# Service Adaptivity through Cross-Domain Reconfiguration of Non-Functional Properties

Josef Spillner
Technische Universität Dresden
Chair for Computer Networks
01062 Dresden, Germany
josef.spillner@tu-dresden.de

Iris Braun
Technische Universität Dresden
Chair for Computer Networks
01062 Dresden, Germany
iris.braun@tu-dresden.de

Alexander Schill
Technische Universität Dresden
Chair for Computer Networks
01062 Dresden, Germany
alexander.schill@tu-dresden.de

## ABSTRACT

Adaptive service execution platforms optimise the consumption of resources by allocating and prioritising resource access according to the requirements of installed services. Service descriptions help in understanding what those requirements are, and contracts are used to estimate and constrain the impact of the requirements. However, service developers rarely specify the constraints in terms of resource use, and the platforms don't have sufficient information about service-specific domain properties. We introduce a reconfiguration mechanism which combines both views based on a cross-domain type system and show the feasibility of its inclusion into a mainstream service execution platform.

## Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Statistical computing; H.3.5 [**Online Information Services**]: Web-based services—*adaptive web services*

## General Terms

Experimentation, Performance

## Keywords

Adaptivity, reconfiguration, web services, middleware, service-oriented architectures

## 1. PROBLEM STATEMENT

Service developers think in terms of the domain a service belongs to. The design and configurability of a flight booking service usually involves variable internal and external properties such as *P(number of flights for display)*. An image processing service may involve *P(resolution)* and *P(compression)* properties. Such a design matches the users' expectations regarding domain-specific flexibility and is popular with developers through domain-specific languages. On the contrary, only generic properties like QoS are understood by execution platforms in the context of contract-bound adaptive service execution. The challenge which providers of such platforms face is keeping the guaranteed properties, avoiding both up-front costs (purchase of extra hardware) and resulting costs (financial obligations due to broken contracts). A number of adaptation strategies allow flexible service, system, network and contract changes when contracts run into the risk of being violated. Reconfiguration is one such strategy which targets the variability of services. Its goal is to modify the values of generic, technical properties such as *P(memory consumption)* so that eventually all active service usage contracts can be fulfilled.

This paper introduces a reconfiguration mechanism for services which accounts for domain-specific properties in addition to generic ones. An overview on the existing approaches will be given in the next section. The concept of cross-domain reconfiguration will then be introduced and explained with examples, tested in a a simulation framework and evaluated by an integration into a Servlet container.

## 2. OVERVIEW ON EXISTING RECONFIGURATION APPROACHES

The adaptation of service execution to constraints imposed by the environment and by contracts is a prominent topic in web service research. Strategies include finding better services in compositions, adding resources dynamically, renegotiating contracts and reconfiguring service parameters. Reconfiguration is a well-established adaptation mechanism on the operating system, middleware and application levels. Most of the approaches regarding dynamic reconfiguration of services found in literature can be classified according to figure 1 which has been derived by extension from an earlier scheme of reconfiguration of components and component containers [5]. In addition, the approaches can be differentiated by the need for cooperation (cooperative and mandatory reconfiguration).

When services expose non-functional properties (NFPs) for external access, reconfiguration can be performed without the need for cooperation from the service. In [2], an adaptive grid scheduling mechanism based on historic monitoring information and application-provided hints is introduced. This makes the system adapt to the application, which is appropriate for grids, but not as much for situations


*MAI* 2009, June 12, Lisbon, Portugal

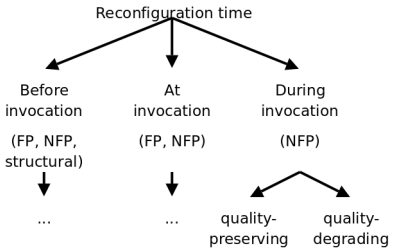


**Figure 1: Reconfiguration mechanism points in time and associated effects**

where the middleware itself cannot scale. Transparent component reconfiguration, including structural changes, has been proposed for J2EE components [5]. This approach affects only new invocations, while old invocations either keep their previous configuration or need to terminate at a synchronisation barrier prior to the reconfiguration effect. Functional reconfiguration at invocation time has been proposed for object request brokers [3]. While this is an improvement and enables SLAs to govern the mechanisms, the proposal also omits the data transmission variability during the invocation which is increasingly important for long-term connections and streaming web services.

