

Adaptive Services Grid Deliverable D5.II-3

Research report about SLA fulfillment concepts and implementation

Peter Tröger

February 11, 2007



EU Project officer	Michel Lacroix		Lead Contractor of Deliverable	HPI-DCL
Program	Information Society Technologies		Contact Author	Peter Tröger
Strategic Objectives	Open development platforms for software and services		Co-Authors	Wolfgang Schult
Project Number	FP6 – 004617		Work Component	C-5
Contractual Milestone	M30		Deliverable Code	D5.II-3
Contractual Date	March 10, 2007		Deliverable Owner	Peter Tröger, HPI-DCL
Contractual Nature	Report		Deliverable Status	mandatory, key
Contractual Dissemination Level	Public		Intellectual Property Rights	Unaffected

DOCUMENT INFORMATION

Authors (Partner)	Peter Tröger (HPI-DCL), Wolfgang Schult (HPI-DCL)		
Responsible Author	Peter Tröger	Email	peter.troeger@hpi.uni-potsdam.de
	Partner 1 HPI-DCL	Phone	+49-331-5509 233
Co-Author	Wolfgang Schult	Email	wolfgang.schult@hpi.uni-potsdam.de
	Partner 2 HPI-DCL	Phone	+49-331-5509 234

Review Date	Reviewer	Partner	Board
	Christoph Ringelstein	UK	Replacement for Architecture Board Member
	Michael Eisenbarth	IESE	General Assembly
	Piotr Grobelny	Astec	General Assembly
	Steffen Staab	UK	Scientific Board

Keywords	
Abstract for dissemination	<p><i>The C-5 work component developed the Services Infrastructure (SI) subsystem, which forms the technological base for a unified atomic service invocation, monitoring, negotiation and deployment in ASG.</i></p> <p><i>SI is based only on wide-spread open source and commercial-off-the-shelf middleware technology (Axis, JBoss, IBM WebSphere, ActiveMQ, JMX) and proven Web service standards (Web Services Resource Framework). We applied a consistent stateful service model to the SI interfaces, development tools and runtime mechanisms. The resulting execution environment supports hosting, dynamic invocation and supervision of stateful internal and proxy atomic services.</i></p> <p><i>Based on the proven architecture of the C-5 Services Infrastructure, this deliverable describes possibilities for SLA fulfillment in the different components of our subsystem. It is split up in two parts: The first part discusses possible SLA fulfillment strategies in the C-5 Services Infrastructure layer. The second part is the result of a study on application of Aspect-Oriented programming tools and concepts in context of the ASG-C5 infrastructure layer.</i></p>

EXECUTIVE SUMMARY

The Adaptive Services Grid (ASG) approach towards semantic service provisioning is a solution to realize the agility and adaptiveness promised by SOA in practice. The C-5 work component developed the *Services Infrastructure* (SI) subsystem, which forms the technological base for a unified atomic service invocation, monitoring, negotiation and deployment in ASG.

Based on the proven architecture of the C-5 *Services Infrastructure*, this deliverable describes possibilities for SLA fulfillment in the different components of our subsystem. We categorize problem classes in such fulfillment strategies, and discuss some theoretical and practical approaches for achieving a negotiated service quality. Our research concentrates on re-using existing mechanisms and approaches from proven operating systems, based on resource-oriented control and partitioning mechanisms in the infrastructure.

This deliverable is split up in two parts: The first part discusses possible SLA fulfillment strategies in the C-5 Services Infrastructure layer. The second part is the result of a study on application of Aspect-Oriented programming tools and concepts in context of the ASG-C5 infrastructure layer.

TABLES OF CONTENTS

DOCUMENT INFORMATION	II
EXECUTIVE SUMMARY	III
TABLES OF CONTENTS	IV
1 INTRODUCTION	1
1.1 Positioning towards the ASG Service Provisioning Features.....	1
2 PART 1: SLA FULFILLMENT IN THE ASG SERVICES INFRASTRUCTURE.....	3
2.1 ASG Services Infrastructure Basics.....	4
2.2 Resource Types in the Services Infrastructure	6
2.3 Resource Scheduling.....	8
2.4 Resource Allocation.....	10
2.5 Resource Partitioning.....	12
2.6 Conclusion	15
3 PART II: DYNAMIC SERVICE ADAPTATION THROUGH AOP.....	16
3.1 Overview of AOP technologies	16
3.2 Examples for Dynamic Service Adaptation in .NET Execution Hosts	18
3.2.1 Monitoring service execution	18
3.2.2 Caching of complex service operations	19
3.2.3 Dynamic service adaptation.....	20
3.3 Conclusion	21
REFERENCES.....	21
PROJECT CONSORTIUM INFORMATION.....	24

LIST OF FIGURES

Figure 1: QoS Attribute Classification	3
Figure 2: Simplified Service Lifecycle in the ASG Services Infrastructure.....	5
Figure 3: Monitoring Data Flow in the ASG Services Infrastructure.....	7
Figure 4: Response times for Denic / Saferpay service in the ASG DSC scenario.....	11
Figure 5 Crosscutting Concerns in Service Components	16
Figure 6 Separating Concerns with Aspect Oriented Programming.....	17

1 INTRODUCTION

Nowadays, middleware technologies such as CORBA or J2EE often build the main platform for distributed IT infrastructures. More innovative approaches, e. g. Web services, obtain more and more acceptance and become more and more important.

Atomic services for a service-oriented software system, implemented with Web service interfaces, demand effective hosting, management and monitoring of their instances. The ASG C-5 work component provides the *Services Infrastructure (SI)* middleware as solution for unified dynamic service deployment, invocation, monitoring and hosting in the ASG architecture.

While Web services and service orientation could be considered as a business trend, a second trend emerges in industry: Business Process Outsourcing (BPO). BPO is the utilization of technology or specialist process vendors to provide and manage an organization's critical or non-critical enterprise processes and applications [2]. Outsourcing implicates to negotiate contracts between partners, so called Service Level Agreements (SLAs). Different definitions for quality-of-service (QoS) and service-level-agreement (SLA) can be found in literature. SLAs could encompass amongst others:

- Partners involved and their roles,
- Type and extent of service provisioning,
- Services usage costs,
- Expected quality of service,
- Legal issues, such as penalties and period of availability.

Depending on the origin of a particular author, these terms are used in a wide range of meanings. For this reason, the ASG C-4 and C-5 work components performed an in-depth analysis of possible QoS and SLA attributes in the past [1][3]. On a more technical level, implementation aspects of SLA monitoring and negotiation in C-5 were discussed by the author in [4].

Based on this previous work about theoretical and practical foundations of SLA's in the *Services Infrastructure*, we want to add a summary of possible resource-oriented SLA fulfilment strategies with this document. Even though most of the described technologies were only tested on a 'proof-of-concept' base from M24 to M30, the results provide a starting point for the continuation of our *Services Infrastructure* research after the finalization of the ASG project.

This deliverable is split up in two parts: The first part discusses possible SLA fulfilment strategies in the C-5 *Services Infrastructure* layer. The second part is the result of a study on application of Aspect-Oriented programming tools and concepts in context of the ASG-C5 infrastructure layer.

1.1 Positioning towards the ASG Service Provisioning Features

The ASG Service Provisioning Features are achievements on the technological level, which are appointed by the application of an ASG compliant platform for service

provision. On business level these achievements lead to a reduction of costs in development and maintenance. Furthermore the platform supports continuous adaptation of service centric applications towards environmental changes and upcoming business needs. The main features are

1. Seamless integration of heterogeneous external services
2. On-demand creation of service compositions
3. Reliable service provision with assured quality of service

This deliverable clearly concentrates on aspects of the third feature. It underlines the C-5 efforts of supporting reliable service provision with assured quality of service by describing possible solutions for SLA fulfillment on the technical level.

2 PART 1: SLA FULFILLMENT IN THE ASG SERVICES INFRASTRUCTURE

The *Service Level Agreement* term originates in legal contracts between business partners, where one party offers a service under negotiated properties to another party. Meanwhile, the term is increasingly used for a collection of technical service quality attributes, especially in the field of Web services [6].

Quality-of-service attributes are one of the most important parts of SLAs in a Web service environment. They describe non-functional demands on service functionality. Many QoS approaches deal only with performance-related quality criteria, like response time and execution duration. Only a few approaches consider advanced criteria's such as security, payment model or execution price. The central aspects of such a quality-of-service attribute classification were discussed in earlier ASG deliverables [1][3]. As one outcome, these reports introduced classification dimensions (see

Figure 1) for such attributes.

