

Relevant systems that deal with the problem of concept drift and hidden contexts include STAGGER [111], IB3 [9], and FLORA [126]. Based on the systems available three different approaches can be distinguished that deal with the problem of concept drift, namely; instance selection, instance weighing and ensemble learning techniques. Furthermore, traditional patterns discovery techniques, such as the extraction of associative, sequential and episodic patterns could be useful as such techniques have been used in the past to extract behavioural patterns and rules from stream based data.

Instance Selection. Based on the fact that a given context changes over time instance selection utilizes only a certain number of "latest" contexts to be compared against an original concept of interest. The set of latest contexts, also known as *window of interest,* is constantly updated with new contexts removing older context as they become obsolete. Those actions may trigger modifications to the supervised concept by constantly comparing the current window of interest with the original concept of interest. The original FLORA algorithm proposed in [126] implements this technique using a fixed window size. This concept has then be extended through FLORA2, FLORA3 and FLORA4 incorporating variable window size, past cases of distinct concept of interests to be used for comparison purposes and a more advanced noise handling mechanism. Other systems that implement a similar technique include FRANN [77] and TMF (Time-Windowing Forgetting) [110].

Instance Weighting. Simplified, instance weighing techniques allocate specific weights to individual properties of the current concept of interest, which are then analyzed by distinct supervised learning methods such as that support the handling of weights. The way weighing takes place for individual properties may vary and can include various measures such as competence, update frequency and the relevance of individual properties with respect to the current context [71].

Ensemble Learning. The purpose of ensemble learning is to build a learning model which integrates a number of base learning models, so that the model gives better generalization performance on application to a particular data-set than any of the individual base models [2]. Compared to the previous two approaches, ensemble learning provides a more effective technique to handle concept drift as it constructs and maintains a set of concept descriptions over different time intervals. Individual predictions thereof are later combined in order to select the most relevant description.

Two basic types of learning approaches exist, incremental learning and batch learning. While the latter analyses large number of instances at once incremental systems evolve over time processing new instances as they arrive [123]. Systems handling concept drift normally use an incremental approach because of the fact that new instances need to be processed as quickly as possible in order to keep a learning model up to date. In essence, an incremental based ensemble method processes an incoming stream of concepts one at a time classifying them into positive or negative instances of pre-defined (or previously occurred) concepts. Any discrepancies between the occurred concept and the predicted concept may trigger modifications to the systems or the underlying concept of interest [126].

Several distinct learning algorithms, such as rule based, decision trees and instance-based algorithms, have been utilized for base models in ensemble learning systems in order to handle concept drift. Almost all of them implement an incremental learning approach in which ensemble members are dynamically created, deleted or modified in relation to the consistency of individual base models with respect to the data used.

STAGGER [111] has been one of the first systems which implemented an ensemble learning approach. It utilizes an initial set of properties as concept features and creates more complex variations thereof in an iterative fashion using feature construction [123]. Based on the relevance to

the current data set the most appropriate features are selected and further processed. The approach described in [125] divides the original data sets into sub-sets of equal size building an ensemble system based on those sub-sets. Alternatively, in [119] individual classifiers of different "age" were used to build an ensemble system. In a nutshell, each base algorithm only utilizes the latest instance rather than all available instances. The IB3 system proposed in [9] implements an instance based technique that calculates the percentage of successful classification attempts comparing it with the frequency of its class. Based on this, the system decides which cases to keep discarding obsolete and noisy cases. Other systems, such as LWF (Local Weighted Forgetting) described in [110] only remove old instance whenever a new relatively similar instance appears. PECS (Prediction Error Context Switching) represents another system which is able to store past instances for later use and also incorporates the accuracy of an instance [110].

Patterns Discovery Methods. Patterns discovery has always been considered a challenging task, which has received great attention not only within the research community but also from various industry sectors. Pioneered by Agrawal et al. in [8] and [7] it has been an active research topic for more than two decades in which countless publications have been published and numerous algorithms where proposed. Multiple techniques have been introduced to extract such patterns and they were utilized for various domains, including telecommunication, life science, chemistry, drug testing, the World Wide Web, etc. Distinguishing between associative, sequential and episodic patterns temporal as well as non-temporal patterns and characteristics can be extracted from different types of data.

Associations represent relationships of a set oriented structure, where the order of items within those patterns is irrelevant, e.g. (A, B) = (B, A). The area was pioneered by Agrawal et al. and most of the current research in that area is based on work presented in [8] and [6]. Some of the most popular methods are based on the apriori-algorithm proposed in [8], which has been later optimized [6] resulting in AprioriTID and AprioriHybrid. Other popular algorithms include the DIC algorithm proposed by Brin et al. in [24]; the DHP algorithm proposed by Park et al. in [100]; sampling based approaches as proposed in [121]; Eclat proposed by Zaki et al. in [130]; FP-growth proposed by Han et al. in [58] and the so called COFI-Tree mining approach proposed by Hajj et al. in [41].

Sequential patterns are similar to associative patterns but incorporate an additional dimension, usually that of time [48], where the order of items is relevant and cannot be ignored, i.e. (A, B) ≠ (B, A). The discovery of sequences can be thought of as association discovery over a temporal database [131]. This area, also pioneered by Agrawal et al. [7][118] has focused on the problem of predicting future events based on past events, where an event could be virtually anything. Naturally, most sequence discovery algorithms are extensions of algorithms that where designed to extract associations. Some of the most popular once are GSP proposed by Agrawal et al. in [7]; PSP proposed in [91]; FreeSpan introduced in [57] and [58]; PrefixSpan proposed by Pei et al. in [103] and Spade proposed by Zaki in [131].

Episodes represent another distinct patterns type combining associative and sequential patterns. First introduced by Mannila et al. in [87] and [88] they provide a powerful technique to analyze time series related data, such as error and status log files or behavioural patterns, which contain related items or in this case events. Examples are found in the telecommunications sector, fraud detection applications or stock market analyses. A number of algorithms and concepts were proposed to extract episodic patterns [11], [20], [56], [80], [86] and [87]. However most of them are based on the same concepts as used to extract associations and sequences.

3.5.3 Visualizing Concept Drifts

The visual presentation of information as well as the interactive exploration of data is an important and challenging task that allows for better understanding and interpretation of data and information.

In [106], Pratt et al. propose an extension of parallel coordinate graphs called "brushed parallel histograms" in order to visualize concept drifts in data. Simplified, a parallel histogram graph is a parallel coordinate graphs with a histogram superimposed on each of its axis, describing the frequency distribution of points of the projected data set. Taking advantage of the fact that parallel

histograms display axes side by side, rather than in a 3-dimensional orthogonal space, they allow visualization of data distributions in high dimensional feature space. Furthermore, advanced interaction methods allow the highlighting of individual values and their relevant links which enables interactive data exploration as well as the presentation of detailed information. In fact, brushed parallel histograms allow the visualization of change rather than state information in high dimensional space, which has been a long standing challenge [106].

3.6 Summary and Conclusions

This report deals with various approaches and results related to Pervasive Supervision and provides a foundation for the further work of WP 2. In the first part, several general approaches have been analyzed and compared according to the implementation of the various subsystems of a reference model, the Viable Systems Model. It turned out that the low-level monitoring and actuation subsystem (VSM System 3) is present in all these approaches as the basic architectural paradigm employed by all compared approaches is that of a closed control loop. The same holds for the analysis subsystem (System 4) that evaluated monitored data with respect to the actual situation of the environment of a system, as long as reactive behaviour is concerned. Only one of the analyzed examples claims that pro-active activities are performed (i.e. future situations are anticipated). None of the considered approaches consider self-adaptation.

Purpose-orientation is represented in the VSM by a high-level system (System 5). We found that only one of the discussed approaches has a rudimentary higher-order policy subsystem.

Pervasiveness, i.e. structural intervening with the supervised system, has to use the (self-organized) structure of the supervised system, but since this structure is considered either as a black box (extrinsic approaches) or as static (intrinsic, i.e. Willow node architecture), this aspect is not investigated in a satisfactory way in any of the considered approaches. Finally, situation-awareness is due to the concentration to the actually supervised system (ignoring its embedding into an environment), is also insufficiently elaborated.

In the second part, we had a deeper look into specific aspects of interest. Monitoring has been mainly investigated from the perspective of advanced service architectures (exemplified by Web Services). We found large variety of the available results ranging from (standardized) monitoring and pre-evaluation approaches and tools.

Event correlation and problem detection is—at least on a generic level—not so well elaborated. Most existing approaches are task and application specific and do not offer a generic systematic algorithmic methodology. However a number of structural and architectural approaches are available that can be exploited in the CASCADAS project. It is however foreseen that this topic will be one of the main research areas for the further work in WP2.

There are a number of approaches concerning the recovery from errors and the determination of corrective measures (which closes the perception—evaluation—reaction control loop). Not only the exploitation of management interfaces (the usual way to interact with a system) is considered but also systematic structural approaches (patterns) and tools for planning are considered.

Finally, we looked into strategies for the self-adaptation of supervision pervasions (in contrast to the self-adaptation of the supervised service configurations). A number of pioneering approaches revolving around the notion of "concept drifts" are available. How to concretize these techniques to work with ACE based service configurations is subject of further work in WP 2.

4 Application Example

To provide a concrete example for a supervision system, this following section describes an application scenario to illustrate the algorithms described in the following sections. The scenario describes a pervasive and distributed application called "Behavioural Advertisement". In particular,

the scenario is chosen to illustrate the use of supervision models for system management, and to describe a basic supervision procedure. It is not intended or capable to capture all the concepts discussed so far:

- The supervision model that is used is supposed to be the result of the negotiation of the system under supervision and the supervision system on a supervision service, i.e. it defines the supervision contract (of class 2, c.f. Section 2.2.1). At the current state of the project, the precise mechanisms of negotiation and contract construction have not been comprehensively investigated. The example therefore does not explain how this contract is obtained.
- The exploitation of hierarchies in the supervised system to define levels of abstractions has been considered from a theoretical perspective (cf. the accompanied document [34]). In order to keep the example simple and to focus on the main ideas of the definitions of supervision cycles, the aspect of hierarchies is not presented.
- We restrict ourselves to supervision activities related to the VSM subsystem 3. Proper mechanisms for the prediction of future situations and long-term system adaptations (cf. Chapter 7) will be considered in a later project phase.