In contrast, reconfiguration concepts in other areas such as operating systems are more sophisticated. Modular kernels use functional reconfiguration to adapt to newer hardware and features such as file systems. Furthermore, modern code swapping mechanisms like the Linux `kexec()` interface even allow for an online structural reconfiguration by replacing kernel code structures with newer versions without the need to reboot [6]. The advantage of uninterrupted operation is directly perceived by users. In addition, the Linux kernel offers an interface `/dev/mem_notify` for subscriptions to memory pressure signals to reduce the usual risk of random applications being terminated when the system runs out of memory. Using the interface, the kernel will coordinate memory usage reduction cooperatively by sending notifications to subscribed applications, although without any context information. Applications have to guess the amount of memory they are supposed to release. Therefore, it is usually only applied to discarding redundant data like caches.

On the middleware implementation level, most service containers are limited to initial configuration and do not offer interfaces for reconfiguration at or after invocation and instantiation time. This means that individual, SLA-governed reconfiguration remains an opaque service-level concern instead of handling it transparently in the middleware. Even modern service-oriented executable processes cannot be configured dynamically per se. Extensions exist to rectify this situation and bring variability into declarative formats [4].

We thus argue that three advancements to web service reconfiguration are needed to increase the impact of adaptive service execution. First, a concept for a transparently enforced cross-domain reconfiguration of services is needed. Second, any mechanism should account for post-invocation instance modification in addition to pre-invocation persistent service modification. Third, service execution containers need to offer reconfiguration interfaces.

# 3. CONCEPT OF CROSS-DOMAIN RECONFIGURATION ADAPTIVITY

Translating between the NFP terminologies of service developers, users and system administrators requires an unambiguous, extensible and expressive language as well as flexible management interfaces in the service container architecture with monitoring and reconfiguration capabilities.

We assume the existence of three sources of information: one being a specification of domain-specific NFPs of each service, the second one being a specification of resources available to the container and execution platform, and the connecting one being cross-domain mapping rules between those other two. No assumption is made about the origin of the specifications. Design-time modelling and monitoring data aggregation can help with the creation of sufficiently precise information. Domain-specific properties can occur as exposed internal service variables, external configuration settings or induced by message contents. The mappings need to be generated upfront or be derived from test results based on how much which service configuration or message property affects which resource consumption. At the end of the paper an initial methodology for deriving mappings from test results will be shown. The concept is depicted in figure 2.

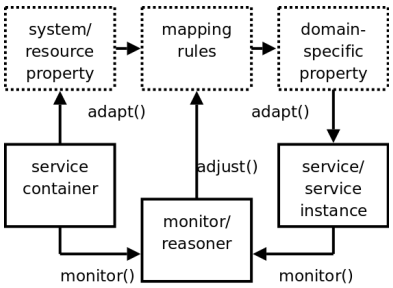


**Figure 2: Concept of NFP mapping for cross-domain reconfiguration**

In order to reconfigure applications efficiently, two interfaces need to be available at the service container: one for reading the current domain-specific configuration values, and one for writing the updated values.

Conventionally, service NFP names of the system-level resource domain can be read from the service description. They can be stored as entries in an OSGi manifest, as a context section in a Servlet descriptor, as an instance of `QoSBase` ontology concepts in a WSML file or as an `appinfo` XML node in a WSDL file. The NFP values can be read from the same locations when formal service and component specifications like CQML+ are used, but can also be derived from monitoring data at runtime.

Reading domain-specific variables is typically more dependent on the implementation. For example, Java annotations, Servlet context and initialisation parameters and OSGi bundle variables can be read from declarative sources. Writing domain-specific variables is however not possible for OSGi manifest header values and Servlet initialisation parameters. We will present a solution for this shortcoming in the implementation section.

The mapping between system NFPs and domain-specific NFPs should happen outside of the scope of the service im-

plementation. The reason is that the service itself will not have any knowledge of the contract for the service instance in question which may prioritise certain configuration settings. For example, assume that a contract for an image processing service specifies a maximum response time of $10s$ and high-quality results. If for some reasons the response time cannot be guaranteed anymore, the service could degrade the quality. However, if the compensation fee for violating this property is higher than the one for delaying the operation, no such reconfiguration should happen.

# 4. EXAMPLES FOR CROSS-DOMAIN ADAPTIVITY

The implications of external cross-domain mappings shall be explained with two examples. Beside the image processing application, a streaming example will be given to demonstrate the need for post-invocation time reconfiguration.