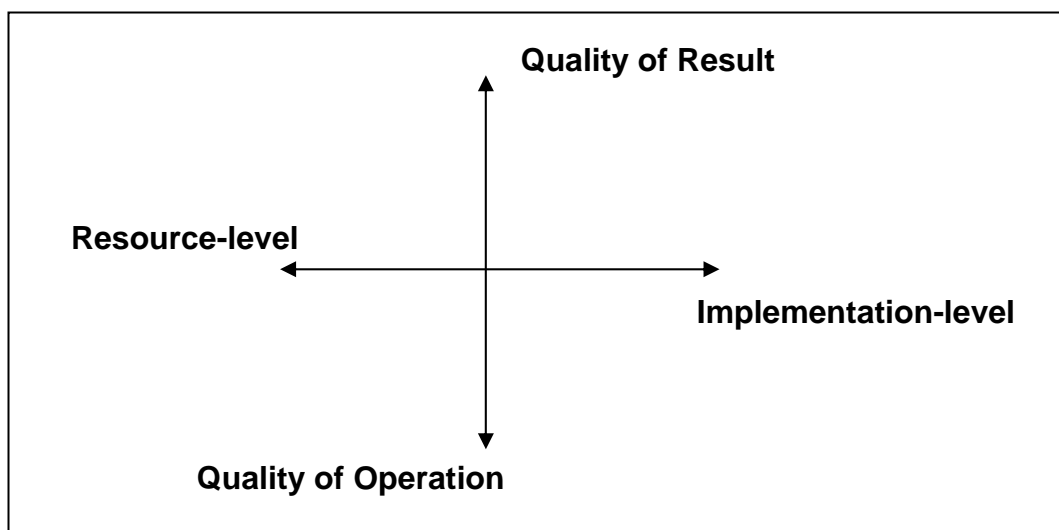


Figure 1: QoS Attribute Classification

The *quality of result* parameters of an SLA give a quantitative representation of non-functional requirements on results of the service operation. These parameters have a domain-specific nature, since they depend on the type of result returned by the service. One example is the resolution of a returned image. The *quality of operation* parameters expresses non-functional requirements on result delivery, such as response time.

As expressed by the second axis, QoS requirements of one or the other kind can be fulfilled either by the execution resources themselves, or by the particular service implementation. The fulfillment of *resource-level quality attributes* depends on mechanisms of the resource that is involved in the service execution. A popular example is the partitioning of network bandwidth for service interactions. The fulfillment of *implementation-level attributes* relies on specific service implementation features and control mechanisms. One example is the consideration of image resolution

demands in the service implementation code, even though it is not part of the functional interface. Typically, the *quality-of-result* attributes are realized with *implementation-level* mechanisms. *Quality-of-operation* attributes are either realized with mechanisms from the execution environment, or specific functionalities in the service implementation.

Since the implementation-level quality demands are bound to the particular kind of service, this must be completely handled in the service code internally. Service development research is a task of the C-3 work component tool chain - we therefore concentrate on resource-level fulfillment strategies in our C-5 research efforts.

Within the next section, we first start with a repetition of relevant ASG Services Infrastructure features. After this, we categorize fulfillment strategies and show examples for feasible approaches in each of the classes, in relation to the kind of resource there are influencing.

2.1 ASG Services Infrastructure Basics

The architecture and implementation of the ASG *Services Infrastructure* was demonstrated in earlier deliverables [35][3]. We therefore give only a short overview of the important concepts here.

Our *Services Infrastructure* approach extends the idea of stateless Web services with the concept of service instances on the infrastructure interface layer. Service implementations are deployed to the *Services Infrastructure* in binary format, together with a description of their acceptable execution environment (e.g. J2EE servlet container, .NET container). The *Services Infrastructure* stores this binary in a data management facility and provides a *service implementation reference* to the deployer. Client applications (such as the C-4 Adaptive Process Management components in ASG) then perform an explicit service instantiation for the service implementation through a factory operation. The resulting endpoint reference document describes all relevant data to perform a particular call to a logical service instance. The client uses a *WS-Addressing* enabled Web service stack (such as the C-5 client library), in order to perform operation calls to the particular service instance.

A logical service instance represents a stateful entity to the client, but does not necessary need to be realized by only one physical service instance on a particular execution host in the infrastructure. This slightly extends the idea of standard Web service frameworks, where services are referenced by an endpoint URI for a particular machine. Instead, all clients communicate with a coordination layer that routes SOAP requests (specifically the SOAP body) to a matching execution host (see Figure 3). This decoupling of *logical service instances* for the client, and *physical service instances* in the infrastructure, allows a dynamic *service placement* on execution hosts. This includes the usage of new service implementations (*service update*) with the same external behavior for existing logical instances.

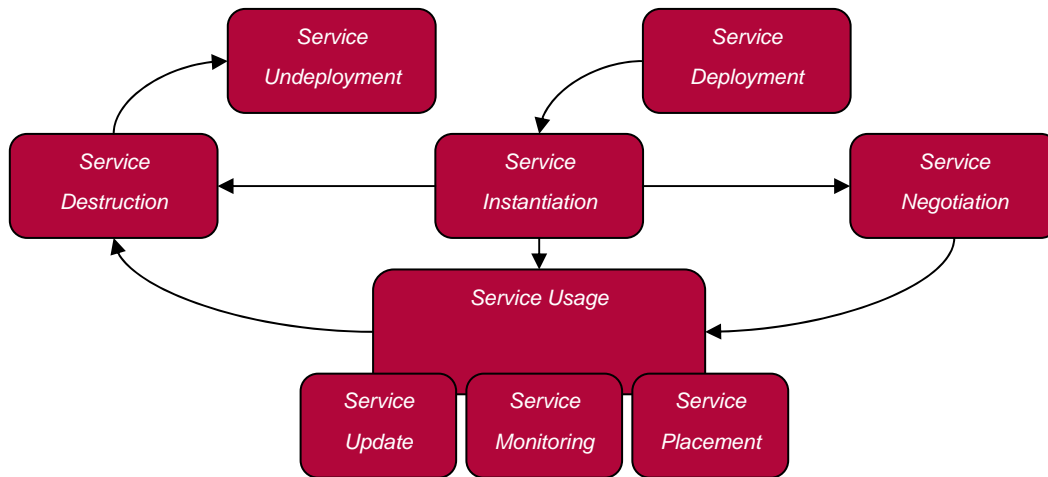


Figure 2: Simplified Service Lifecycle in the ASG Services Infrastructure

In our atomic service model, a logical service instance has queryable state, monitoring properties and a lifetime concept. Client applications, such as the C-4 workflow engine, are enabled to consider service state data in the service workflow decisions, since all according query operations become automatically part of the particular service interface. This implicit extension of the service operations is performed in conformance to the *WS-ResourceProperties* standard.

With a classical stateless execution model, the profiling of multiple service operation calls over a time period cannot distinguish if a measured behavior arises from the service implementation itself, from some delay in the setup procedures, or from the execution environment it runs on. With the logical instance approach, the client (such as the dynamic service profiling and negotiation components in ASG) can now differentiate between behavioral aspects observed for a service instance on a particular resource, the logical service instance in general, or the service implementation. In order to classify profiling results reasoned by some change in the resource binding, our monitoring data model supports the abstract identification of the resource which served the current / last request.

As another (and for this document important) aspect, the service instance factory approach allows the easy integration of service negotiation capabilities, which are researched by partner SWIN in the ASG project. A negotiation procedure can be performed for the same service implementation by different clients, which again may contract different parameters for the functional and non-functional aspects of service operation. With the concept of explicit service instantiation, each client negotiates service aspects with their own logical service instance. Due to the fact that each operation call anyway must refer to its service instance, the negotiated behavioral aspects are referenced automatically by each service call – and remain on the server side. Each of the infrastructure components is then enabled to access negotiated parameters for a service instance, in order to participate in the fulfillment of the according service level contract. The lifetime of the contract is bounded to the lifetime of the logical service instance. This also provides an implicit correlation between the contract lifetime and the validity period of monitored service properties.

A prototypical integration of negotiation algorithms from ASG partner SWIN was already implemented for the M24 ASG platform demonstrator. Anyway, there was no real effect of the negotiated parameters on service execution in the prototype run. While the consideration of service-related negotiation results needs to be part of the C-3 tool chain efforts, the consideration of negotiation results in the hosting environment must be realized by the C-5 *Services Infrastructure*. In the following text, we therefore focus on mechanisms for the assurance of resource-level quality parameters in the C-5 subsystem. The integration of the research results in the further prototypes of the *Services Infrastructure* will be part of ASG successor projects.

