Spillner, Josef, Dipl.-Inf.
(Technische Universität Dresden, Germany, josef.spillner@tu-dresden.de)

PRIVACY-ENHANCED SERVICE EXECUTION

Abstract: Service users have increasing requirements towards the quality of services. The service execution environment needs to translate these requirements to distribution, resource acquisition and adaptivity. We present a platform-level feature called *PrivacyCage* based on monitoring facilities and service replication mechanisms to protect the users' privacy when they interact with services on the platform. This feature will be reflected in the service level agreements.

1. MOTIVATION

Both in personal computing and in enterprise environments, the use of services and remote applications is increasing. A lot of people rely on interactive web applications such as HTML-based mail clients, which don't offer any guarantees regarding availability, privacy or costs. There isn't even a standard mechanism available to retrieve such non-functional information about remote applications before attempting to use them. Any guarantees and conditions published by service providers appear in textual descriptions. Several research projects have proposed and implemented machine-readable SLAs which cover specifications of parts of the system, metrics and guarantees, information about contract duration and the actions in cases of violations such as penalties or compensation. These SLAs need to be simple enough to be used in practice (which arguably isn't the case at the moment), and yet powerful enough to derive negotiation protocol parameters, monitoring configuration and system adaptation hints from them. The service execution environment assumes an important role in providing the coordination facilities to gain an overview about installed services so that SLAs only offer the guarantees which do not exceed the system's capabilities, unless the system can dynamically add resources at runtime.

We are interested in the possibilities and limits of conveying system features into service agreement templates and have designed and implemented a prototype for one such feature: service execution privacy. This document first explains some of the available state-of-the-art service execution platforms with a focus towards increased privacy and autonomy from service providers. It then presents the problems involved and our approach for solving them, in particular how to advertise and enforce guarantees, and concludes with some remarks on their general applicability beyond privacy features.

2. RELATED WORK

We will present existing approaches to SLA handling in current service platforms with a focus on how the content of SLAs or SLA templates prior to contract binding is generated by them.

2.1 Service execution platforms

Both production systems and recent academic work suggest a trend towards more end user-targeted service platforms, leading to instant access to grid systems and other service execution environments with monitoring interfaces.

Environments in practical use today already provide tracing and interception features, notwithstanding scientific scrutiny for the validation of these claims. The *Google App Engine*, a closed-source software, is one such example which «isolates [one's] application in its own secure, reliable environment», and prevents applications from creating subprocesses and writing to the hard disk. According to the documentation, the sandbox is not configurable and doesn't convey its restrictions into any form of machine-readable SLA. The *Gigaspaces* implementation, which focuses on linear scalability based on the tuple space model, uses introspection on event objects to define instrumentation points in the associated SLAs. However, no low-level system or container features are propagated into the SLA automatically. The *TAPAS* architecture for QoS-enabled application servers provides a one-way adaptation between applications, the system and SLAs: For each SLA, a configuration service configures the system in a way that the SLA can be fulfilled, e.g. by reserving resources. However, no system features or restrictions can be propagated to the application to achieve bidirectional adaptivity. In the *COMQUAD* component-based project, contracts between the components, or services, and their managing container were considered separately from contracts between the container and the underlying runtime platform. It was not explained how platform or container features would propagate into SLA templates between the services and the clients [6].

According to the examples above, the discovery and expression of system features by service implementations is very weak compared to research on the discovery of functional and non-functional properties by potential service users (figure 1).