## 4.1 Pre-invocation Reconfiguration on a Service Level

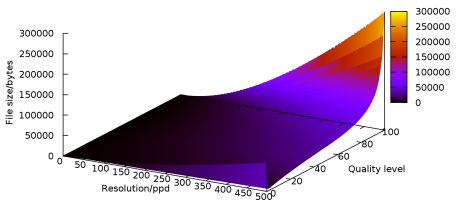


**Figure 3: Image file size depending on resolution and JPEG quality level**

Figure 3 shows the resource consumption of a picture album service which stores pictures as converted JPEG files.[1] While the file size does not grow quadratically with the resolution per dimension as it would with uncompressed images, the chosen quality level has a tremendous effect in the high quality range. Therefore, service providers would have to take this resource consumption into account for the modelling of tariffs. Eventually, the choice of tariffs is a business decision, but resource consumption prediction helps finding suitable offers. A typical scenario is an offer consisting of two tariffs:

```
tariff "standard" {
  quality = 60%;
  resolution = 300px;
  price = {invocation = 20ct;}
}
tariff "premium" {
  quality = 0..100%;
  resolution = 1..500px;
  price = {invocation = 2ct + $quality*10ct +
      $resolution*0.1ct}
}
```

An adaptive service environment requires to honour contractually guaranteed properties while at the same time providing as much flexibility as possible for changes to the service and the environment itself. In the case of the picture album service, the tariffs do not indicate any potential for adaptivity. It depends on whether the contract contains clauses about degrading guarantees and compensation fees. A typical contract looks as follows:

```
contract {
  provider = "p-identification";
  consumer = "c-identification";
  tariff = "standard";
  level = 97%;
  compensation = 20 EUR;
}
```

In this case, up to 3 percent of the invocations may result in violations of individual property guarantees without invalidating the contract as a whole. Only when this level is exceeded, the provider has to pay a compensation fee.

Such flexibility in contracts can be used to drive adaptivity. In the case of shrinking free hard disk space over time, the addition of storage capacity may take too long to avoid loss of image data, hence scaling down images saves disk space. The result is an optimisation problem of finding the right quality parameters to accomplish a predicted capacity allocation slowdown. The decisions range from using a constantly poor image quality, as defined by both resolution and JPEG quality, to a graceful degradation which will not save as much but will also not show the same negative effects. Considering the more dramatic impact of JPEG quality in figure 3, the optimisation problem is simplified to leave the resolution intact and focus on JPEG quality selection policies versus remaining disk space. In figure 4, a range of possible policies is shown. The range is defined by a sigmoid-like function with upper and lower boundaries $degr_\alpha$ and $degr_\beta$ as follows:

$$degr_{step} = val * \frac{(sin(\frac{\pi}{2} + \frac{\pi}{(degr_\beta - degr_\alpha)} * (step - degr_\alpha)) + 1)}{2}$$
$$(step : 0..steps)$$

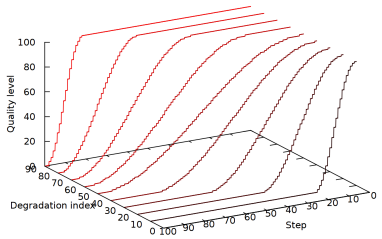


**Figure 4: Comparison of degradation policies depending on parameter $degr_\alpha$ and $degr_\beta$**

However, none of these policies is related to actual resource consumption. To inject the influence of the remaining hard disk space, the formula needs to be altered along the resource and contract constraints. The remaining resource availability shall be known as $totalres$ and the consumption of this resource in step $step$ as $res_{step}(degr_{step})$.

$degr_{step} := res_{step}(degr_{step}) * degr_\beta - degr_\alpha < totalres$

Assuming a remaining disk space of $80kB$ and 200 already stored pictures, therefore allowing up to 6 degraded pictures, the degradation works as follows.

Therefore, instead of the 4 undegraded pictures which would fit into the space, the flexibility to select between

[1] Conversion used libjpeg version 6b14 on the image file `Wartburg_1898.jpg` from `commons.wikimedia.org`, 1st revision.

