Institute of Architecture of Application Systems

Bootstrapping Complex Workflow Middleware Systems into the Cloud

Karolina Vukojevic-Haupt, Florian Haupt, Frank Leymann, and Lukas Reinfurt

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{vukojevic, haupt, leymann, reinfurt}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{INPROC-2014-34
  author    = {Karolina Vukojevic-Haupt and Florian Haupt and
               Frank Leymann and Lukas Reinfurt},
  title     = {Bootstrapping Complex Workflow Middleware Systems into the
               Cloud},
  booktitle = {Proceedings of the IEEE 11th International Conference on
               eScience, eScience 2015, 31. August – 04. September 2015,
               Munich, Germany},
  year      = {2015},
  pages     = {126-135},
  doi       = {10.1109/eScience.2015.69},
  publisher = {IEEE}
}
```


Universität Stuttgart
Germany

Bootstrapping Complex Workflow Middleware Systems into the Cloud

Karolina Vukojevic-Haupt, Florian Haupt, Frank Leymann, and Lukas Reinfurt
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart
Universitätsstr. 38, Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

***Abstract*— The use of Cloud infrastructures together with provisioning technologies can be successfully applied in scenarios where resources are only needed rarely and irregularly, for example simulation workflows in the eScience domain. There has already been proposed a solution for the on-demand provisioning of services required by workflows, but how to automatically provision the needed workflow middleware itself is still an open issue. Although many provisioning technologies are available, it is currently not possible to use them in an integrated, flexible and automated way. The main idea presented in this paper is a multi-step bootstrapping process, starting with a minimal local software component and ending up with a complex workflow middleware running in the Cloud. This minimal software component is called bootware. We define the key requirements for the bootware, present its architecture and discuss the main design decisions and how they fulfil the requirements. The bootware enables to provision complex workflow middleware systems on-demand and automatically in the Cloud and therefore reduces resource consumption and costs.**

Keywords— Bootware, Cloud, Bootstrapping, On-demand Provisioning, Dynamic Provisioning, eScience, SOC

I. INTRODUCTION

The field of eScience is where "IT meets science" [9] in order to advance scientific research by providing novel IT approaches, techniques, and tools to support scientists throughout the life cycle of scientific experiments. One example is the application of the principles of Service Oriented Computing (SOC), where services realize domain specific functions and workflows combine these services in order to model and execute scientific experiments. A basic assumption in SOC is that services are always on and available [11]. For the business domain, especially for production workflows [7], this is a suitable approach. Services are typically used frequently and show a high utilization. There are however some domains where services are only used rarely and not regularly. One example for such a domain is eScience, particularly the application of simulation workflows [2]. In this area, workflow technology is used to model and execute simulation experiments. As simulation experiments in some domains are, depending on the progress of the related research, typically conducted only now and then, simulation workflows are also executed irregularly and rarely [15]. But when a simulation workflow is executed, the required services need a lot of resources as they mostly implement computing intensive algorithms or process huge amount of data. Keeping these services constantly running would result in long periods of underutilization and therefore be a significant waste of resources. As a consequence, when applying workflow technology in some domains, we need new solutions for realizing the principles of SOC.

In our previous research work in the scope of SimTech[1] (Cluster of Excellence in Simulation Technology), which deals with fundamental research in eScience and in particular in the field of simulation technology, we have proposed a solution approach for the sketched mismatch between the principles of SOC and application areas where workflows and services are used rarely and irregularly [18]. The main idea is to provision the services needed by a workflow only when they are needed. Whenever a service is called that is not running yet, it is provisioned automatically. This is achieved by routing all service calls through a specialized Enterprise Service Bus (ESB) which triggers the dynamic provisioning of services that are not running yet. The dynamic provisioning is based on the use of Cloud infrastructures and common provisioning technologies.

The dynamic provisioning of services requires, as described before, a specialized supporting middleware. The assumption that the services are needed rarely and irregularly consequently also applies to the middleware realizing the on-demand provisioning. Following that argumentation, the focus of this work is to expand the on-demand provisioning concept also to the middleware needed to run workflows and provision services. The overall idea is that at any time no workflow is running, there is also no workflow middleware running and therefore no resources required or blocked. Only at the time a workflow shall be executed, the required middleware is provisioned automatically in a Cloud infrastructure. Similarly, during the runtime of the workflow, services are only provisioned when they are needed.