4.1 The Behavioural Advertisement Application

Behavioural targeting is supposed to allow marketers to better grasp customers' needs and interests, by tracking and monitoring consumer behaviours. Currently, those techniques are mainly applied to WEB based applications. We propose to extend behaviour tracking to any communication context supported by Telecommunication operator services where user interests and needs can be grasped (e.g. tracking the preferred shops of a user by means of GSM-based localization services).

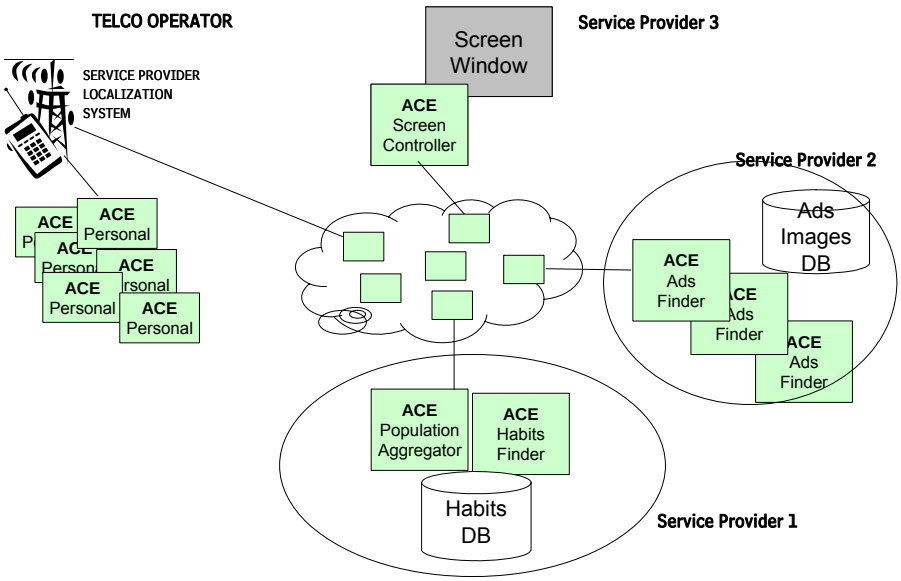


Figure 18. Behavioral Advertisement scenario.

Figure 18 shows an example of a Behavioural Advertisement application expressed as an ensemble of ACEs. Four different actors participate to the scenario: One telecommunication operator and three service providers. The service provider 3 is needed to start the scenario, sending a SMS to the user's device. As soon as a Personal ACE (running on a mobile device) receive the activation SMS, it starts sending personal data. A Habits Data Base is used to determine the personal preferences of the user that owns the Personal ACE, and an Ads Images Data Base is responsible to select

images to be displayed on a window screen the proximity of the initiating device. A GSM based location service is available to identify the devices which are in the range of the screen.

4.2 Supervision

In order to highlight some specific supervision challenges to be addressed, let us consider the following simplified set-up. Consider a mobile device in the range of a localization service which sends data to a content server in order to select the contents to display in the digital ads screen. The ACEs that form the example application (i.e. the Personal ACEs of the mobile devices, the population aggregation, habits finder, and ads finder ACEs explained in Figure 18) form a distributed virtual blackboard (DVB), i.e. an ad-hoc networked structure that propagates personal data and contents toward the display service. This propagation function basically is the service that is in the focus of the supervision activities described below.

Such an ad-hoc network is always in the danger to get congested. The main goal of the supervision task is therefore to keep a congestion level parameter (**C**) between specific thresholds and to ensure some reasonable values for the number of mobile in the range (**N**) and the advertising rate (**R**), which is the average to the advertisement rates (**r**) of the mobile devices.

The figure below provides a detailed view of the scenario:

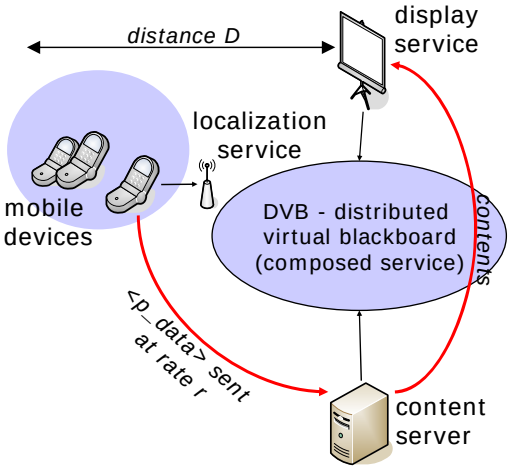


Figure 19. Simplified Scenario

The parameter **D** in the picture above is the maximum distance of a mobile to consider in the range of interest and so that should be involved in the advertisement selection. Thus the targets of a supervision system may be expressed as the task to keep these values in a given range; in the scenario we therefore have two main *objectives* for the supervision system:

- $C_{min} < C < C_{max}$.
- Keep N and R in reasonable boundaries.

Moreover, we allow the supervision system to make use of some additional knowledge related to the domain the supervision system has to operate, which might be obtained from the underlying Knowledge Network or can be part of the supervision contract. In the scenario we use the following knowledge:

- N * R is proportional to C.
- N = D * n (n is a suitable constant).
- decreasing D (and thus N) has a stronger impact to C than decreasing R.

4.3 The Supervision Model

As explained above, contract based supervision makes use of and operational model of the system or service under supervision that explains the ways in which this system can behave, what can be observed by the supervision system, and what can be influenced. To illustrate a possible formalism for those models, consider the following "extended finite state machine" that provides an operational model for a single mobile device (where we concentrate on the aspect of sending personal data, and operations to set the local advertisement rate **r** of the mobile.

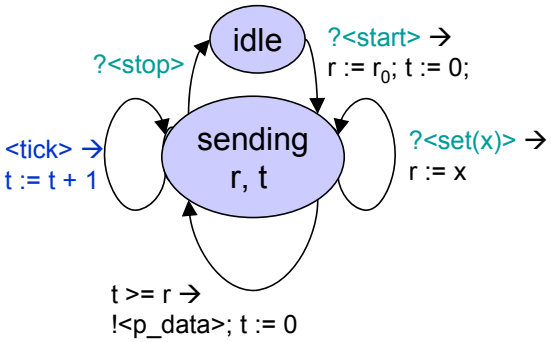


Figure 20. Mobile device model

The "timer" variable t is used to model the sending of the personal data <p_data> on a regular basis. The model exhibits the following actions:

- ?<start> and ?<stop> to initiate and to suppress the sending of personal data.
- ?<set(x)> to set the local advertisement rate r to x
- !<p_data> to send personal data
- <tick> is an internal action to model the event of a time tick.

We now have three different types of actions:

- ?<…> are actions which are controllable by some external entity (i.e. the supervision system)
- !<…> are controlled by the mobile device, but are observable by an external entity
- <…> are internal actions which are neither observable nor controllable

Of course, this mobile device model cannot be used for supervision purposes directly:

1. Firstly, in a real application example, an operational model would contain large a number of additional functions and thus would be much too large and to detailed. Moreover, we do not have a singe device, but probably a large number of those devices, and additional models for the other components of the advertisement service. Thus working with the composed model (which might be of exponential or even super-exponential sizes in terms of the number of components) is not feasible.
2. The model does not express the relationships between the global parameters C, R, D, and N that define the supervision task. And it does not qualify a given state of the system as not suitable, thus it contains no information on when a supervision action is to be performed or not.

We therefore use a different model for supervision as shown in the EFSM below.

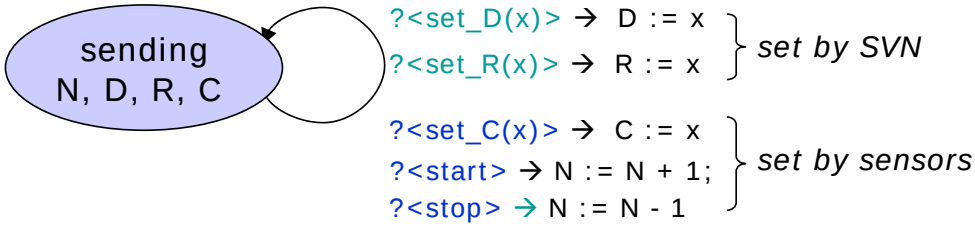


Figure 21. Supervision model

We furthermore impose the following constraints on the set of legal "sending" states:

- N = D * n
- C = c * N * R

(for suitable constants n and c).

We how state the following hypothesis:

> *The model shown in Figure 21 is an abstraction of an ensemble of N mobiles. An application of an appropriate concretization (or refinement) map should yield the model given in Figure 20.*

We are currently not in the position to justify this hypothesis – this will be a work item for the 2nd project phase of CASCADAS. With the notions of "zooms" that are elaborated in the accompanied document [34] we however are able to present a step towards a general theory of automated abstraction mechanisms.

Finally, we give the following notion of the suitability of a system state S = (N, D, R, C) by defining

> **V**(S) := if C > C_{max} or C < C_{min} then 0 else a * N + b * R

for suitable coefficients a and b. We furthermore say that the state S is suitable if **V**(S) is within certain thresholds (also conveyed with the supervision model).

4.4 Supervision Procedure

A supervision procedure is a sequence of steps which should ensure the proper implementation of the supervision tasks and that are triggered by defined states which need supervision. In our scenario for example we need to define a procedure which manages the values of D and R properly in order to keep their values into reasonable range. The procedure is the following:

1. Check if current state is suitable
2. if not, determine new values for D and/or R (to be refined)
3. set new distance D
4. call subsystems for mobiles to adjust local rates according to R
5. Validate constraints (check and improve competence)
6. If N deviates significantly from the expected value, adjust n
7. If C deviates significantly from the expected value, adjust c

Moreover such procedure should be based on suitable heuristics in order to adjust the values of the variables in a proper way. In our scenario some example may be:

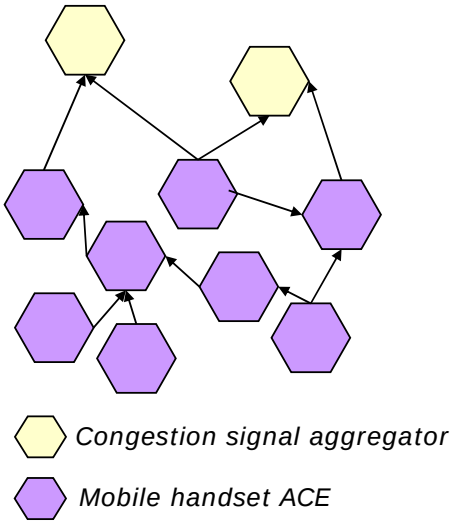
1. If the number of mobiles approaches the thresholds slowly then adjust R
2. If the number of mobiles approaches the thresholds fast then adjust D

3. If C exceeds the thresholds then adjust D, and if this does now work, adjust R (panic mode)

A problem to address in order to implement heuristics is the representation of this heuristics. A way to represent this heuristics may be using history information or include information on the success probability of a certain action into the model.