**Table 1: Resource-bound degradation example**

| $degr_\alpha$ | Remaining size/bytes per picture | Maximum degraded JPEG quality |
|---|---|---|
| step + 0 | 13653 | 43% |
| step + 1 | 10792 | 30% |
| step + 2 | 6826 | 15% |
| step + 3 | 5069 | 9% |

4 and 9 pictures has been achieved. Naturally, a quality level of 9% will immediately be judged as degraded, whereas according to our tests 30% is still an acceptable value compared to an original of 60%. Therefore, the contract limitation on per-invocation quality is counter-productive. An alternative are per-timeslot quality guarantees, specifying that 97% of the year the guaranteed quality levels will be kept. Depending on the occurrence of burst usage patterns, this can be seen as a subjectively better or worse deal for either side, given that during 11 days per year invocations are not guaranteed to succeed. Another alternative are perceived-quality guarantees, specifying that the average perceived JPEG quality needs to be above 59%. This way, up to 57 images with a quality of 30% could be stored without violating the contract. Figure 5 shows the difference compared to the resource-independent degradation policies.

The essence of the example is that flexible resource-aware quality-degrading reconfiguration needs contracts which allow adaptivity for the benefit of both service provider and consumer. This benefit is often a domain-specific metric which requires adaptive systems to honour cross-domain adaptation.

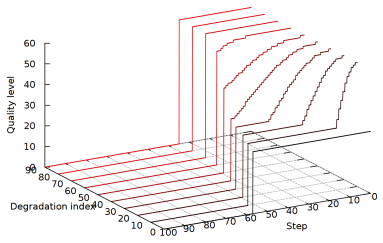


**Figure 5: Comparison of resource-aware degradation policies**

## 4.2 Post-invocation Reconfiguration in Streaming Services

Streaming web services are characterised by offering long-lasting message exchanges which are often consisting of infinitely repetitive data structures. If reconfiguration is restricted to affect only new connections and is not able to adapt existing streams, the overall adaptivity of a system will be severely limited. Going back to the example album service, the adaptive delivery of a series of images can help saving bandwidth in addition to disk space. Both quality-preserving and quality-degrading reconfiguration of non-functional stream properties exist. An example for a preserving reconfiguration is the immediate natural language change during a transmission.

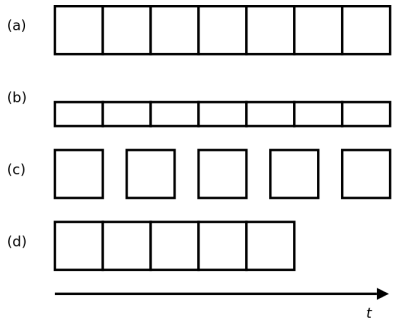


**Figure 6: Quality-degrading reconfiguration mechanism applied to web service streams**

The streaming-related quality-degrading changes, on the other hand, can be used to prevent system overload by minimising traffic and will almost always affect a user negatively, as shown in figure 6. Examples include omission of details within data structure blocks (b), decreasing frequency of repetitive data structures transmission (c) and restriction of the total duration of the transmission (d), e.g. for iterative calculations. Given that today's systems can easily handle tens of thousands of sleeping threads and connections, the combination of these parameter changes opens the whole spectrum of reconfiguring the bandwidth resource consumption. Hence, the goal must be to reduce the transmission volume represented by the area in the figure as much as is required to keep the system running and as much as is possible without violating active contracts. The details of this optimisation problem are out of scope for this paper; the more interesting aspect is how cross-domain property reconfiguration can help achieving the reduction of bandwidth use. Essentially, the information needed by the reconfiguration component is in which direction and by how much service parameters must be changed.

# 5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented the presented concepts and scenarios in a simulation framework, an extension of a Servlet container and a mapping creation tool.

## 5.1 Simulation Framework

The central component for cross-domain reconfiguration is called *NFPcalc*. It is a modular simulation framework consisting of a core written in Ruby and several plugins. The tool reads both NFP specification files and mapping rule files. The NFPs are supposed to express a name, a unit, a domain expressed in XML Schema and a property value. The mapping rules each consist of a service-domain property, a resource property and a directed dependency to specify an aligned or converse resource growth effect when reconfiguring the service-domain property.

Two precision-increasing plugins are available to NFPcalc: A unit converter and an XML Schema inference module. The unit converter facilitates a change of physical or currency units within the same domain - for example, a performance property might be expressed in messages per second or messages per minute. Any transformation between

the units shall be transparent to the calculation. The XML Schema inference makes it possible to use complex and restricted simple databases in the component specification. For example, a colour depth property of an imaging service would usually be an enumeration of 8, 16 or 24 bits, and any reduction or augmentation of the value would have to honour the enumeration over the base domain. In addition to the calculation plugins, two simulation plugins are available: A plot generator and a random system property consumption simulator.

