AN INTEGRATED PROVISIONING TOOLCHAIN FOR THE INTERNET OF SERVICES

Josef Spillner, Anne Kümpel, Stefan Uhlig, Iris Braun and Alexander Schill

Technische Universität Dresden, Faculty of Computer Science

Nöthnitzer Str. 46, 01187 Dresden, Germany, josef.spillner@tu-dresden.de

ABSTRACT

Recent developments in research on service-oriented architectures have yielded proposals and realisations of powerful service platforms with service brokering, execution and presentation functions. These platforms manage services with expressive declarative interface descriptions. For service consumers and resellers, they provide discovery, contracting and delivery capabilities. However, in practice there is little uptake of the platforms, and only a few tradeable services exist. A central reason for this is that service providers require automated processes and convenient tooling to bring more services onto the platforms. Our contribution to solve the problem is a flexible and integrated provisioning toolchain for service modellers and engineers.

KEYWORDS

Web Services, Service Engineering, Service Offering

1 MOTIVATION

The development of interactive applications has accelerated significantly with the advent of user-friendly application catalogues especially in the mobile market. The more popular representatives of both commercial stores and community repositories all provide a a five-digit number of distinct applications. In comparison, such catalogues for programmatically accessible and composable services are rare, small and of low content quality [Legner, 2009].

Looking at both service runtime platforms and service development environments, there is a potential to change this, which is as of yet mostly unused [Cardoso, 2009]. The key improvement would be to codify, specify and standardise more aspects of services along the publishing and provisioning toolchain, starting from the development environments and extending to the service offering and long-term maintenance tasks. In this work, we present an integrated provisioning toolchain and share our experience with its use for scenario service offers. The paper first explains existing work which defines the scope of services, runtime platforms and suitable development environments. Afterwards, we outline the concept for an integrated toolchain and an implementation thereof, followed by a scenario. The paper concludes with an outlook on further use cases.

2 BACKGROUND AND RELATED WORK

In this section, we give a condensed view on existing service definitions, service platforms for brokering and execution, and suitable engineering tools. Our aim is to extract details on the weak spot of provisioning within this service ecosystem in order to motivate the design of our toolchain.

2.1 Tradeable Services

The definition of services is subject to ongoing debate in the service sciences and internet research community [Spohrer et al., 2007, Schall et al., 2008]. Services can be classified along several dimensions of internal and external characteristics, for instance, implementation frameworks and languages, artefacts, distributability and tradeability. Web service implementations reveal a heterogeneity of programming languages and frameworks. Services contain descriptive artefacts (service descriptions, policy documents, contract templates) as well as implementation artefacts (executable code, program data). Services can either be invoked at a single static location or, if the implementation technology allows it, be distributed and replicated to any number of execution platforms. The selection and invocation of services might not require special knowledge and privileges. In many cases though, a systematic selection based on non-technical properties and a contract established prior to the execution will be necessary for flexible usage and composition of services, bringin in the tradeability aspect. For the purpose of this work, we mainly consider tradeable and distributable services with a technical implementation, although parts of our toolchain also support human-delivered services. Our service definition allows for heterogeneous implementation and description formats through any number of artefacts.

2.2 Service Platforms

Service platforms are the software infrastructures to turn service implementations into offers. Located between service providers and consumers, automated systems facilitate the brokering, discovery and delivery of services. A management layer offers functionality for registering services, searching for them, establishing contracts, configuring the behaviour and submitting ratings after their use. An execution layer is typically distributed over multiple local-area computing nodes or distant computing centres for higher horizontal scalability. It performs the authentication of requesters and the monitored execution. The integration between service engineering and runtime platforms is currently often a set of ad-hoc actions rather than determined processes through well-known interfaces. Ideally, service developers would just select which platform operator to publish to, independently from the underlying platform choice. Therefore, we will analyse engineering tools next to assess their capabilities.