4.5 Pervasive Supervision

Each application of supervision activities to the target system requires specific extensions which of that target system or the underlying execution platform (e.g. actuators and sensors).



These extensions are better achieved if already available functions of the target system can be exploited. For example the fact that an ACE should be able to interface specific features of the mobile handset where it runs, in the ACE architecture defined so far, is addressed by its "specific features" module. Moreover, the continuous detection of the congestion level can be done by the perception of the periodically signals sent by the ACEs that form the DVB, e.g. by monitoring of the GN-GA protocol messages defined in Work Package 1. Each ACE sends GA messages based on its capability until an ACE with a service profile matching the GA message (the "GN") is reached. In our scenario we can implement the congestion signal as a GA received by an aggregator ACE which estimates the congestion level by the rate in which GA messages are received. The left hand side figure depicts a possible set-up that makes use of those correlation ACEs.

The Figure 22 below summarizes the relationship of the supervision model (comprising states of the form S = (R, C, D, N)). The parameters N and D are directly received from the incorporated location service. The congestion level C and the average advertisement rate R are computed in a distributed way as illustrated above. For actuation, operations to set a new distance D are directly available to the supervision system, while the breakdown of the average advertisement rate R to the local advertisement rates r can be achieved in a distributed way similar employing e.g. the GN-GA protocol.

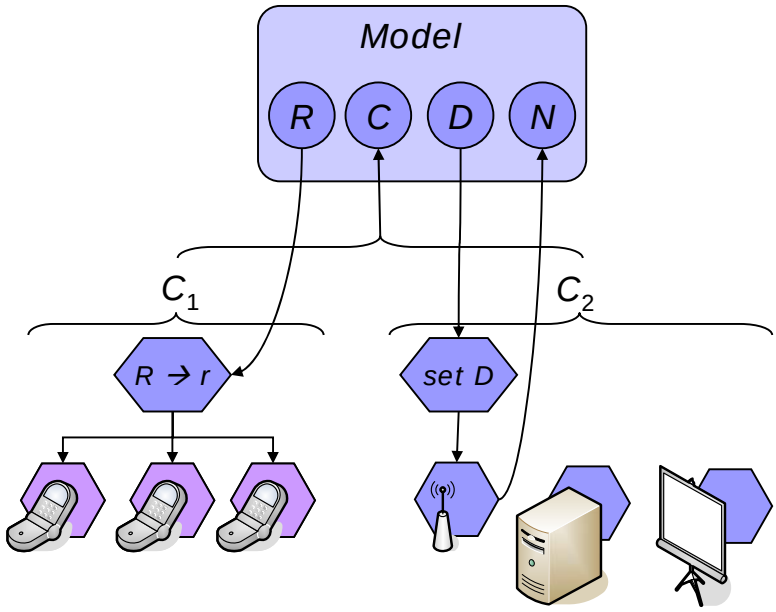


Figure 22. Relationship between supervision model and supervised system.

5 Requirements

In this section, we are going to provide a list of requirements that is to identify open issues and thus defines working tasks for Work Package 2. Before we present these requirements in a more detailed form, let us start with a more general discussion.

In the initial discussion in Section 2.1 we pointed out that current approaches to define of autonomic systems in a way similar to the MAPE paradigm are not sufficient for self-management, as a number of aspects such as autonomy of the supervised system and self-organization are not adequately represented. The analysis of the state (Chapter 3) of the art showed that a large number of approaches that deal with specific aspects of supervision are currently available, but a coherent approach that combines all aspects has not yet developed. The preliminary ideas on contract based supervision (Section 2.2) and the application example (Chapter 4) indicates that a model based approach is an encouraging candidate to overcome current limitations.

The following table details a number of high-level requirements of the comprehensive supervision approach that is in the focus of WP 2 and indicates how they are reflected in the currently available results.

No	Description	Status
1	**Autonomic Supervision** The supervision system should not rely on extensive operator configuration but has to infer the properties to be ensured and the appropriate tasks to be performed as much as possible from run-time information, i.e. by the communication with the ACE configuration to be supervised	Concepts for contract based supervision
2	**Adaptive Supervision** The supervision system has to be able to adapt itself dynamically to changes both in the system under supervision	To be investigated. This requirement is related to the self-organization

No	Description	Status
	and its environment	principles investigated in WP3 that define the dynamics of the system under supervision.
3	**Self-organized Supervision** The supervision system has to be able to self-organize available components into control loops and accompanied additional functional blocks as defined by the VSM.	To be investigated. This requirement is obviously related to the self-organization principles investigated in WP3.
4	**Pervasiveness** The supervision system should be organized along the organization structure of the system under supervision.	Addressed by employing the VSM as conceptual model, and concepts for hierarchical models
5	**Adequate Level of Abstraction** The supervision system should perform its task on an appropriate level of abstraction avoiding global (non-scalable) views as much as possible	Addressed by the developed concepts for hierarchical models
6	**Pro-activeness** The supervision system should be able to predict future problem situations and to enforce appropriate reactions	Addressed by the investigation of drifts in concepts of interests
7	**Self-application** The concept of supervision has to be applicable to supervision systems itself (i.e. a supervision system can be considered as a system under supervision by itself)	Addressed in principle, since the supervision system has been identified as an autonomic system (that will be realized by an ensemble of ACEs) and thus is a possible target for supervision. Details however need to be investigated
8	**Effectiveness** If the system under supervision enters an unsuitable state or is in danger to do so, the supervision system has to be able to enforce a sequence of actions purposed to lead the system under supervision back into a suitable state	Effectiveness is related to the contract that is committed between supervision system and system under supervision.
9	**Timeliness** If the system under supervision enters an unsuitable state or is in danger to do so, the supervision system has to be able to enforce a sequence of actions purposed to lead the system under supervision back into a suitable state within suitable time	Not addressed yet. Improvement of timeliness requires self-optimization features of the supervision

No	Description	Status
	limits (e.g. before the system state degrades further)	system which are related to the ability to select proper (and fast) contingency plans. This aspect will be addressed in the next project phase.
10	**Appropriateness** If the system under supervision enters an unsuitable state or is in danger to do so, the supervision system has to be able to determine an appropriate sequence of actions to lead the system under supervision back into a suitable state	Not addressed yet. Improvement of appropriateness requires self-evaluation features of the supervision system which are related to the ability to assess the suitability of contingency plans. This aspect will be addressed in the next project phase.
11	**Lightwightness** The supervision system should not add significant performance burdens to the system under supervision when it is working within suitable ranges.	Addressed only by the fact that supervision pervasions are defined as ACE based configurations. Experiments with concrete systems are necessary.
12	**Self-assessment** The supervision system has to be able to assess its own level of processing, and has do be able to optimize itself	Addressed by the definition of a number of metrics for "system competence" (see [34])

6 Supervision Algorithms

This section is concerned with:

1. A generic definition of a system model providing for abstraction and composition
2. Definition of supervision algorithms based on this notion

Due to the mathematical nature of these issues and the overwhelming problems to typeset mathematical text with Microsoft Word, we decided to provide these deliverable parts in a separated document [34] (using the LaTeX text processor). Here, we are going to give a summary of the main results presented in this document.

1. The term "model" made its appearance in several contexts through this report. We talked about the use of supervision models as contracts. The application example provided some examples of supervision models. Moreover, the ACE model currently developed in WP 1 makes use of an explicit self-model that defines the possible behaviours and services of an ACE. We thus have to answer the question: "**What is a model**?"

2. The formalism that is used for models is currently under discussion in WP 1. The deliverable document D1.1 discusses a number of candidates. From a more generic perspective, all these formalism (and many others) are basically defined by a (not necessarily finite) set of system states and a set of transitions between states. We adopt an approach that uses this observation as definition of a generic notion of the term "system model": A system is simply defined by a set of states and a set of transitions.

3. **State/transition** systems do not expose (as they are) any build-in notion of composition of concurrent (e.g. distributed) system components, and in particular there is now explicit expression of concurrency or distributiveness. To overcome this problem, we use a special type of state/transition systems, namely those that are defined over so-called distributed alphabets. A **distributed alphabet** provides an alphabet of system actions together with an **independence** relation over actions. Independent actions are supposed to be executable in parallel, or in a distributed way. The complementary notion is that of **dependence**. Intuitively, dependent actions are executed on the same computer, use exclusive resources, or are causally dependent (as a send-receive pair for a given message). We furthermore introduce a special **hidden action** that is use to express the fact that a system or its environment performs an action that is not directly observable but only perceivable because it causes a change in the current state of the system or environment. State/transition systems with actions taken from a distributed alphabet that structurally take the independence of actions as well as certain conditions on hidden actions into account are called **interpretations** of distributed alphabets.

4. The **language** of general state/transition systems, is usually defined in terms of words over system actions as "letters". For interpretations of distributed alphabets, a more attractive notion of language is available, namely that of **partially ordered multisets** (pomsets). A pomset is basically a set of system events that expresses the fact that a certain system action has been executed together with a notion of causality between system events. Informally, occurrences of dependent actions have to be in causal relationship. Events that are causally unrelated stand for the occurrence of independent system actions. Causal relations are mathematically expressed as anti-symmetric and transitive relations – as partial orders.

   We investigate a certain class of pomsets that are executable in a state/transition system, so-called **weak** pomsets. Intuitively, a weak pomset comprises only those causal relationships between system events that are really necessary to correctly express the dependence structure of the underlying distributed alphabet. It can be shown that the class of weak pomsets has the pleasant algebraic property of being a **free monoid**.

5. What is still missing is a system of values that expresses the "desirability" or "suitability" of a given system state or system behaviour. In the current state, we restrict ourselves to assessments of systems states. Entering a non-suitable state means that the supervision system has to get active. We again adopt a very generic perspective in defining those **value systems** as general partial orders (where intuitively, "greater" values are "better"). We are well aware that practically it might be a problem to set-up a relation of desirability between arbitrary system states (for instance, if a number of incomparable criteria are available). It is however necessary to unify the values of different states, even if they are not directly comparable with respect to the desirability order. Thus we stipulate the assumption that for each set of values, a greatest lower bound is available. Informally, a greatest lower bound is the best value that is still worse than any of the values in this set. It is thus the best pessimistic approximation of these values.