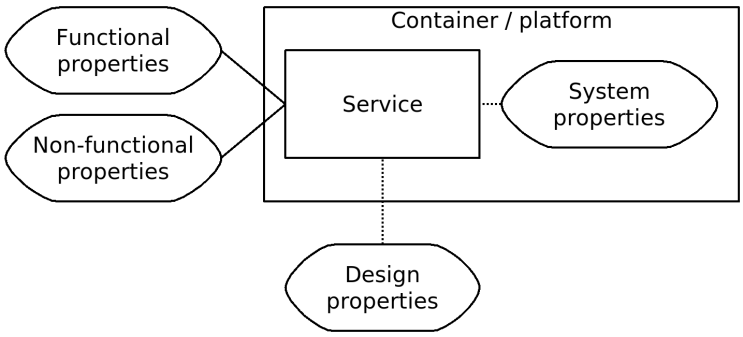

Figure 1: Sources of NFPs at design and execution time

2.2 Service contracts

We do not assume any particular format for the expression of service-level agreements, since there is a number of them available from several projects already, each with specific advantages and disadvantages regarding their negotiation, expressiveness and monitorability. Instead, we are only interested in how to express privacy features and other system features dynamically as a subset of the non-functional specifications in these agreements. They may complement other properties derived from the service model or implementation, measured historic assessment about the service features, and heuristic calculations. We do however not want to assume abstract specifications and hence need a mechanism to put system-level properties into SLAs at runtime.

In many approaches, SLAs are instantiated from SLA templates. The substituted template parameters often encompass only properties with ranges of values which are fixated to certain values for the contract by a negotiation algorithm. In FeMoWS [2], service

feature lists are used to introduce properties into SLAs. In [1], the authors argue against static SLAs and for the dynamic inclusion of properties, which also helps to avoid service substitution in SLA-protected processes by gaining more flexibility during the renegotiation phase. The authors derive their properties from software engineering aspects, but do not cover system-level properties which may be added, changed or removed at runtime. Machine-readable privacy guarantees have been made popular by the P3P format for use in web pages. While P3P can be evaluated programmatically, it lacks several details about the further use of data or contractual obligations in case of not fulfilling the agreement. We therefore aim at an approach of enforced guarantees to avoid such issues. The Open Digital Rights Language and similar languages, some of which are embeddable into SLAs, define a more formal syntax for rules governing the exchange of data with services.

3. CHALLENGES

After the presentation of the state of the art in today's systems, we introduce the topic of privacy-enhanced service execution as a use case to see how system-level features can and should show up in SLAs. The following challenges have to be overcome:

- Data privacy: For simplicity reasons, we define privacy in the context of service execution as the protection of all data exchanged with the service against unwanted leakage to any other party. According to this model, the user will have to trust the execution environment, but not the service itself. We don't consider anonymity or privacy management functionality.
- Service replication: While the concept of Software-as-a-Service promotes the use of remote applications and the storage of data online, and SOA concepts in general favour this approach, it is insufficient in many cases where privacy and code integrity play a major role. An extension to declarative service self-descriptions, such as WSDL files, would make it possible to assess information about the service implementation and hence its replication without breaking existing infrastructures. Replication mechanisms could be based on standardised service packaging formats, including Servlets, OSGi bundles and LSB-compliant services. One challenge here is to extend the replication to configuration data.
- System-level monitoring: The extent of monitoring performed on a service might in some cases directly translate into enhanced service agreement offers. An example would be to be able to offer certain privacy guarantees if a service execution can be traced on an operating system call level, a virtual machine interpreter level, or a higher-level process execution level. System-level monitoring can be distinguished from Monitoring-as-a-Service to further refine the scope of trust in the service execution environment.
- Contractual representation of the infrastructure: While quality modelling languages exist for model-driven approaches of describing individual components, they are insufficient for distributed systems. The system configuration, network topology and previously gathered monitoring information could be used to describe the behaviour of systems more accurately. This will lead to more representative SLAs and less potentially expensive renegotiations. The challenge is how to infer infrastructure dependencies and relationships between systems without having a full model.

These topics are closely related to each other. For example, in order to guarantee privacy by tracing service execution, a service might need to be replicated to the execution platform at first. The security properties of the platform will likely be better if its codebase

remains small, thus favouring a microkernel-style approach of offering as much functionality as possible as a service on top of the platform, such as monitoring.

4. SOLUTION APPROACH

In order to solve the problems identified up to this point, we selected a certain real-world service where both replication and privacy of the exchanged data are desired properties.

