Autonomic Computing in Peer-to-Peer Systems

Bachelor Thesis
KOM-S-0318

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Electrical Engineering
and Information Technology
Department of Computer Science
(Adjunct Professor)

Multimedia Communications Lab - KOM
Prof. Dr.-Ing. Ralf Steinmetz

Submitted by
Michael Behrisch
on 31.05.2009

Advisor
Prof. Dr.-Ing. Ralf Steinmetz

Tutor
Kalman Graffi, Michael Niemann

31.05.2009

1 Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31.05.2009 .....

Contents

List of Figures

List of Tables

2 Introduction

This thesis describes the ongoing efforts in the field of autonomic computing. Autonomic computing's goal is to develop software and hardware systems capable of managing themselves in accordance with high-level policies from their human administrators. Especially, the thesis focuses on the applicability of autonomic computing concepts in self-managing Peer-to-Peer systems. It provides an broad overview about the current efforts and open challenges in all major parts of the autonomic Peer-to-Peer computing field, such as monitoring, policy strategies and decision handling in distributed autonomous systems.

2.1 Motivation

Since the invention of the first computer, research facilities and the industry are pushing the limits of computer science to never foreseen stages. But, at the same time the rapid development in the last decades is leading to ever increasing complexity in both, hardware and software systems.

2.1.1 Software Complexity

Especially, software programs (end-consumer, as well as enterprise applications) and their environments are reaching tens of millions of lines of code. For example, operating systems, like Microsoft's Windows Vista are expected to have reached over 50 million lines of code, whereas Microsoft Windows XP was said to have around 40 million [11], and even open-source projects are reaching this magnitude (Eclipse Europa release consists of more than 17 million lines of code [23]).
However, not only the dimensions of software and hardware projects are becoming a confining factor. Another downside of the evolutionary maturing process is, that an statically increasing amount of highly skilled IT professionals are required to develop, install, configure, optimize and maintain these software systems. Even more, as systems are becoming more and more interconnected and heterogeneous, IT system designers are less able to anticipate interactions among components, leaving such issues to be dealt with at runtime. While today's skilled IT staff is capable of managing these systems with great effort, the complexity will soon become too massive and complex for the most skilled system administrators.

2.1.2 Distributed Computing

In addition to this, another aspect of the upcoming IT system complexity is the distributed computing principle. While the widespread approach to store and process data in a central place has been a successful paradigm in the Internet for many years, the subsequent advances in the computing technology and broadband access technologies built the basis for a further evolutionary

step of the Internet.
Basing on this technological evolution, the music-sharing application *Napster* [49] was introduced the year 1999, enhancing the Internet's rather static and centralized network model. As the first highly successful distributed network application, Napster enabled –the to that time huge amount of– 26.4 million users [4] to connect their PCs for more than just browsing and exchanging information.
Nevertheless, Napster was only *one representative* for the even more far-reaching idea to give computers the ability to contribute *actively* to a network by forming collaborative groups to build virtual supercomputers, ad-hoc communication systems and location independent file systems.
Especially, the immense success of this upcoming paradigm helped Peer-to-Peer (P2P) networks to emerge from status of a research topic to a valuable network architecture alternative. In Peer-to-Peer networks the clients, so called *peers*, are acting with equal rights and follow their own intentions, while simultaneously contributing to the overall system. This concept of distributed computing is merging the clients' and servers' roles and eliminates the need for a central server that manages administrative and routing tasks.

While on the one hand Peer-to-Peer systems have significant advantages, such as reduced infrastructural costs for hardware, cabling and maintenance, as well as an increased flexibility, the network model's disadvantages are having also a significant share in the complexity problem. Not only that the lack of a centralized computing instance makes the administration difficult, the developers also have to deal with uncertainty and unreliability (link-, node-failure) within the infrastructure. Hence, Peer-to-Peer applications have to implement highly sophisticated and decentralized data replication-, recovery-, and control flow mechanisms to achieve the same functionality as a centralized computing instance.

Resulting from the myriad number of potentially networked applications/environments and the ever increasing complexity of IT systems IBM's senior vice president of research, Paul Horn, introduced in 2001 their manifesto for dealing with

> *"... the main obstacle to further progress in the IT industry and the looming software complexity crisis" [38].*

Horn pointed out, that future computer environments will reach far beyond the human's capability to forecast the complex relationships that drive systems internally and externally. As a result of this acknowledgment, IBM stated that the only remaining chance for overcoming the burden of complexity will be self-managing *autonomic systems*.

2.2 The Concepts of Autonomic Computing

Autonomic computing systems are inspired by strategies used in biological, social and economic systems, which –like distributed computing systems– have to deal with similar challenges of scale, complexity, heterogeneity and uncertainty.
Specifically, the human's self-controlling autonomous nervous system was an inspiration for autonomic systems. The human's autonomous nervous system relieves the brain from regulating the lowest-level functions, such as heart rate and body temperature, which are yet vital. Alike,

autonomic systems must be capable of monitoring and evaluating their internal and external states, thus building a self-knowledge about themselves and their embedded environment. Self-knowledge in turn will be the starting point for reactive and anticipatory services, such as self-controlled optimization, configuration, and problem solving.
As another point, the collective intelligence of ant colonies is a strong inspiration for autonomous systems. Here, a single ant is helpless and rather useless, but as a collaborating collective they show tremendous achievements in building astonishing constructions. According to this analogy, *autonomic elements* are thought as service providers of every given granularity. Controlled and externally represented by an so called *autonomic manager*, their service provisioning can reach from giving access to a hardware resource, such as CPU, RAM or a printer, or approaching a software resource, such as a database system, an algorithm or a large legacy system.

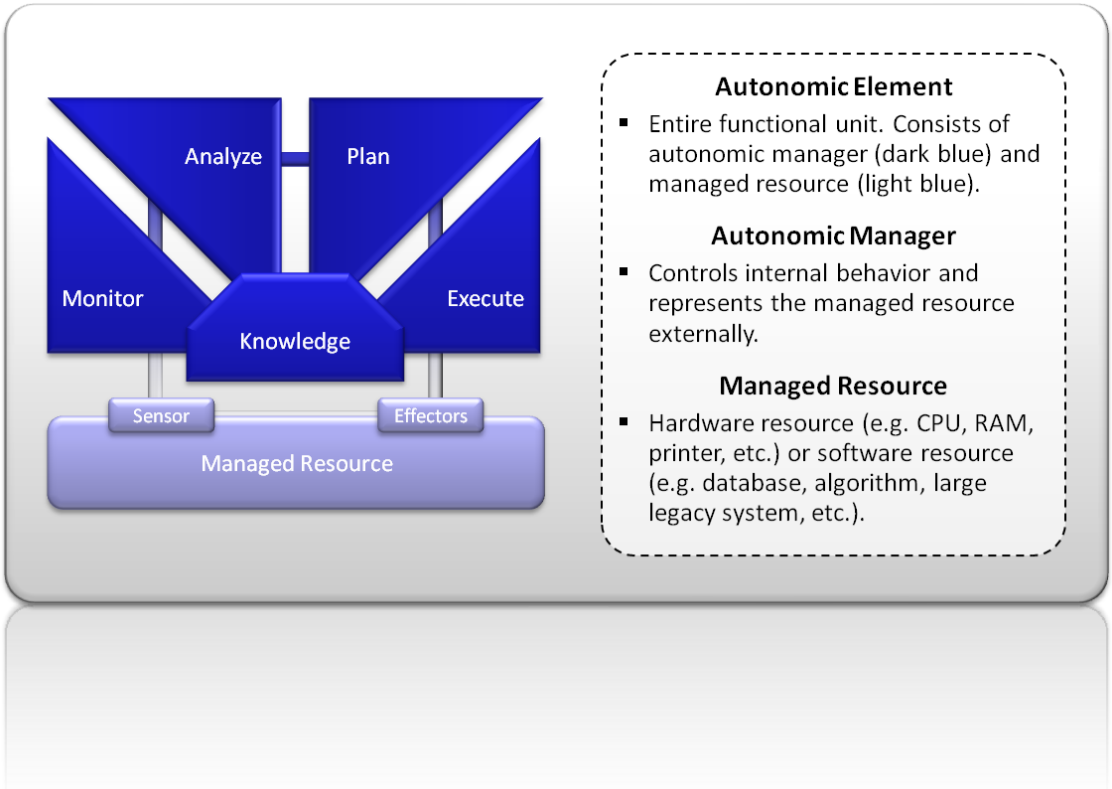


Figure 2.1: Architectural Building Blocks of Autonomic Computing

Composed of a multiplicity of individual subsystems, or autonomic elements, autonomic systems will accomplish tasks by forming dynamic collaborative groups, alike in an ant colony or in the P2P paradigm. Paul Horn envisions autonomic elements as

> *"... individual system constituents that contain resources and deliver services to human and other autonomic elements. Autonomic elements will manage their internal behavior and their relationships with other autonomic elements in accordance with policies that humans or other elements have established" [38].*

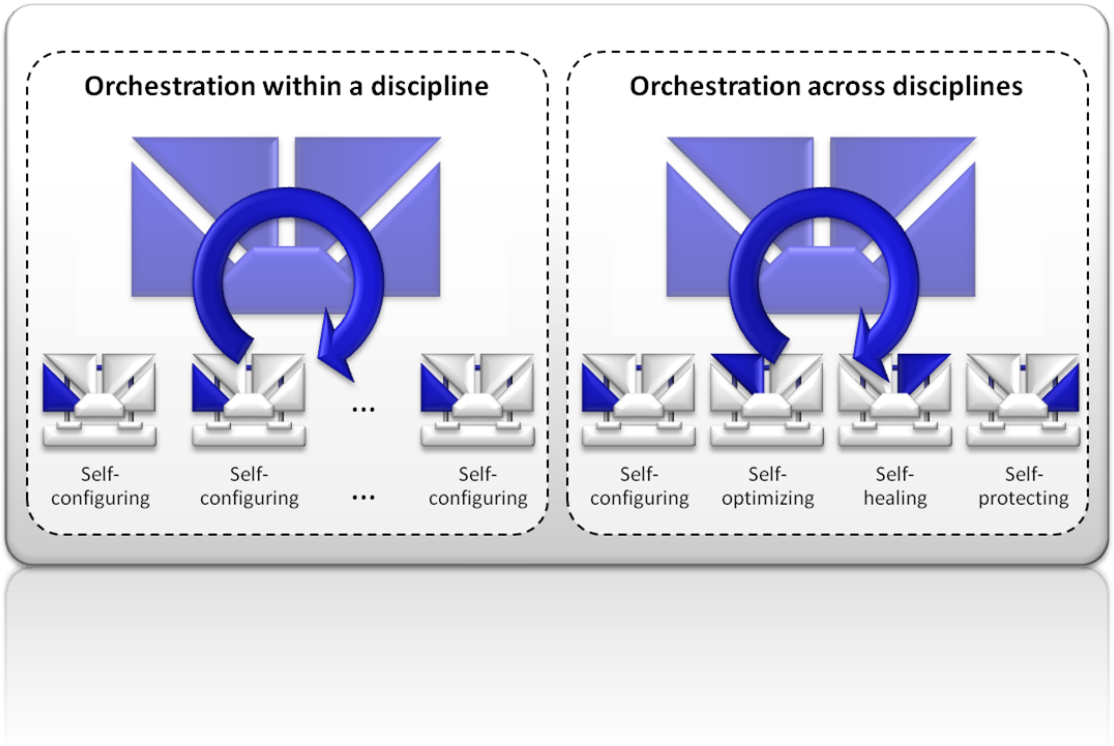


Figure 2.2: Overview Autonomic Systems

As Figure 2.2 depicts, autonomic elements are not only service providers for their own managed resource, but an active contribution to the system's existence is also included in their scope of duties. Even more important is the fact that the emerging great responsibility of executing important system tasks can be shared among several autonomic elements. Thus, it becomes possible to consign various autonomic elements with e.g. the task to configure parts of the system, while one autonomic element manages the coordination between the configuring components. This management concept is called *orchestration within a discipline*.
On the contrary, *orchestration across disciplines* is a management approach to coordinate all necessary task within all self-deciding elements. Hence, a coordinator can e.g. assign one autonomic element with the task to manage all self-protection measures. In turn, it is the assigned element's decision whether or not it needs help for this task (makes use of the orchestration within a discipline approach).

2.2.1 Main Goals of Autonomic Systems

Parashar et al. [51] emphasize that the human body's autonomous nervous system has the main goal of regulating and maintaining the so called *homeostasis*. The biologically connotated term homeostasis refers to open systems that maintain its structure and functions by reacting to every change in the environment, or to every random disturbance, through a series of modifications equal in size and opposite in direction to those that created the disturbance. The objective of these modifications is to the maintain internal balance.

As an example, heart beat, breathing, blood pressure, digestion and metabolism are continuously working together within the body to remain vital variables within their physiological limits.
In accordance to this metaphor, reaching the state of homeostasis will be the main goal of all autonomic systems, too. Correspondingly, autonomic systems' main goals be defined with the two following statements:

1. Survivability: Every autonomic system seeks to adapt its behavior to retain vitality.
2. Equilibrium: Whenever internal or external environment changes unbalance the system's stability, counteractive measures will be invoked in order to return to the equilibrium state.

Even more, since autonomic computing systems are inspired by the human autonomous nervous system, which handles the body's complexities and uncertainties without requiring conscious effort, the goal of autonomic computing research is to design computer infrastructures, that manage themselves with minimal human intervention.

2.2.2 Main Characteristics of Autonomic Systems

Already in 2001 Horn described his autonomic computing vision [31] with a more technical approach, and established thus the vocabulary for describing all aspects of an autonomic computing system. He states that a full-fledged autonomic system must possess the following eight characteristics:

1. Self-Awareness: An autonomic application/system "knows itself" and is aware of its state and its behaviors.
2. Self-Configuring: An autonomic application/system should be able configure and reconfigure itself under varying and unpredictable conditions.
3. Self-Optimizing: An autonomic application/system should be able to detect suboptimal behaviors and optimize itself to improve its execution.
4. Self-Healing: An autonomic application/system should be able to detect and recover from potential problems and continue to function smoothly.
5. Self-Protecting: An autonomic application/system should be capable of detecting and protecting its resources from both internal and external attacks, as well as maintaining overall system security and integrity.
6. Context-Aware: An autonomic application/system should be aware of its execution environment and be able to react to changes in the environment.
7. Open: An autonomic application/system must function in an heterogeneous world and should be portable across multiple hardware and software architectures. Consequently, it must be built on standardized and open protocols and interfaces.
8. Anticipatory: An autonomic application/system should be able to anticipate to the extent possible, its needs and behaviors and those of its context and be able to manage itself proactively.

These eight characteristics form the basis on which autonomic computing can be described. It comprises, for example, that self-managing systems are automatically installing software and hardware components, whenever they are detected as an useful plug-and-play component. Also, they would reactively or proactively install a required software component after detecting that it is missing for a certain –future– operation (self-configuration).
Furthermore, autonomic computing systems would procure behavior for restarting a failed element (self-healing), adjusting the workload distribution in accordance to the free resources (self-optimization), and react with counteractive measures if an intrusion attempt was detected (self-protection).

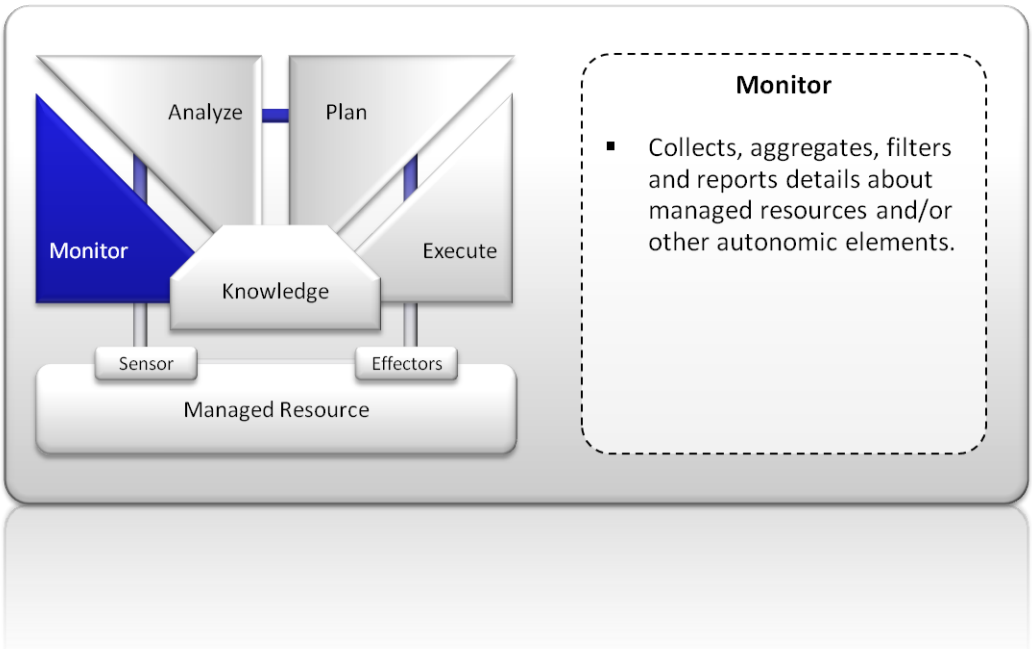
Each of the above mentioned characteristics presents an active research area. As one of the major research challenges the environmental awareness has to be seen, since most of the self-invoked interventions in the standard operating procedures must be planned and anticipated.
In conclusion, all above mentioned characteristics for their own are desirable features, but only in their entirety they enable self-managed infrastructures. In particular, all self-* properties are aspired by the autonomic computing research approaches.

The rest of this thesis is structured as follows: First, in the main part (Section 3 “Autonomic Elements”), all inevitably required functional units of an autonomic computing system are described in detail. Here, the required component’s functions are described and exemplarily research- and industry efforts are pointed out. Section 3.1 “Monitoring” focuses on monitoring of internal and external metrics within the autonomic element, whereas Section 3.2 “Analyze” outlines all necessary analytics and policy-handling mechanisms for autonomic systems. Section 3.3 “Plan” describes algorithmic approaches for an intelligent planning component. Section 3.4 “Execute” concludes the main part by giving an insight into the execution step within autonomic elements and systems.
Subsequently, in Section 4.1 “Standards for Autonomic Computing”, standardization efforts for data models, protocols and security mechanisms are highlighted. Then, in Section 4.2 “Autonomic System Approaches”, three promising autonomic computing prototypes will be presented. The summary in Section 4.3 will conclude the thesis with comments and an outlook into the future of self-managing computing systems.

3 Autonomic Elements

The eight characteristics conducted from Horn's autonomic computing vision (see: Section 2.2.2 "Main Characteristics of Autonomic Systems") motivates a set of mandatory tasks, that must be procured by an autonomic element for enabling self-* properties. Hence, it is necessary for every autonomic system implementation to realize some kind of monitoring-, analyze-, plan-, and execute mechanisms. Each of the tasks makes a contribution to the overall system's goal to seek for viability and releases the human administrators from unnecessary operating, maintaining and optimization tasks.
The following sections will deal with each of the *functional parts* of an autonomic element in detail, comprising an overview about the current (research and industrial) developments and achievements.

3.1 Monitoring



The main goal of the autonomic system's *monitoring* component is to provide mechanisms for collecting, aggregating, filtering and reporting details about managed resources and/or other autonomic elements.
Hence, monitoring is not only a turnkey solution for having insight into the running system, but has rather to be seen as a *knowledge-building* tool, delivering the facts on which the future decisions can based on.

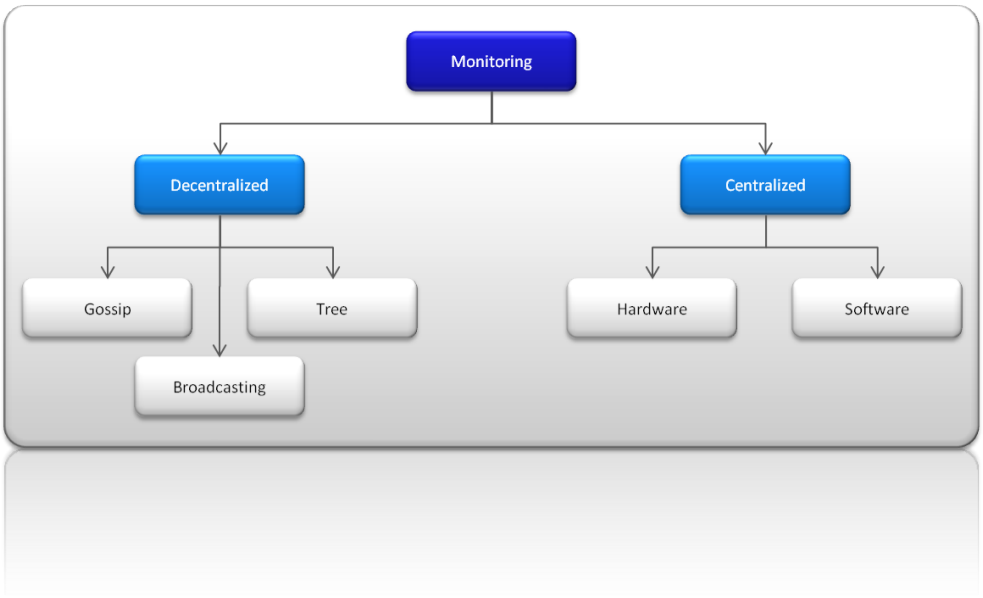


Figure 3.1: Monitoring Classification

As Figure 3.1 depicts, monitoring approaches can be divided in *decentralized* and *centralized* approaches. The following sections will deal with the different solutions in detail.

