IT-AS-A-SERVICE FOR BUILDING VIRTUAL RESEARCH ENVIRONMENTS

Bastian Roth, Matthias Jahn and Stefan Jablonski
University of Bayreuth, Universitaetsstrasse 30, Bayreuth, Germany

Keywords: IT-as-a-Service, Virtual Research Environment, Resource Classification, Cloud Computing, Applied Meta Modeling.

Abstract: Virtual research environments are IT systems which support scientists in their daily workflow. Since there is a great number of different research domains, the quantity of utilized applications and hardware resources is immense. In the following, we present a solution how potentially each kind of IT resource like computing power, storage space, desktop applications and web applications can be integrated in form of services (IT-as-a-Service). The composition of certain services defines a scientist's personal virtual research environment.

1 INTRODUCTION

Virtual collaboration is an important aspect for the success of scientific projects, especially if participating researchers are distributed over the whole globe (Hey and Trefethen, 2002). A common and widely-accepted solution is exchanging documents by means of a file server or email communication. Setting up and managing a file server manually, for instance, is an error-prone and time-consuming task, notably for IT laymen. So, an automatic solution is required. Since data sharing is just one part of a scientist's daily workflow, further challenges also need to be considered. Thus, we interviewed scientists out of many different research areas and discovered the need to have one integrated platform which assists researchers in their daily workflow. We identified the following three challenges and simultaneously three corresponding classes of IT resources to support (for more details, see section 2):

- Easy access to services of domestic university (mainly **web applications**): Especially when scientists start to work at an academic institution they often do not know which kinds of services are offered to them by this institution. According to our experience, in most cases, there is even no single entry point to access all proffered services. Consequently, with our solution we want to provide such a single entry point.
- Further use of existing, in several cases domain-specific applications (mainly **desktop applications**): Since most researchers utilize dedicated software products, it is important to provide access to such kinds of applications as well.
- Support of collaboration, in particular comfortable exchange of files (e.g. using **network storage**)

Since our university is a global player in African studies, we also have to consider requirements which stem from another use case. Thereby, African scientists come to the university for a certain period of time and they directly want to work within the same environment like domestic researchers. This yields the following two challenges:

- Easy usage of the existing infrastructure by visiting scientists, i.e. access to already available resources like group calendars, file servers etc. As one can see, this challenge is quite similar to the first and third one explicated before.
- Possibility to use the same applications like domestic researchers, i.e. well-equipped computers are required (**virtual machines** together with certain software): Oftentimes, visiting scientists even arrive without any laptop such that they need to utilize local available computers. Understandably, these computers are not optimally equipped according to the visiting

Roth B., Jahn M. and Jablonski S..
IT-AS-A-SERVICE FOR BUILDING VIRTUAL RESEARCH ENVIRONMENTS.
DOI: 10.5220/0003913602340239
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 234-239
ISBN: 978-989-8565-05-1

scientist's requirements and sometimes also include outdated hardware. Using virtualization techniques could skirt this out-of-date issue because computing power and storage space is provided by a virtual machine which then is accessed remotely.

These challenges are addressed by offering particular services in a Virtual Research Environment (VRE) as already explained in (Roth et al., 2011). In that, a service encapsulates each above mentioned IT resource (virtual machine, network storage, desktop applications, and web applications) which should be integrated and utilized within a VRE. That is why one needs to ensure generic applicability. Existing VRE solutions do not provide that because they are restricted to certain research domains (Carusi and Reimer, 2010).

In (Roth et al., 2011) a high level solution proposal for a framework to build personal VREs with generic applicability is given, too. Below, we go into greater detail by focusing on Resource and Service layer, and thereby we present how generic applicability can be achieved. At first, IT resources are classified according to particular characteristics and after that we present a short overview of existing solutions and their unsolved issues. In chapter 5, we describe our solution in two steps: At first, we present the similarities of resources and how they can be classified. Afterwards, we show how resources can be integrated by using services as wrappers for them. Finally, we conclude our work and take a short outlook of our future tasks.