4.1 Scenario

A popular and frequently used public web service is the W3C validator for HTML documents. The provider offers the service implementation as source code and recommends its replication to lower the load on the W3C servers. Replication is also desired by some of the service users themselves who do not want to submit potentially confidential documents to a public service. Finally, since the validator uses an HTML parser internally, one has to take care about parser implementations which due to low security awareness and misunderstanding of namespace URIs by the reponsible programmers accidentally fetch DTD files from the network for each validation process, accumulating up to 130 million mostly unnecessary requests per day [3]. Network access restrictions will help lowering this number. While the service itself does not offer a machine-readable service description, there has been made one available by a user of the very similar CSS validator. The description in WSDL format will serve as a basis for the HTML validator, extended with replication information and an SLA feature section.

4.2 Replication

WS-Source, an extension to WSDL, has been specified and was added to the WSDL file for the W3C validator service. The specification has been made available at [4]. The extension refers to the service implementation in a certain packaging format and in addition to its licencing requirements. This way, service users will be able to request its implementation, or refuse to use the service if the code is either not available or the licence is considered not suitable.

4.3 Privacy

PrivacyCage, a system-level monitoring and tracing tool, has been designed to monitor services over their use of system calls. Assuming a compartmentised runtime environment where no service can access data or state information of others, the syscall interception mechanism of this tool presents a reasonable guarantee that the service will not perform instructions which it is not supposed to do. Policies can be configured through a web application and include syscall groups like «must not access the network» and «must not write to disk». Loopholes might exist though due to the poor granularity and semantics of some system calls. We do not claim to have a mechanism right now for absolute guarantees about the enforcement of the desired policies; rather, we recommend using whatever facilities the operating system provides.

4.4 SLA representation

We employ the technique of dynamic SLA templates. Before any (re)negotiation of contracts takes place, system-level features are added to the SLA template by the service execution environment depending on the presence of *PrivacyCage* and other such tools.

5. IMPLEMENTATION

The approach presented in the last section was implemented in order to validate its feasability and usefulness in the context of ad-hoc process composition based on non-functional properties of the potential constituent web services.

5.1 Architecture

PrivacyCage has been implemented as a set of small modules which work together, rather than a monolithic tool. One of the advantages of this approach is that new tracing mechanisms or new interfaces can be added in place (figure 2). For example, Java-based services could be confined into a locked-down JVM. Our implementation only works on the OS level so far, which doesn't allow for fine-grained access control but on the other hand doesn't assume any particular implementation language for the executed service.