6. **Valued interpretations** of distributed alphabets are now defined as interpretations equipped with a value structure, together with a state assessment function that assigns a value to each of the states of the interpretation, and an admissible predicate that defines a threshold of the desirability of a system state.

7. We now have a formal definition of the terms "system" and "system behaviour", and "desirability" of system states, but we still have only a very vague idea of what it means that a formal system is a model of a real system, or is an abstraction of another model. At this point, we use the implicit assumption that for each real system there is some system model that expresses all the aspects of this system in each detail, i.e. is the most concrete model of this system. Thus, we only have to cope with the notion of abstractions between models.
8. As already said, a system comprises of states and actions that define transitions. **Abstraction** thus needs to be defined in terms of these elements. For states, it has to be abstract away from structural details of those states. Abstraction simplifies states and identifies different concrete states. System actions are also identified by the abstraction process, but it is also possible that a certain concrete action has no expression on a more abstract level. Finally, the value structures of more abstract systems are expected to be more coarsely grained than their more concrete counterparts.
9. Another concept is that of the **embedding** of a system into another. Informally, a system component is embedded into its environment. An embedded system appears as a part, or subsystem of the embedding system. As in the case of abstractions, the concept of embedding needs to be explained for states, actions, and values: The state information of the embedded system has to contribute to the state information of the embedding system, and for values hold a similar relationship. But of course all the values of the embedded system are still present in the embedding system.

Abstraction and embedding are explained in terms of functions between actions, states, and values of valued interpretations of distributed alphabets. The mathematical branch that deals with this type of definitions in the most abstract way is category theory. The document [34] (although self-contained) uses extensively the language of commutating diagrams to express the relationship between abstraction and embedding maps (or "arrows").

10. What happens with a subsystem when it gets embedded into some larger system context? This context provides only a subset of all the stimuli to which the subsystem is capable to deal with, thus the embedded subsystem shows now only a subset of its possible behaviours. The resulting restricted system is called the **image** of a subsystem under an embedding. It turns out that the process of embedding can be understood as a concretization of the embedded system in the sense that the "un-embedded" subsystem is more abstract than its image.
11. We finally introduce an important concept, namely that of **zooms**. Consider two systems, the first one is an abstraction of the second one. Suppose furthermore that the abstract system is embedded into some environment. The question we now are going to answer is: Can we use the embedding process that makes the abstract system a part of the environment also to embed the more concrete system into this environment? Is there a notion of local refinement, of decreasing the level of abstraction only for a certain area of an overall system model (the environment in this case). It turns out that the answer is positive. This result provides us with a way to set-up hierarchical models with respect to the level of abstraction, where the shift from a more abstract to a more concrete perspective is done by the application of appropriate zooms.
12. Zooms have a "dual" counterpart, namely that of **anti-zooms** that provides us with a notion of local coarsening. This concept is not used yet, but we anticipate its usefulness when the automated construction of supervision contracts is considered. This is future work.

The discussion of zooms and anti-zooms concludes the first part of the companion document. The second part is concerned with a first definition of supervision algorithms basing on the notion of models as defined in the first part. To relate this work to our reference model, we still remain in the functional domain of the VSM subsystem S3.

1. We start the discussion with a further refinement of the notion of distributed alphabets. We assume that the actions alphabet is separated into several sets of actions:

a. **Observable actions** are those actions of the system under supervision which can be perceived by the supervision system. Not all actions may be directly perceivably by the supervision system, or are visible at all. This may not only have technical reasons but also may be related to the unwillingness of a system to disclose certain actions to the supervision service.

b. **Controllable actions** are those which can be triggered, suppressed, or modified by the supervision system. We assume that if a problem occurs that triggers the supervision system, the system under supervision enters a "controlled" mode in which controllable actions are under the command of the supervision system. Of course, actions like system interrupts, timeouts, etc. are hardly controllable form externally. Moreover, certain management actions may not be desirable to be controlled by the supervision system e.g. because of security issues.

2. **State** in distributed systems is not always a meaningful concept. We nevertheless can use "local states" (i.e. those of the sequential components of a distributed system) to validate whether a system has been successfully executed or not.

3. We continue with the definition of a first supervision algorithm, i.e. a programmatic description of a basic supervision cycle. The cycle extends the basic MAPE-like approach by adding a validation function that assesses whether a certain corrective activity has lead to the desired results.

4. From that, a number of metrics that assess the "competence" of the supervision system can be defined, namely:

   a. **Effectiveness** refers to the ability of a supervision system to enforce countermeasures at all.

   b. **Timeliness** is the ability to react in time.

   c. **Appropriateness** means that the system has the capabilities to determine a sequence of actions that leads to the desired results.

5. A supervision algorithm based on hierarchical models is sketched. The concepts of zooms (and images) are used to explain the relationship between different levels of abstraction.

6. An outlook on further work concludes the companion document.

We finally give a brief summary of the state of this branch of research on WP 2. The presented work was motivated by the need to justify a number of terms that are used informally in the discussion of contract and model based supervision. A number of concepts have been elaborated, but the work is by no means completed. Its application to hierarchical and pervasive supervision is still to be investigated in more details, and application examples are missing yet.

7 Utilising Concept Drift for Pervasive Supervision

The work presented so far deals with functions that are mainly concerned with the definition of supervision cycles that relate to the "here and now" of the system, i.e. with its current state and structure. In this chapter we leave the VSM subsystem S3, and provide a first step in the definition of the functionalities of the subsystem S4, which is concerned with the "there and then", the environment and the future of the system under consideration.

There is strong motivation for new perspectives on generic supervision methodologies in order to provide more resilience in the face of ever more complex systems. In particular, future autonomic systems that ideally operate with no or only a limited user input require such advanced supervision

to control and adapt different variables of existing systems but more importantly to supervise the dynamic aggregation of individual autonomous working components as found in e.g. upcoming service oriented architectures or SOA. This research seeks to explore the requirements for a supervision mechanism that is capable of observing and analyzing complex and dynamically constructed models that reflect a real world service or computational system. Furthermore, the method proposed will be able to operate at different levels of granularity with respect to the model supervised and as such supports the methodological framework for pervasive supervision. Subsequent sections explore the requirements for different observation methodologies for distributed and network like knowledge structures, in particular exploring how such knowledge can be gathered, represented and what type of mechanism can be used to detect so called drift behaviour within the observed data.

A secondary objective is formed by the problem of how such drift behaviour can be used to (a) adapt individual components of a supervised system and (b) achieve a stable state of more global oriented systems, which could then freely evolve within pre-defined boundaries that describe the functional correctness of the system under supervision. In particular, the use of a lower and upper bound as well as the so called ideal state of individual variables will be explored.

7.1 Overall Architecture

In general, state of the art supervision methodologies and systems mainly implement a closed control loop approach which implements the following three concepts.

- Monitoring: Gathering of information from the system that is under supervision. Additional tasks may include correlation and translation activities in order to pre-process incoming information to improve the quality of the monitored data and to reduce information overhead.
- Analytics: Dedicated methods testing for certain conditions, violations etc. that are of interest to the supervision process. Current analytical methods often implement a static rule- or policy-based methodology where individual rules or policies are "hard coded" for each system and as such are not dynamic and often difficult to adapt to changing conditions. While such methods 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.
- Reaction: the reactive part of a supervision system closes the loop to actually achieve supervision. That is guiding a system within the boundaries it is allowed to operate in. The challenge for this part is not to realise and control the so called actuators which realise individual corrective measures on a supervised system. On the contrary the correlation of a given problem that has been detected with the correct countermeasures at different levels of granularity can be seen as the biggest obstacle for pervasive supervision.

The same three concepts are also relevant for a more long-term oriented and evolutionary-based supervision principle as envisioned here. That is with one important extension. In order to allow a system to evolve over time but at the same time assure the correctness of the underlying logic, advanced forecasting and prediction methods are required which allow the system to:

- forecast the "direction" of a supervised system;
- predict individual attributes based on past behaviour or on other attributes;
- and finally, detect critical states before they actually occur.

For that to be realised, it is necessary to build up a history of all monitored attributes of the system that is under supervision. Dedicated forecasting and prediction methods could then be used to predict future states and events based on the past behaviour of the model that is under supervision. Specific reaction mechanism may then be linked to the monitored model in order to register

dedicated corrective measures to specific parts of the system under supervision. Figure 23 depicts schematically the general supervision architecture and highlights individual aspects for each part which will be discussed throughout this section. For convenience, individual monitor or actuator units as relevant for a complete supervision system are not shown in Figure 23.

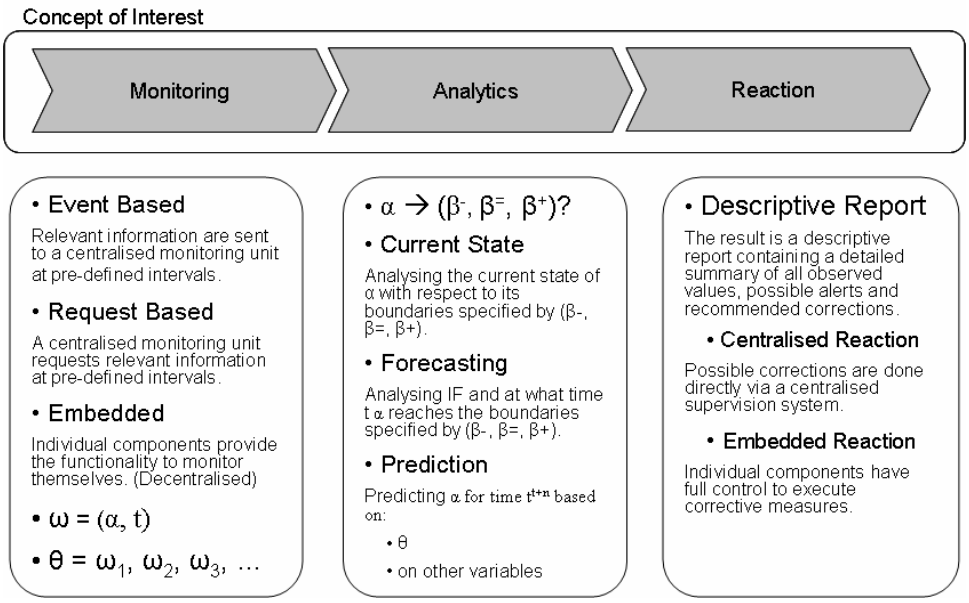


Figure 23: Schematic Supervision Architecture, Concept Drift

7.2 Concept of Interest

The need for such a more long term oriented supervision approach is based on the fact that the real world model of individual services or the underlying data thereof are of a 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 in order to achieve a stable state around some pre-defined boundaries. In order to adapt to such changes effectively a supervision mechanism needs to incorporate a computational model of the real world problem they were originally designed for. Simplified, a concept of interest reflects the underlying model of a given service or application in a machine readable format. Due to the fact that a concept of interest may depend on a hidden or very complex context it is often extremely difficult to design and implement them, let alone the modelling of the system that is intended to supervise it. Another general problem within this area is the handling of noisy data (or even irrelevant attributes) as a supervising system is initially doomed to react on any type of data, relevant or not. 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 false behaviour. Finally the necessity to allow for virtually any type of data as well as structure forms another challenge.