3.1.1 Decentralized Monitoring Approaches

External monitoring aim at unveiling facts about the system as a whole. For example, questions like “To which degree is the Quality-of-Service assured?”, “How many entities in the system provide a certain service or document?” or “Who is my topologically nearest neighbor?” can be issued with such a service.
However, when taking the dynamic and heterogeneous nature of Peer-to-Peer systems into account, it is remarkably difficult to reach every node to retrieve information. Various researchers have addressed peer- and system information management in Peer-to-Peer networks.

- *Gossip-based approaches:* Jelasity et al. [35] propose *Topology Management (T-MAN)* a protocol for constructing and maintaining topologies on-the-fly and on demand. This concept introduces the great advantage that a topology manager could be flexible in allowing to change the managed topology on demand, without having to develop a new protocol for each possible topology from scratch.
  In order to establish a designated topology, a single ranking function needs to be adapted. With the help of this function every mathematically describable object can be used to specify the desired topology. Since the topology is not fixed, only two information dissemination algorithms can be used: Flooding (every peer communicates its updates to every known neighborhood peer) or gossip-based communication themes (when a change happens, a peer sends its update to one or more randomly chosen neighbors).
  The query process works analogous. Whenever a node wants information about a topic, he

asks his neighbor for it, who in turn will ask a random neighbor and so forth.
Experiments have shown that the T-MAN algorithm is capable of establishing topologies that follow a total ordering relation, take proximity information in consideration (yielding to geographical groups) or establish a semantic clustering of peers (according to a given metric).

- *Broadcasting-based approaches:* Idreos et al. [33] propose *P2P-DIET*, a extensible service over Peer-to-Peer networks that unifies data sharing and publish-subscribe. P2P-DIET implements an agent-based system, which supports queries, profiles and notifications at Internet-scale.
  The architecture consists of two node types, super-peers and clients. All super-peers are equal and have the same responsibilities. Each super-peer serves a fraction of clients and keeps indexes on the resources of those clients.
  Whenever a query appears, a client accesses his super-peer who broadcasts the message across all super-peers (a minimum weight spanning tree is used for bounding the complexity).
  Similarly, P2P-DIET provides a publish/subscribe service through continuous queries. A client registers for the receiving of a notification, whenever a assured resource appears in the system.

- *Tree-based approaches:* Graffi et al. proposed *SkyEye.KOM* [28], a overlay structure capable of generating statistics, capacity-based peer search and finding peers for layer-specific role assignments. SkyEye introduces an efficient management Over-Overlay to gather, aggregate and store meta-data information about peers.
  In their topology-independent architecture, the ID space is recursively subdivided into so called *domains*. Each domain has its own *coordinator*, who is chosen according to a deterministic function. Interestingly, this coordinator may not be the only responsible peer for this subcluster. Rather, each coordinator decides himself if he wants to establish helper-coordinators to fulfill his tasks.
  In SkyEye, client-peers send periodically information to their associated coordinator, who collects and aggregates the knowledge. Additionally, each coordinator periodically reports to his higher-level coordinator. Consequently, the tree's root maintains information about the system's characteristics. This information is especially valuable for autonomic systems, because it enables delivers a system-wide view onto the distributed system.
  Whenever a peer issues a request for a service or document, the coordinator will process the request and return an appropriate answer. If no successful answer can be generated, the coordinator handles the request to the next hierarchy level and attach his (partly) list of satisfying peers. With this scheme, questions like "Give me ten peers with the capability X." or "Find the first peer with a upload capacity of 10 MBits/s." can be answered efficiently.

- *Distributed Hash Table Extensions:* Albrecht et al. propose the *Distributed Approximate System Information Service (DASIS)* [3]. DASIS is enhancing a Peer-to-Peer system's Distributed Hash Table (DHT) by building a logical binary tree on top it.
  By means of this tree each peer is associated with its own unique overlay identifier and an according subdivision of the ID space, so called "Domain spaces". A peer is responsible for storing all keys that are in his domain space, thus building a knowledge about the subclusters. Peers periodically exchange sub domain information with their neighbors (neighbor

information according to DHT).
DASIS' query strategy is either *proactive*, meaning that the system instantly acts in case of an event or *reactive*, collecting the requested information before answering a query. The DASIS clients run simultaneously at every peer, resulting in a bottom-up aggregated information collection at every peer.

A further in-depth analysis of information management solutions for Peer-to-Peer systems can be found in [10].

3.1.2 Centralized Monitoring Approaches

Autonomic computing systems are ultimately hardware-dependent computer systems. Hence, to achieve a 24/7 operational status the large amount of multi-agent systems must be actively controlled. Thus, centralized monitoring approaches aim at unveiling facts about the managed resources (CPU load, temperature, bandwidth utilization) in an autonomic element.
Since this topic is highly interesting from a commercial perspective, a wide range of hardware manufacturers established system-on-a-chip or operation system independent solutions for monitoring internal capabilities.

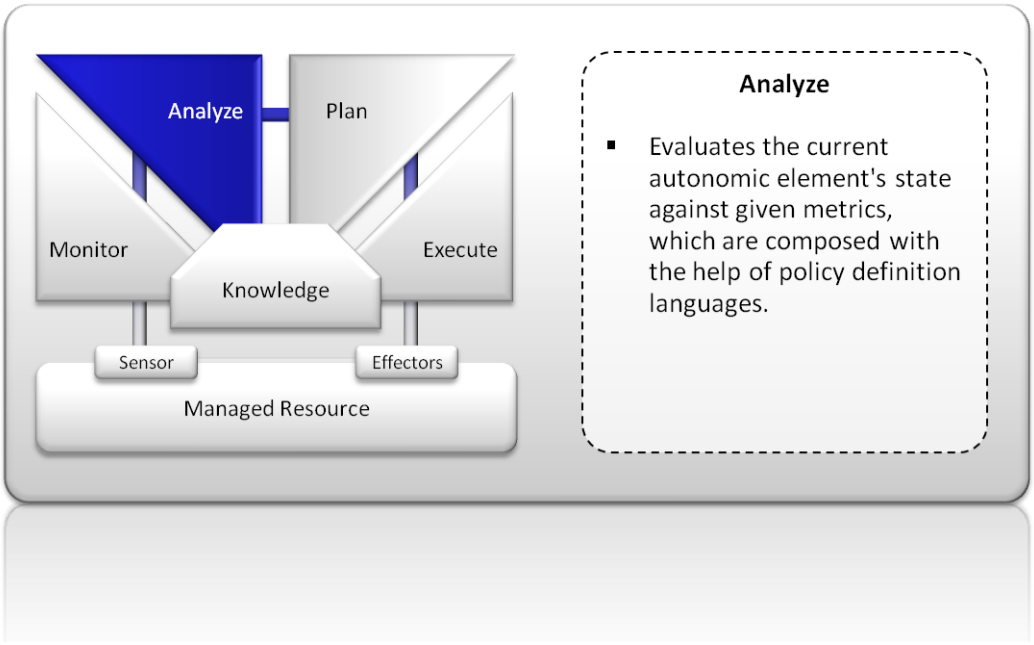
- *Intel Active Management Technology (AMT):* Intel's AMT technology [20] is currently developed under the *vPro technology* label, which introduces autonomic capabilities, such as self-healing and self-protecting, to mainstream PC systems. It features amongst others device discovery, feature tracking, system shut down and restart, and utilization metrics, even when the host's Operation System (OS) is inactive.
  AMT's architecture is based on the concept of a platform container. It is an OS independent computing platform that resides on a dedicated microcontroller in the chipset, or a plug-in card. Accordingly, this platform container has a isolated execution process (including processor and memory), which enables the system to operate in pre-power, pre-OS states, is capable of assisting the OS when it is present, and taking over when it is not.
  In order to achieve broad acceptance and compatibility even in the future, Intel's AMT technology makes use of standardized interfaces. For example, WS-Management is used as an external interface to connect the system to a network. A more detailed information about the WS-Management standard can be found in the Section 4.1.1 "Standards for Autonomic Computing". Furthermore, the platform is able to sensor host metrics, such as the status of running program, the overall OS status, CPU utilization, and hard-disc drive health. Intel's AMT technology is already broadly integrated into desktop PCs with Intel Core 2 processor and also available in laptop PCs with Centrino processor.

- *IBM System z10 Active Resource Monitoring (ARM):* IBM's ARM technology is targeted towards the goal of maintaining continuous operation on IBM servers. Therefore, ARM implements an integrated, automatic resource monitoring software application to track resource, performance, and operational problems. Moreover, the system initiates recovery actions in case of failures.
  In contrast to Intel's AMT approach, IBM's ARM focuses on the high-performance and commercial server market, which is the basis for grid-computing and large enterprise networks.

Conceptually, IBM's ARM architecture contains two uncoupled, but dependent components. First, the hardware management console (HMC), which is a desktop computer running an operating system, communication-, security-, and GUI administrator software. In addition to that, it hosts the "Licensed Internal Code" (LIC), an implementation of IBM's MAPE control loop for their server architectures. Using a local area network (LAN), the HMC communicates with the second component of the ARM architecture: the two support elements (SEs). These SEs are located in the Z-frame server rack. Both the HMC and SEs are closed fixed-function devices, and the customer cannot install any code or applications on the systems other than the firmware supplied by IBM.
Upon detection of a problem First Failure Data Capture (FFDC) information is collected, saved, and a notification of the service being required is generated. In the following, the HMC program will initiate a recovery action or actions. Currently, the actual recovery actions are limited to previously scripted reaction behaviors. They consist of erasing selected files and terminating programs. In case of a highly critical issue, the server can suggest to be restarted (an physically available administrator has to grant the request).

The decoupling of services and the infrastructure has to be seen as a key-enabling factor for the so called utility-computing or cloud-computing paradigm. It enables the grouping of resources into virtual pools, which can be allocated and deallocated dynamically. Thus, grid-like data centers can efficiently use resources according to unsteady computing and I/O requirements. Moreover, this service-oriented architecture allows to establish a pay-per-use environment, which is also an inevitable requirement for the success of autonomic computing systems.

3.2 Analyze



The second building block of the MAPE loop is the *analyze* component. Its main purpose is to evaluate the current autonomic element's state against given metrics, which are composed with the help of *policy definition languages*. In general, policies represent predefined and externalized logic, that determines the autonomic element's future behavior. Correspondingly, the "overall target" is to fulfill the goal policies, which are defined through a single or a set of desired states. Policies govern the decision making process for autonomic systems, indicate which resources are to be monitored or not and how changes need to be propagated in the system. Hence, policies are static descriptors that encode the generic behavior of the system. As an example, policies can be defined to manage multiple aspects of distributed systems, such as their QoS-preferences, configuration, updating- or monitoring behavior, and auditing.

Autonomic elements evaluate their current status with the help of so called *policy engines*. In general, these policy engines comprise a common architecture, which is shown in Figure 3.2 in a highly abstracted manner.

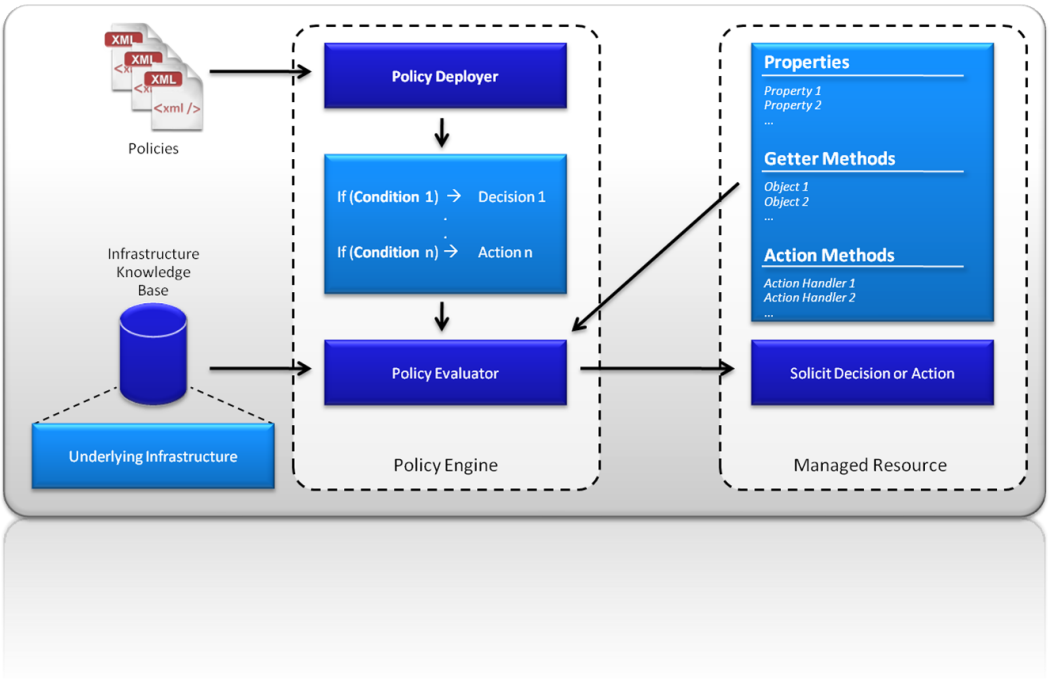


Figure 3.2: Autonomic Element: Policy Engine

Deploying new policies in a self-managing system begins with obtaining a policy or goal statement. In the current state of research, *policy descriptors* have to be written manually by domain-aware administrators for nearly all aspects of the "autonomic" computation. This statement holds especially for the current commercially available autonomic computing systems, such as IBM Tivoli [61] or Fujitsu Triole [24]. Albeit this mandatory step provides the system with a choice of management solutions/processes and thus a basic vocabulary of behaviors, it has to be seen as a major drawback of current autonomic systems. The autonomic computing manifesto, however, envisions a system of self-configuring and self-adapting elements that are acting in concert to reach a certain high-level goal. Research efforts, such as the "Policy continuum" approach [63] are trying to bridge this gap, by allowing an automated transformation of high-level (mostly business-related) policies towards low-level (e.g. device-dependent) goals.

3.2.1 Policy Definition Languages

Policies' main purpose is to guide the behavior of managed entities by extracting and externalizing business logic into a set of rules. As one example for policy definition languages ACPL will be introduced in this paragraph. The *Autonomic Computing Policy Language (ACPL)* is IBM's "language-of-choice" for defining policy descriptors and used throughout their *Policy Management for Autonomic Computing (PMAC)* approach. It is inspired by the *Common Information Model (CIM)* and comprehends influences of OASIS' XACML, WS-Agreement, and the WS-Policy standards. Further information referring to standards for autonomic systems can be found in Section 4.1 "Standards for Autonomic Computing".
In the XML-language ACPL, each policy is a rule composed of four components:

1. *Conditions:* Determines, whether a policy is applicable, or not. The policy is always applicable, whenever the associated condition statement evaluates to true. Otherwise, a further policy evaluation can be skipped.

   Example: The following ACPL snippet describes the condition "Whenever the aggregated order entry over the last month is greater than 4,000 and the aggregated order entry over the last year is greater than 10,000, . . . "

```
<acpl:Condition>
  <exp:And>
    <exp:Greater>
      <exp:PropertySensor propertyName="Orders_Over_Last_Month"/>
      <exp:IntConstant>
        <Value>4000</Value>
      </exp:IntConstant>
    </exp:Greater>
    <exp:Greater>
      <exp:PropertySensor propertyName="Orders_Over_Last_Year"/>
      <exp:IntConstant>
        <Value>10000</Value>
      </exp:IntConstant>
    </exp:Greater>
  </exp:And>
</acpl:Condition>
```

2. *Actions:* If a policy is applicable, then the set of corresponding actions will be executed in the future.

   Example: The following ACPL snippet describes an action, which refers to the previously mentioned condition ". . . then give the customer a discount of 20% . . . "

```
<acpl:Decision>
  <acpl:Result>
    <acpl:Property propertyName="Discount">
      <exp:FloatConstant>
        <Value>.2</Value>
      </exp:FloatConstant>
    </acpl:Property>
  </acpl:Result>
</acpl:Decision>
```

3. *Priority:* The priority component is an ACPL specific tag. A non-negative integer indicates the importance of the associated policy and determines with which priority the correspondent actions shall be executed.

   Example: The following ACPL snippet describes a low business value.

```
<acpl:BusinessValue>
  <Importance>10</Importance>
</acpl:BusinessValue>
```

4. *Role:* A role defines the context in which the policy is relevant. For instance, a policy describing a mail server will not be of interest for a component managing a database system.

   Example: The following ACPL snippet describes a policy's role, here the domain "customer reward".

```
<acpl:Scope>
  <acpl:StringScope>
    <Value>Customer_Reward</Value>
  </acpl:StringScope>
</acpl:Scope>
```

The PMAC reference implementation helps administrators to create and modify policies with the help of a graphical user interface system, called PDT (Policy Definition Tool).

3.2.2 Policy Deployment

The second step for introducing a new policy to an autonomic system is *policy deployment*. Policy deployment consists of two basic subcomponents: (1) *Policy Ratification* and (2) *Policy Storage*. Whereas the second step (storage) manifests the serialization into a storage module (PMAC's Policy Editor Storage supports DB2, Microsoft SQL, Oracle and Cloudscape), the first step of deploying a policy is rather noteworthy.
The policy ratification module certifies policies by examining the relationships with other established policies in the system, before the policy is activated or "ratified". As an example, a system administrator wants to know if his policy for an update process conflicts with backup policies.
In IBM's Policy Management for Autonomic Computing (PMAC) each policy has to pass a series of three evaluation steps in order to get ratified.

1. *Dominance Check:* Whenever the addition of a new policy does not affect the behavior of the system, then the policy is called *dominated* by an already existing policy or policy group. For instance, the policy "Joe has access to the file server from 1 P.M. to 5 P.M." is dominated by the policy "Joe has access to the file server from 8 A.M to 7 P.M.".

2. *Conflict Check:* Whenever the addition of a new policy issues actions, that cannot be achieved simultaneously, the policies are called *in conflict*. For instance, the policy "If a file transfer request is issued after after 5 P.M., then serve the connection with QoS level LOW" stands in conflict with the policy "If a file transfer is issued from the headquarters, then serve the connection with QoS level HIGH".

3. *Coverage Check:* In many application domains, the administrator may want to know if policies have been explicitly defined for a certain range of input parameters. For instance, the administrator may want to know if a policy for managing a printer covers all priority classes, days of the week and all hours of the day.

Further details referring to policy checks can be found in [2].

3.2.3 Policy Evaluation

The policy evaluator represents the most important subcomponent of the *analyze* building block. Here, policies are evaluated against the autonomic element's current state. In contrast to the original PMAC proposal, Figure 3.2 shows an *Infrastructure-Aware Policy Engine*. In the traditional PMAC approach, the policy evaluator accessed its managed resources' states entirely through exposed interfaces, giving the evaluator only access to a limited set of properties. On the contrary, a infrastructure-aware policy engine has access to infrastructural knowledge, which contains relevant information about the underlying resource base. This information can be stored persistently and efficiently in an appropriate format (e.g. database, file format, XML) for future analysis and –more importantly– optimization steps. Furthermore, decoupling the policy engine from the infrastructure has the advantage, that resource-specific or even vendor-specific properties can be taken into consideration. Hence, infrastructure-aware policy engines are able to express policy conditions, containing both infrastructure-dependent and independent operations, as well as vocabulary terms.

3.2.4 Policies and Relationships

Each autonomic elements is not only responsible for its own internal state and behavior, but also for managing its relationship with other autonomic elements in the environment. Thus, a large number of signals and messages from and to other elements must be recorded, evaluated, and processed. An autonomic element will decide about changes in its own behavior and the interaction to others according to three distinct considerations:

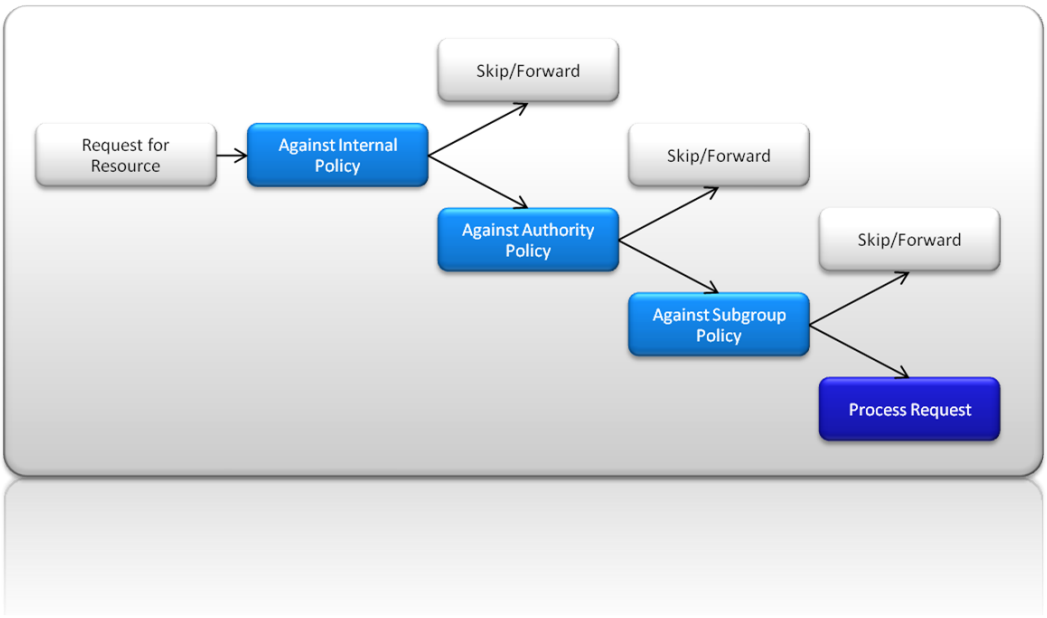


Figure 3.3: Decision Tree Behavior Change

Every time a request reaches an autonomic element it weighs the request against: (1) predetermined or hard-coded goals in the autonomic element itself, (2) the policy of other elements

that have authority over it, (3) or the policy established through explicit or implicit subcontracts to collaborating groups.
As Figure 3.4 depicts, at lower levels autonomic elements will have a hard-coded limited repertoire of behaviors and a predetermined short list of potential connection elements from which they may request assistance. However, at higher levels no behavior or connection pattern and no relationships list will be specified, thus allowing dynamism and flexibility. Accordingly, at higher levels administrator-defined goal-oriented terms, like “Remain this level of Quality-of-Service (QoS).” or “Reach/maintain this Service-Level-Agreement (SLA).”, will determine the systems’ conduct and leave the responsibility of archiving the goal in the hands of each element.

