The architecture of distributed systems driven by autonomic patterns

Marcin Wolski, Cezary Mazurek, Paweł Spychała, Aleksander Sumowski

Poznań Supercomputing and Networking Center,
ul. Noskowskiego 12/14 Poznań, Poland
{maw, mazurek, spychala, sumek}@man.poznan.pl

Abstract. The autonomic computing notion has introduced the concept of self-optimizing, self-healing and auto-diagnosis applications. In this article we would like to present how this idea affects the building of distributed systems. As a reference base, we take advantage of the Data Management System (DMS), which has been developed within the scope of the PROGRESS project. DMS enables the creation of a grid environment capable of storing large amounts of data. The complex architecture of this system, which constitutes a model of loosely coupled components, involves a special approach to its maintenance and management. To address these problems, we have applied the autonomic computing patterns in the DMS architecture. Our solution was designed to be reused in any project dealing with the same issues. It can also act as an autonomic service for any other applications and services.

1 Introduction

Data grid systems have been designed to be up to complex data processing in a geographically distributed environment and exact performance demands. Over the years this class of systems has matured and at present they are offering a wide range of functionality related to the management, collaborative sharing, publication, and preservation of distributed data collections. This wealth of capabilities, however, complicates the managing of such large systems, increases its complexity as more heterogeneous components are added, and makes it more difficult to find and solve any technical problems. They constitute a typical example of an environment where administrators spend too much time doing repetitive tasks, monitoring the system burden, reacting when problems with performance arise or continuously blocking the hackers' attacks.

The concept of the Service Oriented Architecture (SOA) [9,13], which common implementation is based on existing Web services standards and specifications, helps to deal with these inconveniences. The notion of a service is nothing new, but the concept of the SOA has evolved over the past couple of years. It is an architectural style of building software applications that promotes loose coupling between components so that you can reuse them. SOA makes it possible to construct architectures where client applications can simply register, discover, and use the services deployed over the grid.

Please use the following format when citing this chapter:

Wolski, M., Mazurek, C., Spychała, P., Sumowski, A., 2006, in IFIP International Federation for Information Processing, Volume 227, Software Engineering Techniques: Design for Quality, ed. K. Sacha, (Boston: Springer), pp. 49–60.

SOA itself narrows the focus on the overall system maintenance and management but it does not cope with many problems derived from the complexity of distributed systems. We need a solution which enables the system to automatically configure its components, discover and correct faults, monitor and control resources and proactive identify and protect from arbitrary attacks. Autonomic computing (AC) researches offer the most promising approach to addressing such challenges.

The conjecture of AC was inspired by IBM's autonomic computing initiative to deal with the main problem in large and distributed systems - increasing complexity. Autonomic means able to operate without conscious control of a human – similarly to our heart or lungs controlled by our autonomic nervous system. AC generally has two main goals: to reduce the work and complexity associated with a large system and be able to better respond to rapid changes in the system.

In this paper we would like to present how the autonomic computing notion affects the building of distributed systems. Moreover, the solution that we provided can also act as an autonomic service for any other applications and services. As a reference base, we take advantage of the Data Management Suite (DMSuite) [3,4] – a platform of integrated services supporting data management processes in the grid environments. DMSuite has been designed and developed in the scope of the PROGRESS project [2] – an initiative undertaken within the PIONIER National Program [1] and funded by the State Committee for Scientific Research and Sun Microsystems Poland. Currently the Data Management Suite is a part of the Gridge (Grid Enterprise Solutions) [5], which covers the whole grid architecture, from tools and portals down to core middleware.

The remainder of this paper is organized as follows: Section 2 introduces the background and technical aspects of the AC model. Section 3 details the current implementation of DMSuite and indicates its advancement in terms of autonomic computing. Section 4 demonstrates some case studies taken from various projects using the DMSuite software and presents a definition of autonomic patterns and its relationship with the system architecture. Section 5 summarizes the paper with our conclusions about building distributed systems on the basis of the SOA model and AC patterns. It also indicates what will be held in the upcoming release of DMSSuite.