2 RESOURCE CLASSIFICATION

According to the requirements mentioned in the introduction, the resources have to be classified to reduce integration effort. Generally, resources can be categorized into software and hardware. In both categories the resources can be further subdivided into two classes according to the feasibility of access using web technologies. That results in the following classification. On the one hand, there are software resources that can be divided into:

- **Web applications** which directly provide a web interface that is typically accessible via HTTP, and
- **Desktop applications** which do not provide a direct way to access them via the web. This can be solved by dint of separate tools that stream those applications to the user (e.g. ssh -x, remote desktop protocol). Operating systems are also assigned to this class because they can be accessed in the same way.

On the other hand, there are hardware resources which we again separate into two classes:

- **Virtual machines** that encapsulate computing power as well as storage space which both cannot be used directly. Therefore, an operating system has to be installed and the applications have to be streamed as mentioned for desktop applications. This class of resources gives a possibility to provide a personal desktop for visiting scientists on the basis of virtualization techniques.
- **Network storage** which, theoretically, is also viable using a virtual machine. However, this would be a waste of computing power because it is not needed at this point. Instead, lightweight solutions for providing network storage (e.g. provided by NetApp) are utilized that provide typical web protocols (NFS, FTP, WebDAV etc.) to directly offer the network storage to the user.

3 RELATED WORK

As shown in the previous section we discovered four types of resources that have to be integrated into our platform. For desktop applications, this can be done by porting them to web applications using JavaScript, HTML and other web technologies. This approach is also followed, for instance, by Google with Google Docs (Google Inc.) or by Microsoft with Microsoft Office Web Apps (Microsoft Corparation, 2010). Both provide office functionality in the web. But these ports of desktop applications involve great efforts because the whole GUI or in the worst case the whole applications needs to be re-implemented. Thus, it is only applicable for resources that are utilized by many users. For highly specialized resources as we often meet in research environments, this approach is not sufficient.

Another practice is to use Web Services (W3C) to access resources in a well-defined way. Thereby, messages and data are transported over an interface (Stal, 2002). An example for using Web Services to define services can be found in (Foster et al., 2002) where GridServices are specified by a set of interfaces as WSDL port types. Web Services provide a way to make resources available via the internet, but it is not suitable for our requirements because they do not provide a GUI to access the service. So, they are just a way to facilitate

communication between computers. In (Turner et al., 2003) they also pick up the Web Service idea to make existing software available within the web, but afresh only for computers and not for humans.

Once again, a similar approach is followed by the popular cloud computing providers like Amazon and Microsoft. Amazon Web Sevices (Amazon Web Services LLC, 2010) provide a possibility for implementing an application against a Web Service interface and thus it supplies functionality that can be used by software developers. Microsoft Windows Azure (Chappell, 2009) also provides a Web Service API for different concerns, similar to Amazon's offerings. Computation and storage can primarily be rent by users. All in all, cloud providers support an easy way to access web applications, virtual machines and also storage, but they focused on supporting developers, especially regarding the two latter ones. However, our services are completely directed to the end-user. That is why this approach is not suitable for our above mentioned challenges.

Additionally, there are solutions to make desktop applications and the desktop itself available via the internet. They are able to stream GUIs of desktop applications via a security socket. Popular examples are ssh, Microsoft remote desktop protocol and Oracle Secure Global Desktop. These approaches do not support any possibility for integrating other resources like web applications or virtual machines.

Table 1: Comparison of available approaches.

	Web Apps	Desktop Apps	Storage	Virtual Machines
Cloud provider	+	-	+	+
Web Services	-	-	+	+
Application streaming	-	+	-	-

In summary, all presented approaches are limited to a certain kind of resource type they support (Table 1). Cloud providers principally fulfill a couple of our requirements, but they do not allow for integrating existing desktop applications in a rentable way for highly specialized software. Moreover, the different offerings are mainly targeted to developers and not to common users which are mostly IT laymen. The Web Service technology provides a possibility to share data over several computers in a well-defined way. Hence, they are a popular appliance to make virtual machines or storage available (for other computers) via the internet. But as mentioned above, they do not provide a user-friendly GUI. Thus, IT laymen are not able to access them in a convenient way. Furthermore, streaming applications just address desktop applications and do not provide solutions for further resource types.