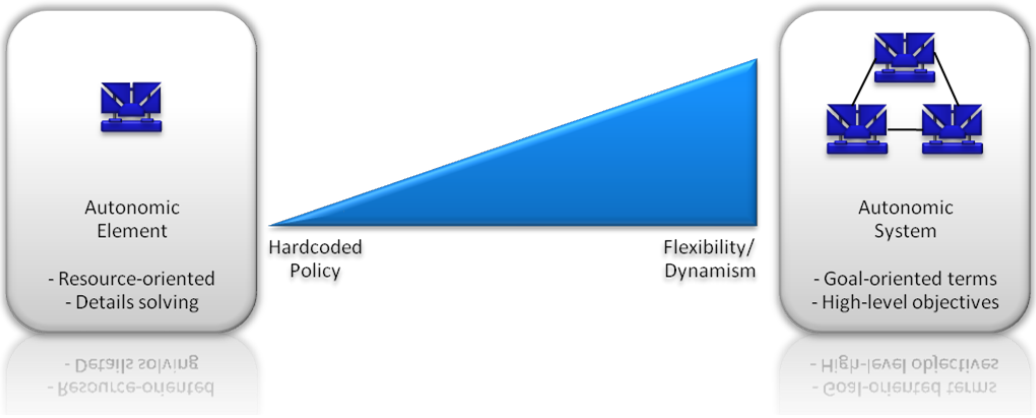


Figure 3.4: Policies: Hierarchy vs. Dynamism

3.2.5 Parameter and Metric Correlation

In a full-fledged autonomic system goal policies are provided by administrators in order to guide the system as a whole, rather than every single resource on its own. Accordingly, policies such as “Remain the response time of service X within a range of 10ms to 100ms.” expresses a system-wide, high-level performance expectation, that is not resource-specific. Nonetheless, system-wide performance expectations imply certain metrics, which have to be abide by the managed resources.
In addition to this, the autonomic system’s overall vision comprises a self-controlled seeking for performance. Hence, every autonomic manager should enforce the system to work at its most sufficient level by adapting a service’s parameters in a reactive or even proactive performance management.
Performance is tried to be improved stepwise, pursuing the well-known tuning principle to change only one aspect at a time, which leads to reduced/eliminated side effects.

In order to analyze a resource’s performance measurement using mathematical methods, a definition of *performance* is required. From the mathematical point of view, autonomic systems are confronted with finding an optimal solution for the *performance function*, which is defined as follows:

$$f_{P,M} : D_{P_1} \times \ldots \times D_{P_k} \rightarrow D_{M_1} \times \ldots \times D_{M_n}$$

with $D_{P_1}, \ldots, D_{P_k}$ as discrete domains of the parameter tuple $P = (P_1, \ldots, P_k)$ and $D_{M_1}, \ldots, D_{M_n}$ as domains of the performance metric tuple $M = (M_1, \ldots, M_n)$. By this means, the function f expresses the functional influence of parameters P_i on the metrics M_i [22].
The mathematical formulation of the problem exposes the three necessary components, which have to be given for self-optimization process:

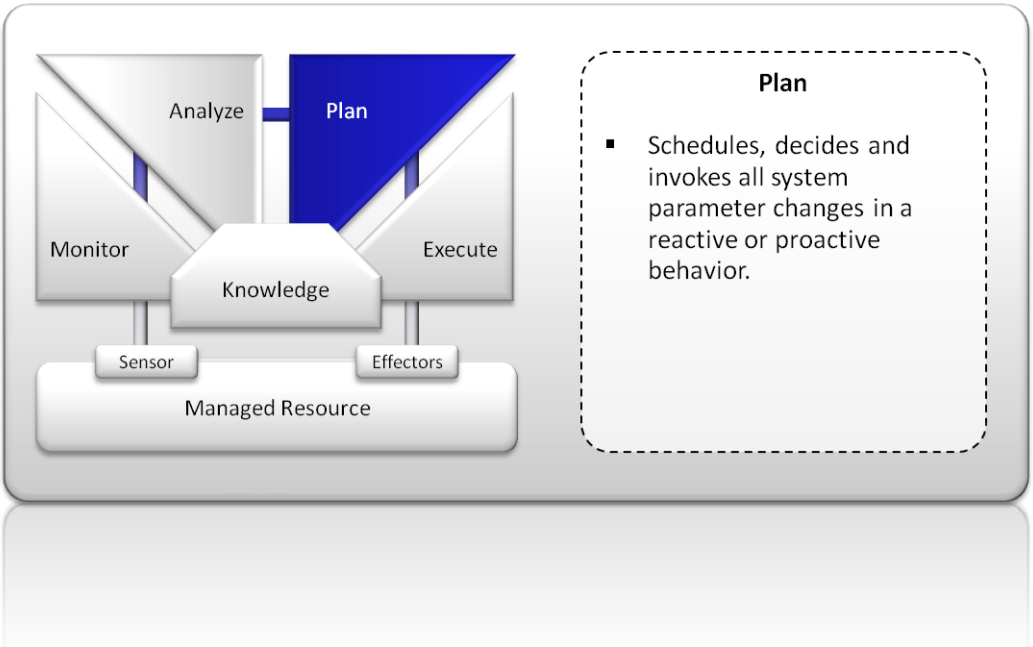
1. *Performance Metric:* A performance metric can be the size of a list or buffer, or its average filling level, the utilization/workload (e.g. of a peer or a service), the average response time for a service, or the hops required to reach a service. The currently analyzed data will be obtained in a reactive (request) or proactive (e.g. scheduled) step in the monitoring phase (see: Section 3.1 “Monitoring”).

2. *Parameters:* Identifying *relevant* parameters may lead to immense efforts. The current research supposes two approaches for identifying performance-related parameters: First, [22] proposes to trivially iterate through either all or a set of possible parameter candidates and choose the one with the best performance increase.
   Second, Brake et al. [12] go one step further by introducing *Software Tuning Panels for Autonomic Control (STAC)*.
   Their goal is to identify tuning parameters and provide automatic rearchitecting of the source code to expose them in a separate control panel module. This control panel module is supposed to be the central point of control for autonomic managers.
   The project phase one (STAC I) already found in 2006 a solution to the latter problem (rearchitecting and exposing parameters). STAC II is focused on the automatic identification of tuning parameters. Basing on the work of T. J. Biggerstaff [7], they use static source code analysis methods for extracting an entity-relationship (ER) model of the entities (e.g. variables, methods, classes) and relationships (e.g. calls, comparisons, assignments) to form a *software graph*. In the subsequent analyzing step, the graph is explored using relational algebra and graph pattern matching to discover deeper relationships.
   The project already delivered a java-based prototype implementation. Related work and tools for software design recovery can be found in Rigi [56] or CrocoPat [5], who use relational programming to extract and infer model graphs and use pattern matching algorithms for analysis.

3. *Performance function:* Finding the dependency- or performance function, which shows the impact of a parameter change, is mostly a pre-deployment step.
   Rabinovitch et al. [52] propose a development of a resource dependency model, allowing precise optimization and decision-support at run-time based on best practices. In their studies they face the problem of self-optimization in database management systems (DBMS). DBMS are especially suited for the academic research, since they expose more than one hundred configurable parameters and a vast number of corresponding metrics.
   In order to determine a qualitative parameter-metric correlation, the researchers perform incremental changes of parameter values so that each combination of the parameters are tested under the same workload conditions. In subsequent processing step, the test results are analyzed for quantitative correlations via Matlab and qualitative dependencies via the

Gauss-Newton non-linear regression method. Finally, the corresponding, empirically determined performance function can be concluded.
Since, the dependency function has to be obtained under controlled environmental factors, (i.e. workload), a mandatory performance evaluation phase needs to be established before a service/component is deployed. Nonetheless, the highly computing intensive process has to happen only once, because it is unlikely that a performance function will change.

3.3 Plan



Subsequent to the MAPE loop's analysis step, which potentially reaches the conclusion, that a system's parameter must be changed, a request to the *planning* component will be issued. The planning components purpose is to schedule, decide, and invoke all necessary changes reactively, or –even more desirable– proactively.
In an ideal case, planning components work proactively, having learned from former good or bad decisions, and draw upon environmental-aware experiences. For example, a planning component could have learned, that the average server load of a web shop increases by 30% from 6 P.M. to 9 P.M., whereas the serverload from 2 A.M. to 6 A.M. is fairly low. Recognizing this time-workload correlation, autonomic systems would issue an overprovisioning/adaptive underprovisioning of resources in the appropriate time.
In general, the planning module is faced with the problem of finding the best possible elements from the space of possible solutions. This space is a set of criteria, expressed with the help of mathematical functions, which might either be obtained through analytical processes (see: Section 3.2 "Analyze") or are tied to the service at development stage. However, these *performance functions*, also called *objective functions*, are very likely to have multiple local optima. Nevertheless, autonomic systems –as already noted– are supposed to seek for steady performance improvements, resulting in the fact, that *optimization algorithms* are supposed to find the (only) global optimum.

The following paragraph will highlight the most promising approaches for autonomic planning in distributed and centralized systems. Moreover, the paragraph will give an insight into the wide variety of optimization and configuration problems arising in distributed systems.

A detailed information source about the general topic of optimization algorithms for distributed algorithms can be found in [66].

3.3.1 Evolutionary Algorithms

Evolutionary algorithms' underlying idea is to improve the system's configuration iteratively in each *evolution cycle*. More specifically, *evolutionary algorithms* are confronted with a given population of individuals and an environmental pressure, resulting in a natural selection. Hence, evolutionary algorithms encode Darwin's "survival of the fittest" idea [67], which demands for a steady rise in the fitness of a population (In this context: find a better configuration in each iteration).
As Figure 3.5 depicts, evolutionary algorithms are generic, population-based, meta-heuristic optimization algorithms, that use biology-inspired mechanisms like mutation, crossover, and natural selection to accomplish a randomly generated reconfiguration of the system. This may lead to an increased performance.

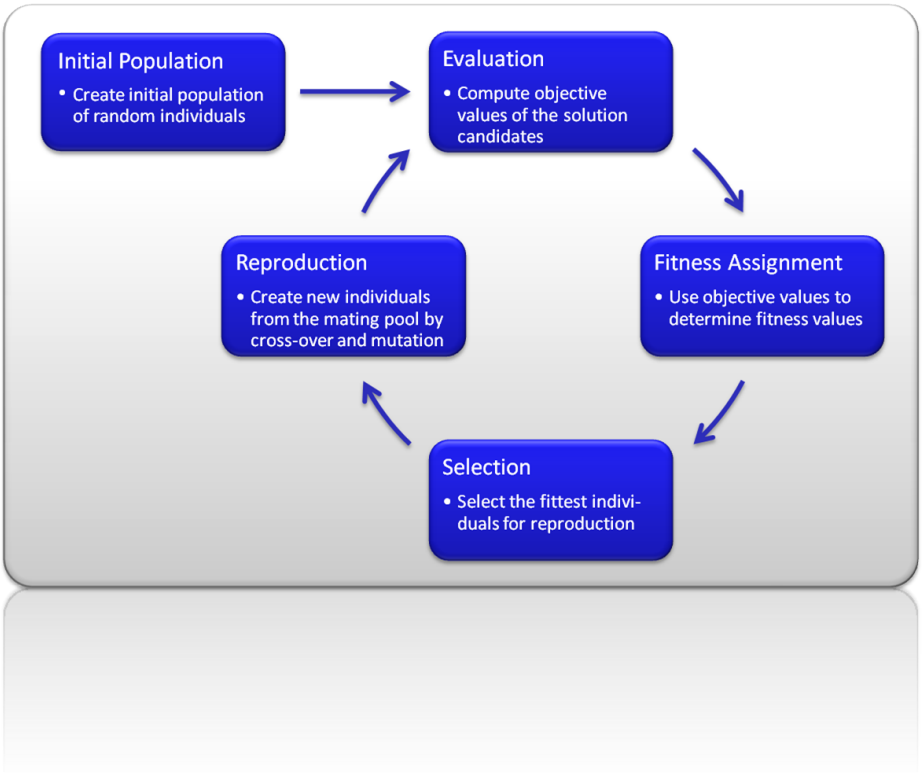


Figure 3.5: Evolutionary Algorithms Cycle

Evolutionary algorithms, in general, subsume two distinguishable subclasses:

1. *Genetic Algorithms:* The early work of Goldberg et al. [26] paved the paths for genetic algorithms. Subsequent fundamental research was conducted by Holland [30]. Both researchers were inspired by the biological evolutionary process. In their *genetic algorithms*,

they distinguish between the search or *genotypes* space $\mathbb{G}$ and the problem or *phenotype* space $\mathbb{X}$.
The set $\mathbb{G}$ corresponds to all possible variations of the configuration, that can be obtained by applying the reproductive mechanisms (recombination and mutation) (see: Figure 3.6). On the other side, the phenotype space $\mathbb{X}$ represents only the feasible solutions of the optimization problem. Every $x \in \mathbb{X}$ can be rated according to the performance function f. Resulting from this fact every $x \in \mathbb{X}$ can be compared to a different solution $y \in \mathbb{X}$ by comparing the evaluation of $f(x)$ to the solution of $f(y)$. Based on this comparison a relative fitness value can be assigned to each individual.
Again copying the nature, genetic algorithms utilize the idea of sexual and asexual reproduction, which are implemented in form of cross-over and mutation operations, to create a child genotype. Since in every step only the two fittest genotypes are selected, it is expected that the offspring will obtain the best characteristics from both parents. Additionally, in every generation step a mutation may occur according to a certain likeliness.

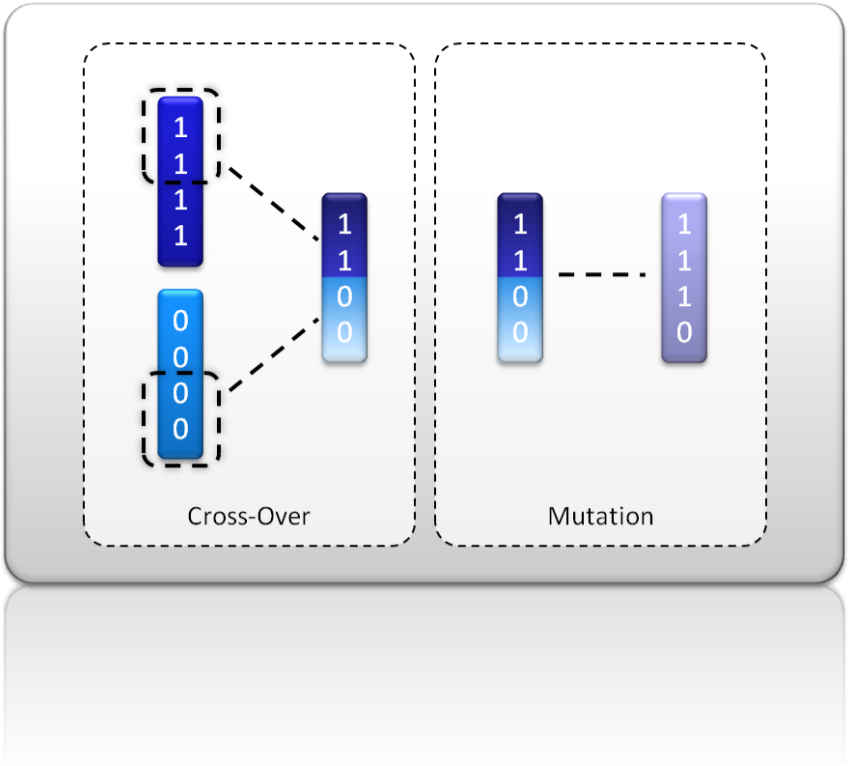


Figure 3.6: Genetic Algorithms: Cross-Over and Mutation

After a genotype $g \in \mathbb{G}$ is generated, according the rules shown in Figure 3.6, it is evaluated whether g is an element of the phenotype space ($g \in \mathbb{X}$) or not. If $g \in \mathbb{X}$ the solution is an applicable recombination of the characteristics and accordingly a feasible solution. Otherwise, $g \notin \mathbb{X}$ implies a non-feasible solution, which has to be discarded.
The cycle of reproduction, genotype-phenotype mapping, objective function computation, fitness assignment, and selection sketched in Figure 3.5 will be repeated until a termination

criterion is met.
Genetic algorithms, in general, are representing their search space with the help of binary strings or arrays of other elementary types $\mathbb{G} = \mathbb{B}^n$, where n is the length of the (binary) array.

2. *Genetic Programming:* The fundamental research achievements by Koza et al. [42] laid the foundation for *genetic programming*, which often subsumes all evolutionary algorithms producing tree data structures as phenotypes (elements of the solution space).
Similarly, it is seen as the group of evolutionary algorithms which automatically generate programs, algorithms, and similar constructs. More generally, genetic programming is a systematic method for automatically solving a predefined problem. The mechanism starts from a high-level statement of "what needs to be done" and automatically creates a computer program to solve the problem. Obviously, these characteristics manifest a remarkable similarity to autonomic system's requirements for self-optimization.

In the following the applicability of evolutionary algorithms to distributed optimization problems should be exemplary reviewed.

Optimizing Network Topology: The applicability of genetic algorithms for network topology optimization was showcased by Khuri and Chiu [39]. Their goal was to determine an optimal solution for the *terminal assignment problem*. The objective is here to determine the minimum cost links to connect a given set of nodes (the terminals) to another disjoint set of nodes (the concentrators). The required capacity of each terminal is known and may vary from terminal to terminal. The capacity of the concentrators is also known and so are the costs for linking them to the terminals. Each terminal must be linked to exactly one concentrator in a way that the maximum capacity of no concentrators is exceeded. An assignment has to be found under the objective of minimizing the total costs.
This $NP-hard$ problem was solved with genetic algorithms by Khuri and Chiu by using a string genome $\mathbb{G} = \mathbb{X} = \mathbb{N}^n$ consisting of natural numbers, where n is the number of terminals. Every value of the gene s_i in the gene string $S = (s_1, \ldots, s_n)$ denotes the contractor's connection to the i^{th} terminal.
Infeasible solutions, violating any kind of constraints, are not as normal eliminated, but penalized, so that they can be separated from the feasible solution set. The results of their work indicated that the original genetic algorithm provided much better results, than the greedy approach.
As another example of network topology optimization through genetic programming and genetic algorithms were shown in the European Optical Network (EON). EON is an optical overlay network capable of carrying the entire international traffic between twenty of the main centers of Europe. The initial design of EON was improved by using genetic algorithms and genetic programming. As results of the work in [54] indicate, genetic algorithms outperform genetic programming, whereas both of them are significantly faster than greedy approaches. Accordingly, especially genetic algorithms are suited for self-managed topology optimization tasks in autonomic systems.

Optimizing Routing: Kirkwood et al. [40] proved the applicability of genetic programming to solve routing problems. Their solution is capable of breeding robust routing rules for networks where links are likely to fail.

In traditional centralized networks methods, like Dijkstra's algorithm, for identifying the shortest path through a static network are used. Albeit this shortest path algorithms are well researched and tuned to an optimal outcome, these algorithms have to be reapplied, whenever a link fails in the system (e.g. whenever a node leaves the Peer-to-Peer network). In contrast to the current solutions, Kirkwood describes an algorithm, which automatically breeds an evolutionary program (here LISP code) to find the near-shortest paths in a given network without reapplying the algorithm again.
However, Kirkwood also states, that their solution is not in competition with Dijkstra's algorithm or any other traditional centralized routing algorithm. The researcher denotes that their algorithm rather "blindly" evolves to find the near-shortest path in the given network subject to link failures. Thus, it delivers a fault-tolerant and robust routing scheme [40].
The solution presented by Kirkwood showed a success rate of 89.5% when almost 30% of the links have failed (solution for a seven-link network with 19 satisfying solutions resulting from breaking two links).
In the planning step of autonomic computing, genetic programming may lead to a performance increase especially in highly dynamic Peer-to-Peer networks. Imaginable would be a hybrid solution based on fault-tolerant routing rules, which fall back onto sophisticated routing algorithms, whenever the evolutionary solution fails.

Optimizing Parameters and Configurations: Nakano and Suda present in their work [48] a self-managing multi-agent system capable of adapting to changes in environmental conditions, such as network failures, varying workload conditions, and changing platform costs.
In their work, network services are represented by mobile agents capable of executing different behaviors, such as replication, migration and termination. The agents main information is self-contained in its *energy level*. Performing services for users causes the energy level to increase, whereas the energy level decreases whenever a resource has to be accessed or a replication process is executed. By this means, network services, such as a HTTP server, are represented by its agent who triggers an action whenever a corresponding weighted sum surpasses certain thresholds. This sum manifests environmental-awareness, since it not only incorporates the agents internal states (age and energy level), but also environmental information (resource costs, request rate, etc.).
The genetic algorithm correspondence is implemented through the agents' replication and reproduction behaviors. The replication process imitates mutation in evolutionary algorithms, leading to a child with slightly changed weight vectors. Reproduction is the implementation of the cross-over operation. It chooses a partner within n hops in the network neighborhood based on measures: (1) the users waiting between request and execution, (2) the number of hops between the execution agent and his requester, (3) and the energy efficiency.
In general, genetic algorithms appear to be a promising approach for the autonomic element's planning component. Not only that these genetic algorithms can adapt to the changing environment, but they are suited for optimization problems, in which the performance function is likely to change over time. More specifically, performance functions are less likely to change in a local context (e.g. database performance will always have the same measurements), but in the global context, performance measurements will change due to time-workload correlations (day/night workload decline) and/or specific task bursts (e.g. VoIP, Video-Conferencing, etc.).