2.3 Service Development Tools

Development tools create deployable services out of higher-level models such as workflows. There are several competing approaches with varying degree of support for trading the resulting services on service platforms. Conventional service engineering tools combine interface modelling with declarative interface description generation and software stub and skeleton generation in bidirectional ways. These tools are however limited by only supporting simple interface-centric service descriptions (such as WSDL files) without extended properties required for global service trading. They also lack technology-agnostic consistent interfaces and support for publishing services on open marketplaces. The Integrated Service Engineering Methodology has been proposed to separate concerns over engineering the data, process, people, rules and description aspects of services. The corresponding workbench consists of editors for all combinations of aspects and roles which are connected by transformations between them, and a simple publishing method. While the methodology is a sound and powerful concept, the workbench has to date not been fully realised and is not publicly available. The Service-Centring System Engineering project offers a toolbox for the description, development, test and operation of services. Services are described either in WSDL with signature facets or in a proprietary extended format which supports QoS, commerce and management facets in an information model. The corresponding editor implements development and test scripts, but lacks a graphical environment and publishing functionality. The SOA4All Studio offers convenient modelling and composition of services with rich semantic annotations. However, it only offers saving the generated processes on the web server or on the local disk.

None of the approaches would be suitable candidates for rapid service engineering or teaching the concepts thereof to students with negligible effort due to missing availability or features. We were however looking for exactly such a solution and therefore designed an open-source integrated toolchain out of existing and newly exhibited tools which reduces the service lifecycle time from the design to a globally discoverable and usable service.

3 AN INTEGRATED PROVISIONING TOOLCHAIN

A service provisioning toolchain needs to cover all important tasks assumed by a provider in a service ecosystem. The tasks include the technical deployment of a service implementation, he registration of descriptive artefacts, the update or deletion of the complete service or parts thereof, and the management of artefacts of the service at runtime, taking feedback from the system and from consumers into account.

The abstract view on the construction of an integrated service toolchain is shown in Figure 1. It implies the roles of a service engineer to the left, interacting with engineering tools, and a service platform operator to the right. The platform serves a consumer in the upper right corner, who interacts with the platform's service directory or marketplace, and relies on a service hoster in the lower right corner to perform its execution tasks. Service artefacts, which might be bundled in a service package, are the main logical exchange units between the components assigned to the roles. A provisioning application serves as central point of exchange for the artefacts. It is highly connected to the engineering tools. The service provider role is determined by whether the engineer directly offers the services or submits them for review by dedicated people who assume this role through the provisioning application. Consumers can give feedback by rating services and contracts, whereas the hosting system produces monitoring data. Both kinds of feedback are aggregated in the provisioning application from which the engineer can fetch it for optimisation or reengineering of the artefacts. Furthermore, for certain feedback like monitoring values hinting at suboptimal SLA templates the application could already semi-automatically update the artefacts by providing retrievable suggestions. For these processes to take effect, the provisioning application should offer both a programmatic web service API and a graphical user interface.