7.3 Modelling

As mentioned, the dynamic modelling and population of individual concepts of interests can be very difficult. This is mainly due to the fact that the type and structure of the system under supervision is normally not known beforehand. Furthermore, as the system under supervision may change the underlying monitoring model has to change too. Another problem is the handling of the potentially very large amount of properties that are required to reflect even relatively small concepts of interests

and, furthermore, the implementation, adaptation and constant supervision of relationships among individual properties can be particularly difficult.

Relevant requirements for a system suitable for the modelling of such concepts of interests can therefore be drawn upon the following criteria:

Intrinsic / Extrinsic Supervision: The system as well as individual variables to be supervised may exist in a distributed rather than in a localised environment which calls for different mechanisms that are able to operate in a centralised as well as in a decentralised fashion.

Cascading: The system should not be limited be any structural or conceptual boundaries in a way that it needs to be able to handle different type of variables, different relational concepts among them as well as different supervision goals at different levels of granularity.

Attribute Correlation: Individual variables may be correlated into higher, more meaningful concepts that are more suitable for later evaluation.

Context History: A history of the variables under supervision has to be created in order to analyse drift behaviour of different kinds. Algedonic signals such as heartbeats signals may be used to build up such a history given that a continuous stream of data can be provided.

Overall Architecture: Although outside the scope of this section it has to be stressed that advanced configuration & feedback mechanisms are key to any supervision architecture.

Within CASCADAS it is envisioned that the concept of knowledge networks as designed by WP5 is reused for the modelling of such concepts of interests. Figure 24 depicts such a modelling structure where individual variables are accessed via the concept of knowledge atoms thus feeding information into the supervision system. Linking individual atoms together via the concept of knowledge containers provides a flexible mechanism to correlate individual information into higher concepts and as such provide general supervision support at different levels of granularity based on the hierarchy of the supervised model. Finally, adding a dedicated service to each atom provides the functionality to build up a history of relevant attributes to be used by dedicated forecasting and prediction mechanism

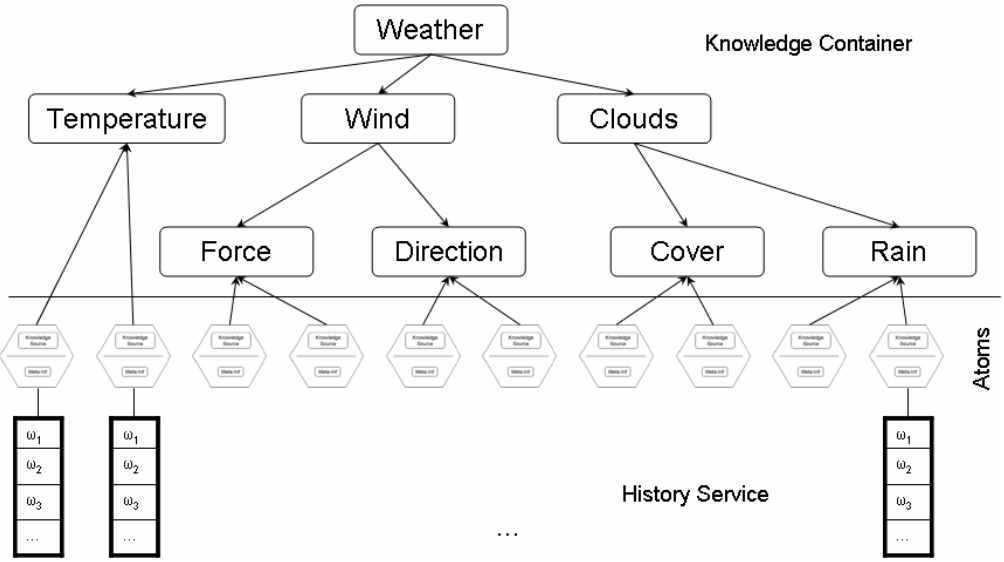


Figure 24: Monitoring Component for Concepts of Interests

Utilising the concept of knowledge networks for the modelling of individual concepts of interests is based on two design facts of the overall CASCADAS framework but in particular the ACE model. Firstly, observable events or attributes for supervision may be extracted from the self-model or from the specific-model of individual ACE's, see specification on ACE's (WP1). Secondly, due to the fact that knowledge atoms (as well as other KN components) will be realised via ACE's, both models

(the self or the specific) may expose the atom interface to publicise individual attributes that are of interest for a specific supervision task as identified via the supervision contract model outlined earlier. Therefore, once a contract is established between any number of ACE's (or their underlying model) a dedicated and private knowledge network may be constructed which only serves the supervision contract it is created for. Simultaneously, standardised methods which are part of the general knowledge network toolkit may be registered to this network to provide additional functionality e.g. the logging of attributes to build up a history of past events. This will be made possible via the service handler concept (see WP5, Building Knowledge Networks) which allows one to dynamically extend the business logic of each network component. Thus specialised supervision mechanism developed within the scope of WP2 may be used via the network framework provided by WP5 and vice versa.

7.4 Monitoring

Independent of the technique used to monitor individual source the goal can be summarised as to collect (a) a pair $\omega = (\alpha, t)$, where α is the observed value and t a timestamp referring to the time the value / event has been observed. As visualised in Figure 24, a registered history service is then able to build up a context history of the observed source such as $\theta = \omega_1, \omega_2, \omega_3, ...$

Despite the fact that internal knowledge network methods will be used to actually access relevant variables of individual concepts of interest's three distinct monitoring mechanisms have been identified to be relevant. These are:

- **Event Based Monitoring:** The observed source posts relevant information at pre-defined intervals or at certain events (e.g. the value of the observed attribute has changed) to a centralised monitoring unit.
- **Request Based Monitoring:** A centralised monitoring unit requests at pre-defined intervals or at certain events (e.g. an outside alert) requests relevant values from observed sources.
- **Embedded Monitoring:** Individual components provide the functionality to monitor themselves.

Table 3 evaluates the three monitoring techniques based on a host of criteria that have been identified to be relevant for the CASCADAS framework as a whole but in particular for the supervision framework. As shown, none of the techniques can be identified as to be best suitable. While event-based and request-based monitoring techniques promote lightweight components and good attribute correlation, and are reasonably easy to control as well as configure, embedded monitoring allows for a decentralised system, a built in cascading mechanism, fast reaction times and basically no delay in monitoring whatsoever.

	Monitoring Technique		
	Event Based	Request Based	Embedded
Decentralised Monitoring	No	No	yes
Lightweight Components	Yes	Yes	no
Controllability & Configurability	Medium	Good	difficult
Time to React	Slow	Slow	fast
Monitor Delay	Medium	High	none
Complexity	Medium	medium	high

Attribute Correlation	Possible		difficult
Cascading	No	No	Build-in

Table 3: Summary, Monitoring Techniques

7.5 Analytics for Concept Drift Detection

As mentioned earlier, a concept of interest is a phenomenon that describes a real world model and is defined by underlying contextual information or raw data. By nature, it is likely to change over time which is referred to as concept drift. Drift may occur in the underlying concept of interest if it:

- is not static e.g. dynamic models
- can not be described in its entirety e.g. incomplete models
- if its values are subject to change in any way e.g. changing context

In general and for real world systems, concept drift can not be avoided and will occur in one way or the other. When dealing with concept drift the following considerations should be taken into account.

- Batch learning is only useful for initialisation procedures or off-line analytics.
- On-line learners are required for "working" systems.
- Tracking concept drift on-line requires a learner to continually monitor the context defined by its "concept of interest" and ultimately adjust itself if necessary. Consequently, the principal task for such a learning system is to incrementally learn about changing contexts **without being explicitly informed about them**.
- Finally, a higher oriented system needs to exist and be capable of realising reactive measures in order to compensate for drift behaviour.

7.6 Types of Concept Drift

For most systems two types of concept drifts are relevant, firstly *continuous concept drift* which may be further divided into slow and moderate drifts (also referred to as concept evolution) depending on the speed of change and secondly, so called *sudden concept drifts* where abrupt and immediate visible changes occur [123]. Furthermore, the literature also differentiates between two different concepts of "concept drifts", namely *virtual* and *real* concept drift [126]. 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 and therefore the execution of some reactive part that adapts relevant components of the system under supervision.

7.7 Detecting Drift Behaviour

Based on the two types of drift behaviour identified above three distinct analytical methods have been identified. These are:

Current State (Sudden drift behaviour): Analysing the current state of α with respect to its own value / state and / or with respect to pre-defined boundaries[9] as specified by (β-, β=, β^+).

Forecasting (Visible, continues drift behaviour): Analysing IF and at what time t α reaches e.g. a critical state as specified by pre-defined boundaries.

[9] See Section 7.7.1 Boundaries

Prediction (Hidden, continues drift behaviour): Predicting α for time $t + n$ based on: θ or more interestingly based on other variables that are correlated to a given α.

7.7.1 Boundaries

Identifying the general 'path' of a system and as such drift behaviour is a powerful method to identify if a system slowly but continually moves into a specific direction or towards an unwanted state. The system is indifferent to whether the state represents only an annoyance or more seriously a critical situation of the system under supervision. In order to (a) identify if a system is in an illegal state; (b) predict the time it takes to reach an illegal state; or (c) to self–organise it around an ideal state which is either pre-defined or the mean of its boundaries. Thus, the boundaries of a given concept of interest specify the states a system can evolve in without violating its general purpose.

Lower and Upper Bound. As depicted in Figure 25, the lower and upper bound (β-, β+) define the borders a system can evolve (operate) in. Overlaid trends would then allow predicting long term directions so that out of bound violations could be identified at an early state. If a system violates the boundaries, as shown at the end of the time line, a possible alarm may be triggered or corrective measures may be induced. Note that individual lower and upper bounds do not need to be static. Depending on a changing context individual boundaries may change as well.