Although Cloud infrastructures are a promising approach to realize this idea, there are still some issues that need to be solved. Provisioning a complex software system like a workflow middleware into a Cloud infrastructure is a challenging task. Today, manual provisioning can be avoided

[1] http://www.simtech.uni-stuttgart.de/

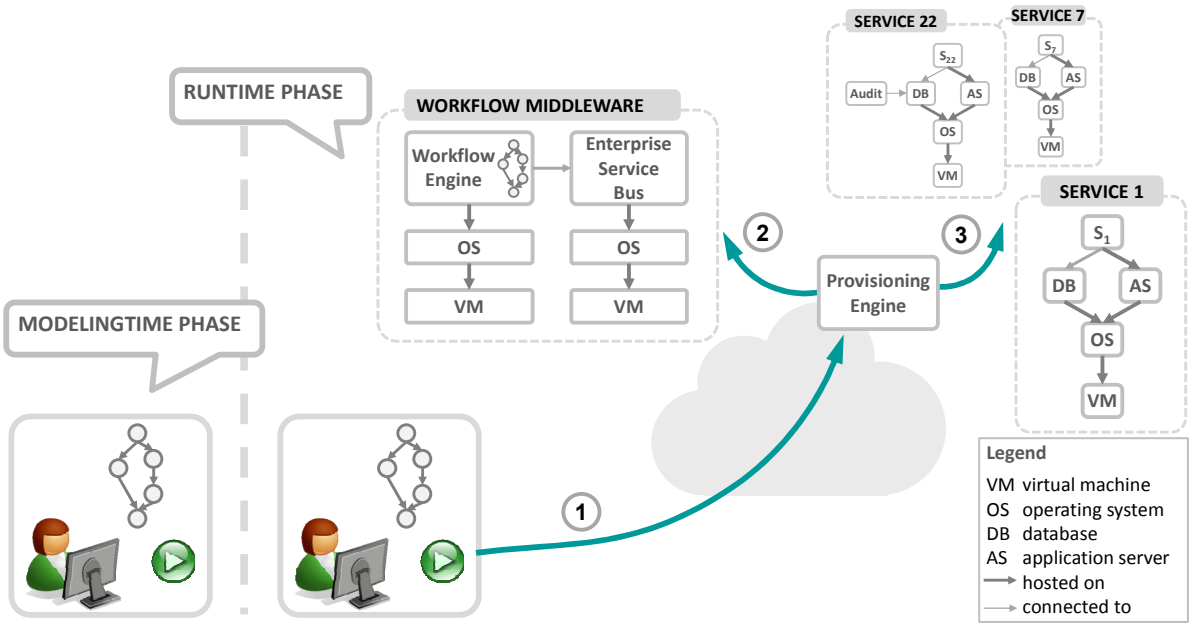

Fig. 1. Approach for on-demand provisioning, figure based on [26]

by using one of several available provisioning technologies or standards such as Chef[2], TOSCA [17] or CloudFormation[3]. However, selecting the appropriate technology and successfully applying it or even combining multiple approaches is a non-trivial task, even for technically experienced users [20]. In addition, many provisioning technologies result in a so called *vendor lock-in*, i.e. they cannot be applied in different Cloud infrastructures without significant migration effort. The ability to dynamically select a Cloud infrastructure is however a desirable feature as it enables to select the best fitting infrastructure for a specific use case.

In this paper we propose to follow a multi-step bootstrapping process to enable the automatic and flexible provisioning of workflow execution middleware into a Cloud infrastructure. We call the system realizing this process the *bootware*. The bootstrapping approach allows starting with a minimal component that kicks off a multi-step process which in the end provisions a complex middleware into a Cloud infrastructure. The bootware is designed to realize this process in an automated and flexible manner, invisible in the background, supporting different Cloud infrastructures and being independent of specific workflow middleware systems.

The research problem we are solving in this work is that in some domains on the one hand scientific workflow management systems are too complex to be managed and operated by scientists, and that on the other hand these systems are only used by small communities so that providing them as a Cloud service is not worth it. Our overall approach is to provision such systems in Cloud infrastructures, but not from a Cloud provider's view but initiated by the user. The bootware, the focus of this paper, is the key enabler of this approach. It allows non-technical scientists from the eScience domain using Cloud resources and running complex systems in the Cloud without having the required technical expertise.