A sample service has been created which processes images. Its functionality is composed of individual operations for scaling images, changing their colour depth and saving them in different formats. All operations are influenced by a number of variable NFPs specific to the service domain, including the resulting colour depth and the use of a fast dithering algorithm.

In Listing 1, a precise type definition is shown in XML Schema notation. The definition is used to define service-specific properties in Listing 2.

**Listing 1: Type definition for a service domain**

```
<xsd:simpleType name="ColourDepthEnum">
  <xsd:restriction base="xsd:unsignedByte">
    <xsd:enumeration value="8"/>
    <xsd:enumeration value="16"/>
    <xsd:enumeration value="24"/>
    <xsd:enumeration value="32"/>
  </xsd:restriction>
</xsd:simpleType>
```

**Listing 2: Strongly-typed NFPs from a service domain**

```
param {
  name = "colour-depth";
  value = "24";
  unit = "bits per pixel";
  domain = "ColourDepthEnum";
}
param {
  name = "fast-dither";
  value = "true";
  domain = "xsd:boolean";
}
```

Finally, Listing 3 shows how the configuration of the service-specific properties influences system properties. The mapping takes into account a partial ordering of the domain-specific NFPs.

**Listing 3: Cross-domain NFP mappings**

```
param-mapping {
  name = "bandwidth-usage";
  depends = "colour-depth";
  direction = "same";
}
param-mapping {
  name = "processing-speed";
  depends = "fast-dither";
  direction = "converse";
}
```

The experiment consisted of evaluating the adaptation behaviour when the available memory became scarce. Due to the mapping information, the reconfiguration handler modified the service parameters to produce lower-quality images. Once this change became effective, a lot of previously occupied memory was freed. This in turn led to a reverse effect of increasing the picture quality again so the available resources could be used efficiently to the benefit of the service consumers. Over time, a repeating reciprocation effect could be seen as is displayed as an oscillating graph in figure 7.

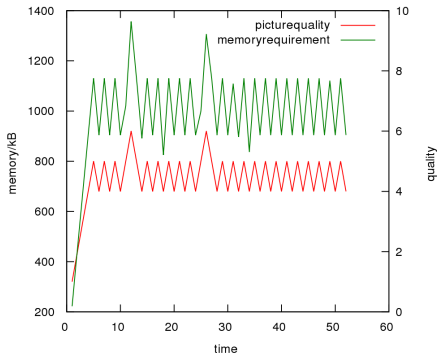


**Figure 7: Reciprocation effect between adapted domain-specific picture quality and available system memory**

This has obviously not been a desired result, given that a permanent reconfiguration activity defeats implicit requirements including system stability and energy efficiency. Therefore, we have designed a reconfiguration stabilisation routine (RSR) based on a two-point hysteresis. Optimised RSRs which honour contractual obligations and tolerances are expected to improve the obtained results in the future.

## 5.2 Application in Reconfigurable Servlets

As outlined in the concept explanation, both OSGi bundles and Java HTTP Servlets already provide internal reconfiguration support, but only through Servlet context attributes and OSGi configuration objects, respectively. Both are only created at runtime and cannot be read from any declarative description. In order to make existing services reconfigurable and to apply our concepts to existing middleware, we have extended the semantics of Servlets to support declarative, cross-domain reconfiguration. Our solution for Servlets involves a new class *AdaptiveHttpServlet* inheriting from the default *HttpServlet*. Upon instantiation, it mirrors declarative initialisation parameters to reconfigurable context attributes. NFPcalc then proceeds to read the property descriptions of all installed Servlets from their `web.xml` descriptors to get an overview on available reconfiguration points. Through the addition of an external interface to the Servlet container, NFPcalc can access these properties for non-functional read and write operations for all Servlets inheriting from *AdaptiveHttpServlet*. First results have shown that this way, a container with several adaptive Servlets can be ordered to release as many acquired resources as possible under its contractual obligations. A limiting factor for the cross-domain calculation is the absence of units. As shown in the example on service streaming, a reconfiguration is nevertheless possible as long as the direction of change is known. We intend to explore the addition of strongly-typed properties to Servlets and the application of the concepts to OSGi in the future.

### 5.3 Methodology for the Creation of Cross-Domain Mappings

The cross-domain mapping of properties understood by the user of a service could be created manually by domain experts or administrators. We have created a monitoring analysis tool which helps mapping creators to find the right dependencies automatically. The behaviour of a service depends not just on resource availability, but also on the structure and content of the messages sent to it. Depending on the element and attribute values of XML messages, QoS properties will either remain constant or relate to the message values.