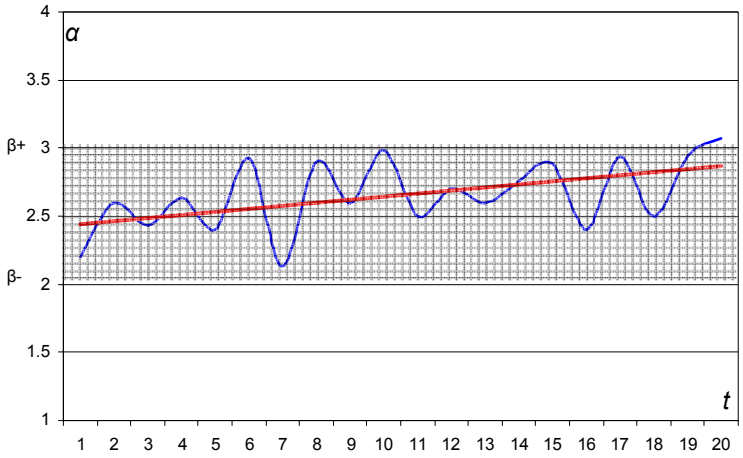


Figure 25: Lower and Upper Bound

Ideal State. More interesting in the context of autonomic computing is the organisation of a system around a so called ideal state, $\beta^{=}$, which is situation a system should attempt to achieve. For instance, an autonomic system regulating the temperature of a building has to react on a multitude of factors, e.g. outside temperature, number of people in the building, open windows, etc. Nevertheless, based on its configuration its ultimate goal could be as simple as "keeping the temperature at 27 degree Celsius". In this case the ideal state of is reflected directly by its goal. Never mind the fact that it would be pretty hot in that building.

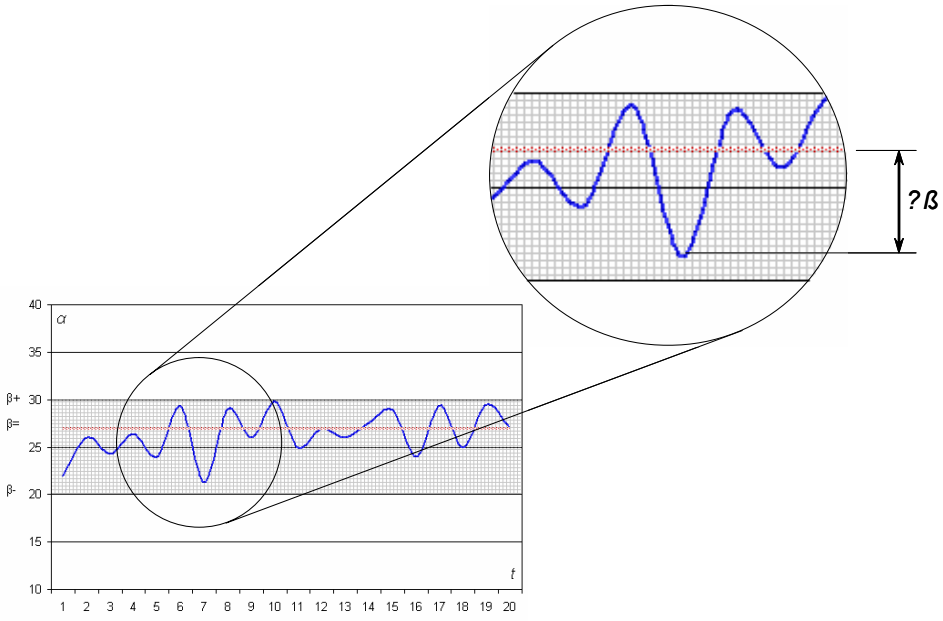


Figure 26: Ideal State

Figure 26 extends Figure 25 with an ideal state threshold where a supervised system attempts to achieve a stable state around $\beta^{=}$, which in this case is around 27 degrees Celsius. The main advantage for such supervision is that relevant countermeasures could be configured more precisely which is due to the fact that the difference between the current state and the ideal state is known. Considering any number of micro supervision systems that are capable of self-organising themselves around a stable state then the overall supervision system too should be able to operate around a stable state.

7.7.2 Summary

Table 4, shows a summary of the identified methods that can be used to detect drift behaviour. As seen each method is relevant for identifying either type of drift behaviour. For numerical values such behaviour can be detected using different mathematical and statistical methods. For symbolic values this may be more difficult, in particular if the complete set of symbolic values and their relationship to each other is not known beforehand. As for a predictive mechanism, a host of different techniques is available to analyse complex data structures. In particular neural networks, support vector machines, associative and sequential patterns discovery algorithms promise to be valuable when testing for drift behaviour in complex data structure. Nonetheless, the suitability of individual methods needs to be evaluated carefully and the selection thereof may depend on specific scenarios.

	Current State	Forecasting	Prediction
Sudden Concept Drift	Relevant to detect	Relevant to predict	Predictive features are relevant for complex structures
Continues Concept Drift	Relevant to detect	Relevant to predict	
History	n/a	Required	
Boundaries	Useful but not required		
Numeric Values	Mathematical Functions	Statistical Methods	Depending on the type of algorithm used

Symbolic Values	Difficult (Set of initial values and ranking thereof is required)		
Complex Structures	Difficult	Difficult	Difficult

Table 4: Summary, Concept Drift

7.8 Reaction

Based on the closed loop approach, the reaction part of a supervision process is concerned with the identification, configuration and execution of relevant measures to counteract incorrect behaviour or to invoke a specific recovery mechanism. With respect to the discussed monitoring and analytical techniques discussed so far two possible reaction mechanisms are deemed relevant.

- Direct Reaction: Corrective measures are invoked whenever an illegal state or violation is detected. This mechanism is particularly relevant for autonomous micro-supervision systems that are fully aware of what to supervise, how to supervise it and finally how to react if something goes wrong.
- Descriptive Reporting: If a system is not able to react on an illegal state or violation or if a system is forced to invoke countermeasures on a more global aspect of a system then individual components may choose to report their current 'health' to conceptually higher oriented supervision components. Obviously, such a reporting mechanism should be as complete as possible containing information about the sender, the fault, possible reasons (if the fault already has been analysed locally) and if known, relevant corrective measures.

Both mechanisms may be realised in a centralised way where possible corrective measures are identified and executed via a centralised system or, alternatively, in a decentralised system that is embedded, where individual components have full control to execute corrective measures. The latter obviously requires that each component is aware of the reactive measures it can invoke.

7.9 Summary

While the problem of concept drift is only one possible method to allow for long term supervision, the *concept of interest* and the real world problem they reflect are key to enable autonomic services to self evolve in context aware environments.

The main problem in this area is that a system under supervision can, at different levels of granularity, contain any type of information. It is therefore not possible to create a generic evaluation function that is capable of evaluating any type of information. This is based on the fact that an evaluation value is likely to be meaningless rather than due to different types of information. Thus, it may be better to allow a system to try and self organise itself in a way that micro versions of the whole supervision system exist at different levels. If such micro supervision systems maintain a stable state, the overall system should be stable too. On the other hand, if this state changes in any way, a system may recognise this as odd behaviour and may react on this. Realising a current state analysis combined with more advanced forecasting and prediction mechanisms will allow the detection of sudden as well as gradual drift behaviour. If embedded in a virtual realisation of a system under supervision such drift behaviour could be detected at early stages and effective countermeasures or a fail back mechanism could be invoked.

Within this section the requirements for such a system have been discussed and possible directions have been outlined. Subsequent steps will include the realisation of individual components in order to directly embed a supervision mechanism into individual components of a system under supervision. Finally the use of predictive methods will be explored in an attempt to predict critical states of important variables based on their own history or, more importantly, based on other variables altogether.

8 Reference Software Architecture

In this chapter, we introduce the software architecture that implements the supervision approach discussed in Section 2. First of all, we introduce the software components that are needed for the supervision and the relations among them. Then, we show how to distribute the entire set of components onto the levels of the Viable System Model. We validate identified components and relations through a simple example, which is a simplified version of the case study on advertisements proposed by WP 6. The last part of this section describes a possible implementation for the proposed architecture based on the event-based paradigm.

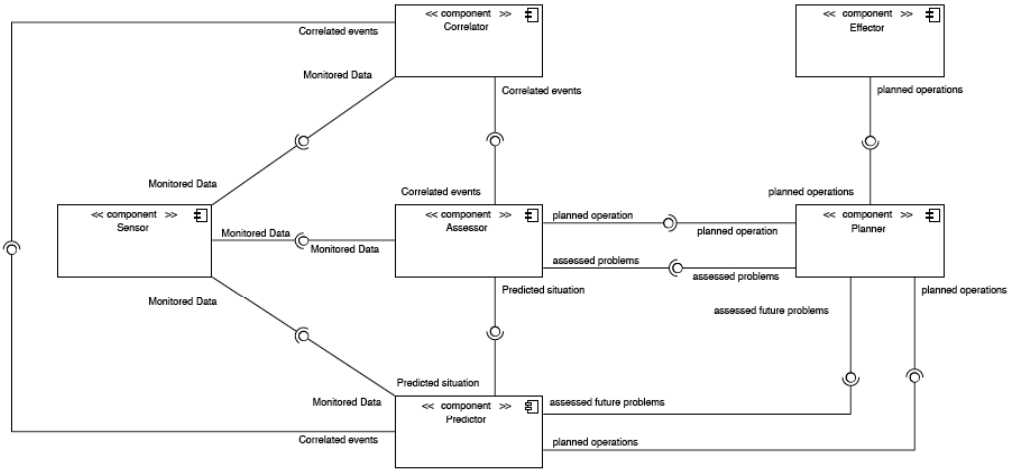


Figure 27. Components of the Supervision System

8.1 Components and Relations

The **Sensors** capture the data from ACEs, and the communications among them. They also send *monitored data* to the other components of the supervision system. The **Correlator** analyzes the history of monitored data to construct a coherent picture of the supervised system. This component has a ***repository*** of collected information and a ***reasoner*** to extract important information from collected data. The **Assessor** creates an abstract model of the system under supervision based on monitored data and correlation analysis. It is also able to detect if the status of single or composed elements is "correct". As soon as a problem is detected, the **Assessor** declares it. This means that it can detect both the status of a single element under supervision and problems with the environment of the system under supervision.