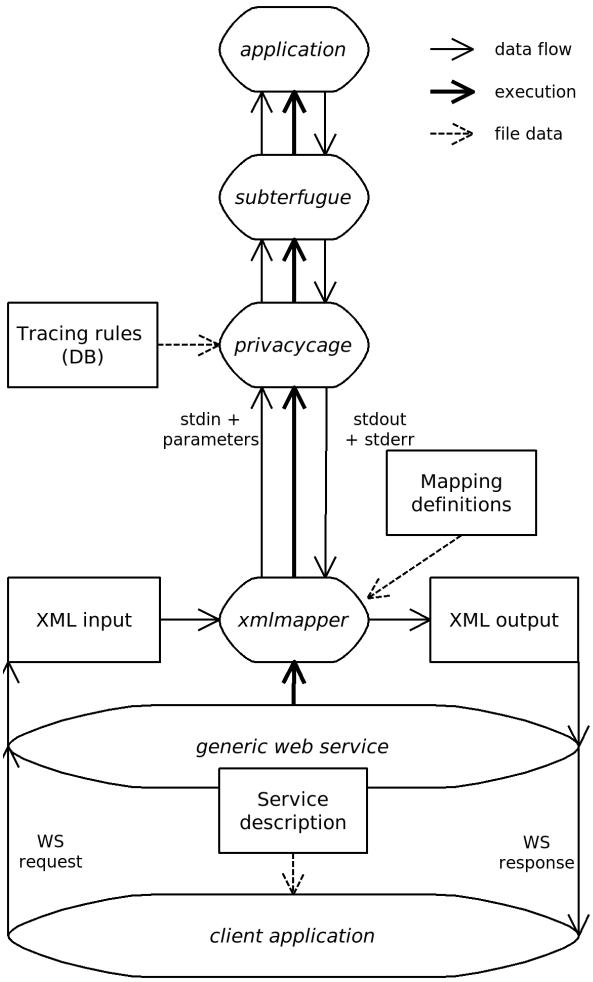

Figure 2: Architecture of PrivacyCage

PrivacyCage itself is a small Ruby application which loads the tracing rules and invokes *Subterfugue*, an already existing syscall tracer for Linux. This tool can report syscall usage by monitoring or intercepting them and handing over these events to plugins. There was no plugin readily available with it for the interception of arbitrary syscalls, so we extended the tool with one called *SyscallInterceptorTrick*. The application is finally launched under the supervision of *Subterfugue*. The first version of *PrivacyCage* included a web service interface. We changed this for several reasons: First, one might want to run the tool from a non-web service context. Second, one might want to use different web service protocols. Third, the fine-granular design will exhibit the same advantages as descibed above regarding substitutability. Therefore, we reimplemented it as a command-line application and wrote a generic web service mapper called *xmlmapper* for it. Both tools are usable independent of each other but work well together for our scenario. The mapping tool is similar in spirit to *gFac*, the Generic Service Toolkit [5], although it isn't limited to SOAP. In fact, *PrivacyCage* has been mapped to a REST-style service.

An application called *wssource-loader* has been written as an implementation of the *WS-Source* extension. It scans through the service description and install the service implementation. No configuration takes place yet, as it is assumed for implementation packages to be completely self-contained including code, data and configuration parameters. The implementation has been made available under a free licence to gather suggestions for improvement [7].

5.2 Tracing rules

The association between syscalls and the actions performed on them is stored in so-called tracing rules. A web interface has been implemented to make their configuration as easy as possible. Associations can be grouped to higher-level rules, which is useful for often-used expressions such as the prohibition of network access or hard disk write operations. To give an example, the following (incomplete) rule will prevent network access:

```
netrule := socketcall | nfsservctl | sendto; on(netrule) abort();
```

For performance assessment, the rules would have to be written accordingly:

```
perfrule := prof | sendfile; on(perfrule) log();
```

6. FUTURE PLANS

The first implementation of *PrivacyCage* is functional, even though it has only been tested for simple cases including the scenario presented in this work. Additional tracing layers on other levels like virtual machines for Java (JVM) and other bytecode-interpreted languages, or on the service message level, could provide for more powerful SLA guarantees. The *WS-Source* specification needs to be validated using more real-world services. In general, replication through *wssource-loader* would only be considered one use case for *WS-Source*. A second one is the strategic advantages for the client like independence from the service provider. Once enough feedback on this topic has been collected, we will consider a separate publication about *WS-Source*.

7. CONCLUSION

We have shown how to increase the privacy for web service consumption by claiming guarantees in SLAs and enforcing them on the service execution layer. In addition to that, we have touched on service replication as an infrastructure-level prerequisite for effective privacy-enhanced service execution in dynamic service environments. While we cannot claim to have solved the problem of privacy-aware service execution in particular or the propagation of system-level features into service contracts in general, it is worth to point out that our implementation can serve as building blocks for future research on this topic.

BIBLIOGRAPHY

[1] N. C. Narendra, Karthikeyan Ponnalagu, Jayatheerthan Krishnamurthy, R. Ramkumar: Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming, Proceedings of the ICSOC 2007.

[2] M. Fantinato, I. M. de S. Gimenes, M. Beatriz F. de Toledo: Supporting QoS Negotiation with Feature Modeling, Proceedings of the ICSOC 2007.

[3] G. Oskoboiny, T. Guild: W3C's Excessive DTD Traffic, W3C article, online: http://www.w3.org/blog/systeam/2008/02/08/w3c_s_excessive_dtd_traffic

[4] J. Spillner: WS-Source: an implementation-revealing extension to web service description languages, software project, online: http://rcswww.zih.tu-dresden.de/~spillner/software/.

[5] G. Kandaswamy, D. Gannon: A mechanism for just-in-time creation of web services for scientific workflows, Workshop on Web Services-based Grid Applications, 2006.

[6] S. Zschaler, S. Röttger: Types of Quality of Service Contracts for Component-Based Systems, Proceedings of the IASTED SE 2004.

[7] J. Spillner: PrivacyCage: SLA-based confined service execution, software project, online: http://rcswww.zih.tu-dresden.de/~spillner/software/.