2 Autonomic computing design

Autonomic computing was introduced by IBM as a response to overwhelmingly increasing complexity of novel systems [12]. The process of growing complexity threatened that at some future point of time computer systems would become a burden, covering its initial use.

The autonomic computing vision is based on an autonomic nervous system. Autonomic computer systems are supposed to be able to operate without human attention. They are supposed to automatically interoperate between each other without the need to tweak large amounts of switches and XML configuration files.

The system architecture built according to autonomic computing principles should limit the hands-on intervention to uncommon cases which occur during the system's regular work. This postulate could be fulfilled by applying predefined policies for

administrative operations which can take decisions leading to the system reconfiguration, basing on gathered knowledge during the system work. Such vision of the fully self-managed system seems to be hard to realize, but it allows to determine the aim, an ideal system architecture which uses different technologies and solutions for achieving the assumed autonomic computing level. The autonomic computing architecture can be understood as a continuum for a system.

2.1 Self-CHOP paradigm

There are four components that comprise the autonomic computing vision [11]:

- Self-configuring – means the ability to dynamically adapt to changing environments. Self-configuring components use policies provided by the professional staff to perform self-configure procedures. Such changes could include the deployment of new components or the removal of the existing ones, or even remarkable changes in the system characteristics.
- Self-healing – means the ability to discover, diagnose and react to malfunctions. Self-healing components can detect system disruptions and initiate policy-based repair procedures without any influence on the rest of the environment. Corrective action could involve a product altering its own state or effecting changes in other components.
- Self-optimizing – means the ability to monitor and tune resources automatically. The tuning actions could imply, for example, reallocating resources (such as in reaction to dynamically changing workloads), improvement of the overall utilization, or ensuring that particular transactions can be completed in a timely fashion.
- Self-protecting – means the ability to anticipate, detect, identify and protect against threats from anywhere. Self-protecting components can identify hostile behaviors as they occur and take appropriate actions to make themselves more resistant. The hostile behaviors can include unauthorized access and use, virus infection and denial-of-service attacks.

Those four ideas together form a self-CHOP paradigm which, in short, stands for configuration, healing, optimization and protection.

2.2 Maturity levels

The autonomic computing architecture ideas could be realized in the developed system in a different way, in a different scope and on a different level. Following the [14], there are five levels of maturity that refers to the state of implementation of the autonomic computing recommendations. These levels are: basic, managed, predictive, adaptive and autonomic. Although the distributed applications constantly evolve along these stages, the general state of the novel system remains at the basic and managed levels. These two base levels do not allow the application to be aware of the system environment state.

The basic level defines an architecture which still requires human intervention and expertise basing on their knowledge. The managed level is achieved when the envi-

ronment is equipped with some scripting and logging tools, allowing to automate routine execution and reporting. The plans and taken decisions are based on this gathered information; however, it still needs an individual specialists review.

Systems at a predictive level of autonomic computing maturity have a basic intelligence, which bases on predetermined thresholds and knowledge base, suggesting solutions according to the set of events stored at a centralized base and their common occurrences and experience. The adaptive level defines environments that allow them to take action themselves basing on predictive system capabilities according to the arising situations.

The highest level of the autonomic computing system architecture is defined as autonomic, which is understood as a policy-driven system, which is able to e.g. allocate resources according to priorities.

It is worth underlining that the system maturity levels evolve and there is no approach to make a self-optimizing, self-protecting, self-configuring and self-healing system.

3 Data Management Suite

DMSuite is a middleware platform providing a uniform interface for connecting heterogeneous data sources over a network. It may stand for the backbone on which a computational grid would perform its operations. The following figure depicts the main components belonging to this integrated platform.