2.2 Resource Types in the Services Infrastructure

Following some earlier discussion [5], we define a *resource* in the infrastructure in the following way:

“A resource is an identifiable physical or virtual component of a system, which is limited in its availability or amount for a given time period, and which refreshes afterwards.”

Typical examples for resources are network bandwidth, CPU cycles, or even whole machines. The scheduling of such resources for different load conditions is usually described as *capacity planning*, which is an established research field for clusters and distributed systems [7]. Capacity planning denotes a tool-supported manual planning of typical workloads and their scheduling on available execution resources, before the actual infrastructure is installed and used. In the context of ASG, we want to provide a runtime-based load balancing and dynamic resource management approach instead. This applies to the ASG-understanding of a global dynamic platform, which adapts to changing runtime and environmental conditions in an automated way.

Most of the resource-level SLA fulfilment strategies from real-world use cases concentrate on the improvement of ‘quality of operation’ attributes, like response time, service request throughput or fault tolerance. The ‘quality of result’ attributes, like image output quality, numerical precision or information detail level, are typically not influenceable by infrastructure mechanisms. For the remaining text, we therefore concentrate on operational resource-level SLA attributes. This class of SLA parameters can be fulfilled if the infrastructure ensures a large enough amount of available resources, according to the particular negotiated parameter.

We distinguish in general between *host resources*, *server resources* and *service resources*, following our basic C-5 SI architecture concepts.

Hosts are physical machines with their typical resources, such as CPU, memory, storage or I/O capacities. For the current SI implementation, we support the monitoring of Windows, X86 Linux, and Sparc Solaris host systems. Monitoring agents collect the according performance and load indices from the host systems, and provide them based on our unified monitoring data model to the SI coordination layer.

Servers are container applications which act as execution environment for deployed and placed service implementations. One host can contain one or more servers, which themselves can contain one or more physical service instances during runtime. Typical server products have their own understanding and management of execution resources, for example with thread pools, object pools, or pre-allocated memory. We currently

support the monitoring and usage of J2EE application servers (specifically JBoss 4, IBM WebSphere 6 and Apache Geronimo 1.0), and a custom implementation of a .NET execution server.

A service is hosted by a particular server instance, and offers its functionalities after the deployment and placement through the coordination layer functionalities. The request processing component distributes incoming requests to available services, and therefore has a specific amount of service resources available. Services expose their own monitoring data, or public state information, to the central coordination layer storage.

The monitoring of all three resource types is implemented in the M24 prototype of the *Services Infrastructure*, and uses the same asynchronous messaging mechanisms for host, server and service monitoring information (see Figure 3).

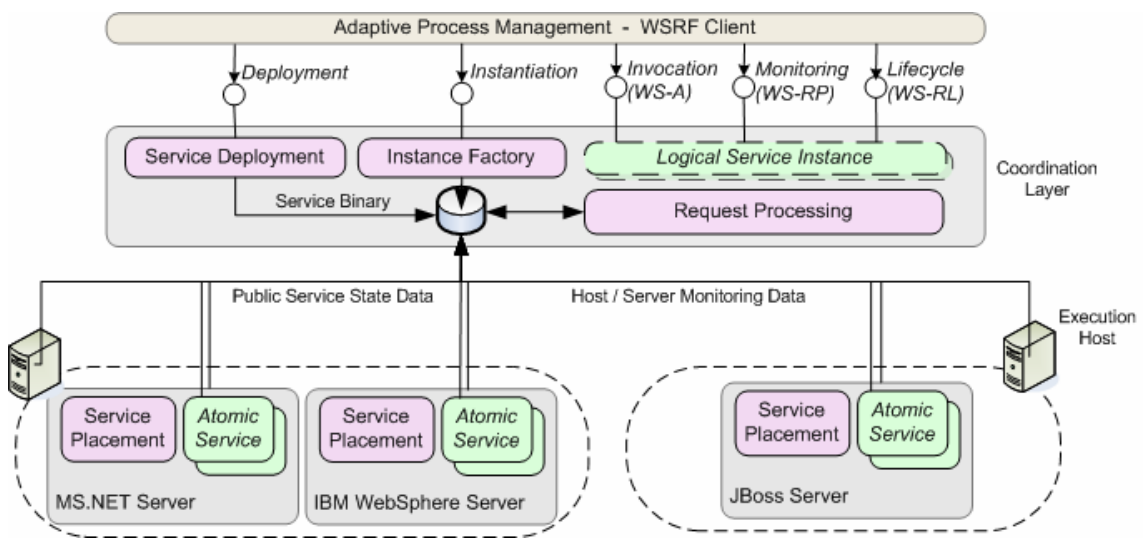


Figure 3: Monitoring Data Flow in the ASG Services Infrastructure

All three resource classes influence the non-functional aspects of service execution at run-time. An increased number of available physical service instances (== service resources) might allow a higher request throughput for the service type. Also an increased number of server resources could allow a more scalable behavior. An increased number of hosts might be beneficial in terms of fault tolerance aspects.

All three resource types can help in ensuring different SLA attribute classes. We take the most promising attribute types from various ASG project meeting discussions as starting point – the *performance-oriented*, *availability-oriented*, and *security-oriented* service quality attributes. Even though other SLA attribute classes might be possible, we will concentrate our discussion from here on around these three prominent examples.

Security attributes are usually handled by built-in middleware and operating system mechanisms on the resource-level. The aspects of security fulfillment are therefore only covered by according C-3 deliverables [13]. Availability-oriented attributes quantify the probability of a non-usable service for the client. Performance-related attributes specify the timing behavior of service operations.

Taking into account that a resource has a limited amount over a time period, we assume that resource-level quality attributes for performance and availability can be fulfilled by one of the following strategies: *resource allocation*, *resource partitioning* and *resource scheduling*.

2.3 Resource Scheduling

The fulfillment of performance guarantees on resource level through *resource scheduling* has a long tradition in several research areas. In cluster and grid computing, incoming computational jobs are scheduled on several execution resources [8]. Beside idle-time computing facilities [9], these resources are typically fixed in their amount. Modern cluster frameworks support a fine-grained prioritization of incoming jobs, depending on their finishing time, submitter identity or additional resource requirements. Scheduling algorithms such as back-filling and advanced reservation use job meta-data in order to maximize the utilization of the available resources. Only some of the schedulers support non-functional timing requirements, like latest start time or latest finishing time.

In the *Services Infrastructure*, the scheduling of service requests takes place in the *request processing* module in the coordination layer. A request for particular logical service instances must be forwarded to a matching physical service instance on an execution host. The physical service instances on the execution hosts act here as resources, which are assigned to different incoming requests within the scheduling operation. Since placement is centrally controlled by the coordination layer, the request processing always has an overview of used and unused physical service instances on execution hosts, assuming (as usual in scheduling) the absence of host or server failures.

Logical service instances in the C-5 layer can be used to negotiate non-functional requirements before the first operation call. In order to fulfill performance promises for such service instances (e.g. response time), the scheduling could prioritize requests for them in the queue of the service implementation. Scheduling algorithms for such scenarios are well-known from cluster processing research. All such approaches assume the availability of worst-case execution time data for service operations, either provided by the client, the service implementer or a statistical analysis of earlier service calls.

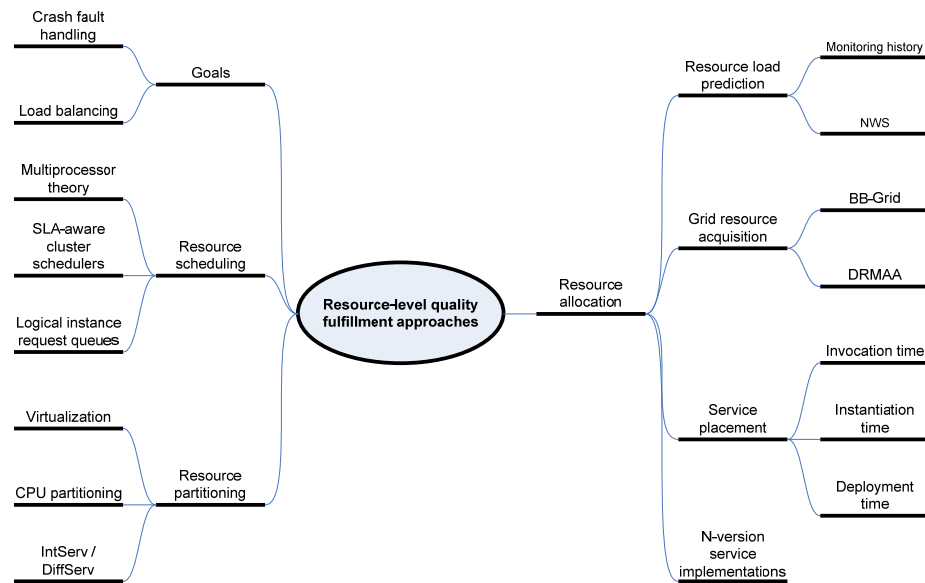


Figure 4: Overview of investigated SLA fulfilment approaches

The simplest (and the currently implemented) solution for such a request scheduler is a round-robin algorithm, with an own queue for each type of service implementation. Every time a new request for a logical service instance arrives, it is inserted at the end of the matching service type queue. When a physical service instance becomes available, then the request processing takes the request from the head of the according service type queue and sends the request to the according server. This allows the insertion of newly arrived requests at the head of the queue, if they are prioritized in some way due to a previous SLA negotiation cycle.