4 RESOURCE INTEGRATION

One of the main challenges for our framework to solve is to provide an easy way to integrate existing IT resources. Since existing resources generally cannot be adapted, we need to see them as black boxes. So, they have to be integrated by means of corresponding wrappers which have to be specified, implemented and added to our system. We call these wrappers services because they provide functionality of the IT resources in the VRE as a service (IT-as-a-Service (Lin et al., 2009)). To define the structure of such a service, we have to cover all classes of resources that we presented in section 3.

4.1 Similarities between Resource Classes

Our first step is to extract similarities of the four different resource classes that must be handled by a service. If one wants to utilize and accordingly instantiate a resource, a well-defined set of information (data) has to be specified before. This can be realized by defining associated meta data for the resource. Those meta data can be seen as a template for the resources and thus we call it resource template. Examples for resource templates are setup bundles or web service interfaces which are used, for example, to create virtual machines, storage spaces or web application accounts.

Each resource must be set up before first usage and therefore, a corresponding template must be provided. After setup, most resources can be modified according to evolved user requirements. If resources are not needed any more they can be "disposed". After setup and before disposing, a resource may be started and stopped. This feature is not necessarily supported by every resource. Summing up, resources have a common **lifecycle** which has to be handled by the corresponding service. In addition, the lifecycle constitutes one of three functional aspects when describing the existence of a resource. It is called the state-based aspect.

In some cases, resources need to interact with other resources. These relationships can be divided into two different types: Firstly, resources can depend on other resources (**dependency relationship**), i.e. one resource needs another

resource for successful installation or execution. This kind of relationship affects a resource's lifecycle. With it, resources can build hierarchies, for example Microsoft Office depends on Microsoft Windows which again depends on a computer with x86-architecture (e.g. represented by an according virtual machine). Dependency relationships can also be found in many web applications (project management systems, wikis, groupware etc.) that depend on a Lightweight Directory Access Protocol (LDAP) (Koutsonikola and Vakali, 2004) and/or a database. A second relationship between resources is the **usage relationship** which represents loosely coupling of resources. Thereby, one resource does not necessarily need the other resource for execution and hence, the lifecycle of both resources do not affect each other. A common example for that relationship is the integration of the EndNote (Thomson Reuters) plugin into Microsoft Word. Both applications work independently, but by means of the plugin mechanism they can benefit from each other.

For the first setup or if reconfiguration is possible a resource contains a set of **properties** which can be set or manipulated. Examples for properties are installation directory, user credentials for an initially required super user and particular relationships to other resources. Properties and relationships form the second functional aspect, the data based aspect.

Each resource possesses certain functionality which is provided via a well-defined interface, so called **operations**. In some cases, operations need or accept further data that mainly is delivered in form of parameters. In particular, a couple of desktop applications can handle parameters, e.g. Microsoft Word can be invoked in combination with a document filename. Hence, it is necessary to know which kind of parameters a resource can handle to check whether the parameter's type is valid. A special parameter type which must be considered separately is an authentication token because with it, single sign-on can be facilitated. This scenario is especially useful when using web applications in combination with our system. Then, users merely have to login one time when accessing their personal virtual research environments which results in generating an authentication token. Afterwards, this token can be used for further authentications at the different utilized services, but without entering the user credentials again. Operations are the building blocks of the behavioral aspect which represents the third and last functional aspect.

4.2 Services as Resource Wrappers

According to these circumstances given by the resource layer, we now specify how a concrete resource can be integrated in our system as a service in a generic way. This takes place in the service layer which is the next abstraction stage after the resource layer. For more information about the resource layer's constitution see (Roth et al., 2011).

First of all, we give a definition of our service comprehension: A service wraps an IT resource to provide a standardized interface (used by our system) which captures the aforementioned resource similarities (IT-as-a-Service).

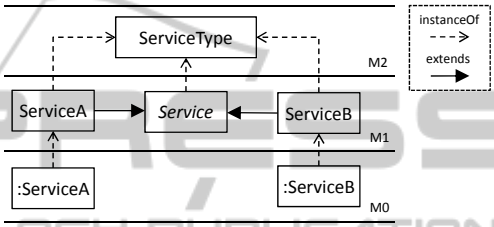

Figure 1: Abstract service description using meta modeling.