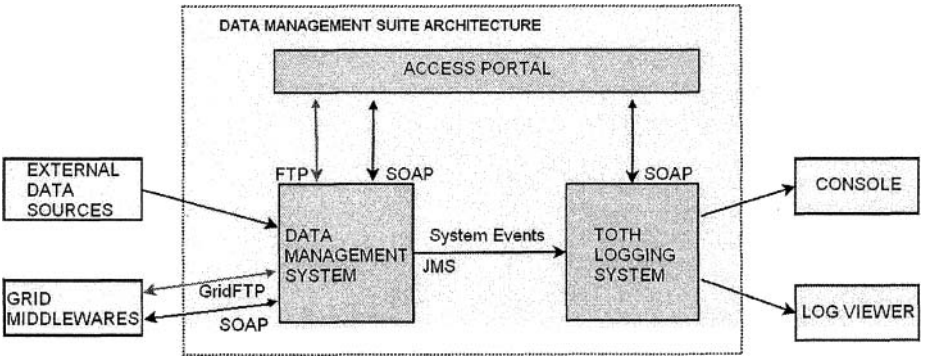

Fig. 1. Data Management Suite architecture

The Data Management System and Toth constitute a base for our architecture that combines autonomic and data grid technologies. The Data Management System is the main part of Data Management Suite solutions. It is a middleware application based on the SOA model that determines loose coupling between reusable components. Similarly to computing and network resources, DMS provides services to manage and retrieve data files in order to support grid jobs. The computational resources managed by DMS can be described by metadata schemas which allow the creation of an abstract, semantic and explorable layer of resources.

The Toth Logging System simplifies the system administration process by gathering all events incoming from distributed components of the DMSuite. It is based on the JMS technology [15], which assures simple exchange of binary messages in the asynchronous way.

3.1 Autonomic design

DMSuite has been built using some autonomic computing principles, such as self-management, fault detection and self-configuration. The Data Management System (DMS) - the key element of DMSuite - contains three logically distinguished modules: Data Broker, Metadata Repository and Data Container [16]. Together they create the basic layer of the data management environment, the so-called DMS core. DMS architecture is comprised of distributed modules, each of them can be treated as a separate service using XML messages to communicate with other applications.

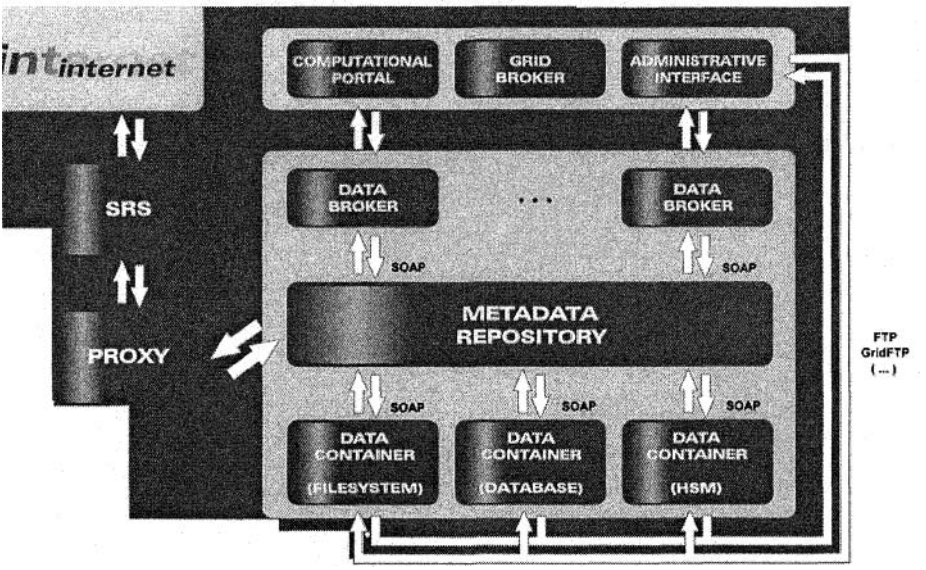

Fig. 2. Data Management System

Metadata Repository stores various type of information about resources managed by the system. Data Broker is a distributed entry point to the system. Data Container is a storage element, which arranges data on various types of media. Proxy provides an uniform interface to external data sources with a diversified structure. Within the PROGRESS installation Proxy enables access to biological databanks managed by SRS (Sequence Retrieval System) [19]. All modules belonging to the DMS middleware automatically register themselves in the Metadata Repository.