3.3.2 Simulated Annealing

As another algorithm for computing optimal solutions for combinatorial problems *Simulated Annealing* is of relevance.
Simulated Annealing (SA) is a partly random technique of search and optimization, usually used for hard combinational optimization problems.
The technique bases on the idea of iron annealing. Here, the goal is to reach a minimum energy state upon cooling the iron within the minimum amount of time. If the iron cools too quickly, the molecular structure of the metal will be highly disordered, leading to an increased brittleness and thus a low-quality product. However, cooling the metal too slowly, e.g. by adding heat in a heating furnace, results in an economically bad solution, whereas the molecule structure is ordered, stable, which is correspondent to a high-quality product.
As shown in Figure 3.7 simulated annealing, here used to find a global maximum, is a heuristic search, allowing non-improving moves to a neighbor. This disadvantageous steps are often necessary to overcome local minimum or maximum. Despite of all, SA decreases the probability for these non-improving steps continuously. The rate of this decrease is determined by the so called *cooling schedule*, often just a parameter used in an exponential decay (in keeping with the thermodynamic metaphor). With some small assumptions about the cooling schedule, this method is highly likely to converge towards a global optimum.

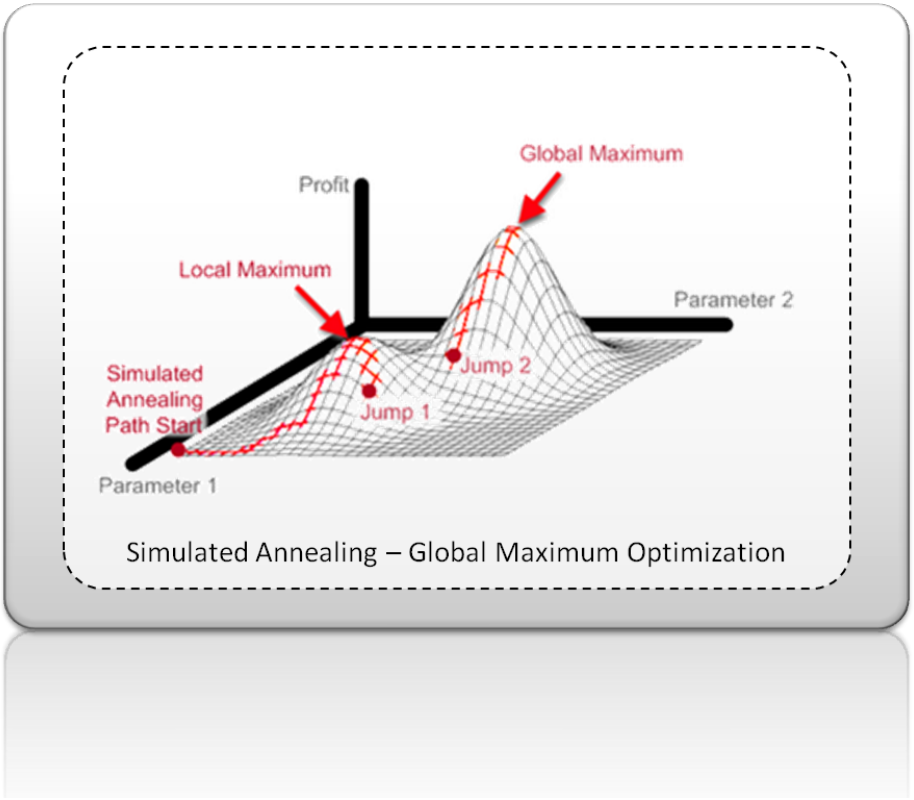


Figure 3.7: Simulated Annealing: Global Maximum Optimization

Matossian et al. [45] propose in their study "Autonomic Oil Reservoir Optimization on the Grid" a fast and reliable optimization algorithm. Their solution, aimed at finding the optimal placement for oil drilling wells, builds on top of an algorithm called *Very Fast Simulated Annealing (VFSA)*, an enhanced *simulated annealing* method.
VFSA is a variant of SA, which speed the annealing process significantly. It differs from this parent algorithm by introducing a temperature-dependent Cauchy-like distribution. This improvement in the cooling schedule leads to a larger sampling of the model space at early stages (faster convergence towards the optimum) and a much narrower sampling in the model space, when the area around the global optimum is reached. Additionally, VFSA introduces the concept of parameter-dependent cooling schedules. Each parameter in the parameter space can have its own cooling schedule and model space sampling, allowing individual control for each parameter and therefore embracing of a priori information. Especially in autonomic systems, where a certain amount of knowledge is inherent and ubiquitous, VFSA could have great advantages.

In the following the applicability of simulated annealing algorithms to distributed optimization problems will be exemplary reviewed.

Optimizing Network Topology: Zweig is presenting in [71] a self-optimizing network topology approach, which can be considered as a *hill-climbing* approach with evolutionary aspects. In contrast to genetic approaches, Zweig's algorithm is not population-based and delivers per iteration only one network, that must be considered in the subsequent steps.
In each step of the optimization process, one node's edges will be modified by either adding one connection (from this node to a neighbor) or deleting an edge attached to a neighbor. If this modification step seams to be advantageous for the node, it will be accepted for the next iteration, otherwise it is rejected. One can clearly see, that the decision is made locally and exclusively builds on the perspective of the node that is modified. Zweig coined for this purpose the notion of "Singular, Selfish, and Self-Organized (S^3) Networks".
In summary, Zweig presented a method for self-organizing networks, which are capable of minimizing/maximizing given objective functions. This characteristic is especially interesting for distributed self-managing systems, where a central optimization instance might not necessarily be available or meaningful.

Optimizing Routing: Simulated annealing finds application in several shortest path problems approaches. Especially, the *Traveling Salesman Problem*, which is known to be a NP-complete problem, can be efficiently solved by SA algorithms.
However, an outstanding implementation for optimizing routing schemes can not be found in the literature. One of the reasons for this could be, that the SA algorithm draws upon the class of meta-heuristic algorithms. These algorithms are not necessarily delivering a 100% correct solution, but rather an approximate result. However, in the context of routing it is not desireable to make guesses about the routing path. In addition to this, better algorithms are known –and discussed later in this section– to solve routing optimization problems more efficiently and more accurately.

Optimizing Parameters and Configurations: In the work of Xi et al. [68], the researchers developed a smart hill climbing algorithm which was subsequently applied to an application server configuration. The general aim of their algorithm is to find a (near-)optimal solution

for a given application as a black-box optimization problem. As the researchers point out, the challenge of black-box optimization problems is to obtain the global optimal solution, while the objective function is usually high-dimension, non-linear, non-convex and multi-modal. Moreover, local optima are typically not the global optimal solution.
The developed algorithm from Xi et al. is subdivided into two main phases, a global search phase and a local search phase. The global search phase's aim is to cover the search space as broadly as possible, delivering a starting point for the subsequent local step. In contrast to the normal approach to start with an arbitrary sampling (random generated starting points for optimization), the researchers are employing Latin Hypercube Sampling (LHS), which generally provides high quality sampling coverage. Starting from a promising global solution, the local phase uses a gradient-based sampling method to search around its neighborhood for a even better solution.
In experiments basing on a running IBM WebSphere brokerage system the optimization process delivered near-optimal results after calculating only ten performance samples and provides thus very fast fine-tuning of system parameters. However, not only the performance of this approach should be emphasized. Rather it is remarkable, that the solution deals quite easily with the black-box problem, which is inherent in Peer-to-Peer systems.

3.3.3 Multi Agent Algorithms

Inspired by the research on real ant colonies, Dorigo et al. [19] developed the idea of *ant colony optimization* based on cooperative agents randomly populating a computer network. In the studies the researchers found out, that these kind of *multi agent algorithms* were capable of determining a solution to problems, that can be reduced to finding optimal paths in graphs. More interestingly, ant colony optimization algorithms have great advantages in environments, in which no knowledge about the environmental influences and topology is available.
In the literature, these algorithms are also denoted as population-based metaheuristics or *probabilistic multi agent algorithms*, in which every solution is represented by an ant moving in the search space.
The basic concept of ant colony optimization is based on the metaphor of ants seeking for food. Whenever the animals leave the anthill, they will begin to wander into a randomly chosen direction. While doing so, the insects are leaving trails of *pheromones* behind that will guide them back to their anthill. If an ant finds food, it will return to the anthill on the same successful path, which lead it to the food. On its journey back it will increase the pheromone density on the path, because it excretes another layer of pheromones on the same path. The pheromone density on a trail directly affects other ants' decision whether to follow it or not. As a consequence, trails with a high pheromone density are more likely to be visited/followed than other trails with a low pheromone denseness. However, the pheromones vaporize after some time. If all the food is collected, the successful path will no longer be renewed and it will disappear after a while. Now, the ants will head to new, random locations.
Ant colony optimization algorithms are especially of interest for self-managed planning and optimizing. This class of algorithms depends on problems which can be represented by a (directed) graph. If this kind of problem representation is available a population-based, iterative ant optimization technique can be applied. In the first iteration, a set of ants performs randomized walks through the graph. After reaching the goal, the optimality of the solution is evaluated

and denoted in the network graph by assigning a pheromone density to each edge on the successful path. This pheromone density changes the probability of walking on these trails for the next generations. As further ant generations populate the network graph it is possible that they deviate from these routes by choosing other edges, since their decision is partly randomized. However, after a sufficient amount of populations the solution will converge toward the optimum, denoted by the path with the most pheromone density.
Remarkable is, that ants are flagging the best solutions, while taking previously made solutions into account. This can be seen as a probabilistic multi agent learning by altering probability distribution functions.

In the following the applicability of multi agent algorithms to distributed optimization problems should be exemplary reviewed.

Optimizing Network Topology: In [59], Tamaki et al. propose a pheromone approach for an adaptive discovery of sensor-network topology. By focusing on three main objectives (anytime characteristics, adaptability, and robustness) for their research Tamaki and his team are capable of constructing network infrastructures without any manual assistance. In their research project, Tamaki emphazises *anytime characteristics* as their aim of delivering an optimal solution through the analysis of results in a finite period of time even if the network's size is changing. Secondly, *adaptability* denotes the approach's ability to function without any special tuning to accommodate all types of environments. Thirdly, *robustness* expresses characteristic of maintaining accurate analysis results even if the noise ratio (caused by e.g. sensing error) is increased.
In their biologically inspired approach the researchers propose a new methodology for constructing adjacent relations in sensor networks by using an ant colony optimization algorithm. This methodology is used for extracting human motion in a sensor network system, where sensor nodes do not know their adjacent neighbors. The adjacency likelihood is represented by the amount of pheromone, accumulated from the past sensor reaction data and the presumed distance between sensors (standard travel time). A positive feedback loop in each sensor is used for gradually increasing its adjacent relations by the accumulation mechanism. Interestingly, the pheromone communication approach removes failure information (i.e. sensor noise) on its own, since the aging noise-information pheromones evaporate until they are finally removed.
In the context of topology optimization, ant colony algorithms prove to be robust against and adaptable to dynamic changes in the environment. This fact makes them especially interesting for Peer-to-Peer systems, where no assumption about the node churn can be made.

Optimizing Routing: In the studies of Umlauft and Elmenreich [62], a Quality-of-Service aware ant routing algorithm for wireless mesh networks is supposed.
In their paper, the researchers enhance the ant routing pheromone concept with a color coding, which corresponds to a particular class of traffic. By doing so, it is possible to establish routing information, which treats packets according to the application-specific requirements on packet delay, delay jitter, and bandwidth.
Like in other ant routing algorithms, the ants mark their paths in the network by depositing pheromone at each node/path along the way, thereby marking a trail along which data traffic can be routed. However, in the colored pheromone approach the ants additionally mark the paths through the network by depositing pheromone with different colors depending on the suitability of the path for the corresponding traffic class. As shown in the Table 3.1 each traffic

class is mapped to a color. For example, a path with high bandwidth, low jitter, and low delay is suitable for traffic in the *conversational class* and therefore marked with pheromone with the color A.

Class	Name	Color	Critical Requirements	Uncritical Requirements
1	Conversational	A	Bandwidth, Jitter, Delay	
2	Streaming	B	Bandwidth	Jitter, Delay
3	Interactive (Web)	C	Delay	Jitter, (Bandwidth)
4	Background (Data)	D		Jitter, Delay, (Bandwidth)

Table 3.1: Mapping of Colors to Traffic Classes

With the help of the Colored Pheromone Ant Routing (CPANT), the researchers were able to enhance ant routing algorithms with a necessary differentiation of traffic classes. This differentiation is mandatory for the future success of Quality-of-Service and Service-Level-Agreement requiring applications, such as VoIP and Video-Conferencing.
Moreover, ant routing algorithms are capable of dealing with churn in dynamic networks. But, alike the current networks' characteristic that "bad" news travel slowly (are propagated slower than "better" messages) ant algorithms are deleting out-aged information with the times, rather than in the moment they happen. Nevertheless, it is remarkable, that ant routing algorithms are a field of research, which might be of interest for unstructured Peer-to-Peer networks, in which a sophisticated virtual network topology, like e.g. in Pastry [60], does not help for solving high-performance routing problems.

Optimizing Parameters and Configurations: Due to ant colony optimization's nature of finding optimal solutions for graph problems, it was explored as a means for computing optimal scheduling in distributed computing environments (multi-processor hardware, grid computing environments, etc.). For example, in the work of Lorpunmanee et al. [44] the researcher use ant colony optimization for dynamic job scheduling in grid environments.
The effective usage of computing resources in a grid data center is certainly a highly complex scientific problem. Even more, the problem's complexity is increased by the rapidly growing adaption of virtualization and sharing of large-scale resources. An effective utilization of computing resources (especially the scheduling of these resources) leads to an improved economic value and is therefore intensively supported by the business.
Lorpunmanee et al. present a general framework of grid scheduling using dynamic information and ant colony optimization to improve the decision of scheduling.
Generally, submitted (work) jobs are not equally distributed, leading in burst and idle time frames. Hence, the scheduling is much more difficult than dealing with the static load of a computing system. The supposed ant colony optimization algorithm solves this problem by first considering the requirements of each job independently, then establishing correlation to the other waiting jobs and then distributing the workload so that only one job is scheduled at any unit per time slot.
In their empirical studies the researchers revealed that ant scheduling algorithms can outperform current available scheduling algorithms. The results presented an average 17% idle time reduction in comparision to the other scheduler algorithms, such as First Come First Served (FCFS), Minimum Time Earliest Due Date (MTEDD), Minimum Time Earliest Release Date

(MTERD).
Autonomic systems are supposed to distribute their work to all available and appropriate autonomic elements (equally-principle). Especially for autonomic systems, an effective scheduling is thus not only important, but has to be seen as a mandatory characteristic. Furthermore, the dynamic workload variation will be one of the upcoming scientific challenges, which will arise in the self-configuration context. Ant colony optimization algorithms proved their applicability and robustness in these cases. However, it remains to be seen, if these computational-intensive algorithms will become the method-of-choice for these kind of problems.

3.3.4 Tabu Search

Tabu search (TS) is based on the idea of enhancing the search for global optima with a historical knowledge. The word “tabu” stems from Polynesia and describes a sacred place or object. Things that are tabu must be left alone and may not be visited or touched. TS extends the classic iterative hill climbing search by this concept –it marks an object as tabu, whenever it should not be investigated again for a certain amount of time. Hence, tabu solutions must not be visited again and the optimization process is less likely to get stuck on a local optimum.
Tabu search is mostly implemented with a finite-length First-In-First-Out list of candidates, which stores all tested candidates for a certain amount of time. If a newly created solution can already be found in the list, it will be rejected. After the list has reached its maximum capacity, the first added solution candidate is deleted and the new solution candidate is marked as a tabu.
More complex approaches store specific properties of the individuals instead of the solution candidates themselves in the list. This will not only lead to more complicated algorithms, but may also reject new solutions, which actually are very good. Therefore, certain criteria can be defined to override the tabu list and allow certain individuals.
In the following the applicability of evolutionary algorithms to distributed optimization problems should be exemplary reviewed.

Optimizing Network Topology: In [14], the researchers use tabu search to search for the best coverage for fixed cellular networks. The goal of their research is to find an optimal division of the geographic area into disjoint and hierarchically organized, controlled by one base station controller (BSC).
In a typical cellular network, the area of coverage is geographically divided into network cells. Each cell is equipped with a base transceiver station (BTS) that contains the radio transceivers, providing the radio interface for the devices. One or more BTSs are connected to a base station controller (BSC), that provides a number of functions related to resource- and mobility management, as well as operation and maintenance interfaces for the overall radio network.
In the next hierarchical step, one or more BSCs are connected to a mobile switching center (MSC) that controls call setup and call routing, next to many other functions provided by a conventional communications switch. An MSC can be connected to other networks such as the public switched telephone network (PSTN) in order to provide a larger coverage.
Apparently, for cellular network operators finding an optimal solution is a trade-off between two extremely important facts: First, remaining low CAPEX and OPEX (acquisition- and maintenance costs: acquire, install and maintain the facilities (BTS, BSC, MSC, etc.)), while secondly

achieving the best possible satisfaction rate for the customer (best coverage in all areas).
While several researchers, such as in [17], [47], and [6], found solutions for parts of this problem by e.g. solving the knapsack- and network flow subproblems, a global approach to the cellular network design problem was missing. However, Chamberland et. al proposed an algorithm that deals jointly with selecting the location of the BSCs and MSCs, selecting the BSC and MSC types, designing the network topology, and choosing the link types.
Undoubtedly, in next generation networks the border between fixed and wireless network components will more and more blur and a larger percentage of fixed devices will get a high-performance wireless access channel. Even more, in the future static cell-based base stations, so called "FemtoCells", will enhance and cooperate with ad-hoc networking approaches to deliver even "more bars in more places".

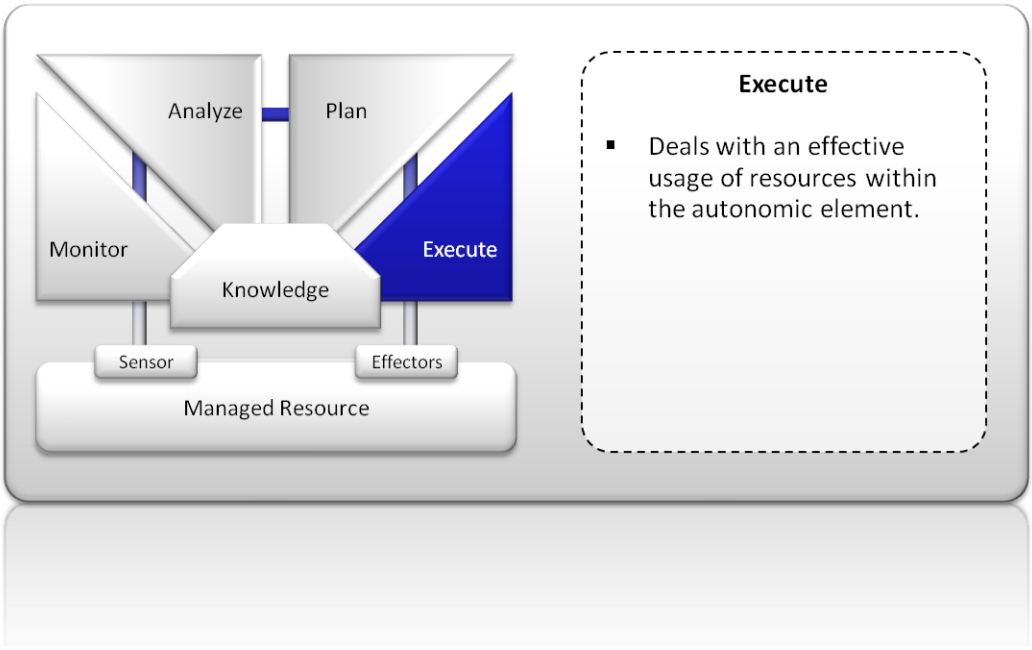
Optimizing Routing: In [25], Ghaboosi and Haghighat propose an algorithm to solve the "bandwidth-delay constrained least-cost multicast routing problem" with Tabu Search.
In their paper the researchers are describing the necessity of QoS-based multicast routing for various real-time multimedia applications in high-speed networks and highlight the two most important QoS constraints, bandwidth and end-to-end delay.
In order to solve the problem, the researchers are formulating a *constraint steiner tree* as the mathematical structure behind the problem, which is a well-known NP-complete problem. In order to achieve simultaneous delivery of information to a group of destinations, a multicast tree structure is required. One of the main goals of this tree structure is to minimize the communication resources used in multicast sessions. The problem of determining a multicast tree with the least costs is known as the *minimum Steiner tree problem*.
In their paper, Ghaboosi and Haghighat propose a tabu search based algorithm, which proves to outperform other existing algorithms. It produces an optimal solution rapidly, after only a few iterations in contrast of the existing algorithms, which require many iterations to achieve a similar outcome.
An effective multicast routing is tried to achieve by novel approaches for Peer-to-Peer TV (see also Joost [36] and Zattoo [36]). Here, it is also desirable to find minimal load situations, while distributing the channel signal to all interested nodes. In the autonomic system context multicast will be necessary for distributing effectively the decisions which are made throughout the planning stage.