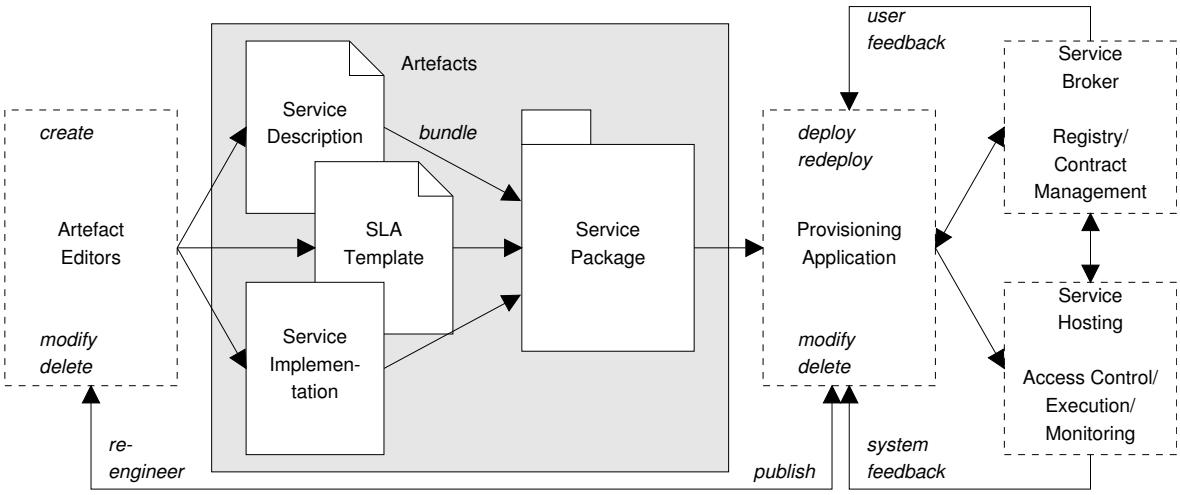

Figure 1: Concept for a provisioning toolchain from the service engineer to the service hoster and consumer

The design principles for the client side of the toolchain match the ones recommended for service-oriented systems: Loose coupling, composability (through scripting languages), discoverability (by using application metadata) [Groza et al., 2007] and separation of concerns (through per-artefact editors) [Mili et al., 2004]. A key concept for the increased integration is direct support for submitting modelled and implemented artefacts to service marketplaces. For increased flexibility and loose coupling within the toolchain, the deployment can happen both for each single artefact and for a complete service package which allows service providers to migrate to other platforms without much effort. Loose coupling is also achieved through file-based artefact exchange between editors, following proven system design guidelines and matching service platform decomposition trends. Another key aspect is the alignment with the service lifecycle. Evolution in service packages is supported by automatically versioned artefacts during redeployment at the provisioning application. This way, tracking the development of a service happens independently from the service engineers' versioning schemes and tools. Especially for collaborative service improvements and service compositions which benefit from having access to the most recent or particular version of its constituent services, this is a major advantage, although the versioning also needs to be properly supported by the runtime environment.

4 PROTOTYPICAL REALISATION

The implemented prototype instantiates the toolchain approach for service provisioning and implements a best-of-breed methodology of independently developed tools instead of requiring a single monolithic tool to be used for all aspects of service development. It consists of a number of stand-alone desktop editors for modelling and implementing services, a package creation tool, and a provisioning web application to deploy service packages and manage service offerings over their lifetime. Each editor typically requires certain configuration parameters: The submission URL at the corresponding platform service, the engineer's authentication credentials (username and password), and the association to a service. In our prototype, we introduce a submission application which can perform this task on the editor's behalf. The toolchain does not include a complete service platform; instead, it interacts with an existing platform. This way, the actual chaining is kept very lean and the result is an integrated open-source service lifecycle environment around the central toolchain.

Service Description Editor. We recognise the need to express service characteristics and properties in versatile file formats. There are already graphical modelling environments for interface descriptions (Eclipse WSDL, NetBeans WADL) and semantic concepts (WSMO Studio, Protégé) [Feier et al., 2005]. In the context of our work, we contributed to the development of the USDL Editor[1]. The Unified Service Description Language is a recent specification from the Internet of Services research community. It is specifically targeting tradeable services by offering description modules for business and legal aspects.

SLA Template Editor. Contract-bound service execution requires the use of formal contracts between service providers and consumers which are derived from SLA templates. Thus far, no graphical editor for templates has been publicly available. Therefore, we developed one which targets the WS-Agreement specification. The editor offers modelling extensions to match the language extensions, making it is suitable for adapting to evolving SLA languages. It is supplemented by a transformation from USDL service descriptions into skeleton SLA templates to achieve integration even between artefact modellers.

Service Implementation Editor. Due to the choice and free availability of editors and IDEs for service implementation development, we did not develop a custom tool. Instead, we integrate existing editors and others into our toolchain by using a temporary file-based calling convention.

Service Package Bundling. Collaborative service engineering benefits from having service artefacts published as a combined package which can be conveniently redeployed, modified and archived. For descriptive artefacts, no standard package layout exists so far, and the formats differ for executable artefacts such as Java servlets. We assume that placing these artefacts into auxiliary folders will not negatively impact existing deployments but will positively enable the automatic deployment in advanced service platforms. For the integrated toolchain, we have created a tool to assemble descriptive artefacts into an already existing implementation package. Artefacts can be added from files or created and modified from registered editors.