The **Planner** elaborates the set of actions that must be executed on the supervised system when the **Assessor** declares a problem. It uses the data received from both the **Correlator** and the **Assessor**. The **Effector** simply translates planned recovery actions into executable actions and messages that are then sent to the supervised components. The **Predictor** retrieves information from the **Sensors**, **Correlator**, and **Planner** to predict the likely behaviour of planned recovery actions. Like the **Assessor**, the **Planner** is also able to raise future problems to the **Planner**

Independently of the allocation of each component in the environment of the supervised system, the main relations among these components are:

- The tasks of the **Correlator**, **Assessor**, and **Predictor** are based on *monitored data* provided by the **Sensors**.
- Both the **Assessor** and the **Predictor** require correlated events from the **Correlator** and planned actions from the **Planner**.

- Starting from collected information, the **Assessor** informs the **Planner** that there is a problem and that a reaction strategy must be executed.
- The **Planner** sends the actions that must be executed on the autonomic elements to the **Effector**.
- The **Predictor** requires the results from the **Planner** to predict their effect.
- The **Predictor** uses collected information to inform the **Planner** about a possible future problem and that a reaction strategy must be executed.

8.2 Components and Viable System Model

The components described in the previous section fully comply with the requirements of the Viable System Model (VSM). Figure 28 describes how each component can be associated with the different levels of VSM.

Levels S1 and S2 define the autonomic system we want to oversee and are not presented in Figure 2. Level S3 represents the structures and controls that are put in place to establish the rules, resources, rights and responsibilities of the system under supervision and to provide an interface to the upper level. This level comprises the capability of acquiring data (**Sensor**), the capability of detecting problems (**Correlator**), the capability of planning necessary recovery actions to deal with detected problems (**Planner**), and the capability of executing them (**Effector**).

Level S4 is responsible for looking outwards to the environment to monitor how the federation of components needs to adapt to remain viable. It is crucial to understand that S4 has two main interfaces: the interface to S3, which provides information and actuation facilities concerning the inner structure of a viable system, and the interface to the environment. Furthermore, S4 is responsible for anticipating and pro-actively adapting to future situation. In the context of supervision, the ability to foresee and extrapolate future problems and requirements is of major importance since instantaneous reaction to problems on a global distributed basis is hampered by communication delays and by the complexity of data correlation and problem detection. The **Predictor** is the key S4 component. It retrieves data from S3 and it also informs S5 about the status that the system under supervision is likely to assume in the future. It also informs S3 about the future behaviour of the supervised system. This way, S3 is able to anticipate the problem by means of appropriate recovery strategies.

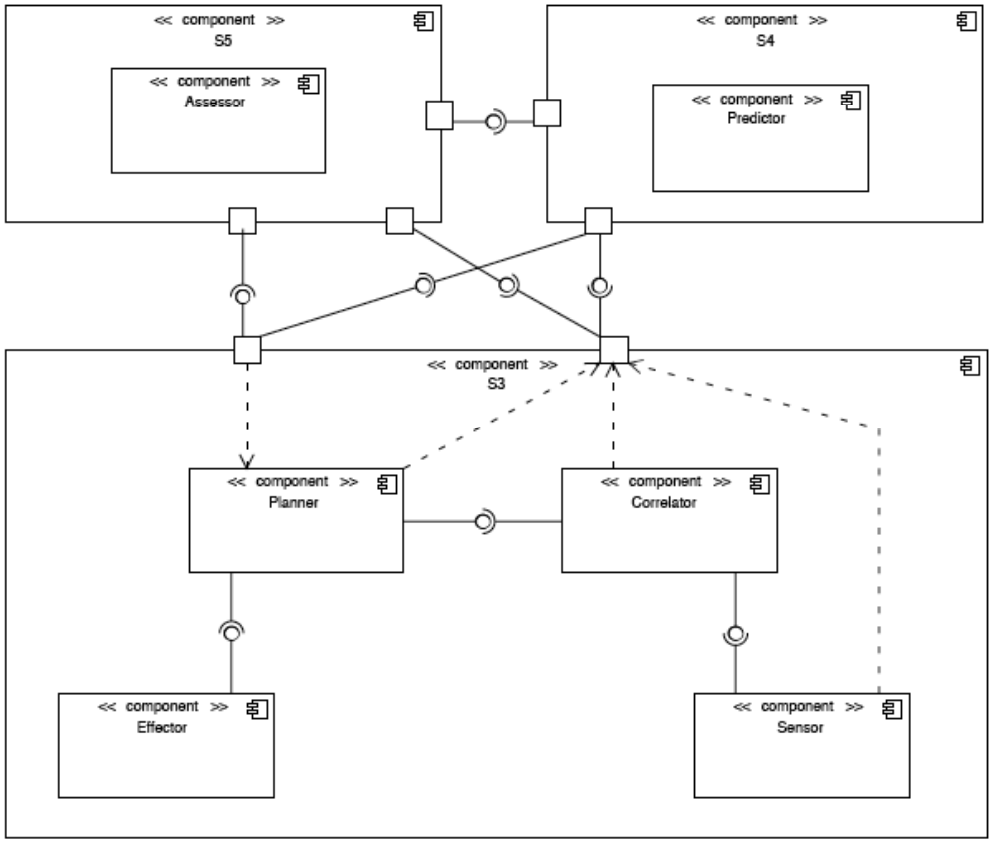


Figure 28. Supervision components and the VSM model

S5 is responsible for policy selection on the system as a whole to balance the demands from the different parts of the system and steer the organization as a whole. These policies are considered being non-technical in the sense that they are not directly represented by rules related to adaptation and optimization parameters (those rules are part of S3 and S4), but they define the purpose of a system and its intended relation to other systems in terms of general requirements. Hence, S5 comprises the **Assessor,** which analyzes the vision of the system under supervision provided by the other levels and assesses whether planned actions and their predicted results are within an acceptable range.

8.3 Example Application

The effectiveness of the software architecture must be validated through examples. In this case, we have chosen the case study about smart advertisements presented in Section 4.1. In this example, there is a supervision system that oversees the behaviour of the Advertisement System (ADV), which comprises a set of mobile devices and a screen; a Distributed Virtual Blackboard (DVB) is used as communication bus; it is an ad-hoc network that links all mobile devices; thus, the DVB is the composed service.

The internal model of the supervised system comprises four different parameters:

- `D` is the maximum distance of a mobile device that must send personal data. It is a parameter inside the DVB.
- `N` is the number of mobile devices in the range `D`.

- R is the rate with which personal data are sent. A mobile device sends its personal data every R milliseconds.
- C is the current congestion level. This parameter is derived from the other parameters by the formula C = c*N*R, where c is a suitable constant.

The goal of the supervision system is to regulate R and D in order to maintain C between two thresholds (C_{min} and C_{max}). The system detects the value of N from the DVB by means of a sensor. A suitable mathematical model evaluates whether the system is viable or not. When an abnormal situation is detected by the supervision system, it regulates the value of variables C and R.

Components. Figure 29 illustrates the data exchanged among the different components in the example application. This is a static view of the software architecture: here we identify the messages exchanged among the different components to carry out the scenario described above.

A **Sensor** is inserted in the DVB: it detects the value of N (number of mobile devices) and sends it to the supervision system. Given the previous value of N and the time of arrival of this information, the **Correlator** detects the *rate of change* for this variable. Afterwards it sends this information to the **Assessor** and **Predictor**. The **Assessor** evaluates whether there is a problem in the system under supervision by analyzing the values of N and R that come from the **Sensor** and **Correlator**. In particular, it calculates the value of C and detects whether it is over the threshold; moreover it detects whether the number of mobile devices in the range D grows too fast. **Planner** receives the detected problem from Assessor and decides how to react. This component evaluates how to adjust the value of variables D and R. Effector translates the planned decisions into an executable format. Therefore, Effector sends messages set(r) and set(d) to the DVB. **Predictor** uses relations N~n*D and C~c*N*R to predict the effect of planned reactions.

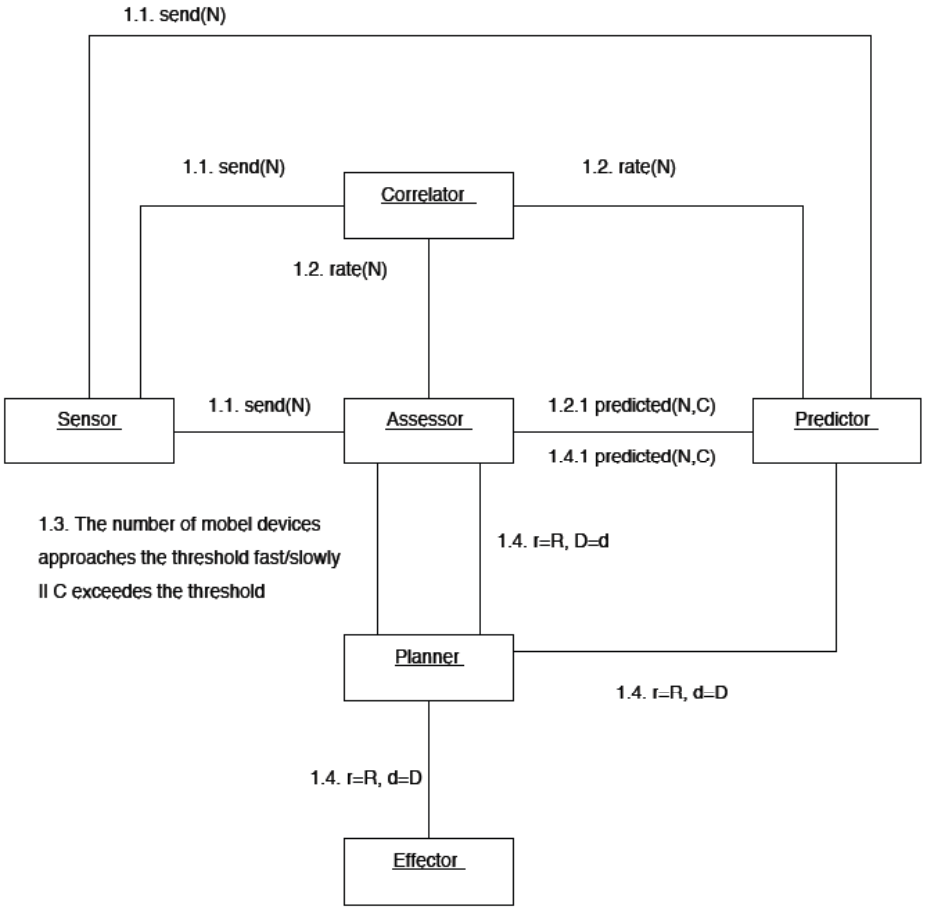


Figure 29. Communication diagram of supervision component for the example scenario

8.4 Possible Implementation

Before describing the proposal for implementing the software architecture, we analyze the context where introduced components are supposed to operate. In the general vision of supervision, there are three different kinds of supervision tasks. (Figure 30):

- *Embedded supervision* oversees the internal behaviour of each autonomic element;
- *External supervision* addresses the behaviour of each autonomic element externally;
- *Coordinated supervision* controls and coordinates the overall coordination of an autonomic system.