With a more fine-grained control of service execution on the servers, the request processing might even decide to interrupt a running request processing in some physical instance, in favor of a more important incoming service request. The existence of such a mechanism would allow applying aperiodic multiprocessor scheduling algorithms [10] to our problem. However, most of these algorithms assume the pre-emption of task without any overhead, which is an unrealistic assumption for a distributed system such as our infrastructure. The multiprocessor scheduling theory also typically assumes similar processors, which is contrary to our assumption of heterogeneous execution hosts, where physical service instances might not be available on all execution servers. Here, the multiprocessor scheduling theory is missing a consideration of additional request metadata. Several real-time researchers already identified the problem, and developed multi-dimensional models like Q-RAM and PRAM for the scheduling of QoS-annotated requests to resources.

Beside the consideration of resource scheduling for request load balancing, it can also be used to achieve an improved availability for logical service instances. Under the assumption of a reliable request processing and reliable central data storage, a detectable crash fault of execution hosts or servers can be handled by re-submitting the according request to another physical service instance. This is possible due to the C-5 atomic service programming model, which clarifies the management of persistent state outside of the physical service instance. However, it demands a transactional behavior

of the state storage mechanisms in the execution layer – the externalized state of the service should only be persisted if the service response was successfully retrieved by the request processing component. If not, then the state storage transaction should be rolled back and the request could be safely forwarded to another execution host.

As a third option for resource scheduling, typical job schedulers from cluster and grid computing could be re-used in order to provide a more sophisticated implementation of request processing under the described preconditions. One example is the MAUI scheduler [14], which works as external scheduler implementation without any relation to a specific resource manager. It supports relevant functionalities like fairshare (consider usage targets to limit resource access and adjust priority based on historical resource usage), exclusive allocation, priority management, scheduling optimization including backfill (better use of available resources by running jobs out of order), resource charging or throttling policies (limits on exactly what resources can be used at any given instant).

In sum it is clearly given that request scheduling is a well-researched topic in other areas of distributed systems. The scheduling of Web service requests, especially in our ASG subsystem, could benefit from algorithms and tools in the cluster and grid field.

2.4 Resource Allocation

The second possibility for the fulfillment of performance guarantees is the dynamic *resource allocation* in the Service Infrastructure. Our architectural concept already integrates the idea of dynamic *service placement*. Each server in the execution layer provides a placement web service, which allows the installation and removal of physical service instances during runtime.

An initial service placement could theoretically happen at three points in the service lifecycle - at deployment time, at instantiation time, or at invocation time. The delivery of the service implementation to an execution host at deployment time would contradict the late binding concept in our dynamic infrastructure. It might be needed when negotiated SLA attributes demand the usage of a specific host, e.g. with specific security properties.

A placement during the first service invocation would provide the latest possible binding. However, due to the fact that a service placement takes significantly longer than an operation call, the user would experience a performance penalty only on the first operation call. This would be a conflict to our transparent notion of a logical service instance, where the instantiation process promises to be the only needed ‘preparation step’ before a service invocation.

Within the current implementation, a service placement therefore happens during the service instantiation. The service factory checks if some of the execution layer servers already provides a matching physical service instance for the instantiated service type - if not, a placement operation on a chosen free server takes place, and the factory operation returns only successfully if at least one physical service instance was successfully created. This base mechanism is implemented in the current version of the *Services Infrastructure* demonstrator.

If the request processing detects a possible violation of SLA constraints due to overload of the existing physical service instances, it could decide to add another physical service

implementation for the overloaded service type. Specifically for performance-related SLA parameters, it must be ensured that the overhead for an additional placement does not outperform the possible improvement for the request turnaround time. Therefore, this SLA fulfillment strategy depends heavily on the precision of execution time estimation and on the time taken for a placement.

The analysis of ASG's M20 use case implementation showed that for constant input data, the jitter of response time for a (successful) service operation call is comparatively small (see examples in Figure 5). We therefore derive the general assumption that for typical Web service implementations, the usage of historical data for calculating the average service response time (on the same machine) is a valid solution. With the additional assumption of constant placement time (per machine), it would be possible to calculate the feasibility of an additional service placement during the request processing. According scheduling algorithms, based on a similar assumption, are part of current web server clusters research [11].

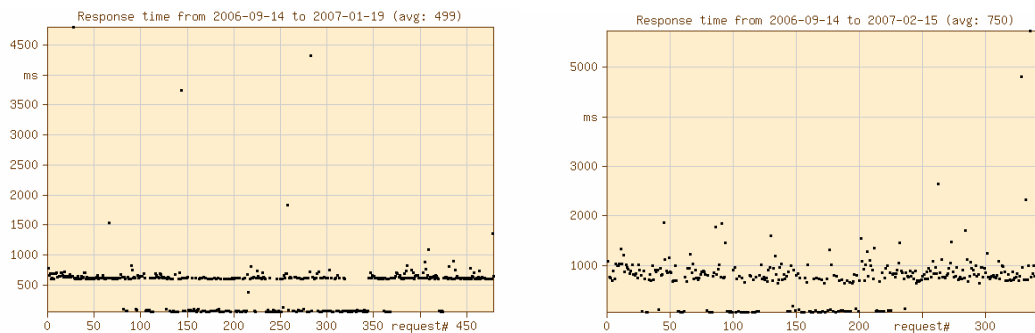


Figure 5: Response times for Denic / Safepay service in the ASG DSC scenario

Some of the scheduling techniques in literature also consider the usage of resource availability predictions, based on a stochastically analysis of historical usage data. The most prominent example for such a framework is the Network Weather Service (NWS) [15], which provides short-term predictions about future resource availability for network bandwidth and CPU resources. An earlier analysis in the context of ASG showed [16] that these predictions are only useful for very specific periodic work loads. The data quality decreases dramatically with the extension of the prediction time frame from some seconds to one minute. Our study showed that such generic resource prediction technologies do not provide a stable base for our service scheduling scenarios. Other theoretical approaches utilize locally weighted learning or automated code analysis for performance predictions.

As an extension of the dynamic resource allocation strategy, another possibility is to acquire new execution hosts during runtime through the usage of grid computing technologies. Similar to the service placement scenario, the coordination layer could decide to add another execution host machine from a pool of machines in a connected grid. According initial experiments with our M24 prototype where realized in the ASG testbed [17]. BB-Grid relies on a Globus 4 infrastructure, and allows the dynamic usage of machines from different universities in Germany through grid mechanisms. Our prototypical dynamic host allocation feature is based on the unified usage of grid

resources through the standardized *Distributed Resource Management Application API (DRMAA)* [18] from Open Grid Forum¹. The coordination layer deploys server implementations (for example a prepared JBoss 4 package) over the DRMAA interface to a grid host, and uses the integrated service placement functionality afterwards for the distribution of physical service instances to this new host. Typical firewall issues with remotely used machines did not appear in our first experiments, since Globus 4 anyway assumes the reachability of all interconnected grid nodes through WSRF-enabled Web service calls. This allows the communication of the *Services Infrastructure* coordination layer with such a grid host in the usual manner, namely through Web service calls to the Placement service or the particular physical service instances. Further experiments need to prove the feasibility of such an approach in terms of overhead for the automated server installation and execution over grid interfaces.