To concretely describe our understanding of services, we resort to meta modeling (Clark et al., 2008) as a basic principle (Figure) because a meta model provides information about a designated set of models which is also mentioned in the definition of a meta model by Seidewitz: "A metamodel makes statements about what can be expressed in the valid models of a certain modeling language". (Seidewitz, 2003) Due to the reason that each model can make statements about other models, a model itself can be seen as a meta model as well. We explicitly use this fact to constitute the base of the service specification: A *Service* is an abstract concept that provides fundamental infrastructural functionality. Specializations of a service may provide some resource-specific structure and functionality that concerns the similarities above. How these similarities are captured is defined by the *ServiceType* concept at *M2* because concrete services are instances of this concept. So, *M2* represents the meta model for *M1*. This *instanceOf* relationship between service and service type is comparable to the relation between class and object in object-oriented programming languages (Armstrong, 2006). According to MOF (OMG, 2006), this is the only valid relationship for elements at a certain meta level to elements at the next higher level. An instance of a concrete service (like *ServiceA* and *ServiceB*) supplies access to a concrete

resource (installation, invocation of provided operations, modification of properties, reconfiguration etc.). Hence, *M1* is the meta model for *M0* and therefore, attributes declared by concrete services can be assigned by associated instances within *M0*.

Based on this approach, in the following, we detail the specification of the service interface more precisely. Both, service and service instance contain a unique **name** for identification purposes on scientist's site.

Furthermore, a service has to be configurable and consequently needs to have **properties**. The possibility to declare properties which can be utilized by the framework is constituted by the association between *ServiceType* and *Property* at level *M2*. Thus, properties can be defined at *M1* and the corresponding assignment of values takes place at *M0*. These values define the state of the Service. Since they correspond to properties of the underlying resource in some way, modifying these values certainly influences the state of the underlying resource, too.

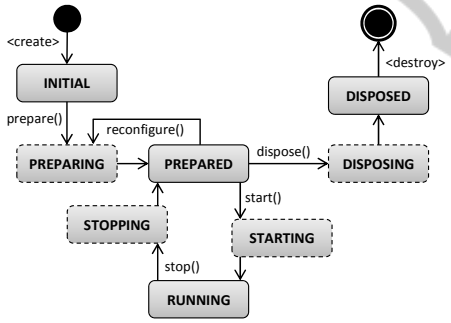

Figure 2: State diagram of service lifecycle.

Analog to the properties, a service must provide an interface that can be used by the system to address some of the resource's functionalities in the form of **operations**. They are also defined at *M1* as instances of *Operation*. Delegation of operation invocations to the encapsulated resource accesses its functionality. This must be specifically implemented in the corresponding method by the developer who integrates a certain resource into the VRE system. However, with this concept, it is possible to define system independent methods as well. Then, such methods must not be instances of *Operation*.

However, some methods like the ones that control the lifecycle (*onPrepare()*, *onDispose()* etc.), have to be implemented by every service. The generic service lifecycle is shown in Figure in form of a state diagram. According to that, each service starts with *INITIAL* state where the user has to assign proper values to the required properties. After that, she starts the *PREPARING* step which is potentially intended to be a long running process. In case of a wrapped desktop application, this represents the installation. After successful completion, the state automatically passes into its successor state. That is true for all states with a dotted border. If a service resides in *PREPARED* state, it can get into *PREPARING* state again after reconfiguration. Once more, in case of an underlying desktop application, this mates with adding or removing features by means of the setup program. *RUNNING* state mainly is relevant when wrapping virtual machines or desktop applications because it tells whether a resource currently is executed or not. Since starting and stopping of web applications often is not necessary or even not allowed, for the corresponding service this state is superfluous. If a service is not needed any more, it can be released and then it gets into *DISPOSED* state. In case of encapsulated desktop applications, that forces an attendant uninstallation. All state transitions must be implemented accordingly by every concrete service.

5 CONCLUSIONS AND OUTLOOK

In this article, we present a solution that allows for the integration of arbitrary IT resources (primarily web applications, desktop applications, virtual machines and storage spaces) into a framework to build virtual research environments. As mentioned above, a scientist's personal VRE is constituted by a set of projects in which she participates with a certain role (e.g. project leader). Each project, in turn, consists of a number of service instances that represent wrappers for concrete occurrences of IT resources (e.g. an instance of a virtual machine or a concretely installed office application). Due to the generic applicability of our solution, the framework handles resources as black boxes but with a well-defined interface implemented by the corresponding services.