Optimizing Parameters and Configurations: The similarity of tabu search and simulated annealing as both enhanced hill-climbing algorithms, implicates the applicability to the grid computing workload distribution (load balancing) problem. Accordingly, researchers, such as Subrata et al. [58] are comparing tabu search algorithms to other alternatives for efficient workload distribution in grid computing environments.
The empirical comparison in Subrata's paper [58] reveals that *tabu search* algorithms in general perform well for these kind of problems. In order to analyze the different load balancing algorithms (tabu search, genetic algorithms, min-min, max-min, and sufferage) the researchers presented a detailed and realistic evaluation model. It comprises randomly created networks in the range of 40 to 60 nodes, each with a relative processing power, a communication link latency model, a bandwidth link variation, and an associated background workload model. The virtual networks are tested under a realistic task distribution model (varying completion times and complexity).

The experiments' results revealed that tabu search reaches the least deviation (1.48%) to the optimum solution. The second best deviation was reached by genetic algorithms (4.37%), which outperformed Min-min, Max-min and Sufferage load balancing algorithms significantly.

As already stated in Section 3.3.3 "Multi Agent Algorithms", autonomic systems must distribute their task onto the appropriate autonomic elements. Each of these workers has to deal with a dynamic workload situation, which might also lead to subcontracting other helper elements in the case of computational intensive tasks. Accordingly, an effective scheduling and workload distribution has direct impact on the system's overall performance.

3.4 Execute



The *executing* building block of the autonomic system's control loop deals with an effective usage of resources within the autonomic element itself. However, since every autonomic element is part of a larger autonomic system construct, system-wide considerations must be taken into consideration, as well. Hence, for Peer-to-Peer systems' nodes it is necessary to act on the one hand selfishly –with the aim to achieve its own tasks as fast/resource-conserving as possible– but on the other hand with an active contribution in mind, e.g. as a provider for vital services (message relay router, part of a DHT, etc.).
In the control loop's sequence the analyze component retrieves an exceptional circumstance (malfunction, service response-time drop, unnecessary over-provisioning, etc.) and decides reactively or proactively (see: 3.3 "Plan"), the counteractive measurements, which are invoked in the execution step. As an example, the monitoring component could trace the response time of a distributed service. While monitoring the service's responsible peers, a drop in this response time may occur. Analyzing that the response time exceeds a policy-given threshold, the planning component decides to invoke counteractive measurements that effect the service's response time. In the example, the planning component could decide to replicate the service onto other peers, leading to a more balanced workload on the service layer, but stressing more peers' computing resources.

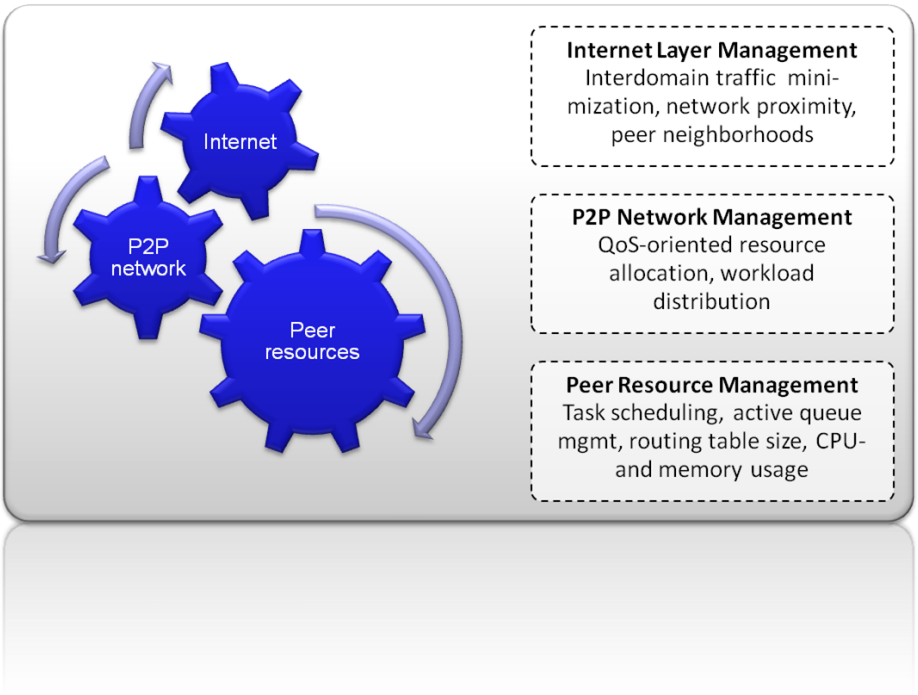


Figure 3.8: Resource usage control layers

As Figure 3.8 denotes autonomic systems' resource management and efficiency control can be invoked on three jointly correlating layers: (1) Peer resource management layer, (2) Peer-to-Peer network management layer, and (3) Internet management layer.

3.4.1 Peer Resource Management Layer

The peer resource management layer comprises device-specific resources, such as computing power, memory and bandwidth. Alike "non-autonomic" Peer-to-Peer or grid-computing systems, scheduling and queuing adjustments can improve the performance in this case.
Emphasizing the point of Quality-of-Service (QoS) assurance, Graffi et al. introduce HiPNOS.KOM [29], a mechanism for providing *overlay bandwidth management* in an arbitrary overlay network. In their work, an active and dynamic scheduling and queue management mechanism is introduced, which is capable of assuring the provisioning of latency- and failure-sensitive applications, such as emergency calls over a Peer-to-Peer network.
It was shown, that an automated improvement of resource consumptions on the node level is possible, as long as the optimization goal is known to every peer. In the special case of an emergency call, message priorities can be set per default, but an autonomic reevaluation of message priorities is only possible if the current network situation is known to every peer in the network. In the framework of autonomic computing a system-wide monitoring and evaluation will be accomplished by the monitoring component (see: Section 3.1 "Monitoring").
Another example of resource efficiency improvement for Peer-to-Peer systems is the work of Lo et al. [43], who are using the peers' idle time to calculate an optimal scheduling for inciden-

tal tasks. The so called “Wave Scheduler” exploits large blocks of idle time at night to provide higher QoS for deadline-driven tasks. Taking advantage of a geographic-based overlay, which includes peer separation into timezones, the researchers are capable of improving the workload distribution to a near optimum.
Furthermore, Eger and Killat propose in [21] a distributed resource allocation algorithm in which peers control their own service rates. The algorithm is based on the congestion pricing principle known from IP networks. Every time a service is requested, a price bid for the service allocation has to be attached. Based on the offered price(-s), a service provider allocates its resources to the highest bidding customer. In that way, a *spot-market* for rare service capacities emerges, which takes fairness problems (free-riding) into account.

3.4.2 Peer-to-Peer Network Management Layer

In most cases, a task can be accomplished by more than one peer in a Peer-to-Peer network. Even more, due to the peer’s heterogeneity regarding its network access capabilities, computational power, and fail-safeness, considerations how to distribute the workload are likely to improve the system-wide performance.
In [27], a QoS-oriented resource allocation for overlays is introduced. The mechanism abstracts from specific overlay characteristics, yielding in an applicability for every structured Peer-to-Peer network. The technique enables to optimize the service provisioning for three important metrics: fair workload distribution, node heterogeneity, and network uptime.
The peers, who are maintaining the Distributed Hash Table (DHT), are also responsible for a list, which maps service providers to their services. Each time queries for a certain service is issued to the *service brokers*, an extensible allocation function evaluates the optimal workload distribution and delivers the most appropriate peer. This allocation function is not fixed to one hard-coded optimization goal, but can be dynamically changed as a response to the network’s dynamism.
As a further outcome of the project the researchers are mentioning, that the overarching issue shifts from “How can QoS-assurance be accomplished in a Peer-to-Peer network?” to “Which criteria must be optimized –on which level– in order to assure system-wide QoS and SLA?”. This issue is even more complex since each peer acts as a single and partly selfish entity, deciding on its own which internal parameters shall be adjusted or which task shall be executed. All these decisions are influencing not only short-term but also long-term service level agreements.
In contrast to the structured Peer-to-Peer network approach of Graffi et al., the research work of Wang et al. [65] focuses unstructured Peer-to-Peer infrastructures. The researchers propose to position related data chunks into Peer-to-Peer neighborhood groups, leading to a *popularity-aware prefetch caching*.
Resulting from the fact, that most large-scale distributed applications are dependent on *range selection queries* for providing their services, the researchers are emphasizing, that the retrieval of poorly-replicated data items accounts for the dominant part of the processing costs. Thus, a prefetch-based approach can effectively facilitate the caching of poorly-replicated data items that are potentially requested in subsequent range queries.
In the experiments basing on their distribution mechanism, the researchers are showing that the overall range query processing costs are decreasing significantly, even under various query load settings.

Unstructured Peer-to-Peer systems are a relative new research topic. However, it remains to be seen whether this Peer-to-Peer network infrastructure can be integrated into the autonomic computing vision, or it will co-exist next to the other approaches. Nevertheless, the advantages of structured Peer-to-Peer infrastructures (ordered topology, hierarchies, DHT, etc.) offer better contact points for effective and coordinated resource allocation mechanisms.

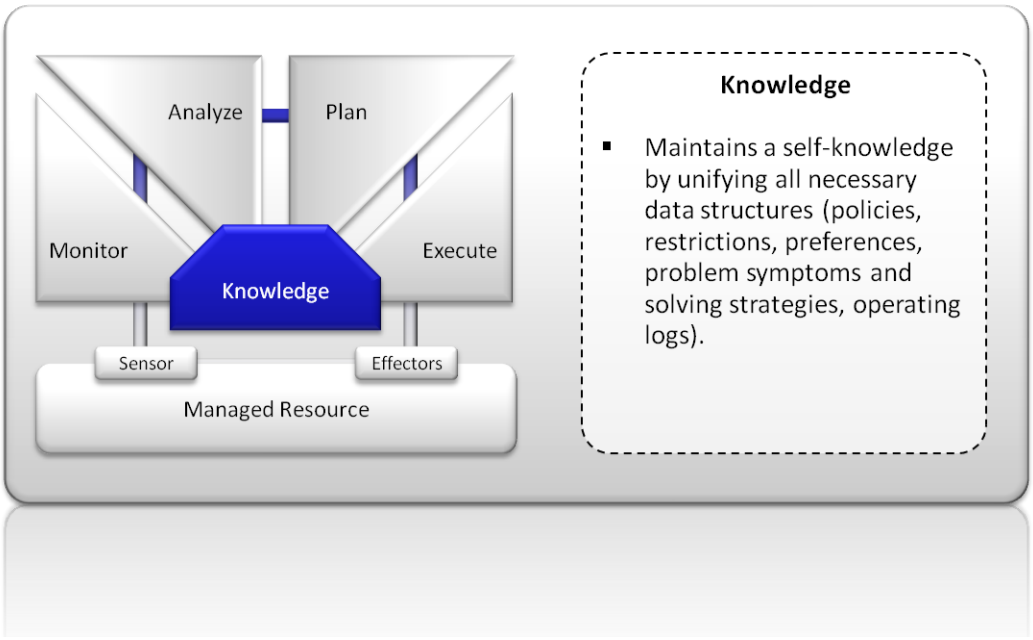
3.4.3 Internet Management Layer

On the Internet management layer execution parameters, for example the choice of peer neighborhoods can influence the network's effectiveness, performance, and also economical aspects. However, in general an optimal network topology for Peer-to-Peer networks does not exist, since the performance function is tied to the provided services' requirements. Obviously, a VoIP Peer-to-Peer network can have a significantly different optimal structure compared to content delivery network (CDN). The choice of network topology configurations can be influenced by factors such as network position proximity (i.e. according to a geographic metric), the network delay, same-interest-neighborhoods, the bandwidth to the neighbors or even the Internet Service Provider (ISP) affiliation.
As an example for the choice of neighborhoods in a Peer-to-Peer network, Zhang et al. [69] are proposing a routing mechanism, which embraces information about the next hops' capabilities, features, reputation, and even affiliations of administrative domains into the routing decision. To achieve their goal, the researchers introduce *Grouped Tapestry (GTap)*, a novel Tapestry-based [60] DHT, that supports the organization into groups of participating nodes. With the help of this routing mechanism, the researchers are able to introduce performance, availability, security and path-constraint considerations on the Internet layer.
From an economical point of view, the Internet layer optimization is also important for ISPs. Their optimization goal is to minimize interdomain traffic to competitors. Unfortunately, Peer-to-Peer networks have the biggest share in the amount of cross-domain traffic. To tackle this problem Bindal et al. [8] propose a *biased neighbor selection* algorithm, which enables BitTorrent [9] peers to choose the majority of its neighbors within the same ISP domain. As a result of the traffic locality, ISPs are not forced anymore to throttle the BitTorrent traffic to control their costs.
As another example of Internet layer effectiveness for execution, location-based overlay like [37] and [41] have to be mentioned. Here, the optimization criteria is to find a peer within a pre-defined radius, which obviously can also lead to reduced interdomain traffic or low latency searches (e.g. in mobile ad-hoc networks).
In general, location-based overlays are based on a hierarchical tree-based Peer-to-Peer overlay structure, enabling location-based operations in an efficient (logarithmic) scale.

All in all, the execution component has to assure that invoked tasks are completed in comprehension with all restrictions and user-expectations. Apparently, local execution considerations are not alone sufficient, but important. Especially in the scope of Peer-to-Peer networks or grid computing the next layers' parameters must be examined and adjusted in order to reach efficient short and long-term SLA and QoS.
However, it is inherent that mechanisms for QoS and SLA assurance are available, but a system-wide applicability in autonomic computing systems, especially on the distributed infrastructure

foundation, is far out of reach. Up to now, a peer's local decisions do not take system-wide execution goals into account, because an all-embracing overview about the current network status is –most often– not available. However, for the autonomic systems' self-optimization, -healing, and -configuration goals not only local, but also far-reaching global QoS and SLA considerations are necessary.

3.5 Knowledge



The autonomic system's knowledge component is a central entity, helping to improve all components' decisions. It unifies policies, restrictions, preferences, problem symptoms and -solving strategies, operating logs, and all other knowledge-building data structures. With the help of the knowledge component, autonomic managers are able to improve their decisions basing on a history of earlier successful operations. Accordingly, it provides the means for accessing the *self-knowledge*.

Without a system's knowledge about its internal structure and functioning, other *self-* properties*, especially self-healing, -configuration, and -management, are not realizable. Only the self-knowledge feature allows autonomic systems to decide, that a service has failed or is about to fail. More specifically, self-knowledge enhances the system with an internal consciousness about what is expected as an operational output. In other words, it allows to determine if a current behavior is consistent or expected with respect to the environment.

Furthermore, the knowledge component is not envisioned as a static information source, but also *learns* through every updating, appending, or deleting step. As a consequence, new environmental conditions may lead to new behavior (see: Section 3.3 "Plan"), which is learned (or stored) if it proves successful. Each successful change, leads to an self-knowledge update and lets thus the system evolve continuously with the addition of new components, updates, and

usage scenarios.
In summary, the knowledge component is capable of mirroring a systems' limitations and –more importantly– capabilities.

As stated beforehand, the knowledge component helps every other component of the MAPE loop to improve their decisions by adding self-knowledge about the systems' environmental influences. Consequently, a separate view about the influent information must be established for every part of the MAPE loop.

3.5.1 Self-Monitoring Information

An autonomic system's monitoring process collects data and takes measurements at selected key points. It is used to probe the vital signals and data continuously or periodically.
Due to this, Cofino et al. propose in [16] a *System Task Library (STL)* containing test scenarios for monitoring the system's health. A STL continuously runs one or more specialized system tasks. Cofino et al. propose an organization of tasks into categories of functions: (1) self-identification tasks (comprised components and their capabilities), (2) self-diagnostic tasks (benchmarks and expected behaviors), (3) self-monitoring tasks (to be monitored touch points and their frequency of monitoring).
The self-monitoring module aims to capture vital, as well as operational information from the system's sensors. Thus, it requires a knowledge about what to monitor. In the ideal case a self-monitoring component delivers data about everything, that is crucial to the system's stability and liveliness.
Basing on the *continuous* monitoring of information, a knowledge component has to deal with a huge and ever increasing amount of structured and unstructured data. To limit the required storage capacity and filter unnecessary information, various points have to be considered:

1. A service delivered by means of an autonomic system must have predefined and self-describing interfaces following a standard definition (see: Section 4.1 "Standards").
2. The self-monitoring knowledge must be based on *dynamic information sampling*, which can be adjusted to load (CPU, bandwidth), storage space, or system health (e.g. reduce sampling rate during attack).
3. An autonomic system must be subclustered into (disjoint) structural levels at which they operate. Thus, sensor data can be layered hierarchically, which makes it possible to monitor every aspect of autonomic system on each particular layer (module, component, object, data structure or variable).

To achieve the above mentioned goals, the system has to establish an effective and ideally non-intrusive instrumentation, that reduces or eliminates the extra overhead entailed by self-monitoring. Some techniques are already mentioned in Section 3.1 "Monitoring". On the conceptual level, *event notification* mechanisms have to be seen as most promising or even the key concept for large system monitoring endeavors. They combine the developers' knowledge at development time and an effective practicability even in large scale networks. A proof-of-concept

for Internet-scale networks is exemplary given in [53] and [13].
Furthermore, system information should be presented in a self-explanatory machine- and human-readable representation. This data is usually collected long enough to run statistical analysis. Resulting from this fact, time becomes an important factor, which has to be considered closely. Whenever a time interval for the to-be-monitored component is too small a performance-drop is indispensable, but if the interval is chosen too large other side effects may occur. Due to this, dynamical sampling of information, e.g like Wang's and Kankanhalli's experience based sampling technique [64], becomes a performance-boosting requirement.
All in all, self-monitored data delivers *vital information* about the system's characteristics (such as CPU and memory usage, response times, types and numbers of threads and processes). A well-formatted and self-explanatory representation in a structured language, derived at key points, and at the right time allows to determine the health of autonomic systems without human intervention.

3.5.2 Problem Detection and Self-Diagnosis Knowledge

In "Towards Knowledge Management In Autonomic Systems" [16], the researchers propose a modularization into several knowledge components, which can easily be mapped onto IBM's Monitor-Analyze-Plan-Execute (MAPE) loop subclusters as indicated in Figure 3.9.

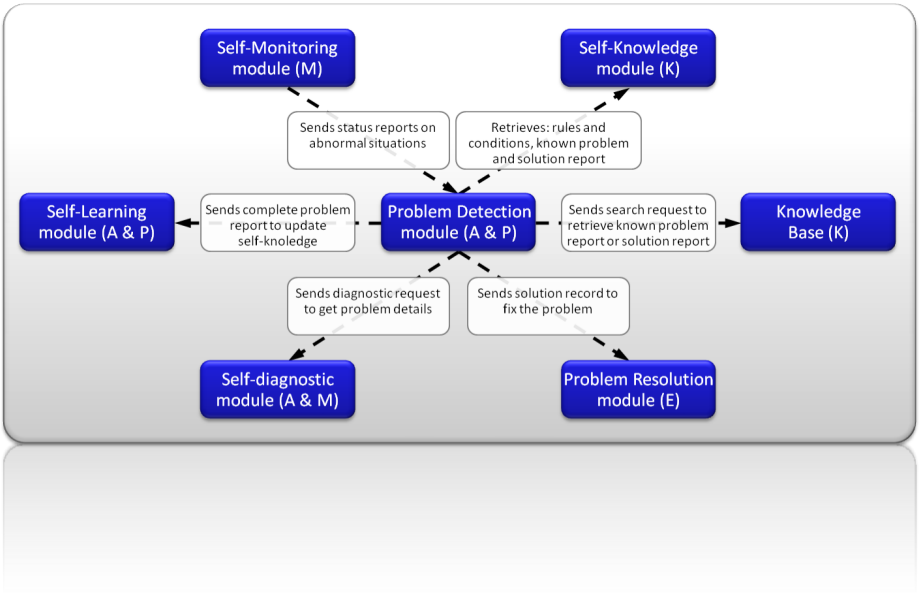


Figure 3.9: Knowledge Flow in Autonomic Systems

In accordance to the previous chapters, "M" indicates the MAPE loop's monitoring component (see: Section 3.1 "Monitoring"), "A" the proximity to the analyze component (see: Section 3.2 "Analyze"), "P" the planning component (see: Section 3.3 "Plan"), and "E" the execute component (see: Section 3.4 "Execute").
Figure 3.9 draws an image about the knowledge and information flow within an autonomic el-

ement. As a central unit the *problem detection module* analyzes the input data, which is derived from the *self-monitoring module*. The problem detection module's task is to collect and analyze the input data in order to diagnose an abnormal system response or failure. Since the problem description granularity may be not sufficient, a *problem diagnosis module* may be invoked to retrieve a more detailed report of the problem. In this process step the "problem root cause" analysis can also be invoked to retrieve not only the failed component, but also the stack trace of invokers, who could have been responsible for the problem.
After collecting all necessary problem descriptors and environmental influences, the problem detection module issues a *problem detection request*. The standardized data model and its pre-classification into problem-subsegments helps the planning and knowledge components to retrieve default/policy-defined course of actions or stored successful *solution reports* from earlier similar problems. If no complete solution record, associated with the given problem can be found, a pending problem needs to be handled by a human expert.
Once the problem solution is found, this record is passed to the *problem resolution module* to initiate the *problem resolution cycle*.
As a prototypic implementation of the above mentioned knowledge flow approach, IBM presented *DataCase*, a knowledge management system, used for information retrieval and hardware problem diagnosis in IBM call centers and customer service centers worldwide.
In this field of application one centralized knowledge pool for all self-configuring/ self-healing products of the manufacturer's portfolio exists. Every time a component detects a malfunction, it sends a symptom report to the provider's problem management system. Here, an problem record will be filed and forwarded to the DataCase server for analyzing. The DataCase server diagnosis the problem by taking advantage of an aggregation of counteractive measures (knowledge/experience pool) and enhances the problem report with an according action plan. After executing the contained sequence of actions on the failed component, the customer's system acknowledges the repair and resumes its normal execution process.