[2] https://www.chef.io/
[3] https://aws.amazon.com/cloudformation/

The rest of this paper is structured as follows. Section II introduces the approach for on-demand provisioning and de-provisioning of workflow middleware and services including their underlying infrastructure and middleware. Section III defines the requirements for the bootware and section IV shows its architecture. In section V we discuss the main design decisions for the bootware and section VI gives some details of its realization. The paper closes with a discussion of related work in section VII and a summary and outlook in section VIII.

II. On-Demand Provisioning

The main contribution of this paper, the bootware, is one of the building blocks that enable the on-demand provisioning and de-provisioning of workflow middleware and services including their underlying infrastructure and middleware. The goal of this section is to introduce the main idea of this approach [18] and to provide the context in which the bootware will be applied. A high level sketch of the whole on-demand provisioning approach is shown in Fig. 1.

In the beginning the user models a workflow on the local system. The workflow execution middleware and the services are not running at this time (Fig. 1, left side). After that, the user starts the workflow execution with one click, everything else is handled invisibly and automatically in the background. In the first step the *provisioning engine* is provisioned into a Cloud infrastructure (Fig. 1, right side). A provisioning engine is a component which is able to provision any kind of service into a Cloud infrastructure. In the second step the provisioning engine provisions the *workflow execution middleware* including the underlying infrastructure into the Cloud (Fig. 1, right side). After that, the workflow is deployed on this middleware and instantiated.

Every time a *service* is called by the workflow, the provisioning engine first provisions the service including the underlying middleware and infrastructure (Fig. 1, right side, third step). To be able to provision a service, the provisioning

engine requires a so called *service package*. A service package contains all artifacts needed to provision a service, for example the description of the topology of the service, the description of the provisioning process, or the implementations of the components of the service.

If a service has processed a request and is not needed anymore, the provisioning engine de-provisions the service including the underlying middleware and infrastructure. Similarly, when the workflow is finished, the provisioning engine de-provisions the workflow middleware including the underlying infrastructure. Finally, the provisioning engine itself is de-provisioned. Thereby, at the end again only the local modeling tool is running (Fig. 1, left side).

To summarize, the concept of on-demand provisioning of workflow execution middleware and services enables users to run their simulation workflows in the Cloud "with only one click". Local on the user's machine there are only the modeling tool and the so-called bootware. The bootware is a small component which triggers the on-demand provisioning process. The workflow middleware and the simulation services are provisioned in the Cloud automatically and on demand.

In [18] we proposed a basic architecture for our on-demand provisioning approach. As part of the architecture, we already introduced the idea of a bootware component. We further developed and detailed this architecture in [19], but only focused on the on-demand provisioning of the services and without giving any additional details about the on-demand provisioning of the workflow middleware. Until now the bootware has only been defined as a placeholder in our architecture. We sketched some basic capabilities of this component but designated the architecture and design of the bootware as one important part of our future work. In the following we will address this open issue.

III. Requirements

The architecture and design of the bootware is driven by a set of requirements. Before presenting the architecture of the bootware and its integration in the existing on-demand provisioning system in section IV, we will first present and discuss the requirements that influenced our solution.

A. "be lightweight" (R1)

The motivating scenario for the bootware and the on-demand provisioning approach is the execution of simulation workflows. One special feature in the context of scientific workflow management systems (SWfMS) is that they are typically operated by the same person during all phases of the lifecycle of a workflow (e.g. modeling, technical refinement, deployment, or execution). This person is typically a specialist in her domain (e.g. physics, biology or engineering) but however has no deep knowledge about the technical foundations of SWfMS or about Cloud technologies. Therefore, it is a basic requirement for SWfMS to hide their technical foundations in order to allow the user to focus on his domain specific tasks (e.g. the modeling of simulations). The same applies to the bootware.

The bootware extends the modeling tool with the capability to execute workflows in Cloud infrastructures following the on-demand provisioning approach. Doing so, the bootware should not increase the complexity of the modeling tool with respect to its installation, configuration and operation. The bootware, as well as the workflow modeling tool, runs locally on the user's machine. Therefore, the resource demand of running these tools should not be significantly increased.

To summarize, the first requirement for the bootware is to support an easy installation, configuration and operation as well as being "small", i.e. occupying as little resources as possible.