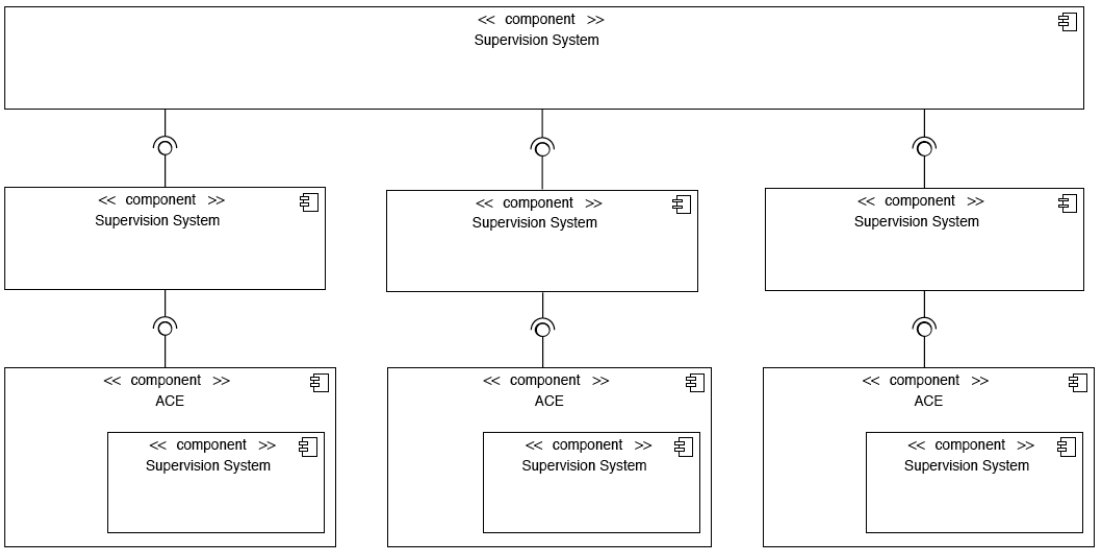


Figure 30. Levels of supervision system

These tasks do not really impact the software architecture. They mainly differ in the way sensors are deployed and oversee the behaviour of the different parts. More precisely:

Embedded supervision controls the common functionality of an ACE, such as GA/GN protocol. Sensors are embedded in the *Common Part* of the *Message Handler* and get information about the common functionality of an ACE. A Sensor works like an intermediary between the *Message Handler* and the *Reasoning Engine* to intercept the communications between the two components and send these data to the supervision system.

External supervision inserts sensors in the *Specific Part* of the *Message Handler* and thus provides data to supervision system about the behaviour of the specific parts of the ACE. This external supervision controls the lifecycle of ACEs.

Coordinated supervision, differently from the two previous options, imposes that sensors collect: (a) information about messages exchanged among the ACEs, (b) the status of the supervision system, and (c) the status of controlled ACEs. Hierarchical supervision requires that sensors be placed in both the interfaces of the different ACES, and controlled supervision (sub)systems.

8.5 A Solution Based on Knowledge Networks

To accommodate the different supervision tasks presented in the previous section, and let them be organized hierarchically, the supervision infrastructure must be flexible enough to manage

information detected by the sensors both in the different ACEs and in the supervision subsystems. Furthermore, the relations among identified components show that different kinds of components require the same information: for example the **Assessor**, **Correlator,** and **Planner** require monitored data. To cope with these requirements, the supervision system is built on top of the Knowledge Network developed in WP5.

The Knowledge Network provided by WP5 provides two different features exploited by WP2. First of all, a way to publish specific information to a virtual knowledge layer that is accessible by other ACE's. This feature allows the construction of private sub-networks that (a) serve specific (local) supervision tasks, (b) allows other ACE's to serve or retrieve information from this sub-network and as such enables task specific knowledge provisioning.

The Knowledge Network acts as a request-driven architecture, but provides also functions (such as a push functionality) that can be exploited to define event-driven concepts. An *event-driven architecture* (EDA) defines a communication paradigm for designing and implementing software systems in which components are loosely coupled and their interactions are governed by exchanging events (messages) through an intermediary event manager. Event consumers subscribe to the intermediary event manager for the types of events they are interested in, and event producers publish their events onto this manager. When the event manager receives a new event, it forwards it to the consumers registered for receiving it.

This kind of architecture allows certain degrees of freedom in the definition of the supervision system architecture. All the components introduced above become data/event consumers and data/event producers. For example, the **Sensor** produces data/events for the **Correlator**, **Assessor,** and **Predictor**. The **Planner** produces data/events for the **Predictor**, **Assessor,** and **Effector**. This model allows us to easily federate different components by registering them onto the Knowledge Network. Similarly if the sensors deployed in a given supervision subsystem are attached to the Knowledge Network, we can easily create hierarchical supervision infrastructures.

Figure 31 shows the interactions between WP2 components and the Knowledge Network in a reflective manner. The KN is seen as a set of features provided to the other components. **Sensors** can insert important sensed data about the System Under Supervision into the KN through the push functionality provided by the KN. This data can then be retrieved by the other components of supervision system A through the KN. In the same way, a **Sensor** of the Supervision System B senses the events dispatched in the Supervision System A; so that, all the important data dispatched in Supervision System A can be analyzed by the Supervision System B.

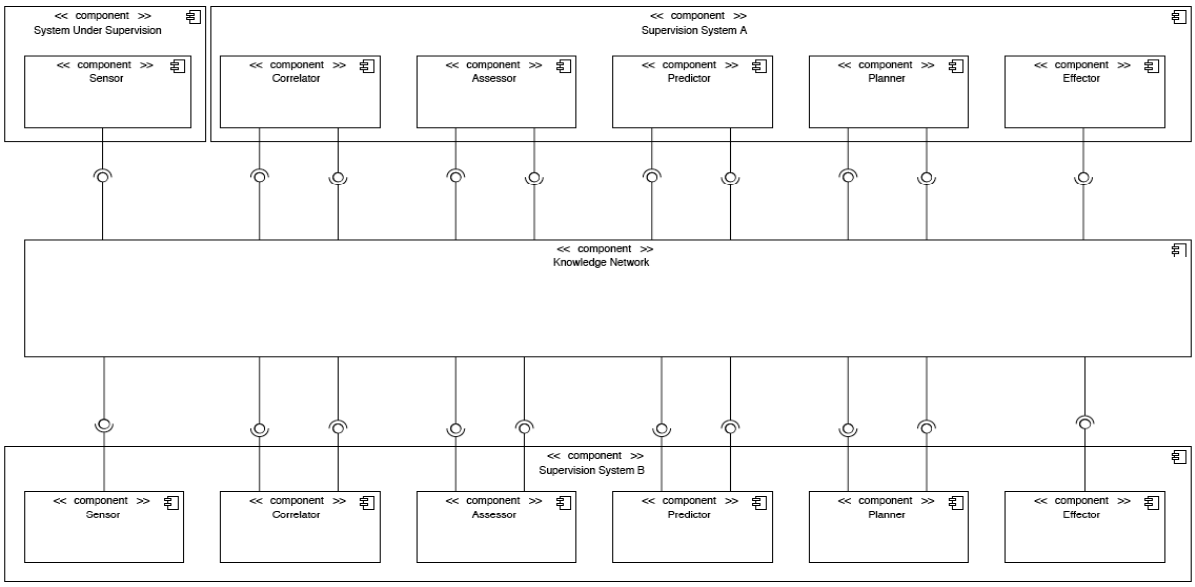


Figure 31. Data and event dispatching among Supervision System components

9 Summary

This report is necessarily about "work in progress", as in the current state of the project final results cannot be provided or are even desirable. As described in the initial Chapter 1, we started from an initial vision of a pervasive supervision architecture as described in the Viable System Model. From that, three main work tasks have been defined:

1. Supervision as a service. This idea leads to the use of supervision models as contracts between the system under supervision and the supervision system. We developed a formal framework for the mathematical work with models, abstractions, and embeddings.
2. Prediction of future situations to detect possible problem situations and to perform pro-active system adaptation. We presented a framework based on concepts of interest.
3. Software development. As a first step to an implementation of the theoretically elaborated concepts, a high-level software architecture has been developed.

Now since all work packages in CASCADAS are able to present initial results, the integration of the WP 2 activities with that of other WPs becomes possible. The following list provides a road map for the further work of WP 2.

1. What are the technological pre-requisites to build and to interact with supervision pervasions, i.e. which abilities are required from ACEs and ACE configurations to support (a) monitoring and interaction, but also (b) to model and to implement a supervision architecture using ACEs? This question demands a close cooperation with WP 1.
2. Self-organization (WP 3) is one of the envisioned properties of service configurations, and—following the pervasive supervision paradigm—supervision subsystems are thus (a pervading) part of such configurations. Thus they have to follow basically the same organization rules than the supervised subsystems. On the other hand, interaction and modification of service configurations by exploiting self-organization rules is a mean to interact with those configurations.

   ACE configurations will not be statically formed networks but base on dynamic and continuously changing interrelationships. Thus a major problem is how to identify and to build up an internal image of the actual configuration that is needed to correctly percept monitored data. An accompanied question is how to define long-term trends in such an environment and to support pro-active self-adaptation of the supervision pervasion.
3. From the perspective of security, supervision is of course a very critical approach. Thus cooperation with WP 4 is required to perform supervision tasks in the presence of security demands. On the other hand, security modules and components are—as being software packages—error prone and might non-functional, add unacceptable performance bottlenecks, etc. Thus security is by itself an application area for supervision techniques. Resolving this mutual relationship is another foreseen research topic for WP 2.
4. Finally, as already noticed, situation-awareness is not reflected by the most existing supervision approaches. In CASCADAS, the concept of a knowledge network (WP 5) exists as a structure that intends to provide and to distribute the necessary information. How this network can be exploited for supervision is yet another open issue to be investigated in more detail.
5. The implementation work proposed by WP 2 has to be done in the context of one (or several) of the application examples developed in WP 6. We will concentrate on the Behavioural Advertisement example.