The dynamic availability of additional execution hosts to the *Services Infrastructure* can also improve the resilience of the platform against host or server crash faults. With many spare servers and hosts, it would be possible to apply redundancy concepts for improved service availability. While the classical replication approach (using the same implementation in multiple instances) is already an inherent concept of the *C-5 Services Infrastructure*, it might here also be possible to handle software-related faults by an N-version-programming [21] approach. Different implementations of the same service functionality could be used in parallel, in order to check the correctness of the computed results. This approach is supported by our explicit consideration of heterogenous execution environments for services. The N-version approach for service implementations then enables us to handle not only *crash faults*, but also *incorrect computation faults* [36] for service implementations through voting mechanisms in the service response processing.

2.5 Resource Partitioning

The third major idea for the fulfillment of non-functional service level guarantees in the *Services Infrastructure* is dynamic **resource partitioning**. Partitioning here means that the underlying resources of a host or server are divided into distinct parts, in order to ensure a limited but guaranteed amount of physical resources like CPU time or storage capacity for the service execution. The basic concepts for such solutions arose from the hard real-time and soft real-time research, where the ensured availability of hardware resources enables timeliness of software execution. Within the Service Infrastructure, we want to stick with our basic principle of using only commercial-of-the-shelf middleware and operating systems. For this reason, real-time-enabled operating system kernels, network stacks or Java runtimes are no feasible solutions for our infrastructure. Instead, we want to rely on user mode – based resource partitioning, which is either part of the operating system or works only with default operating system API's.

One of the most prominent examples for non-intrusive resource partitioning is the DiffServ algorithm [24]. It allows allocating chunks of network bandwidth to specific communication partners in an IP network. According experiments in the ASG testbed showed that today's networking infrastructures (specifically routers) demand explicit

¹ The author works in a leading position in the DRMAA working group, in order to ensure the best-possible application of ASG research outcomes on standardization documents.

configuration and management steps in order to utilize such functionalities in an efficient manner. All of the network elements through which a request packet flow passes — such as network interface cards, switches, routers, and bridges — must support the according networking QoS mechanism. If a network device along this path does not support QoS, the traffic flow receives the standard first-come, first-served treatment on that network segment. As a second issue, the operating systems of client and execution host must support the chosen network QoS mechanism. For Unix-based systems, both relevant protocols IntServ and DiffServ are supported by the TCP stack implementation. Windows 2000 supports both the IntServ and DiffServ models and includes resource reservation, admission control, and traffic control mechanisms. Windows XP and Windows Server 2003 support only the DiffServ model and include traffic control mechanisms [23]. Based on our experimental experience, we rate network partitioning to be an effective approach in case all participating network infrastructure in an ASG setup is centrally managed.

The second possibility for resource partitioning is the slicing of available CPU power on a host or server, giving a guaranteed amount of processing time to the executing service implementation.

In the last years there were some attempts made to use Windows NT as a hard real-time operating system. All available systems modify or extend parts of the core operating system to provide the necessary functionality. VentureCom offers their product RTX (real-time extension) for Windows NT and 2000 [29]. The original HAL is enhanced with some features to a real-time HAL, which provides fast timers (100 ns resolution), isolation of interrupts and the continuation of real-time processes in case of a windows stop-event (“blue screen”). The RTSS (real-time subsystem) is implemented as a windows device driver, providing an own thread scheduler. The interrupt handlers are implemented as threads, providing full control over interrupt priorities by setting the according thread priorities.

Beside commercial products and traditional hardware-based approaches in Mainframe systems, there are also different proposed solutions for CPU partitioning from research, especially from the area of multimedia streaming systems. Ford and Susarla implemented a framework for hierarchical scheduling [30], where different scheduling policies can coexist in the same system at the same time. The idea is realized by giving special threads the possibility to lease their execution time to other threads while they are waiting for an event to occur (timer interrupt). At this moment they act like a scheduler. The receiving thread could also donate the execution time to other threads, so it is possible to implement a hierarchical cascade of schedulers.

Goyal, Guo and Vin created a hierarchical scheduler on Solaris 2.4 [31]. The nodes in the scheduler tree are weighted. Every node scheduler has to ensure that the processor time is fairly distributed in proportion to the weight of each controlled node. This must be done regardless of dynamic changes in their resource needs. The authors used a specialized scheduling algorithm for the intermediate nodes in the tree – Start-time fair queuing (SFQ). It assures bounds on minimal throughput and maximum delay of the nodes. This is only possible with the quantification of the CPU bandwidth variances that can occur because of interrupts. The system also includes overrun mechanisms that ensure the protection between the applications if some of them are misbehaving.

In the implementation the authors changed the operating system scheduler for using it as one leaf scheduler. They showed that SFQ is usable for multimedia video applications.

The Processor capacity reserves were implemented by Mercer, Savage and Tokuda [32]. The authors focused on scheduling requests in micro kernel operating systems, for the realization they used RT Mach. The applications request CPU time (period and percentage of processor time) from the scheduler. This assignment is guaranteed if the request succeeds. The request, called reserve, can be changed dynamically during execution. Charging of CPU time for the thread does not only include the execution time of thread code but also times spend in user-level operating system services. The scheduler uses the calling threads reserve while the operating system service thread is executed. To avoid problems with the measurement of execution time, the authors used a specialized hardware timer board in their experiments. The implementation needs non-trivial kernel modifications.

Nieh and Lam implemented their SMART scheduler for multimedia applications with real-time constraints on Solaris 2.5.1. In the experiments they compared their SMART scheduler with the standard operating system scheduler, which is normally replaced by SMART. The kernel was modified to increase the timing resolution for both of them.

Lin, Chu and Nahrstedt realized their scheduling framework on Solaris, Irix, Windows NT, and Linux [33]. A multiprocessor user-level scheduler runs on the highest fixed priority in the system. The active controlled thread is boosted to the second highest priority in the system while the scheduler sleeps. After the end of the client quantum the scheduler awakes and lowers the priority to the lowest value, allowing the rest of the system to do its work. A client requests execution time from the scheduler with period length and CPU percentage. The main advantage is the pure user-level implementation, allowing an easily portable solution. No changes in the operating system kernel are needed.

Our earlier ASG-related study of major operating systems and their resource partitioning capabilities [34] clearly showed that most solutions come with huge overhead and unclear prerequisites on the scheduling and allocation strategy. We therefore decided to apply to the *Services Infrastructure* our own soft-real-time CPU partitioning framework, which is based on former research efforts [27][26][25] on a similar idea to Lin et.al.. Our scheduling server concept allows the slicing of CPU resources on standard operating systems (Linux, Mac OS X, Solaris, Windows, Mach) without any change to kernel or system libraries. We enabled the monitoring of the mapping of Java threads to their according operating system threads on the execution host, by using the standardized JVMTI interface of the Java runtime. This allows the partitioning of operating system CPU resources for all threads of the Java application server, especially for those running a physical service instance operation. Technical details and experimental results are out of scope for this deliverable, but will be part of a future publication.

As another new trend in resource partitioning, modern operating systems provide virtualization support for running multiple operating systems at the same time on one hardware node. From the viewpoint of the *Services Infrastructure* architecture, all such guest systems in a virtualization environment act as own host machine. The availability of resource-restricted virtual machines simply increases the number of available hosts,

which can enable the same fault tolerance and performance improvement features as already discussed in Section 2.4.

2.6 Conclusion

The distributed and dynamic architecture of the *C-5 Services Infrastructure* supports the extension with different resource-based SLA fulfillment strategies. In the context of this document, we discussed our classification of different types of execution resources and fulfillment strategies in the ASG use case, and introduced practical approaches from the area of resource scheduling, resource allocation and resource partitioning. The most promising ideas, such as the dynamic allocation of grid resources, will be implemented as part of the ASG C-5 successor project *Adaptive eXecution Platform* at HPI-DCL. This framework will include extensions for new execution host types (.NET, Corba), grid placement, request queuing and resource partitioning support. The resulting infrastructure will act as future platform for the *Distributed Control Lab*, and also provides a base for further research on dynamic SOA infrastructures in collaboration with Software AG and DaimlerChrysler research.

3 PART II: DYNAMIC SERVICE ADAPTATION THROUGH AOP

This second part of the D5.II-3 deliverable is the result of an additionally funded study on the application of Aspect-Oriented programming tools and concepts in context of the ASG-C5 infrastructure layer. We will show how AOP techniques can help in enriching .NET service implementations for an improved monitoring and resource control support. In contrast to the discussion of AOP usage for Java-based atomic services in C-3, we consider only our newly supported .NET service execution environment. The results of work therefore apply to the C-5 successor project *Adaptive eXecution Platform*.