B. "be generic" (R2)

The concept of on-demand provisioning and de-provisioning of workflow middleware and services, including their underlying infrastructure and middleware, has been developed to be independent of any specific workflow management system or Cloud infrastructure. The corresponding architecture presented in [18] and [19] has also been defined to be generic and reusable. It follows that the bootware, as an important part of this architecture, should be generic and reusable as well.

C. "be robust" (R3)

The bootware realizes the transition from local workflow modeling to the execution of the workflow in a Cloud infrastructure. This process requires communication between local components and remote components in the Cloud. Given the fact that this communication cannot be assumed to be reliable together with the fact that the provisioning process typically takes some time, it is a requirement for the bootware to be robust against failures in this remote communication.

D. "be extensible" (R4)

The on-demand provisioning approach should be able to allow the user to select an appropriate Cloud infrastructure for the execution of a workflow. The suitability of a Cloud infrastructure for the execution of a specific workflow may depend on costs, performance, or privacy policies. As a consequence, the on-demand provisioning system and therefore also the bootware have to support multiple Cloud infrastructures. In addition we require that the set of supported Clouds has to be extensible, allowing to flexibly adapt to the user's needs in the future. Supporting multiple Clouds may also require supporting different provisioning engines. Therefore, the bootware is in addition required to be extensible with respect to provisioning engine support.

IV. Bootstrapping Enabled Architecture

In section II we presented the concept and basic operation of the on-demand provisioning of workflow middleware and services. An architecture for the on-demand provisioning of services was already introduced in [18] and [19]. In this paper we expand this architecture by the bootware, realizing the on-demand provisioning of the workflow middleware (i.e. we will replace the bootware placeholder that was part of our previous architecture). In the following we will introduce the bootware

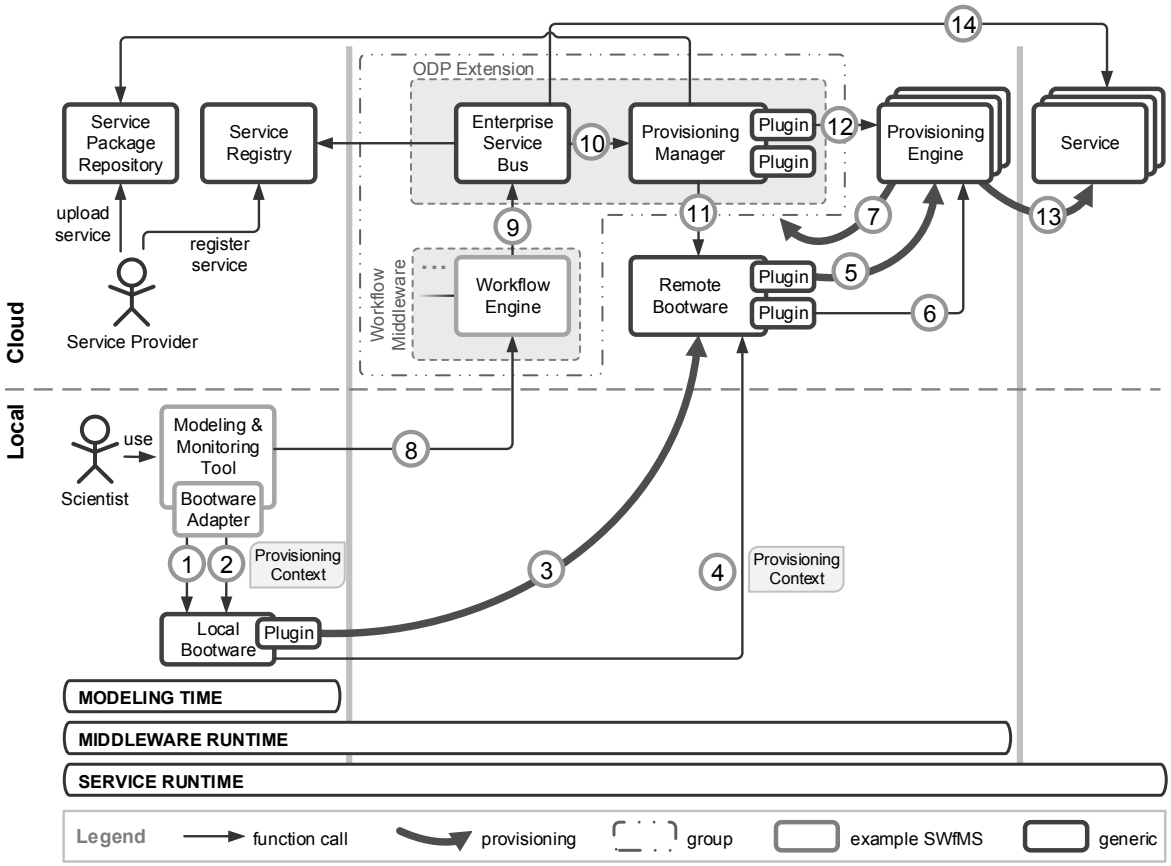

Fig. 2. Bootstrapping enabled achitecture for on-demand provisioning

system as well as its integration with the existing architecture for the on-demand provisioning of services. The bootware system comprises altogether three components: the *bootware adapter*, the *local bootware*, and the *remote bootware*.