Service Provisioning Web Application. We have realised a web application which serves as central frontend to service deployment and portfolio management. It consists of separate pages for provider-related profile updates (such as address and tax numbers), a view on the currently offered service portfolio, and a wizard for deploying new services. Artefacts can be updated and versioned individually.

The resulting realised toolchain is shown in Figure 2. An advantage of using a separate package bundler is that the editors and IDEs themselves don't necessarily require publishing support since the bundler can perform this task for them. The toolchain is easily extensible by adding more modelling and development editors and registering their supported files in the bundler. Not all conceptual features have been realised. The feedback-driven reengineering process requires further refinements of the messages from the system, and the per-editor publishing and transformation functions are only realised in part.

[1] Eclipse-based USDL Editor: `http://sourceforge.net/projects/usdleditor/`

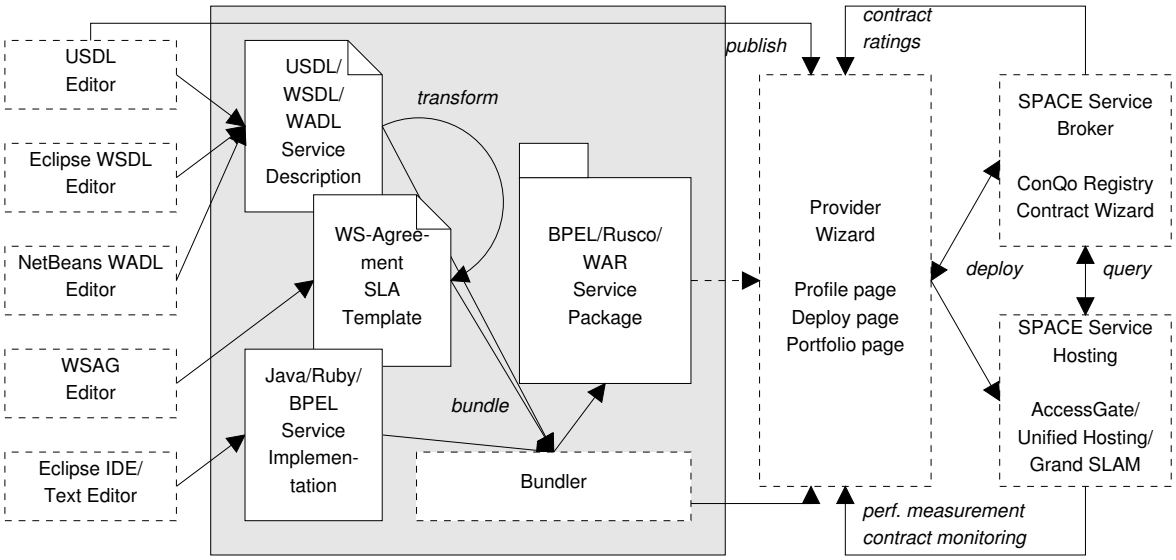

Figure 2: Realised toolchain as set of integrated open-source tools and service platform components

5 APPLICATION SCENARIO: CLOUD STORAGE SERVICE

We have evaluated our toolchain in various configurations through demonstrations within the THESEUS/TEXO research project. We describe the provisioning of a simple cloud storage service to illustrate the toolchain's usefulness. The purpose of TinyStorage is offering a method to allocate a storage area and another one to receive the file and store it in this pre-allocated area. The service consists of a Ruby service module, a corresponding set of WSDL interface description, USDL and WSML behaviour descriptions, two WSAG agreement templates and some auxiliary files. The service artefacts are bundled and deployed into a service platform. TinyStorage is available for review in the Framework Diversity Web Services repository[2]. Using negotiated contracts as authentication parameter for the invocation at runtime, the platform generates system-side monitoring data and eventually a rating from the user. When the service provider decides to move hosts, it becomes possible to include the submitted ratings into the package.

Fig. 3 shows a screenshot of the service package bundler just before submitting the package to an instance of the SPACE service platform. The submission triggers the installation of the Ruby service module into its respective execution container, the registration of the USDL file and other descriptions at the marketplace's service directory and the referencing of the WSAG template from the SLA manager. The configuration template is installed alongside the SLAT and contains mappings to calculate the storage area size from business properties.