According to the five-degree maturity levels of autonomic computing (section 2.2), DMSuite currently comes in the second (managed) level. The first (basic) level of maturity is assured by the presence of the Toth Logging System. The Toth is responsible

for gathering all events that occur in the monitored system. It stores the messages in the internal structures and provides an interface to explore them. But it is not enough to achieve the second level of maturity, which ensures that systems management technologies can be used to collect details from managed resources, helping to reduce the time it takes the administrator to collect and synthesize information. Therefore Toth has been equipped with additional functionality related to the messages processing. It performs advanced parsing on each of the received events, constructs a set of metadata on their base, and exposes an interface for searching the collected events, according to the specified criteria.

Our initial solution aimed to fulfill the basic AC principles is completed by a single access interface to the whole data components. It allows the end users (professional staff, administrators, researches) to manage the entire data grid infrastructure as easily as managing one application running on one computer. This interface has been developed in a form of a Web portal which offers a single and efficient tool to simplify the management of the distributed components.

The ideas described above have introduced the concept of autonomic computing in the DMSuite environment. This includes self-configuration, that is automatic registration of distributed components in the Metadata Repository, and self-healing, which stands for restoring coherency in the distributed file system. But these features constitute only a part of the autonomic computing model and are appropriate to achieve the basic level of AC implementation. On the basis of some scenarios taken from various perspectives (users, developer, administrators), we will point out which features are still missing, and describe some extensions to the DMS architecture which will take advantage of advanced capabilities from the autonomic computing area.

4 Toward the concept of autonomic patterns

In the beginning we would like to recall a few general principles which were formulated to design the data grid architectures (following the [6]). These are: mechanism neutrality, policy neutrality, compatibility with grid infrastructure, and uniformity of the information infrastructure.

These principles were the underlying reason for the creation of DMSuite. But taking into consideration real case studies taken from various projects using our software, we noticed a lack of AC capabilities which are necessary to fulfill the enterprise requirements. These are:

- Self-configuration: self-discovery and self-configuring of the system components. For example: providing and maintaining the current information about active Data Brokers and their hierarchy, automatic detection of inactive Data Containers.
- Self-healing: automatic discovery of errors and their corrections. For example: searching for files with stale properties[1] and their automatic refreshing, system recovery after Metadata Repository failure.

[1] In case of internal failure the file properties may become invalid. Dealing with this problem requires hands-on reaction. This is an internal DMS feature.

- Self-optimization: continuous monitoring and control of resource usage, which assures their optimal utilization, for example, automatic file replication, file transfer optimization in terms of speed and bandwidth.
- Self-protection: proactive identification and protection against the attacks. For example: active detection of incoming threats, response to the specific events in a form of a system message (mail, log journal etc.).

Owing to the above assumptions, we perceive a necessity to define two additional guidelines, pointing at directions of the building of novel data grid systems (generally distributed systems). These are:

- The component model – instead of building a monolith architecture and thus assigning all resources to a specific application, the software should be treated as a set of logical, reusable services that can dynamically utilize (and share) the underlying hardware resources. These services should be platform-, language-, and operating system-independent;
- Autonomic patterns – system design patterns should lead to building self-management service architectures, being able to runtime adaptation to the changing environment conditions. It indicates the presence of the event services, capture and sharing of state information between sub-systems, integrity and autonomy of self-management systems. This idea will be revealed further in the next sections.

With regard to the first notion (the component model), in the previous section we introduced the Data Management System as an example of a distributed system based on loosely coupled components. The implementation on the second assumption - autonomic patterns - involves an extension in the present DMS architecture. This topic will be described more precisely in the next section.

4.1 AC principles implementation

Before we get down to the autonomic computing implementation, we discuss the initial principles that lay down at the basis of the autonomic patterns. We base this list on the well-known CHOP model (section 2.1):

Self-configuration – distributed components configure themselves without any human intervention in the form of configuration dialogs or external files. We can assume that each element possesses a high-level description of its behavior in a standardized form and the address of the central information repository. This repository stores all information about the services and resources belonging to the distributed environment. A new element retrieves the appropriate information that it needs to function, configures itself on this base and then registers itself in the repository.

The Web Services platform (WSDL, UDDI, WS-Addressing and more) [17] seems to be the most viable option to implement self-configuration patterns.