The tool calculates the correlation of these values with measured QoS properties by using a clustering algorithm over a set of stored SOAP messages. Variations of the messages can result in a combinatorial set of QoS variations which is simplified by the clusters. In figure 8, the influence of one message property on two QoS properties is shown. The circle positions represent the values of a measured property, and their diameters describe the number of occurrences. The algorithm is a variant of the Quality Threshold algorithm which originates from biological research but is not uncommon for distance-based grouping of characteristics in distributed systems [1]. The corresponding message properties for each cluster are examined and a type-dependent correlation is established by assigning sets of causation tuples containing all combinations of message properties expressed as XPath terms over the original messages to each cluster. The fewer sets there are, the easier it is to detect dependencies and the easier the mapping rules become.

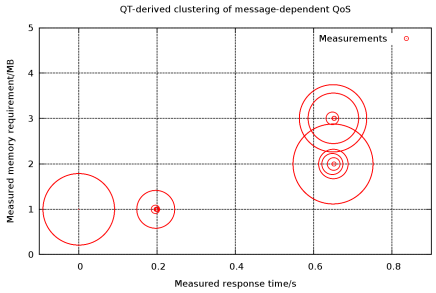


**Figure 8: Clusters of multiple QoS property dependencies on message content property**

In figure 9, the result of the algorithm on the requests with two properties influencing the transmission volume of the response is shown. For any known NFP, the transmission volume can be regulated by reconfiguring the properties before the messages are passed to the service. Additional research will be needed to deterministically tell FP from NFPs in requests. Using the clustering algorithm, server providers can optimise their offered features depending on how resource-consuming the transmitted message parameters are.

## 6. CONCLUSION

Dynamic reconfigurability is a desirable factor for adaptive service execution. We have shown that using cross-domain mappings, system property changes can be translated into configuration changes in the domain of the service. We also demonstrated that the creation of these mappings can be automated to some extent based on observed service

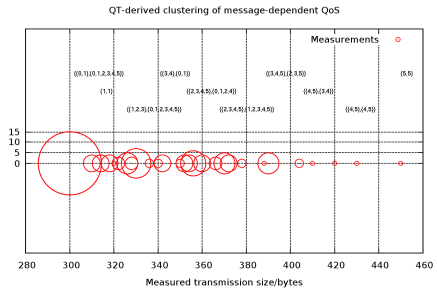


**Figure 9: Clusters of single QoS property dependency on multiple message content properties**

behaviour, and that their use with modified mainstream service execution containers is feasible. Future work will see optimised RSRs and more precise mapping rules including impact quantity in addition to impact direction.

## Acknowledegments

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference “01MQ07012”. The authors take the responsibility for the contents.

## 7. REFERENCES

[1] F. Cantin, B. Gueye, M. A. Kaafar, and G. Leduc. A Self-Organized Clustering Scheme for Overlay Networks. In *Proceedings of the 3rd International Workshop on Self-Organizing Systems*, volume LNCS 5343, pages 59–70, December 2008. Vienna, Austria.

[2] V. Janjic, K. Hammond, and Y. Yang. Using Application Information to Drive Adaptive Grid Middleware Scheduling Decisions. In *ACM International Conference Proceeding Series/Proceedings of the 2nd workshop on Middleware-Application Interaction (MAI)*, volume 306, pages 7–12, June 2008. Oslo, Norway.

[3] B. N. Jørgensen, E. Truyen, F. Matthijs, and W. Joosen. Customization of Object Request Brokers by Application Specific Policies. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing - MIDDLEWARE*, April 2000. Hudson River Valley, New York, USA.

[4] M. Koning, C. ai Sun, M. Sinnema, and P. Avgeriou. VxBPEL: Supporting variability for Web services in BPEL. *Information and Software Technology*, 51(2):258–269, February 2009.

[5] J. Matevska-Meyer, S. Olliges, and W. Hasselbring. Runtime Reconfiguration of J2EE Applications. In *DECOR'04 - 1ère Conférence Francophone sur le Déploiement et la (Re) Configuration de Logiciels*, October 2004. Grenoble, France.

[6] C. A. N. Soules, J. Appavoo, K. Hui, D. D. Silva, G. R. Ganger, O. Krieger, M. Stumm, R. W. Wisniewski, M. Auslander, M. Ostrowski, B. Rosenburg, and J. Xenidis. System Support for Online Reconfiguration. In *Proc. USENIX Annual Technical Conference*, June 2003. San Antonio, Texas, USA.