6 CONCLUSION

In this publication, we have extended the notion of flexible engineering toolchains to service engineering tooling integrated into service platforms. As it was shown, this approach helps to reduce duplicate modelling, development and maintenance. We also publicly provide the realisation of our toolchain for future experiments and improvements. Among other use cases, it is now part of the SPACEflight live demonstrator for the Internet of Services [Spillner, 2011]. In the future, we expect the integration to become even tighter. Globally distributed version control systems can mediate between service engineer commits and service platform checkouts, causing a locally modified artefact to be propagated automatically to the staging area of a marketplace.

[2]Framework Diversity Web Services Repository: `https://gitorious.org/framework-diversity-web-services`

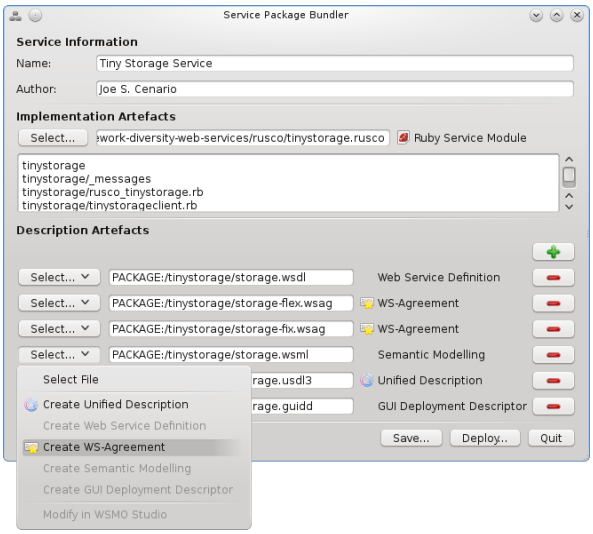

Figure 3: Invoking the package bundler on the scenario service artefacts

ACKNOWLEDGEMENTS

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

REFERENCES

[Cardoso, 2009] Cardoso, J. (2009). The Internet of Services. In *Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT)*, volume 1, pages 7–10. Sofia, Bulgaria.

[Feier et al., 2005] Feier, C., Roman, D., Polleres, A., Domingue, J., Stollberg, M., and Fensel, D. (2005). Towards Intelligent Web Services: Web Service Modeling Ontology (WSMO). In *Proceedings of the International Conference on Intelligent Computing (ICIC)*. Hefei, China.

[Groza et al., 2007] Groza, T., Handschuh, S., Möller, K., Grimnes, G., Minack, L. S. E., Jazayeri, M., Mesnage, C., Reif, G., and Gudjónsdóttir, R. (2007). The NEPOMUK Project- On the Way to the Social Semantic Desktop. In *Proceedings of International Conferences on new Media technology (I-MEDIA) and Semantic Systems (I-SEMANTICS)*, pages 201–210. Graz, Austria.

[Legner, 2009] Legner, C. (2009). Do Web Services Foster Specialization? - An Analysis of Commercial Web Service Directories. In *Tagungsband der Konferenz Wirtschaftsinformatik*, pages 67–76. Vienna, Austria.

[Mili et al., 2004] Mili, H., Elkharraz, A., and Mcheick, H. (2004). Understanding separation of concerns. In *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*. Lancaster, United Kingdom.

[Schall et al., 2008] Schall, D., Truong, H.-L., , and Dustdar, S. (2008). Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet Computing*, 12(3).

[Spillner, 2011] Spillner, J. (2011). SPACEflight - A Versatile Live Demonstrator and Teaching System for Advanced Service-Oriented Technologies. In *To appear in Proceedings of the 21st International Crimean Conference on Microwave and Communication Technology*. Sevastopol, Ukraine.

[Spohrer et al., 2007] Spohrer, J., Maglio, P. P., Bailey, J., and Gruhl, D. (2007). Steps Toward a Science of Service Systems. *Computer*, 40(1):71–77.