3.5.3 Problem Resolution Knowledge

The problem resolution module's purpose is to execute the *problem resolution cycle*. For this purpose, the component is invoked with problem solving instructions, which are executed on the associated module. Subsequently, the problem resolution module initiates a diagnostic process to verify that the applied solution has fixed the problem.
For instance, after detecting a service failure a deployed policy could initiate a service re-install within the autonomic element. Moreover, it would be possible that the self-knowledge history automatically retrieves a correlation between the monitored malfunction and an earlier incident with the same erroneous results. Accordingly, the solution record is stored within the self-knowledge, containing the execution steps to download a new version from a repository location, uninstall the old service version and re-install the new version from the previously downloaded package. After the procedure is completed, the problem resolution module is forced to invoke the self-diagnostic module to verify the solution's success. For this purpose, it may be necessary to run (several) benchmarks and function tests with predefined maintenance data.
All test results will be recorded in an action report, which will be led back to the knowledge component. More specifically, as Figure 3.10 depicts, a learning mechanism is invoked to enhance or update the self-knowledge.

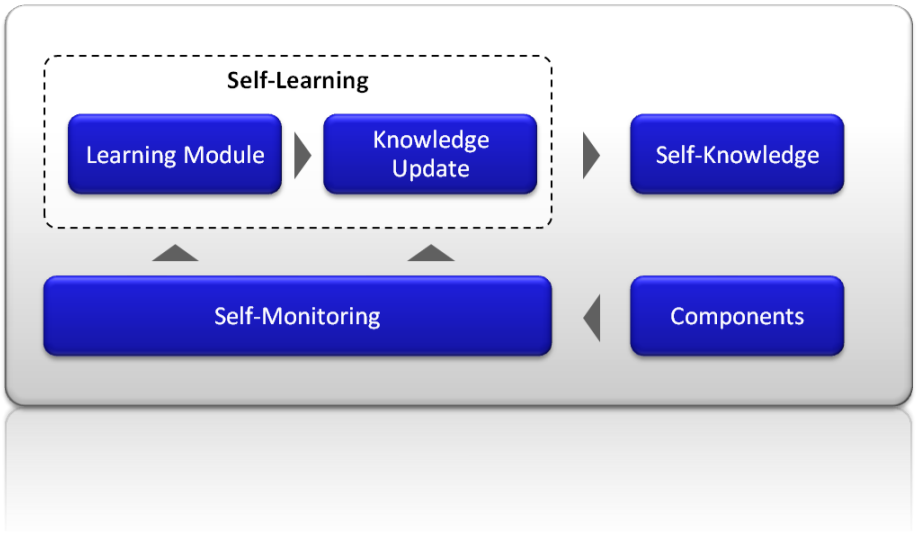


Figure 3.10: Self-learning and Component Interaction

The self-learning process, illustrated in Figure 3.10, is inherent in every autonomic and non-autonomic (human intervened) decision process. The human-connotated term *learning* can here be applied, because the system can improve its performance with the help of experience. And experience is best understood as the knowledge gained from the analysis of several tasks performed during a period of time [16].
In the context of autonomic systems, this self-learning process updates and improves the knowledge about the system's internal and external characteristics.

In summary, the knowledge component provides the *self-knowledge* information, which distinguishes autonomic systems from non- or semi-autonomic systems. Moreover, it builds the foundation on which self-healing, self-managing, self-configuration and self-optimization can be based.
It contains essential system information, which is at least a list of components that the system is made of, such as software and hardware components and their associated capabilities. In addition to this fundamental information, the knowledge component also contains sets of rules, conditions, and operational patterns expressed as functions or scripts. These rules are derived from the functional specifications or the system's design goals. Furthermore, the self-knowledge component has the ability to represent this information with the help of standardized description formats.
However, the sum of knowledge is not omnipresent from the point of the initial setup, but acquired during the life cycle of the system. As a matter of fact, self-knowledge is one of the most important enhancements, leading to autonomics and self-controlled government of IT systems.

4 Autonomic Systems

4.1 Standards for Autonomic Computing

This section will highlight standards for autonomic computing, which will be necessary for implementing the autonomic computing vision.
Resulting from the given autonomic systems' local structure –autonomic elements will be independent although integrated into a group/network of autonomic elements– and the desired global structure –multiple autonomic elements will communicate amongst each other simultaneously and concurrently– it is implicit that autonomic managers will have to manage a myriad of resources from a multiplicity of vendors.
Already today, distributed computing systems are comprising a large number of heterogeneous interconnected devices, that in turn provide numerous and increasingly complex services to their users at a global scale.

4.1.1 Service-Oriented Architecture (SOA)

In today's Internet the concept of building applications and services using the principles of service orientation is gaining more and more momentum. The overarching goal of this technique is to establish interoperability amongst implementations from multiple vendors. The *Organization for the Advancement of Structured Information Standards (OASIS)* defines the concept of the so called *Service-Oriented Architecture (SOA)* as

> *"... a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM (SOA Reference Manual) specification bases its definition of SOA around the concept of "needs and capabilities", where SOA provides a mechanism for matching needs of service consumers with capabilities provided by service providers." [50]*

Due to SOA's abstractions and the accompanied descriptive power it gained significant following in the industry and has to be seen as a quasi-standard for the definition of interoperable Web-service based standards. Intel and IBM state in [34] and [32], that it is both pragmatic and desirable to use the same or interoperable Web-service based standards for both domains (Autonomic computing and Internet services). The significant investments, sophisticated implementations, established tools, and a broad industry support should be the foundation for service-oriented architectures in autonomic environments.

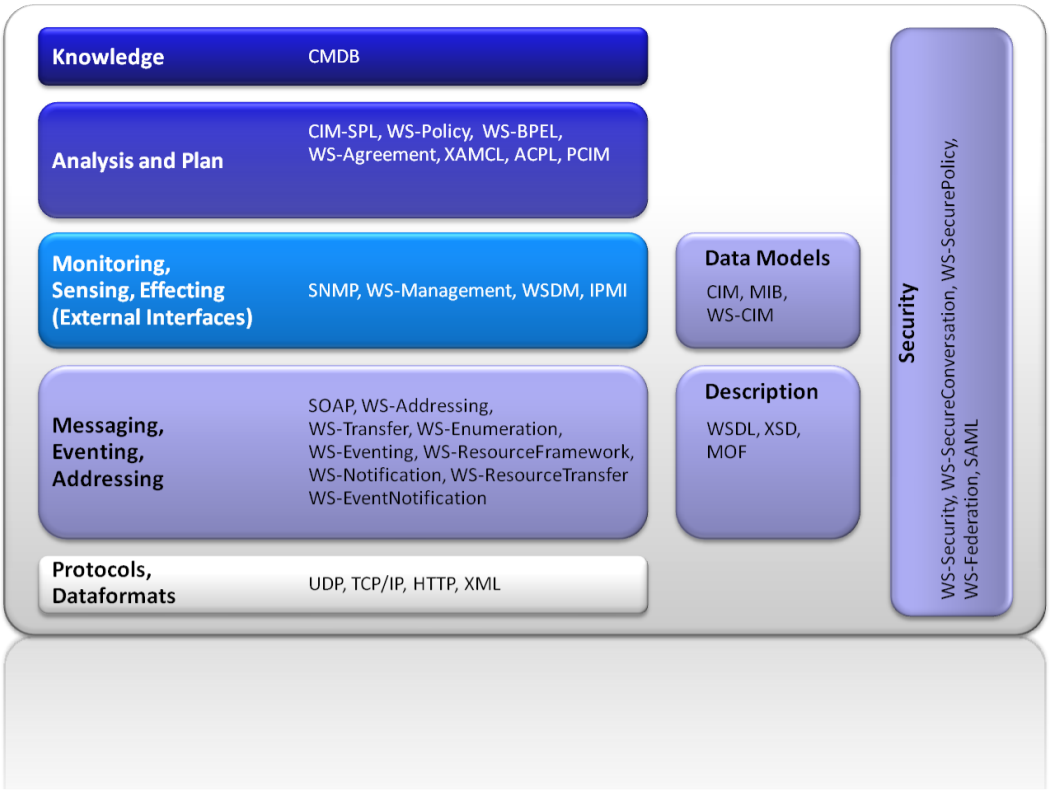


Figure 4.1: Overview Standards for Autonomic Computing

As Figure 4.1 depicts a stack of standards will be necessary to establish accessibility across the various hardware and software layers. However, the mentioned enumeration of standards should not to be seen as an exhaustive list, but as a selection of the most promising standards in the sphere.

Protocols and Data Formats

As the bottom-most layer of the stack, autonomic system are basing on proven and established protocols and data formats for the exchange of all messages. This section describes the common protocols and data formats.

- *TCP/IP and UDP:* Basing on the packet-switched Internet Protocol (IP) (current version IPv4 with 2^{32} addresses and the future version IPv6 with 2^{128} addresses) the connectionless UDP and the connection-oriented TCP protocol will be used for end-to-end communication.
- *Hypertext Transfer Protocol (HTTP):* The W3C standardized HTTP protocol gives a set of rules for exchanging text, graphic images, sound, video, and other multimedia data. It is the most popular request/response protocol and comprehensively used in the World Wide Web.

- *Extensible Markup Language (XML):* The W3C standard XML enhances unstructured plain text with meaningful structure. It introduces a general-purpose format (syntax and semantics) for structuring data with the help of so called *tags*.

Messaging Eventing and Addressing

This section briefly mentions massaging and addressing standards. Usually, the abbreviation "WS" is used for "Web-Services", unless noted otherwise.

- *Simple Object Access Protocol (SOAP):* A W3C standardized protocol for exchanging XML-based messages over (secure) HTTP, abstracting away from the underlying hardware and software requirements.
- *WS-Addressing:* A W3C recommendation for defining a transport neutral mechanism to address Web services and -messages.
- *WS-Transfer:* One of the three main parts of Distributed Management Task Force (DMFT) "WS-Management". WS-Transfer defines a XML-representation of a WS-Addressing compliant resource, as well as the basic management operations on the managed entities (create, update, read, delete).
- *WS-Enumeration:* The second main part of DMFT's "WS-Management". WS-Enumeration defines generic operations to enumerate entities. The protocol allows to iterate over a sequence of XML elements for traversing logs, message queues, or other linear information models.
- *WS-Eventing:* The third main part of DMFT's "WS-Management". WS-Eventing establishes a publish/subscribe service for event notifications.
- *WS-ResourceFramework:* One of the two main parts of Organization for the Advancement of Structured Information Standards' (OASIS) "WSDM". This specification defines a WS-Resource, which describes the relationship between a Web service and a resource in the WS-ResourceFramework.
- *WS-Notification:* The second main part of OASIS's "WSDM". An event-driven, or notification-based, interaction pattern for inter-object communications between WS-ResourceFramework compliant resources.
- *WS-ResourceTransfer:* A not yet released W3C standard. Unifies DMFT's "WS-Transfer" and OASIS's "WS-ResourceFramework". WS-ResourceTransfer will provide basic create, update, read, delete functionalities and an uniform definition of a XML-compliant resource representations. In addition to that it provides support for operations on partial object representations.
- *WS-EventNotification:* A not yet released W3C standard. Unifies DMFT's "WS-Eventing" and OASIS's "WS-Notification". WS-EventNotification will provide a publish/subscribe/deliver mechanism for events in an Internet-scale environment. In addition, the new specification will comprise event filtering, policy-based subscriptions and the treating of subscriptions themselves as WS-ResourceTransfer compliant resources with all its modification operations.

The two specifications from Distributed Management Task Force (DMFT) "WS-Management" and Organization for the Advancement of Structured Information Standards' (OASIS) "WSDM" have shown a similar scope and a great overlap in their architectural approaches. Consequently, the specification's authors have started working together on a proposition to unify their results in two new specifications, called "WS-ResourceTransfer" and "WS-EventNotification".

Resource Sensing, Monitoring and Effecting

This section gives an insight into standards, which enables the monitoring and controlling of resources through standardized external interfaces.

- *Simple Network Management Protocol (SNMP):* IETF protocol SNMP is designed to give administrators the ability to manage a computer network remotely. By accessing SNMP agents operating on the endpoints the administrator can adjust values and monitor network events.

- *WS-Management:* Distributed Management Task Force's (DMFT) approach for a general SOAP-based protocol for managing systems, such as PCs, servers, devices, Web services and applications, and other manageable entities. The first version of the specification was introduced in June 2005. Microsoft's current operation systems Windows Server 2008 and Windows Vista is shipped with "Windows Remote Management", which is their implementation of the WS-Management standard.
  Various Linux versions (e.g. SUSE Linux Enterprise) are integrating OS support, too. Wiseman and SOA4D are open source Java and C implementations of the WS-Management stack. OpenWSMan offers Apache HTTP server support and even Ruby bindings.

- *Web Services Distributed Management (WSDM):* Advancement of Structured Information Standards' (OASIS) Web Service Distributed Management 1.0 specification was introduced in March 2005. Its purpose is to define how the management interface of any resource can be accessed via Web service protocols (Management using Web services) and how the management interface of any Web service can be accessed (Management of Web services). WSDM is highly favored by IBM. The manufacturer stated that more than 30 products have a support for WSDM (i.e. WebSphere Application Server, Tivoli Enterprise Control, Virtualization Engine).

- *Intelligent Platform Management Interface (IPMI):* Intel's IPMI standard is enhanced with functions-support since its first version in 1998. The current version v2.0 was introduced in February 2004 and is focused on monitoring a server's physical characteristic (e.g. thermal sensors, fans, voltage) and workload.
  Intel states a number of 162 adopters, comprising Dell, HP, NEC Cooperation.

- *Software monitoring:* The monitoring of software components is a fragmented space due to the large number of programming languages and their accompanied runtime environments. However, the open source examples, such as Wiseman and OpenWSMan, show that the open source community willingly supports promising standards. Mentionable are in this field also platform dependent approaches, such as Java's Management Extensions (JMX) and the System.Management namespace in the .NET framework.

Description Standards

The highly dynamic structure of autonomic elements can only be facilitated, if a binding of services can be achieved while the system is operating. Inevitably, every information exchange must be described in a machine-readable manner.
The following section describes the common standards for providing this capability:

- *Web Services Description Language (WSDL):* WSDL is a W3C standardized XML-based language for describing network services as a set of endpoints. These endpoints are operating on messages containing either document-oriented or procedure-oriented information. A WSDL file provides information on the purpose of the service, where it resides, and how it can be invoked.
- *XML-Schema (XSD):* The also W3C standardized XML-Schema exposes the possibility to define structure, content, and semantics in conjunction with the possibility to validate the content of a XML file against the recommended document structure (schema).
- *Managed Object Format (MOF):* DMFT defines a language called Managed Object Format (MOF) for the description of its *Common Information Model (CIM)*. It provides ways to describe object-oriented class and instance definitions in machine- and human readable forms. Alike XSD and WSDL, it offers qualifiers, meta-data about classes, properties, methods, and more.

Data Models

Nearly every IT system needs a way to represent data in a processable manner. In order for autonomic managers to monitor and control elements, they require a well-defined and structured representation of the objects. Even more, the data model is a building block for one of the most essential parts of autonomic systems: *The knowledge base*.

- *Common Information Model (CIM):* CIM v1.0 was defined and published by the DMTF. The most recent version 2.2 was released in November 2008. CIM is a common data model of an implementation independent schema for describing overall management information in a network/enterprise environment (software and hardware).
  CIM consists of a specification and a schema part. The specification defines the details for integration with other management models (i.e. SNMP's Management Information Base or the DMTF's Management Information Format, while the schema provides the actual model descriptions.
- *Web Services Common Information Model (WS-CIM):* This DMTF standard defines a translation from the MOF format for CIM into XML-Schema. It allows CIM model definitions to be transported over Web Service management protocols, such as WS-Management.
- *Management Information Base (MIB):* SNMP also functions on top of a hierarchical (tree-structured) database, which stores all objects in a (virtual) table used to manage entities in a network.

On top of the descriptive parts of SOA, further intelligence can be established. Especially, in the autonomic computing environment this layer is responsible for analysis and decision making. Especially, the decision making process in this layer will have effect on all managed elements and resources. The reasoning in this layer will be directly interconnected with so called high-level goals, such as “Remain a certain level of QoS” or “Reach a certain Service-Level-Agreement”.

- *WS-Policy:* The W3C recommendation WS-Policy, released September 2007, proposes a general purpose model and a corresponding syntax to describe policies for entities in a Web service based system. The XML format enables clients to exchange information about requirements, capabilities, and constraints (e.g. required security level, Quality-of-Service, in-/compatibility) in a standardized format.
  Moreover, this framework for expressing policies establishes a set of operators, which makes it for example possible to intersect two policies to find an acceptable sub-policy for all parties.

- *WS-Agreement:* WS-Agreement is a specification from the Open Grid Forum (OGF) with the purpose to describe a Web Services protocol for establishing agreements between two parties. The XML-compliant language specifies the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties.
  The specification consists of three parts, which may be used in a composable manner: a schema for specifying an agreement, a schema for specifying an agreement template, and a set of port types and operations for managing life-cycle agreements, including creation, expiration, and monitoring of agreement states.

- *CIM Simplified Policy Language (CIM-SPL):* CIM-SPL (released October 2007) provides an *if-condition-then-action* style policy rules to manage computing resources, using constructs defined by the underlying model of the Common Information Model (CIM).
  The rules’ conditions may contain the direct access to the managed objects’ properties, which can be accessed through various types of CIM data repositories. The action part of a CIM-SPL policy can invoke any operation or function call.

- *Policy Core Information Model (PCIM):* PCIM is a DMTF standard, proposed in February 2001. It defines two hierarchies of object classes: structural classes representing policy information and control of policies, in addition to association classes, that indicate how instances of the structural classes are related to each other. PCIM was developed as an extension to the Common Information Model (CIM).

- *eXtensible Access Control Markup Language (XACML):* OASIS’ version 2.0 of the XACML standard was released in February 2005. The standard describes both a policy language and an access control request/response language (both encoded in XML).
  The policy language is used to describe access requirements and has extension points for defining new data types, functions, or combination logic. With the request/response language an administrator can define whether certain actions should be allowed, or not.
  Various open-source implementations can be found in the Internet (e.g. HERAS-AF, sunxacml, pam_xacml).

- *Autonomic Computing Policy Language (ACPL):* ACPL is a XML-compliant policy language proposed by IBM. Its purpose is to establish a best-of-breed selection of all major policy definition and management languages. Therefore, IBM selected best-practices of DMTF's PCIM standard, OASIS' XACML, the WS-Agreement specification, and the Web Services Policy Framework (WS-Policy).
  The ACPL result is a XML-language, containing four tuples: scope (representing the managed elements), condition (the occurrence of a circumstance), business value (a metric representing the decision priority), decision (can be an action, a configuration profile, and results).

- *Business Process Execution Language (WS-BPEL):* WS-BPEL, or plainly BPEL, is a XML-based language for defining enterprise processes as Web services, allowing businesses to implement a service-oriented approach to business processes.
  BPEL is designed to enable task-sharing for distributed- or grid computing environments using combinations of Web services. Apache ODE is an open-source Web server, fully compliant to the BPEL standard. Further commercial products (i.e. Active Endpoints' ActiveVOS, Oracle BPEL Process Manager) and open-source implementations (e.g. ActiveBPEL 4.0) can be found in the Internet.

Knowledge Provisioning

The knowledge pool will be a fundamental component for autonomic computing systems. Its purpose will be to reflect the overall system's status and establish an always accessible information source on which incidental decisions can be made.

- *Configuration Management Database (CMDB):* The IT Infrastructure Library (ITIL) presents a quasi-standard selection of best practices. ITIL promotes the CMDB as *the* emerging repository for information IT systems.
  The CMDB will be a database, that contains all relevant details of an infrastructures' components and details about the important relationships between managed objects.

Security Standards

Open and heterogeneous systems are most likely a target for security treats. However, a robust security architecture must prevent compromised agents from impairing the overall function. This is especially challenging, since all elements in autonomic systems are distributed and decoupled.

- *WS-Security:* OASIS's WS-Security standard describes enhancements for SOAP's messaging functionality. It adds message integrity, confidentiality and authentification. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. For instance, WS-Security describes how to attach signatures and encryption headers to SOAP messages. Additionally, it describes ways and means to attach security tokens, including binary security tokens, such as X.509 certificates, and Kerberos tickets to messages.

- *WS-Trust:* WS-Trust is an extension to WS-Security, providing a framework for requesting and processing of security tokens and to mediate trust relationships across multiple elements.
- *WS-SecureConversation:* WS-SecureConversation is built on top of WS-Security and WS-Policy models to provide secure communication between services. The purpose of WS-SecureConversation is to allow secure conversations between sites by sharing of security contexts through Web services.
- *WS-SecurityPolicy:* WS-SecurityPolicy's current version 1.2 is an OASIS standard. It is not intended as a complete security solution for Web services. Rather, WS-SecurityPolicy is a building block, that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models.
- *Transport Layer Security (TLS) and Secure Socket Layer (SSL):* Transport Layer Security (TLS) is a protocol that ensures privacy between communicating applications and their users over the Internet. When server and client are communicating over a TLS-secured connection, it ensures that no third party may eavesdrop or tamper exchanged messages. TLS is the successor of Secure Sockets Layer (SSL) protocol. TSL/SSL is being commonly used to provide messaging integrity and confidentially over the HTTP protocol.
- *Security Assertion Markup Language (SAML):* OASIS' SAML standard defines a XML-based framework for communicating user authentication and attribute information.
  SAML allows business entities to validate identities, attributes, and entitlements of a subject.
- *WS-Federation:* The WS-Federation standard proposal defines mechanisms for allowing security realms to broker information about identities, identity attributes, and authentication. Especially for autonomic managers, it will be necessary to communicate confidentially across security domains to request or offer secure services.