Self-healing – we assume that the distributed environment should be capable of dealing with the failure of any of its components. It is important, however, to distinguish between the local and the global approach. The former is related to the creation of a reliable and robust single entity, which involves using the appropriate architectural techniques or hardware protection. The latter, which remains our interest, as-

sumes the presence of a monitoring element responsible for determining if the distributed components are performing properly, according to their desired behavior. If the monitoring element detects any inconsistent service, then it reacts, possibly terminating the failure element or updating appropriate records in the information repository.

Self-optimization – similarly to the previous case, we should also distinguish between local and global tuning aspects. It is obvious that each element must utilize the underlying resources efficiently but it does not assure that the whole environment will work properly. Therefore we assume that on the global level we will take advantage of the policy-based management [18]. It involves the presence of an autonomic manager which will perform the self-optimize actions according to the desired policy. The policy should be expressed in an abstract language, for example "On average, users will not wait more than 5 seconds for the response", and the autonomic element will translate it into the system commands (or workflow) and execute in the environment.

Self-protection – the self-protection aspects cover two distinct issues: undesirable system behavior due to bugs or other unexpected conditions and unauthorized access by attackers. Regarding the former issue, some of the self-healing or self-optimize patterns are suitable for protection from this type of event. For example, self-regenerative clusters may be useful when a single node is down because of internal failure. It is also recommended to take advantage of the intrusion detection system which is responsible for preventing from any unauthorized access. Similarly to the other computing systems, the autonomic environment requires strong security control. It can be realized by defining the security policies which are a part of the self-tuning policies described in the previous paragraph.

Event service. As a base for our autonomic architecture we will use the central log repository gathered events from the distributed components. The Toth Logging System, which is ready to use in various environments and accommodated to cooperate with different kinds of applications, perfectly fits our needs.

The main functionality, which is storing events coming from many wide-spread modules, fulfills only one basic assumption – a central message gathering. It does not treat the statistical analysis of this data. This feature is particularly important in reference to AC patterns when we must distinguish between many types of events which may occur in the distributed system. These can be, for example, situations in which:

- The user waits for data transfer longer than the expected value.
- The user failed to logon into the system.
- The amount of data transferred daily for one user exceeded a given value.

To handle these issues Toth provides a context analysis mechanism of collected messages. It is based on a set of attributes which are passed in the event body in a form of the key-value pairs. The sequence of operations, comprising the message processing and drilling for the attributes according to the specified criteria, is realized on the database level. These criteria may be a type of operation, name of the file, preferred file pr transfer protocol.

In this paragraph we have outlined the general concept of Toth architecture, and the next sections provide a detailed description of its main components with reference to the autonomic computing.

Predictive level. To achieve level 3 (predictive) of autonomic computation, Toth has been equipped with two specialized modules. First of all, it is the Knowledge Base built on the basis of recent activities of managed resources. This part of the application acts as a foundation of further actions and defines the global environment state. It is very important to note that this knowledge does not comprise only log messages from the registered modules, variables values or states and measurements. It has to be considered as a real knowledge, which is a base set of conclusions that are drawn from the collected data. To accomplish this assumption we have introduced the System Diagnostic Monitor – a separate Toth module characterized by the following features:

- It analyzes the gathered logs, monitors the system components and creates recommendations.
- It generates alerts on the basis of several thresholds. It assures proactive monitoring, which means reaction to problems before they may appear.
- It runs at regular intervals (autonomic control loop).

On the basis of the CHOP model (section 2.1), we can present a few examples of system rules which act as a rationale to create recommendations. These are:

- Self-optimization: if a file is accessed frequently, then it can be spread among different nodes.
- Self-configuring: if a request passed to distributed component finishes with a network error, then it may indicate its failure.
- Self-healing: if a file is locked more than reservation time, then it is probably stale[2].
- Self-protection: if the administrator tried to log in from the machine outside the secure zone, then it may indicate an attempt to break in.

The rules are encoded according to XML methodology and apply to the form of IF(condition) THEN (action). Additionally, we define a set of alerts in the system, which are triggered when a certain condition (threshold) takes place. It can be, for example, "low free space" which raises an alarm when the space usage at the Data Container is higher than 90 percent. By default, the alert notifications will be sent to the console, but Toth will also support the email or SMS notifications.