3.1 Overview of AOP technologies

The concept of aspect-oriented programming (AOP) offers an interesting alternative for specification and realization of non-functional service properties. Non-functional properties are usually crosscutting-concerns of the service business logic. For example if an atomic service implementation needs to provide some operational monitoring values, a programmer would implement this usually within the business logic code. The problem of crosscutting leads to a tangled and scattered code (Figure 6).

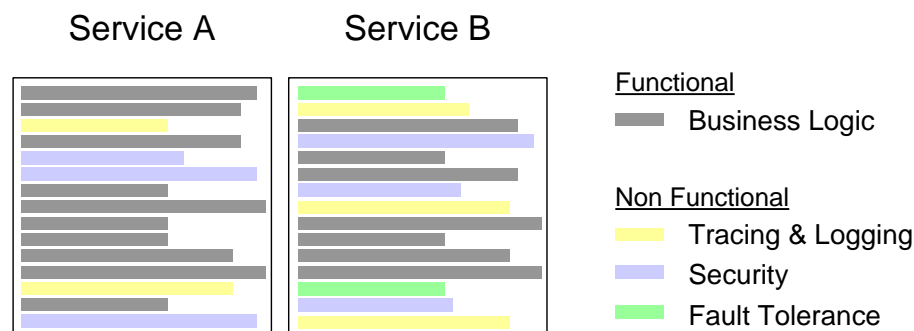


Figure 6 Crosscutting Concerns in Service Components

This example shows a typical service implementation. Besides the implementation of business logic, it also contains a non-functional code which is scattered and tangled. In a service environment, there are typically many services which have the same or similar non-functional properties. This is especially the case when an execution host demands that a service provides general monitoring or state properties which are usually not part of its business logic. This leads to a situation where a service programmer has to implement the same piece of functionality into each new service repeatedly.

With AOP it is possible to separate every non-functional property from the business logic and implement it in discrete units, so-called aspects (see Figure 7).

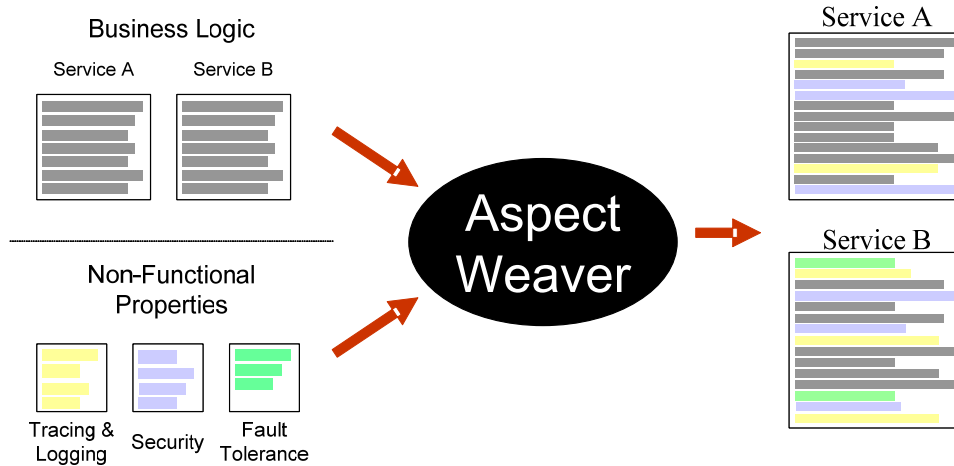


Figure 7 Separating Concerns with Aspect Oriented Programming

There are a variety of language extensions to deal with AOP. One of which, AspectJ, a Java extension, can be cited as the most prominent example. The central concept of most AOP-frameworks is a join-point model described in [38]. Our ASG-oriented study showed that the AspectJ join-point model has some drawbacks.

Mehmet Aksit has developed the Composition Filter's object model, which provides control over messages received and sent by an object [37]. In this model, the component language follows traditional object-oriented programming techniques. The Composition Filter's mechanism represents an aspect language that can be used to control a number of aspects including synchronization and communication. A recent implementation of this approach for .NET is Compose* [39]. Compose* uses its own filter definition language with its own build process, which complicates the integration of existing development processes. Furthermore, it is not possible to use sufficient aspect weaving control mechanisms during runtime of the .NET service container.

The JBoss application server includes a framework for aspect-oriented development called *JBoss-AOP*. It supports the instrumentation of Java classes at load-time through the manipulation of the utilized class-loaders. The *JBoss-AOP* hot-swap mode [Vasseur04] allows enabling aspects during runtime of an application. Application classes must be prepared before runtime. Due to the usage of JBoss as primary J2EE execution host in the *C-5 Services Infrastructure*, we see this AOP solution as a good candidate for introducing aspect-driven code into the atomic service implementations during runtime.

For our ongoing work on support for .NET service implementations, an according aspect weaver for .NET comes from the Rapier-Loom.Net project. The advantage of Rapier-Loom is that it allows for interweaving aspects upon instantiation of components, independently of the particular .NET executing environment. There is no need for a special framework or a particular application server like IIS. Aspects and join points are defined by .NET mechanism, which allows a consistent programming model for services and aspects. Since Rapier-Loom currently provides the most powerful AOP solution for .NET environments, it will be part of our currently developed .NET service container for the *C-5 Services Infrastructure*.

One important question with the application of AOP techniques in a dynamic hosting infrastructure is the appropriate point in time for interweaving aspects and components. AspectJ and Composition Filters are compile-time weavers, where aspect weaving happens before the component is deployed. In contrast, JBOSS-AOP and Loom.Net are classical examples for run-time time weaving. In these frameworks, aspect weaving happens either when the component is loaded or when the component is initially instantiated. The latter paradigm seems more preferable for service execution hosts like in the C-5 infrastructure. In the case of compile-time weaving, a deployment unit has to be interwoven before it can be deployed. Run-time weaving allows for interweaving aspects depending on the execution host's needs, which gives the flexibility of activating and deactivating aspects at runtime.

3.2 Examples for Dynamic Service Adaptation in .NET Execution Hosts

Within the environment of a service execution host, we have identified several non-functional service properties which might be implementable as aspects. Our first example is a monitoring aspect which can easily be used to collect generic service monitoring data. Secondly, we will present a result-caching aspect which enables acceleration of complex service operations. Finally we will show how aspects are useful in adapting service implementations into the environment of an execution host.

3.2.1 Monitoring service execution

AOP uses two fundamental concepts to interweave code (of non-functional properties) into a target (the service). The first concept is *advice* which means that code will be added at a particular *join-point*. A join-point is the call or execution of a service operation. The second concept is *introduction* or *inter-type declaration* and will be used to implement new interfaces, or data member to a target.

```
public class Monitoring : Aspect
{
    [IncludeAll]
    [Call(Advice.Before)]
    public void Start_Monitor<T>([JPContext] Context ctx, params object[] args)
    {
        ctx.Tag = DateTime.Now;
    }

    [IncludeAll]
    [Call(Advice.After)]
    public void Stop_Monitor<T>([JPContext] Context ctx, params object[] args)
    {
        TimeSpan executiontime = DateTime.Now - (DateTime)ctx.Tag;
        RegisterPerformanceData(ctx.CurrentMethod, executiontime);
    }
}
```

Figure 8: Example for Monitoring Aspect

Figure 8 shows an excerpt of a monitoring aspect. The following steps happen in the aspect implementation: Every operation of a service becomes interwoven with the *Start_Monitor* and *Stop_Monitor* method. The former is called before and the latter called after the service is executed. The *Start_Monitor* method records the time when the call to the service occurs. When the service finishes, the difference between the recorded time and the current time stamp is reported to the execution host (via a *RegisterPerformanceData* call of the ELS library).

3.2.2 Caching of complex service operations

Sometimes services have a very long response time. This can be caused by complex operations and a small network bandwidth. If the service behavior depends only on input data, and the input parameters of each service call could be repeated, it might be a feasible strategy to cache the result values of the operation call. The caching function maps the input parameters into a result-set, i.e. whenever a service call with the same parameters is executed twice; the aspect returns the cached value. Furthermore the caching aspect implements caching strategies like LRU, in order to prevent exhaustive use of memory. Figure 9 shows a draft for a possible implementation of the caching aspect. The aspect calls the original service method only, if a lookup to the cache fails.

```

public class CachingAspect:Aspect
{
    [IncludeAll]
    [Call(Advice.Around)]
    public T Cache<T>([JPContext] Context ctx, params object[] args)
    {
        object res = MethodCache.GetValue(ctx.CurrentMethod, args);
        if (res == null)
        {
            res = ctx.Invoke(args);
            MethodCache.AddValue(ctx.CurrentMethod, args, res);
        }
        return (T)res;
    }
}

```

Figure 9: Excerpt from a Caching Aspect

The applicability of such an aspect to service implementation could be specified by the service developer in the ASG C-5 deployment descriptor.