In this section we highlighted the standardization efforts, which may lead to an accelerated adoption of autonomic computing paradigms.
The highly interconnected characteristics of autonomic managers, who in turn are managing autonomic elements of a myriad of vendors, require standardized interaction mechanism, defined through external and internal interfaces. A further examination of this topic can be found in [34].

4.2 Autonomic System Approaches

The following section will give an exemplary insight about promising or sophisticated implementation approaches for the autonomic computing vision. It is noteworthy, that this section should neither be seen as a exhaustive list of research projects, nor as a complete enumeration of research groups within the field. Rather, it should give a holistic overview about the current state of research and present interesting approaches towards the problems, occurring in the targeted problem domain.
Moreover, this section will emphasize the differences (scope, infrastructural requirements, and solving strategies) between the mentioned projects.

4.2.1 Focale

FOCALE stands for Foundation, Observation, Comparison, Action, and Learning Environment [57]. The project was introduced in 2006 by Motorola in collaboration with the University of Evry and the TSSG Lab.
Its architecture is closely related to the original autonomic computing vision of enhancing computer systems with self-* properties. But as an enhancement, Focale's actively tackles the problem of dealing with complexity, derived from accommodating legacy hardware and software.

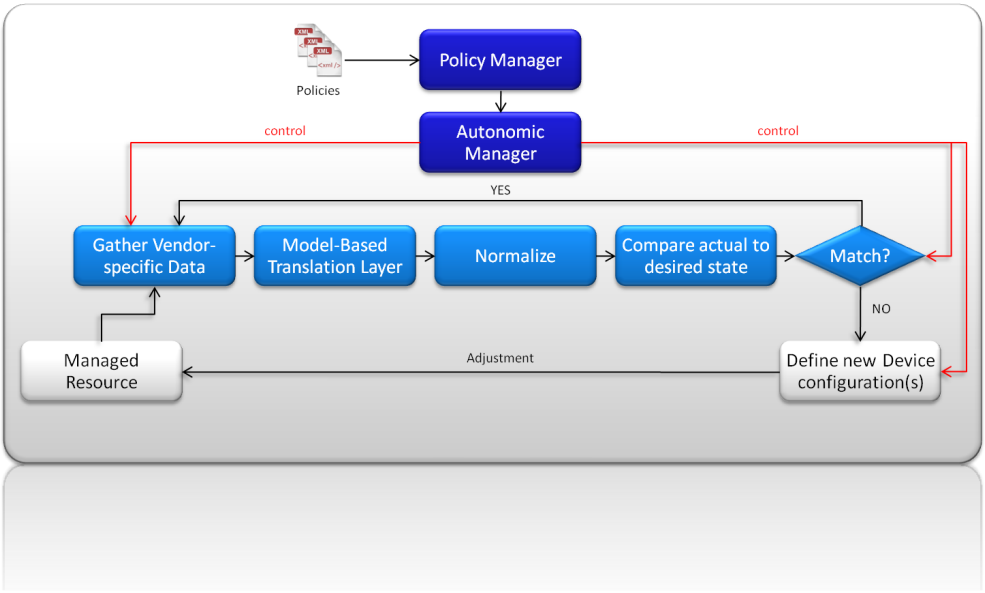


Figure 4.2: Focale: Autonomic Element's Control Loop

As Figure 4.2 depicts, Focale's control loop is closely related to the MAPE control loop introduced in the previous sections (see: Chapter 3 "Autonomic Elements").
The monitoring and analyze components are manifested in Focale's "Gather Vendor-specific Data", which is analyzed in the latter "Compare Actual State to Desired State" step.
Between the mentioned steps, Focale introduces a so called *Mediation-* or *Model Based Translation Layer (MBTL)*. MBTL allows to build networks of heterogeneous devices, no matter if autonomic capabilities and -controlling was intended from the vendor side. For example, in networking architectures it is common to use a SNMP-based system (see: Standards 4.1 "Standards for Autonomic Computing") to monitor necessary network components. However, since many devices are not SNMP-compliant, a vendor-proprietary command line interface (CLI) must be used to retrieve or establish configuration data. Resulting from the fact that a common *lingua franca* (i.e. a standardized, and well-established management information model) is missing, and assuming that any device might have to communicate with any other device, there are at least n^2 programming models to manage. MBTL solves this problem efficiently by introducing a translation mechanism, which accepts vendor-specific data and translates it into a normalized XML representation for further processing by the autonomic manager (see: Figure 4.3).

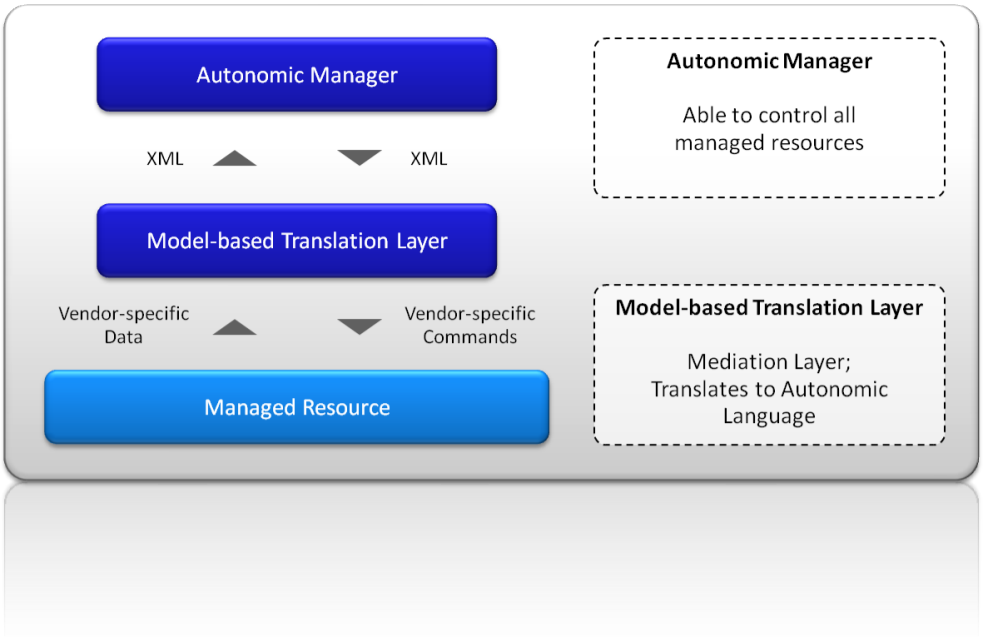


Figure 4.3: Focale: Model Based Translation Layer

As another interesting fact of the architecture, Focale places the actual autonomic manager component outside of the main control loop. This has to be seen as a significant improvement over the current MAPE control loop, since it allows to handle failures in the managed resource from within the autonomic element. In contrast to this, a failed autonomic element would have to rely on another autonomic element, which administers the problem solving process. For example, an autonomic element based on Focale could handle the following scenario: The monitoring component observes actively an interface, which goes down and stays offline until the resource's problem is fixed. In comparison to the current MAPE loop approaches, Focale recognizes the failure, stops monitoring (releases thus computational power), and instead starts to manage the recovery process (problem analysis, -solving). Other MAPE loop implementation, on the other hand, would actively track the failed interface and contract another autonomic element with the repair process. Consequently, this problem solving process would force two autonomic elements to work actively to solve the problem.

Focale's research is driven by the autonomic networking group, which bundles the efforts in the field of autonomic control mechanisms for future networks. Hence, Focale was deploy in [57] to "beyond 3G networks". In this context significantly different wired and wireless access technologies, like UMTS, WiFi, WiMAX, or xDSL are forming management domains, which in turn are subsumed in the overall autonomic management environment.
In the prototype, Focale is recursively deployed as a logical bottom-up decomposition, in which the autonomic management environment (AME) is responsible for ensuring end-to-end service communication, the autonomic management domain (AMD) is responsible for the self-controlled management of a particular administrative domain, and the autonomic computing component (ACC) is responsible for an autonomic management of resources. Each leaf domain

itself can be decomposed into a number of ACCs, that control each individual component in the leaf's domain [57].

4.2.2 AutoI

AutoI is the *Autonomic Internet* approach conducted by the European Community's research facility CORDIS (Community Research and Development Information Service). In the 24 month research project various non-commercial research facilities, such as the University College London (UK), University of Passau (Germany), University of Patras (Greece) and commercial research facilities, such as Hitachi (France), Motorola (USA), Ginkgo Networks (France), TSSG (Ireland), and UPC (Spain) are building a consortium for the ambitious project.
Their main research objective is "the creation of a management resource overlay with autonomic characteristics for the purposes of easy, fast and guaranteed service delivery" [1] for the future Internet.
The heart of AutoI's architecture will be a *virtual* resource overlay, which abstracts away from the more and more aging Internet infrastructure. Due to this, AutoI has to expunge the weaknesses of the current Internet core, which are the lack of built-in service support, -provisioning, -discovery and -management functionality, the missing network and device mobility, and the inherently insufficient QoS- and security support. AutoI's framework description [1] sums the feature set as the following:

- Continuing service delivery in a dynamically changing resource environment and context.
- Resource support for changes in service requirements or introduction of new services.
- Seamless service mobility across multi-domain (technology, operator) (inter-/ intradomain) resource environments.
- Mechanism for unification and separation of the virtual resources at the management level.
- Exhibit self-* functions in terms of supporting autonomic communications.
- Ensuring secure, fast and guaranteed services delivery.

From a conceptual point of view, AutoI consists of a number of distributed management systems, described with the help of five abstractions, the so called *OSKMV* planes:

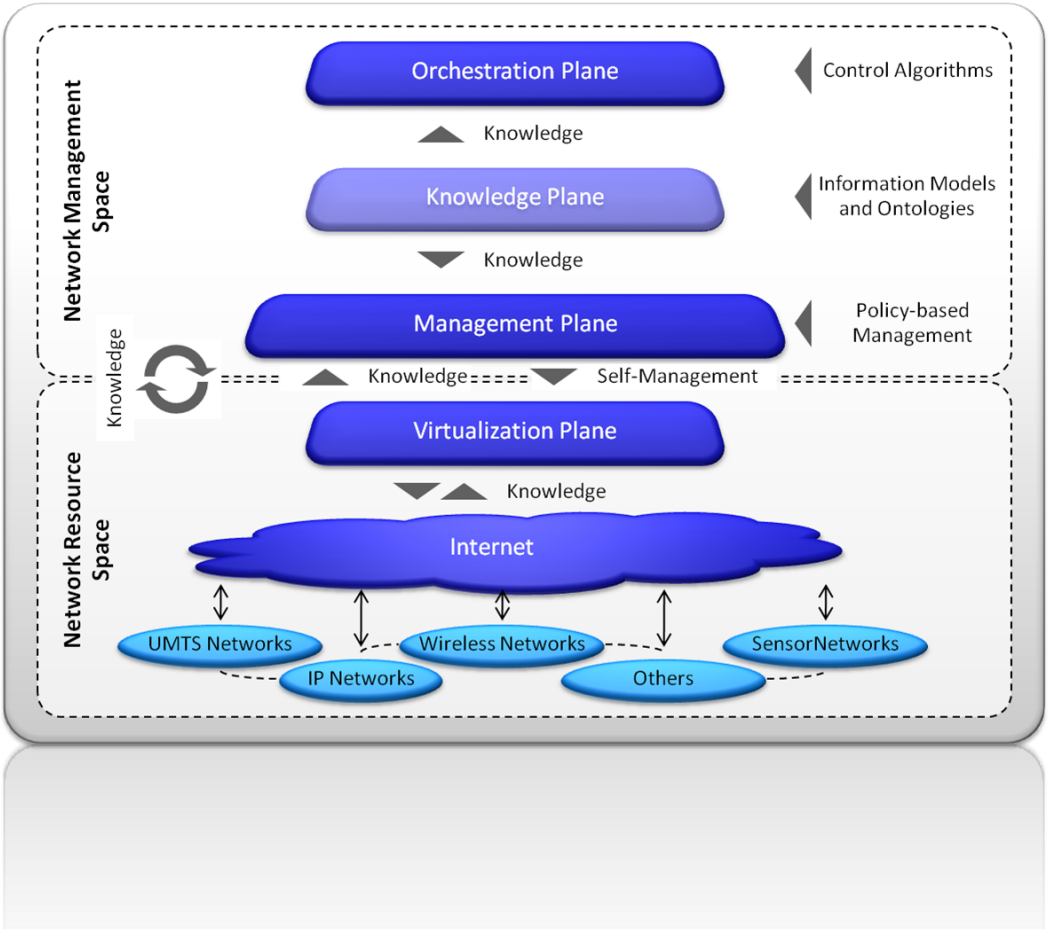


Figure 4.4: AutoI: Layered Management Approach

Beginning from the top, the *orchestration plane's* goal is to offer a component-driven control framework in which components can be plugged into in order to achieve the required functionality. It governs the *overall system's behavior* by responding to context or business goal changes. Therefore, it supervises and integrates all other planes's behavior. Furthermore, the orchestration plane is intended to build an accessible collection of control algorithms and functions as a business-goal "translator", assuring that every entity in the system interprets the policies in a correct way. Being responsible for all managed autonomic subsystems, it "conducts" the system as a whole.
Further tasks are the provisioning of access to all proprietary data models and ontologies, user interface services, and lifecycle management services.

The *knowledge plane* is a concept by Clark et al. described in [15]. Its purpose is the provisioning of knowledge to enable the network to be self-analyzing, self-diagnosing, self-maintaining and self-optimizing.
AutoI is taking over Clark's ideas in parts. By bundling information- and context services with models and ontologies, the knowledge plane provides services for automated reasoning, accessible through unified data structures.

The knowledge plane's main task is to provide a system-wide accessible *universal lexicon*, containing all forms of facts (business-policies, resource descriptors, and infrastructural knowledge).

The *management plane* is AutoI's main autonomic management component. It builds the basis for every autonomic subsystem by incorporating a *Control-Analyze-Enforce-Change* loop, which can be seen equivalent to the MAPE loop.
Using the knowledge of the planes on top and underneath, it provides educated decisions leading to self-awareness, self-adaptiveness, self-control and extensibility. For the lower levels it enables a recursive subclustering for logical groups of entities (same access channel, same device functionality, etc.).

The *virtualization plane* consists of software mechanisms to treat selected physical resources as a programmable pool of virtual resources, which can be organized by the orchestration plane into appropriate sets of virtual resources to form components (e.g., increased storage or memory), devices (e.g. a switch with more ports), or even networks. Every (re-)organization is executed in order to realize a certain business goal or service requirement [1].

Together all mentioned distributed management systems form a software-driven network approach that runs on top of the current networks with all its differentiations (i.e. fixed, wireless, and mobile networks) and services. AutoI's framework description [1] states, that with this reference architecture it is aspired to deliver the first prototypical approach, that reaches the goal of a holistic autonomicity in networking.

4.2.3 SkyEye.KOM and SkyNet.KOM

SkyEye.KOM and SkyNet.KOM are two sequential projects of the Technische Universität Darmstadt (Germany), focusing on an implementation of the autonomic computing principle in structured Peer-to-Peer networks.
As a matter of fact, especially the Peer-to-Peer environment seams to be one of the most difficult –but yet promising– fields of application for the emerging autonomic system paradigm. In addition to the primarily unsolved challenges stated throughout this thesis, the researcher have to deal with node churn, effective message propagation, uncertainty, responsibility- and performance considerations.
On the contrary to the most approaches in the field of autonomic computing, SkyEye.KOM and SkyNet.KOM are captivating with its advance not to deliver a full-fledged autonomic framework from the very beginning. Rather, the research work can be considered as the more promising approach to steadily enhance existing solutions with more and more autonomicity and self-* properties. Nevertheless, in [28] the researcher denote, that their overall project goal is to establish a self-controlled monitoring and management in structured Peer-to-Peer systems.

Currently, in the field of Peer-to-Peer systems it is difficult to monitor and manage the quality of a system, because system-wide considerations are not easy to provide. Especially, a node's limited knowledge about the overall network seams to be a restricting circumstance for the aggregation of knowledge. But above all, this knowledge and experience aggregation (see: Section

3.5 “Knowledge”) has to be seen as the turnkey solution for a self-managed optimization process. As a solution to this problem Graffi et al. propose in [28] SkyEye.KOM, which presents a resource-preserving and reliable monitoring approach on top of a variety of underlaying DHT-protocol overlays (such as Chord [55], Kademlia [46], Tapestry [70]).
In order to operate on nearly every DHT overlay, SkyEye.KOM takes advantage of the quasi-standard Key-Based Routing (KBR) Interface [18] and introduces the following *ID Space Mapping* that makes it independent from the underlaying specifics:

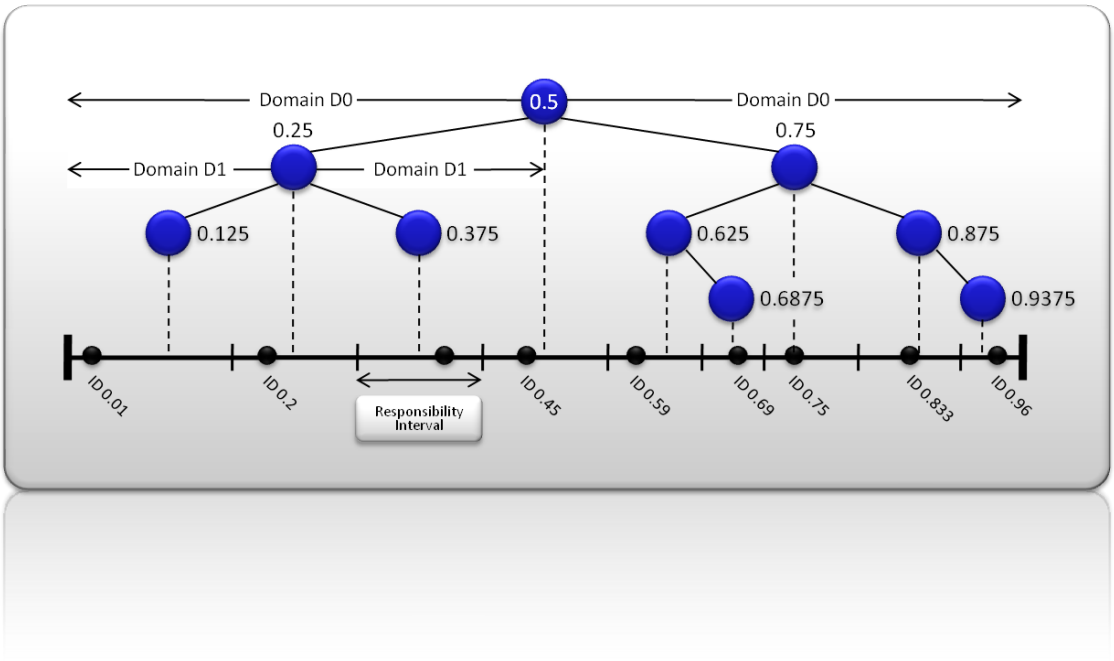


Figure 4.5: SkyEye.KOM: Monitoring Tree Topology

As Figure 4.5 depicts, the various overlay-specific ID spaces –e.g. Chord uses $0, \ldots, 2^{160} - 1$ IDs, Kademlia uses $0, \ldots, 2^{128} - 1$ IDs– are mapped onto an unified ID space in the interval of $[0, 1] \subset \mathbb{R}$.
With the help of this overlay independence mapping, the ID space is deterministically subdivided into intervals, the so called *Domains*. Each domain has one responsible node, the *Coordinator*. With one operation call every inner node or leaf in the binary tree structure can determine its own coordinator.
The actual monitoring functionality is established as follows: After the metric update period, which denotes the continuous measurement of a node’s metrics, the node propagates his metric-update to the parent/coordinator. After receiving messages from its children, the coordinator node computes the tree-branch wide metrics by aggregating the children’s metrics to his own. Since the parent node itself propagates its metric update to its parent, each metric update is propagated until the root-coordinator is reached.
In contrast to other monitoring applications for Peer-to-Peer networks (see: Section 3.1.1 “Decentralized Monitoring Approaches”), SkyEye.KOM enables the system-wide aggregation infor-

mation in an unprecedented level of detail. For example, metric evaluation functions such as min(x), max(x), sum(x), sum_of_squares(x), count(x), avg(x), and standard_deviation(x) are automatically applied and propagated. Hence, a comprehensive and significant system-wide metric examination is established.