A service's interface can be classified into three functional aspects of which the first one is mandatory. It is the state based aspect that encapsulates the lifecycle of the underlying resource. The second one is called data based aspect and contains all relevant data for performing certain state transitions concerning the lifecycle (e.g. parameters needed for installation like the target path). The behavioral aspect is the last one; it comprises all

operations which a service provides. For specifying the generic service interface, we use concepts stemming from the meta modeling approach. Thereby, we define the so called service type which represents the meta class for each service. Again, a service itself constitutes the meta class for corresponding service instances.

Programmatically, we have implemented this meta hierarchy by using annotations or alternatively external configuration files. As an example, the usage of XML files to describe interfaces is explained in (Bramley et al., 2000). For a proof of concept, we implemented a prototype of the proposed VRE framework in context of the African Studies use case. Therefore, we supply services for several resources, i.e. concretely for virtual machines using VMware vSphere, network storage with NetApp, Microsoft Windows Server 7 and Microsoft Office 2010.

Future steps concern the generic applicability challenge again, i.e. integration of further resources is the major task. This addresses widely-used desktop applications (e.g. existent office tools) as well as groupware solutions already provided by the university.

ACKNOWLEDGEMENTS

The proposed framework is called ViATOR which is also the name of a research project funded by the Deutsche Forschungsgemeinschaft (INST 106535/2-1). So, we thank this institution which has kindly facilitated our work.

REFERENCES

AMAZON WEB SERVICES LLC. 2010. Overview of Amazon Web Services. [Accessed 06.12.2011].

Armstrong, D. J. 2006. The quarks of object-oriented development. Communications of the ACM, 49, 123-128.

Bramley, R., Chiu, K., Diwan, S., Gannon, D., Govindaraju, M., Mukhi, N., Temko, B. & Yechuri, M. A component based services architecture for building distributed applications. In: *9th International Symposium on High-Performance Distributed Computing*, 2000 Pittsburgh, USA. Published by the IEEE Computer Society, 51.

Carusi, A. & Reimer, T. 2010. Virtual research environment collaborative landscape study.

Chappell, D. 2009. Introducing the azure service platform.

Clark, T., Sammut, P. & Willans, J. 2008. Applied metamodelling: a foundation for language driven development. CETEVA.

Foster, I., Kesselman, C., Nick, J. M. & Tuecke, S. 2002. The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration. Journal Computer, 35.

GOOGLE INC. Google Docs [Online]. Available: http://docs.google.com/ [Accessed 06.12.2011].

Hey, T. & Trefethen, A. E. 2002. The UK e-science core programme and the grid. Future Generation Computer Systems, 18, 1017-1031.

Koutsonikola, V. & Vakali, A. 2004. LDAP: Framework, practices, and trends. Internet Computing, IEEE, 8, 66-72.

Lin, G., Fu, D., Zhu, J. & Dasmalchi, G. 2009. Cloud computing: IT as a service. IT Professional, 11, 10-13.

MICROSOFT CORPARATION 2010. Microsoft Office Web Apps Product Guide.

OMG 2006. Meta Object Facility (MOF) Core Specification. Version 2.0.

Roth, B., Hecht, R., Volz, B. & Jablonski, S. 2011. Towards a Generic Cloud-based Virtual Research Environment. In: *2nd IEEE International Workshop on Applied Cloud Computing*, Munich.

Seidewitz, E. 2003. What models mean. Software, IEEE, 20, 26-32.

Stal, M. 2002. Web services: beyond component-based computing. Communications of the ACM, 45, 71-76.

THOMSON REUTERS. EndNote - Bibliographies Made Easy [Online]. Available: http://www.endnote.com/ [Accessed 06.12.2011].

Turner, M., Budgen, D. & Brereton, P. 2003. Turning software into a service. Computer, 38-44.

W3C. Web Services @ W3C [Online]. Available: http://www.w3.org/2002/ws/ [Accessed 06.12.2011].