The predictive level ensures faster and better decision making providing appropriate recommendations for the professional staff members. But the realization of the autonomic computing vision involves to automate the processes of self-* procedures, which stands for the adaptive level.

Adaptive level. At level 4 (adaptive), the distributed environment can automatically take actions based on the available knowledge. The decisions are taken on the basis of the knowledge base and have to fulfill the assumed policy and defined base rules.

To provide a consistent view of this level of AC, we have to firstly define a term of a policy with reference to autonomic computing methodology. The anatomy of the

[2] File management within DMS is based on the reservation of physical storage on a specific amount of time

policy defines the system ability for high-level, broadly scope directives, which are translated into the specific actions to be taken by the elements. Policy-based management is an active research topic among the scientists. In the autonomic computing approach this refers to the policy-based self-management.

System rules introduced in the previous section actually constitute a basic form of policy (based on actions IF-THEN). In order to satisfy the requirements of AC level 4 we define a goal policy which describes the conditions to be attained without specifying how to achieve this (for example, the time of a file restoration after the failure must not exceed 60 seconds). This notion is much more flexible than system rules, because the human or autonomic element can perform specific actions in the monitored components without knowing of its inner behavior.

This set of rules and policies allows to define the demanded system characteristics and it is a base for performing self-* procedures. Actions that are taken during self-healing or self-tuning operations are performed by individual components. Thus it is necessary to equip these components in manageability interfaces which provide various ways to gather details and change the behavior of the managed resources.

The service-oriented architecture defines a number of standard interfaces, but in order to fulfill the autonomic computing requirements we need to provide the additional interfaces as well. Because our idea concerns the SOA model and grid technologies, we plan to base the final solution on the OGSA [10] architecture. In terms of the Web Services Description Language (WSDL), OGSA defines interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification.

We also define an additional Toth module which will take desired actions on distributed components. This module – the so-called Change Manager – will be responsible for two main actions:

- Planning – generating the appropriate change plan according to the assumed policy and recommendations (made by the system diagnostic monitor). The plan function can take on many forms, from a single command to a complex workflow.
- Executing – scheduling and performing the necessary changes to the monitored system. We must consider that part of the execution of the change plan involves updating the Toth knowledge base. It is necessary to indicate the actions that were taken as a result of the analysis and planning and how these actions affected the managed resources.

Autonomic level. Level 5 (autonomic) is closely defined with the business and industry demands. It is characterized by a closed loop with the business processes level, business policies and objectives governing the whole infrastructure operation. Users interact with the autonomic technology tools to monitor business processes, alter the objectives, or both.

At the highest level of autonomic computing we need to extend the meaning of the goal policy (see the previous section) and provide a way to automatic determination of the most valuable goal in any given situation. To achieve this intention we define the utility policy which makes use of an additional attribute – a value expressing the relative priority of a policy.

Now it is very complicated to create a system which would be 100 percent compliant with the highest AC level. We notice the fact that there is still a need for research in this area. Our team performed some effort in this direction and as a possible solution we see the combination of the SOA capabilities (service workflow, service bus, BPEL) and autonomic computing concepts. A detailed presentation of our visions, however, is beyond the scope of this article (it involves providing a solid background of the Service Oriented Architecture), so we shall outline only a brief description of this idea.

Let us suppose that we have an environment built according to the SOA principles. It means that the services and processes can be decomposed into workflows of activities and tasks that are used to realize them. According to one of the main SOA assumptions, these workflows are created, managed and monitored by professional staff which has specialized tools to perform these operations. Having such an environment, we can take advantage of the SOA concept and combine it with the AC model. It means, in short, that the recommendation performed by the system diagnostic monitor can be translated into the SOA-specific workflow in order to perform the desired activity in the system. This workflow may be automatically deployed on a specialized runtime engine and executed. Thanks to the existing tools designed to manage the SOA, professional staff have a possibility to analyze, check, redefine or monitor these autonomic activities.