Basing on the monitoring functionality provided by SkyEye.KOM, the researches are introducing knowledge processing functions, such as analyze (see: Section 3.2 “Analyze”) and plan (see: Section 3.3 “Plan”), to the Peer-to-Peer system. Due to this, the project SkyNet.KOM closes the MAPE control loop cycle, as Figure 4.6 reveals:

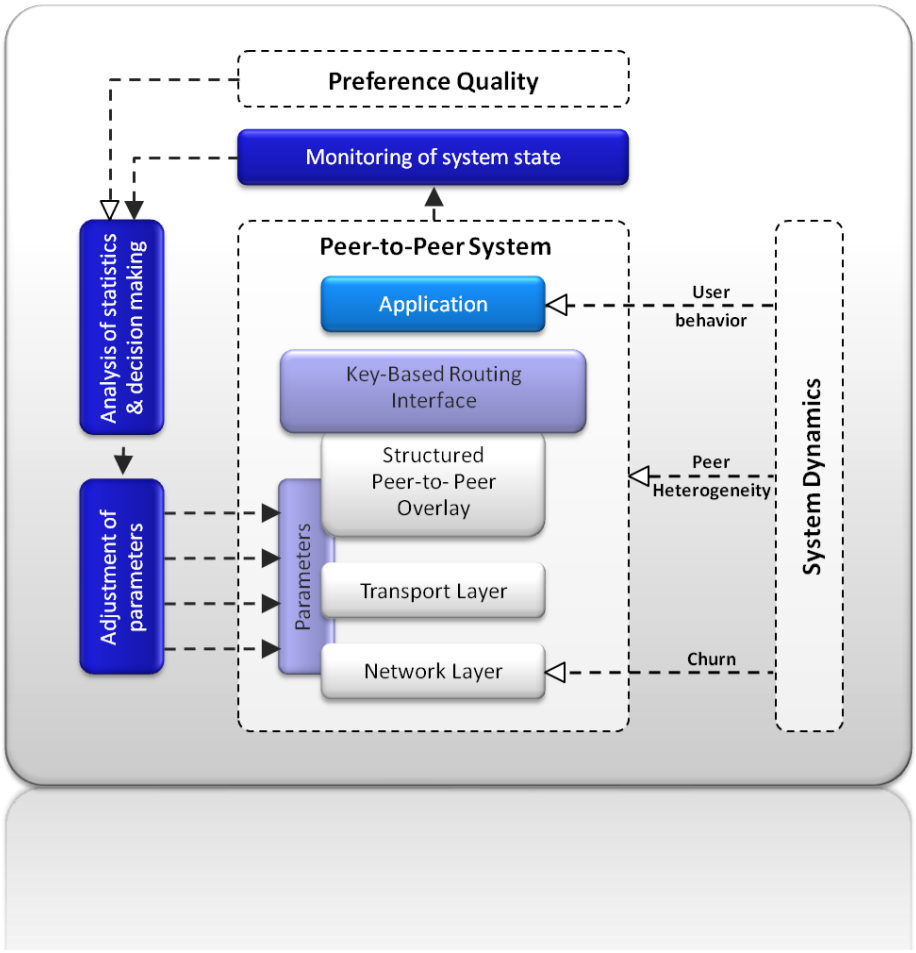


Figure 4.6: SkyNet.KOM: System Architecture

After abstracting away from the Peer-to-Peer specific details, the architecture shows a remarkable similarity to original MAPE loop vision.
Since all system metrics are propagated to the root, SkyNet.KOM implements the analyze, as well as the planning step within the root node. Here, the current system state is evaluated against preset quality intervals for a set of metrics (e.g. response time, average bandwidth, variance of bandwidth usage). Within the MAPE loop context, these preset quality intervals manifest the policies, as described in the Section 3.2.1 “Policy Definition Languages”.

In the subsequent planning step, it is decided which parameters need to be changed. Currently, several implementations for a sustainable planning mechanism are applied, in accordance to the promising developments in the field of artificial intelligence (see also: Section 3.3 “Plan”). While in the first SkyNet.KOM implementation decision rules were denoted by static decision code, current research projects are implementing genetic algorithms, machine learning and neural networks to increase the “intelligence” of the planning component.
As the last step in the SkyNet.KOM implementation a parameter-change request is spread out to all nodes in the network. This is done effectively by attaching the change request descriptors to the metric update acknowledgments. Thus, a remarkably low overhead is produced by the SkyNet.KOM implementation.
All in all, SkyNet.KOM presents means and mechanisms for enabling structured Peer-to-Peer systems to reach and hold present quality standards.
A more specific description and further details can be found in [28].

4.3 Summary

Since Paul Horn coined the notion of *autonomic computing* in 2001, significant advancements have been made. Prototypes, standardization efforts and commercial products promote the emerging advent of intelligent self-managing systems.
However, it is worth noting that autonomic computing architectures do not exclude the idea of human-intervention in extraordinary situations. In contrast, it is rather the goal to support the –nonetheless– required educated staff by carrying out reoccurring administrative functions (i.e. analyze, plan, optimize). Hence, the overall goal of autonomic computing is to limit the hands-on intervention of administrators to a minimum.
But especially the diversity of management tasks in computer systems requires close collaborations of various computer science disciplines (i.e. AI-, complexity theory-, network- and software-engineering research, etc.), as well as completely new approaches and perceptions. The magnitude of this collaboration leads to a slow, but steady, maturing process of autonomic systems, which can be described by the following Figure 4.7:

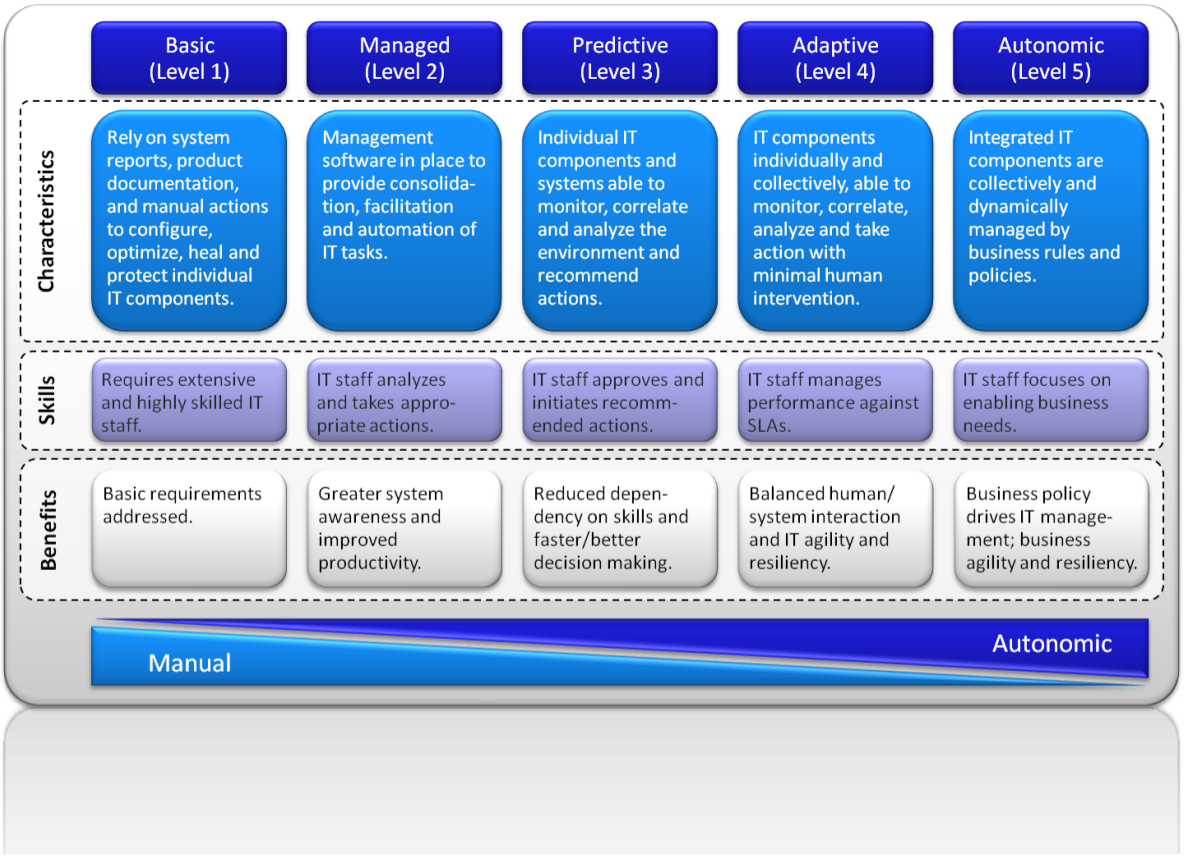


Figure 4.7: Autonomic Systems Maturity Levels

As Figure 4.7 depicts, the autonomic computing vision will not be reached overnight, but it will take several years of research to reach the ultimate goal.
As the foundation of all autonomic approaches the *basic* level represents products and environments, that are dependent on the human minds of their administrators or require consultation of humans for everyday operation processes.
As the next milestone, *managed* systems are enhanced by scripting and logging tools to automate routine execution and reporting. The staff's task is to review the gathered information in order to initiate plans and make educated decisions.
Next, *predictive* systems will forecast potential upcoming events and raise warning flags if a certain threshold is excelled. In this step reactive mechanisms (aggregated in a knowledge base) are also recommended. Due to this the global knowledge pool will need to incorporate common occurrences and experiences.
Adaptive systems are constructed on top of the predicted level, deciding itself about appropriate (re-)actions.
Finally, in order to reach the last and fifth level of maturity, all efforts have to be combined to build on top the previous "adaptive" step. In this highest level of self-managing, *autonomic systems* will incorporate sophisticated mechanisms, driven by business policies, to manage the system without human-intervention. At this point, the system will decide on its own whether or not resources need to be allocated, if task priorities are appropriate, or which actions have to be

executed in order to reach SLA agreements.

While examples of predictive and even adaptive systems exist today, the general state of the industry remains at the basic and managed levels. The IT industries' slow reaction times, and economical leadership-ambition leads to considerable overhead, duplication of effort, and missed opportunities. At the same time, non- and semi-commercial organizations are pushing the limits of autonomic systems beyond the third level.

Like any form of maturation, the autonomic computing research has to reach milestone by milestone. However, during this process a comprehensive knowledge about the advantages and disadvantages of the model can be aggregated and evaluated. But, during this process the basic knowledge of the autonomic computing philosophy can be spread out to find an increasing adaption in the IT landscape.
Nevertheless, in the efforts of applying an autonomic skeletal structure on top of existing infrastructures one important aspect may not be missed: It is important –yet vital– for autonomic computing technologies to reach a better value for the customers investing in IT resources. Thus, it is sustainably important to reduce the management complexity in all fields without shifting necessary efforts from one point to another. It does not make sense to relieve administrators with optimization and configuration tasks, while they have to define and adapt policies for every managed device in return to a system's counter move.
The overarching objective must be, that the system does more work for the user than the user does for the system. Consequently, this will lead to a shift from a *system-centric* perception towards a *human-centric* approach.

Autonomic maturity is starting at every conceivable maturity level of a computer system by incorporating tools to move to the next level. By applying autonomic computing tools, mechanisms or architectures to the computing infrastructure, one can deliver better value to the customer. The greater risk is a hardly manageable, unintegrated system, expensive to maintain, and hard to change. Autonomic computing technology represents a way to alleviate practical problems today, while laying the foundation for the integrated systems architecture for tomorrow.

A Appendix

Bibliography

[1] M. Abid, Andreas Berl, Z. Boudjemil, Abderhaman Cheniour, Steven Davy, Spyros Denazis, Alex Galis, Jean-Patrick Gelas, Claire Fahy, Andreas Fischer, Javier Rubio Loyola, Laurent Lefèvre, Daniel Macedo, Lefteris Mamatas, Hermann de Meer, Z. Movahedi, John Strassner, and H. Zimmermann. Autonomic internet initial framework - deliverable d6.1. Technical report, INRIA, August 2008.

[2] Lobo Jorge Agrawal, Lee Kang-Won. Policy-based management of networked computing systems. *IEEE Communications Magazine*, 43:69 – 75, 2005.

[3] Keno Albrecht, Ruedi Arnold, Michael Gahwiler, and Roger Wattenhofer. Aggregating information in peer-to-peer systems for improved join and leave. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 227–234, Washington, DC, USA, 2004. IEEE Computer Society.

[4] BBC. Napster use slumps 65 percent., July 2001.

[5] Dirk Beyer. Relational programming with crocopat. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 807–810, New York, NY, USA, 2006. ACM.

[6] P.S. Bhattacharjee, D. Saha, and A. Mukherjee. Heuristics for assignment of cells to switches in a pcsn: a comparative study. pages 331–334, 1999.

[7] Ted J. Biggerstaff. Design recovery for maintenance and reuse. *Computer*, 22(7):36–49, 1989.

[8] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. Improving traffic locality in bittorrent via biased neighbor selection. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 66, Washington, DC, USA, 2006. IEEE Computer Society.

[9] BitTorrent. http://www.bittorrent.com, 2009.

[10] Ruben Fraile Blazquez. Literature survey on information management systems. Bachelorthesis, Technische Universität Darmstadt, 2008.

[11] Microsoft MSDN Blog. http://blogs.msdn.com/philipsu/archive/2006/06/14/631438.aspx, June 2006.

[12] Nevon Brake, James R. Cordy, Elizabeth Dan y, Marin Litoiu, and Valentina Popes u. Automating discovery of software tuning parameters. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 65–72, New York, NY, USA, 2008. ACM.

[13] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 219–227, New York, NY, USA, 2000. ACM.

[14] Steven Chamberland and Samuel Pierre. On the design problem of cellular wireless networks. *Wirel. Netw.*, 11(4):489–496, 2005.

[15] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A knowledge plane for the internet. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10, New York, NY, USA, 2003. ACM.

[16] Thomas Cofino, Yurdaer Doganata, Youssef Drissi, Tong Fin, Lev Kozakov, and Meir Laker. Towards knowledge management in autonomic systems. In *ISCC '03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, page 789, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Louis Anthony Cox, Jr. and Jennifer Ryan Sanchez. Designing least-cost survivable wireless backhaul networks. *Journal of Heuristics*, 6(4):525–540, 2000.

[18] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. pages 33–44. 2003.

[19] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.

[20] L. Durham, J. van de Groenendaal, J. He, J. Hobbs, M. Milenkovic, and Y. Mazin. Platform support of autonomic computing: an evolution of manageability architecture. *Intel Technology Journal*, 10:253 – 263, 2006.

[21] Kolja Eger and Ulrich Killat. Fair resource allocation in peer-to-peer networks (extended version). *Comput. Commun.*, 30(16):3046–3054, 2007.

[22] Exner. Techniques for approximation and optimization of ibm db2 udb performance functions (in german). Master's thesis, University of Jena, Germany, 2007.

[23] Eclipse Foundation. http://www.eclipse.org/org/press-release/20070627_europarelease.php. Press Release, June 2007.

[24] Fujitsu. www.fujitsu.com/global/services/solutions/triole/, 2009.

[25] Nejla Ghaboosi and Abolfazl T. Haghighat. A tabu search based algorithm for multicast routing with qos constraints. In *ICIT '06: Proceedings of the 9th International Conference on Information Technology*, pages 33–39, Washington, DC, USA, 2006. IEEE Computer Society.

[26] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[27] Kalman Graffi, Sebastian Kaune, Konstantin Pussep, Aleksandra Kovacevic, and Ralf Steinmetz. Load balancing for multimedia streaming in heterogeneous peer-to-peer systems. In ACM Press, editor, *18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '08)*, page 6, Technical University of Braunschweig, Braunschweig, Germany, May 2008. ACM SIGMM.

[28] Kalman Graffi, Aleksandra Kovacevic, Song Xiao, and Ralf Steinmetz. SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems. In *The 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*. EEE Computer Society Press, 2008.

[29] Kalman Graffi, Konstantin Pussep, Sebastian Kaune, Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Overlay bandwidth management: Scheduling and active queue management of overlay flows. In USA IEEE Computer Society, editor, *IEEE Local Computer Networks '07*, pages 334–342. IEEE Computer Society, Oct 2007.

[30] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[31] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology, 2001.

[32] IBM. An architectural blueprint for autonomic computing. 2006.

[33] Stratos Idreos, Manolis Koubarakis, and Christos Tryfonopoulos. P2p-diet: an extensible p2p service that unifies ad-hoc and continuous querying in super-peer networks. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 933–934, New York, NY, USA, 2004. ACM.

[34] Intel. Standards for autonomic computing. *Intel Technology Journal*, 10:275 – 284, 2006.

[35] Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Proc. of ESOA'05*, 2005.

[36] Joost. http://www.joost.com/, 2009.

[37] Sebastian Kaune, Tobias Lauinger, Aleksandra Kovacevic, and Konstantin Pussep. Embracing the peer next door: Proximity in kademlia. In *P2P '08: Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 343–350, Washington, DC, USA, 2008. IEEE Computer Society.

[38] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[39] Sami Khuri and Teresa Chiu. Heuristic algorithms for the terminal assignment problem. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 247–251, New York, NY, USA, 1997. ACM.

[40] I. M. A. Kirkwood, S. H. Shami, and M. C. Sinclair. Discovering simple fault-tolerant routing rules using genetic programming. In George D. Smith, Nigel C. Steele, and Rudolf F.

Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference, ICANNGA97*, University of East Anglia, Norwich, UK, 1997. Springer-Verlag. published in 1998.

[41] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.kom - a p2p overlay for fully retrievable location-based search. In *P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*, pages 87–96, Washington, DC, USA, 2007. IEEE Computer Society.

[42] John R. Koza. Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford University, Stanford, CA, USA, 1990.

[43] Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *In Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Systems*, pages 227–236, 2004.

[44] Siriluck Lorpunmanee, Mohd Noor Sap, Abdul Hanan Abdullah, and Chai Chompoo-inwai. An ant colony optimization for dynamic job scheduling in grid environment. 2008.

[45] Vincent Matossian, Viraj Bhat, Manish Parashar, Malgorzata Peszyńska, Mrinal Sen, Paul Stoffa, and Mary F. Wheeler. Autonomic oil reservoir optimization on the grid: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(1):1–26, 2005.

[46] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[47] Arif Merchant and Bhaskar Sengupta. Assignment of cells to switches in pcs networks. *IEEE/ACM Trans. Netw.*, 3(5):521–526, 1995.

[48] Tadashi Nakano and Tatsuya Suda. Applying biological principles to designs of network services. *Appl. Soft Comput.*, 7(3):870–878, 2007.

[49] Napster. http://www.napster.com, 2009.

[50] Oasis. Reference model for service oriented architecture v1.0, November 2008.

[51] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*, pages 247–259. Springer Verlag, 2005.

[52] Gennadi Rabinovitch and David Wiese. Non-linear optimization of performance functions for autonomic database performance tuning. In *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 48, Washington, DC, USA, 2007. IEEE Computer Society.

[53] David S. Rosenblum and Alexander L. Wolf. A design framework for internet-scale event observation and notification. *SIGSOFT Softw. Eng. Notes*, 22(6):344–360, 1997.

[54] M. C. Sinclair. Minimum cost topology optimisation of the cost 239 european optical network. In *In*, pages 26–29. Springer-Verlag, 1995.

[55] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[56] Margaret-Anne D. Storey, Kenny Wong, and Hausi A. Müller. Rigi: a visualization environment for reverse engineering. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 606–607, New York, NY, USA, 1997. ACM.

[57] John Strassner, N. Agoulmine, and E. Lehtihet. Focale: A novel autonomic networking architecture. In *Latin American Autonomic Computing Symposium (LAACS)*, 2006.

[58] Riky Subrata, Albert Y. Zomaya, and Bjorn Landfeldt. Artificial life techniques for load balancing in computational grids. *J. Comput. Syst. Sci.*, 73(8):1176–1190, 2007.

[59] Hiroshi Tamaki, Ken-ichi Fukui, Masayuki Numao, and Satoshi Kurihara. Pheromone approach to the adaptive discovery of sensor-network topology. In *WI-IAT '08: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 41–47, Washington, DC, USA, 2008. IEEE Computer Society.

[60] Apache Tapestry. http://tapestry.apache.org/, 2009.

[61] IBM Tivoli. www.ibm.com/tivoli, 2009.

[62] Martina Umlauft and Wilfried Elmenreich. Qos-aware ant routing with colored pheromones in wireless mesh networks. In *Autonomics '08: Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[63] S. VanDerMeer, A. Davy, S. Davy, R. Carroll, B. Jennings, and J. Strassner. Autonomic networking: Prototype implementation of the policy continuum. In *InProc of 1st IEEE Intl Workshop on Broadband Convergence Networks BcN*, 2006.

[64] Jun Wang and Mohan S. Kankanhalli. Experience based sampling technique for multimedia analysis. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 319–322, New York, NY, USA, 2003. ACM.

[65] Qiang Wang, Khuzaima Daudjee, and M. Tamer Özsu. Popularity-aware prefetch in p2p range caching. In *P2P '08: Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 53–62, Washington, DC, USA, 2008. IEEE Computer Society.

[66] Thomas Weise, Hendrik Skubch, Michael Zapf, and Kurt Geihs. Global optimization algorithms and their application to distributed systems. Kasseler Informatikschriften (KIS) 2008, 3, Distributed Systems Group, FB 16, University of Kassel, Wilhelmsöher Allee 73, 34121 Kassel, oct 2008. A technical report on the usage of global optimization algorithms such as genetic algorithms, genetic programming, tabu search, simulated annealing, and ant colony optimization for improving features of distributed systems (such as topology, routing, ...

[67] Wikipedia. Survival of the fittest — wikipedia, the free encyclopedia, 2009. [Online; accessed 29-May-2009].

[68] Bowei Xi, Cathy H. Xia, Zhen Liu, Li Zhang, and Mukund Raghavachari. A smart hill-climbing algorithm for application server configuration. In *13th Int. Conf. on WWW*, pages 287–296, 2004.

[69] Yiming Zhang, Dongsheng Li, Lei Chen, and Xicheng Lu. Flexible routing in grouped dhts. In *P2P '08: Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 109–118, Washington, DC, USA, 2008. IEEE Computer Society.

[70] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

[71] Katharina Anna Zweig. *On Local Behavior and Global Structures in the Evolution of Complex Networks*. PhD thesis, Fakultät für Informations- und Kognitionswissenschaften der Eberhard-Karls-Universität Tübingen, July 2007.

Index