3.2.3 Dynamic service adaptation

A service programmer often asserts that various resources are available on an execution host, which might not be possible in a dynamic execution environment such as the *C-5 Services Infrastructure*. For example, if a service is deployed behind a firewall, it is impossible to acquire a direct TCP connection to the coordination layer or external services, which the implementation acts as proxy for. It could also be a problem if a service tries to use the file system, but is restricted by the underlying operating system.

Typically, the service developer has to manage all such possible error cases. This approach is obviously error-prone in a dynamic hosting environment, because the programmer would have to consider all the possible variations of the underlying system configuration. With aspect-oriented techniques, typical problems of legacy resource access or networking could be resolved by an according aspect implementation. The idea is to interweave an aspect that patches, depending on the system configuration, the resource access to the needs of the underlying system configuration. In the case of the TCP-connection problem, the aspect could open a specific channel to a proxy server to pass the firewall. For the file system case, the aspect could forward the access operation to a protected area of the file system on an unrestricted site (i.e. a 'temp' folder).

```
public class CachingAspect:Aspect
{
    [IncludeAll]
    [Create(Advice.Around)]
    public T CreateResource<T>([JPContext] Context ctx, params object[] args)
    {
        if (Environment.IsRestricted)
        {
            return Environment.GetRestrictedResource(typeof(T), args);
        }
        return (T)ctx.Invoke(args);
    }
}
```

Figure 10: Resource Access Alternation Example

Figure 10 shows the basic principle: The aspect needs to be interwoven into the creation of new resource objects (like TCP sockets). If there is a restriction, the aspect can create the proper version of the resource object. This depends on the particular execution host and server, which need to configure the aspect implementation accordingly. For the service developer, the access to legacy resources remains the same as usual. This whole concept is especially important with non-EJB containers for service implementations, since only the EJB standard restricts the programming model in order to avoid legacy resource access. All other types of atomic service implementations (e.g. servlets or .NET assemblies) can benefit from such an automated adoption of service implementations to the needs of the *Services Infrastructure*.

3.3 Conclusion

The usage of aspect-oriented programming techniques facilitates a seamless integration of non-functional properties into services. We have evaluated dynamic aspect weavers, and showed three exemplary aspects which solve specific problems of non-EJB service implementations in the C-5 Services Infrastructures. The results of our study, especially the application of the Rapier-Loom .NET aspect weaver, or now part of the C-5 successor project *Adaptive eXecution Platform*.

REFERENCES

- [1] Jun Yan, Mariusz Momotko, Guido Lares: Report on the requirement analysis and evaluation of service agreement negotiation and re-negotiation. ASG Key Deliverable D4.I-1, February 2005.
- [2] Markus Debusmann: Modellbasiertes Service Level Management verteilter Anwendungssysteme. Online Dissertation, March 2006.
<https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-200603157573>
(Last visited March 2007)
- [3] Peter Tröger, Marcus Flehmig, Helmut Jorke: QoS criteria for the services within the Adaptive Services Grid, ASG Key Deliverable D5.II-6, July 2005.
- [4] Marcus Flehmig, Peter Tröger, Alexander Saar: Design and Integration of SLA Monitoring and Negotiation Capabilities, ASG Key Deliverable D5.II-7, August 2006.
- [5] Alexander Saar. Leistungsüberwachung von Web-Service-Anwendungen in heterogenen Java Enterprise Umgebungen. Master thesis, December 2006.
- [6] Keller, A., Ludwig, H., The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management, Special Issue on E-Business Management, Volume 11, Number 1, Plenum Publishing Corporation, March, 2003
- [7] Daniel A. Menasce and Virgilio A. F. Almeida. Capacity Planning for Web Services: Metrics, Models, and Methods, 2nd Edition. Prentice Hall PTR, 2001. ISBN-10: 0130659037.
- [8] Jarek Nabrzyski, Jennifer M. Schopf and Jan Weglarz. Grid Resource Management. State of the Art and Future Trends. Kluwer Academic Publishers, 2004.
- [9] M.J. Litzkow, M. Livny and M.W. Mutka. Condor - A Hunter of Idle Workstations. In Proceedings of the Eighth International Conference on Distributed Computing Systems, page 104-111, 1988
- [10] Björn Andersson, Tark Abdelzaher and Jan Jonsson. Partitioned Aperiodic Scheduling on Multiprocessors. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03). April 22 - April 26, 2003. Nice, France.
- [11] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko and A. Tantawi. Dynamic placement for clustered web applications. In Proceedings of the 15th international conference on World Wide Web, 2006, pages 595-604, ACM Press, New York, USA.
- [12] Jane W. S. Liu. Real-Time Systems. Prentice-Hall Inc. 2000. ISBN 0-13-099651-3.
- [13] Rüdiger Grimm. SOA Security. ASG Key Deliverable D3.K-9, January 2007.
- [14] David Jackson, Quinn Snell and Mark Clement. Core Algorithms of the Maui Scheduler. In Lecture Notes in Computer Science, volume 2221, page 87ff. 2001.
- [15] Rich Wolski and Neil T. Spring and Jim Hayes. The Network Weather Service: a distributed resource performance forecasting service for metacomputing. In Future Generation Computer Systems, volume 15, pages 757-768, number 5-6, 1999.
- [16] Marcus Roscher. Prediktionsmechanismen in Grid-Umgebungen. Master thesis. July 2005.
- [17] Peter Tröger. Testbed demonstrator for the project partners. ASG Internal Deliverable D5.III-2. August 2005.
- [18] Peter Tröger, Hrabri Rajic, Andreas Haas, Piotr Domagalski. Standardization of an API for Distributed Resource Management Systems. To appear in Proceedings of the Seventh IEEE

- International Symposium on Cluster Computing and the Grid — CCGrid 2007. Rio de Janeiro - Brazil / May 14-17, 2007.
- [19] M. Iverson, F. Ozguner and L.Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In *Proceedings of the Heterogeneous Computing Workshop*, 1999.
- [20] V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, M. Hereld, I. Judson, R. Stevens. Prophecy: Automating the modeling process. In *Proceedings of the Third International Workshop on Active Middleware Services*, 2001.
- [21] Algirdas A. Avizienis. The Methodology of N-Version Programming. Chapter 2 of *Software Fault Tolerance*, M. R. Lyu (ed.), Wiley, 23-46, 1995
- [22] Kelvin Nilson. Adding real-time capabilities to Java. *Communications of the ACM*. volume 41, issue 6 (June 1998), pages: 49 - 56, ISSN:0001-0782
- [23] Microsoft Corporation. What Is QoS? Microsoft TechNet. March 28, 2003.
<http://technet2.microsoft.com/WindowsServer/en/library/1c1f53a6-da9e-496f-be84-b91e2763dbeb1033.aspx> (Last visited March 2007)
- [24] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W.Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998
- [25] Peter Tröger. CPU partitioning on Windows NT and SUN Solaris. Term paper, June 2002
- [26] Andreas Polze, How to Partition a Workstation. in *Proceedings of Eight IASTED / ISMM International Conference on Parallel and Distributed Computing and Systems*, Chicago, USA, October 16-19, 1996, publication of IASTED, pp.: 184-187(4), ISBN 0-88986-213-3.
- [27] Jan Richling. Scheduling Server for Responsive Computing. In *HUB Informatik-Berichte No. 94/97*, Institut für Informatik, Humboldt-Universität zu Berlin, December 1997.
- [28] Dr. Martin Timmerman. Windows NT as Real-Time OS ? In *Dedicated Systems Magazine*, 1997
<http://www.dedicated-systems.com/magazine/97q2/winntasrtos.htm> (Last visited March 2007)
- [29] Microsoft Corporation. Hard Real-Time with Venturcom RTX on Microsoft Windows XP and Windows XP Embedded.
<http://msdn2.microsoft.com/en-us/library/ms838583.aspx> (Last visited March 2007)
- [30] Bryan Ford, Sai Susarla. CPU Inheritance Scheduling. in *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, October 1996, pp. 91-106
- [31] Pawan Goyal, Xingang Guo, Harrick M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. in *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, October 1996.
- [32] C. W. Mercer, S. Savage and H.Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994
- [33] Klara Nahrstedt, Chih-han Lin, Hao-hua Chu. A Soft Real-time Scheduling Server on the Windows NT, in *Proceedings of the 2nd USENIX Windows NT Symposium*, Seattle, Washington, August 3-4, 1998. <http://cairo.cs.uiuc.edu/software/DSRT-2/dsrt-2.html> (Last visited March 2007)
- [34] Matthias Lendholt. Ressourcenpartitionierung für Grid-Systeme. Master thesis. January 2005.
- [35] Peter Tröger, Harald Meyer, Ingo Melzer, Marcus Flehmig. Dynamic Provisioning and Monitoring of Stateful Services. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies - WEBIST 2007*. Barcelona - Spain / March 3-6. 2007.
- [36] Jean-Claude Laprie and Brian Randell and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions On Dependable And Secure Computing*, 1, January – March 2004.
- [37] M. Aksit, L. Bergmans and S. Vural An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach, ECOOP '92, LNCS 615, Springer-Verlag, pp. 372-395, 1992.
- [38] AspectJ Homepage - <http://www.aspectj.org/> (Last visited March 2007)
- [39] Compose* Homepage - <http://janus.cs.utwente.nl/twiki/bin/view/Composer/WebHome> (Last visited March 2007)