5 Conclusions

In this paper we have described an approach to creating a distributed environment composed of loosely coupled components and capable of performing self-managing actions. This solution may constitute an example of a novel application (data grid system), which faces the problem of increasingly complex systems.

We have provided a step-by-step solution describing how to achieve the desired goal. We have started from the basic level of autonomic computing implementation and finished with the most advanced issues, referring to the business processes level and business policies. The DMSuite platform served us as a reference base for implementing autonomic patterns. As it was mentioned, the current release of this software supports the basic level of the AC model. But we are currently working on the extensions which were described in this article, and will implement a part of advanced autonomic computing technology. It will include the proactive reactions to some well-defined situations, detections of any "unusual" events, generating recommendations for administrators and many more.

There is still much to be done within the scope of the self-aware environments. This is not only because of the scale of distributed applications and systems, but also because QoS (Quality of Service) support needs to be specific to the requirements of individual end-users. In our opinion, in the near future much of research work will pursue the full vision of autonomic computing systems, and this will rely on aggregated grid resources and autonomic computing software platforms. This may be a crucial step to pass from the academic to the enterprise environment.

References

1. Rychlewski, J., Weglarz, J., Starzak, S., Stroinski, M., Nakonieczny, M.: PIONIER: Polish Optical Internet. Proceedings of ISThmus 2000 Research and Development for the Information Society conference. Poznan Poland (2000) 19-28
2. Bogdański M., Kosiedowski M., Mazurek C., Stroiński M.: Facilitating the process of enabling applications within grid portals. Grid and Cooperative Computing (GCC 2004) ed Jianhua Sun et all Proceedings of Third International Conference, Wuhan, Chiny, October 2004, Springer, Lecture Notes in Computer Science, 3251, pp.175-182
3. Kosiedowski, M., Malecki, M., Mazurek, C., Spychala, P., Wolski, M.: Integration of the Biological Databases into Grid-Portal Environments, Workshop on Database Issues in Bioological Databases DBiBD. Edinburgh UK (2005)
4. Grzybowski P., Mazurek C., Spychała P., Wolski M.: Data Management System for grid and portal services. Submitted to Grid Computing: Infrastructure and Applications special issue of The International Journal of High Performance Computing Applications (IJHPCA), Cardiff University, UK, http://progress.psnc.pl/English/DMS.pdf
5. Journal of Computational Methods For Science and Technology no. 12 vol. 1 – Grid Applications - New Challenges For Computational Methods
6. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S.: The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. J. Network and Computer Applications, 2000
7. Kephart J.O, Chess D.M., "The Vision of Autonomic Computing," Computer, vol. 36, no. 1, 2003, pp. 41–50
8. Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, Jeffrey O. Kephart,: An Architectural Approach to Autonomic Computing", International Conference on Autonomic Computing (ICAC'04), May 17 - 18, 2004.
9. Foster, I., Kesselman C., Tuecke S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 2001. 15(3): p. 200-222
10. Foster I., Kesselman C, Nick J.M., Tuecke S: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," a research paper, Globus Project; http://www.globus.org/alliance/publications/papers/ogsa.pdf
11. An architectural blueprint for autonomic computing, a white paper, IBM corporation, http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf, June 2005, third edition
12. Automating problem determination: A first step toward self-healing computing systems", a white paper, IBM corporation, http://www-03.ibm.com/autonomic/pdfs/Problem_Determination_WP_Final_100703.pdf, October 2003
13. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Prentice Hall, Upper Saddle River, NJ, USA (2004)
14. Ganak A. G., Corbi A. T.: The dawning of the autonomic computing era. IBM Systems Journal, 42(1):5–18, 2003
15. JMS, the Java Message Service, http://java.sun.com/products/jms/index.jsp
16. Data Management System Portal, http://dms.progress.psnc.pl
17. Weerawarana S., Curbera F., Leymann F., Storey T., Ferguson D. F.: Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall, Upper Saddle River, NJ, USA (2005)
18. Sloman M.: Policy Driven Management for Distributed Systems, Journal of Network and Systems Management, Vol.2 (1994)
19. SRS, the Sequence Retrieval System, http://www.biowisdom.com/solutions_srs.htm