- [40] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect oriented programming. In European Conference on Object-Oriented Programming (ECOOP), Finland, June 1997. Springer Verlag LNCS 1241.
- [41] Andreas Polze and Dave Probert. Teaching Operating Systems: The Windows Case to appear in Proceedings of ACM SIGCSE Technical Symposium, Houston, TX, March 2006.
- [42] Andreas Rasche, Wolfgang Schult, and Andreas Polze. Self-Adaptive Multithreaded Applications - A Case for Dynamic Aspect Weaving in Proceedings of the 4th Workshop on Adaptive and Reflective Middleware (ARM 2005), Grenoble, France - November 28, 2005
- [43] Wolfgang Schult and Andreas Polze. Speed vs. Memory Usage - An Approach to Deal with Contrary Aspects in Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (AOSD2003).
- [44] Wolfgang Schult and Andreas Polze Aspect-Oriented Programming with C# and .NET in Proceedings of International Symposium on Object-oriented Real-time distributed Computing (ISORC) 2002, pp. 241-248, Crystal City, VA, USA, April 29 - May 1 2002.
- [45] Alex Vasseur, Dynamic AOP and HotSwap.
http://blogs.codehaus.org/people/avasseur/archives/000615_dynamic_aop_and_hotswap.html, 2004.
(Last visited March 2007)

PROJECT CONSORTIUM INFORMATION

Partner	Acronym	Contact
University of Potsdam, Germany		Dr. Regina Gerber Universitaet Potsdam Am Neuen Palais 10 D-14469 POTSDAM Germany Email: rgerber@rz.uni-potsdam.de Tel: +49-331-9771080
University of Leipzig, Germany	 Faculty of Economics and Management Information Systems Institute	Prof. Bogdan Franczyk Universitaet Leipzig Ritterstrasse 26 D-04109 LEIPZIG Germany Email: franczyk@wifa.uni-leipzig.de Tel: +49.341-33720
University of Innsbruck, Austria		Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck Austria Email: dieter.fensel@deri.org Tel: +43-512-5076488
Fraunhofer IESE, Germany	 Fraunhofer Institut Experimentelles Software Engineering	Dr. Dirk Muthig Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e. V. Hansastrasse 27C, D-80686 MUENCHEN Germany Email: muthig@iese.fhg.de Tel: +49-6301-707161
DaimlerChrysler Research, Germany		DI Jens Weiland DaimlerChrysler AG Postfach 2360 D-89013 Ulm Germany Email: jens.weiland@daimlerchrysler.com Tel: +49-731-5052404
HPI at University Potsdam, Germany		Hasso-Plattner-Institut fuer Softwaresystemtechnik gGMBH Prof.-Dr.-Helmert-Strasse 2-3 D-14482 POTSDAM Germany Prof. Mathias Weske Email: Mathias.Weske@hpi.uni-potsdam.de Tel: +49-331-5509191 Prof. Andreas Polze Email: andreas.polze@hpi.uni-potsdam.de Tel: +49 331 5509 231
NUIG, Ireland	 Ollscoil na hÉireann, Gaillimh National University of Ireland, Galway	Prof. Christoph Bussler National University of Ireland Science and Engineering Technology Building Galway Ireland Email: chris.bussler@deri.ie Tel: +353-87-6826940

<p>Swinburne University, Australia</p>		<p>Prof. Ryszard Kowalczyk Swinburne University of Technology PO Box -218 AUS-3122 HAWTHORN Australia Email: rkowalczyk@it.swin.edu.au Tel: +61-39-2145834</p>
<p>Thueringer Anwendungszentrum fuer Software-, Informations- und Kommunikations-technologie GmbH, Germany</p>		<p>DI Holger Krause Thueringer Anwendungszentrum fuer Software-, Informations- und Kommunikations-technologie GmbH Langewiesener Strasse 32 D-98693 ILMENAU Germany Email: Krause@transit-online.de Tel: +49-3677-845109</p>
<p>NIWA, Austria</p>		<p>DI Alexander Wahler NIWA-WEB Solutions Niederacher & Wahler OEG Kirchengasse 13/1a A-1070 VIENNA Austria Email: wahler@niwa.at Tel: +43-131-9584311</p>
<p>Telenor Communications II AS, Norway</p>		<p>Dr. Rolf. B. Haugen Telenor ASA Snaroyveien 30 N-1331 FORNEBU Norway Email: rolf-bjorn.haugen@telenor.com Tel: +47-900-51101</p>
<p>Siemens AG, Germany</p>		<p>DI Klaus Jank Siemens AG Corporate Technology, Software and System Architecture Otto-Hahn-Ring 6 D-81730 MUENCHEN Germany Email: klaus.jank@siemens.com Tel: +49-89-636-50573</p>
<p>Rodan Systems, Poland</p>		<p>Mariusz MOMOTKO Rodan Systems Spolka Akcyjna Ul. Pulawska 465 PL-02-844 WARSZAWA Poland Email: Mariusz.Momotko@rodan.pl Tel: +48-58-5502024</p>
<p>University Jyväskylä, Finland</p>		<p>Prof. Jari Antti VEIJALAINEN Jyvaskylan Yliopisto Seminaarinkatu 15 FI-40014 JYVASKYLA Finland Email: jari.veijalainen@titu.jyu.fi Tel: +358-14-2603021</p>
<p>Telekomunikacja Polska, Poland</p>		<p>Bogdan BANSIK Telekomunikacja Polska S.A. Ul. Obrzezna 7 PL-02-691 WARSZAWA Poland Email: Bogdan.Banskiak@telekomunikacja.pl Tel: +48-22-6995340</p>

Marketplanet, Poland		Otwarty Rynek Elektroniczny S.A. Ul. Domaniewska 41 PL-02-672 WARSZAWA Poland Email: info@marketplanet.pl Tel: +48 22 576 88 00
ASTECSp. z o.o., Poland		Janusz MICHALEWICZ ASTECSp. Z O.O. Ul. Piaskowa 14 PL-65-209 ZIELONA GORA Poland Email: J.Michalewicz@astec.com.pl Tel: +48-68-3298031
The Poznan University of Economics, Poland		Prof. Witold ABRAMOWICZ Akademia Ekonomiczna W Poznaniu Al. Niepodleglosci 10 PL-60-967 POZNAN Poland Email: W.Abramowicz@kie.ae.poznan.pl Tel: +48-61-8569333
FH Furtwangen, Germany		Prof. Ulf Schreier University of Applied Sciences Furtwangen Rogert-Gerwig-Platz 1 D-78120 FURTWANGEN Germany Email: schreier@fh-furtwangen.de Tel: +49-7723-920153
Polska Telefonía Cyfrowa, Poland		Longin BRZEZINSKI Polska Telefonía Cyfrowa SP. Z O.O. Al. Jerozolimskie 181 PL-02-222 WARSZAWA Poland Email: lbrzezinski@era.pl Tel: +48-22-4135808
University of Koblenz-Landau		Prof. Steffen Staab Institute of Computer Science Universität Koblenz-Landau PO Box 201 602 56016 Koblenz Germany Email: staab@uni-koblenz.de Tel: +49-261-287 2761
Erik Lillevold		Erik Lillevold Åsheimneien 33 2016 Frogner Norway Email: erlille@online.no Tel: +47-9134-4641