A fundamental characteristic of the existing architecture for on-demand provisioning of services is its independence of a specific workflow management system and a specific Cloud infrastructure. This characteristic should be preserved in the design and integration of the bootware.

In Fig. 2 we present the architecture of the overall system. In the lower part, all components running locally on the user's machine are shown, whereas the components in the upper part are running in a Cloud environment. In addition to this spatial division we can also structure our architecture in course of time. Not every component of the architecture exists all the time. The components of the *modeling time phase* are always available. In contrast, the components specific for the *middleware runtime phase* and the *service runtime phase* are provisioned and de-provisioned on demand at runtime.

In the lifecycle of the whole system we can identify three use cases for the bootware. The *Bootstrapping* use case describes the usage of the bootware to provision the workflow middleware. The use case *Provisioning Engine Management* describes the tasks of the bootware during the execution of a workflow. The use case *Shutdown* describes the usage of the bootware to de-provision the workflow middleware. In the following we will discuss these three use cases in detail.

A. Bootstrapping

The modeling of workflows is performed during the modeling time phase. The workflow middleware, needed for the execution of the workflows, is not provisioned or running at this time. If a workflow is to be executed, the workflow middleware first has to be provisioned into a Cloud infrastructure. In our approach this is done automatically and invisibly, and is one of the main tasks of the bootware.

After modeling a workflow, the scientist can start it directly in the *modeling tool*. Starting a workflow triggers the *bootware adapter* which connects the modeling tool with the bootware. The bootware adapter first starts the *local bootware* (Fig. 2, step 1) and then passes the so-called *provisioning context* (step 2). This context contains user-provided configuration data, for example access credentials for Cloud infrastructures or a reference to the service package of the required workflow middleware. The local bootware then provisions the *remote bootware*, depending on the given provisioning context, into a specific Cloud environment (step 3) like for example Amazon Web Services[4], OpenStack[5], OpenNebula[6] or CloudMesh [6]. Afterwards, the local bootware passes the provisioning context to the remote bootware (step 4).

[4] http://aws.amazon.com
[5] https://www.openstack.org/
[6] http://opennebula.org/

Based on the provisioning context the remote bootware determines which service package is needed to provision the workflow middleware. The service package is described in a specific provisioning language, such as Chef, TOSCA, HEAT or CloudFormation, or e.g. available as Docker image. For every provisioning language a corresponding provisioning engine is needed that is able to process this service package format (and therefore to provision the described service into a Cloud environment). The remote bootware determines, based on the passed provisioning context, a suitable provisioning engine and provisions it in the user-specified Cloud environment (step 5). The service package of the workflow middleware is stored in the global *service package repository*. The remote bootware passes this service package (or a reference to it) to the just provisioned *provisioning engine* (step 6), which then provisions the *workflow middleware* into the Cloud (step 7). As illustrated in Fig. 2, the middleware that is provisioned comprises an arbitrary workflow middleware as well as a generic extension enabling the on-demand provisioning of the services (*ODP extension*). After the middleware has been successfully provisioned, the provisioning engine returns some information, like endpoint addresses of the workflow middleware, to the remote bootware. The remote bootware then forwards this data to the local bootware, which in turn passes it to the bootware adapter. The bootware adapter uses this information to connect the local modeling and monitoring tool to the workflow middleware. Finally, the workflow to be started can be deployed on the workflow middleware and afterwards be instantiated (step 8).

B. Provisioning Engine Management

During its execution, a workflow calls several services. Every service call from the workflow engine is forwarded to the *ESB* (Fig. 2, step 9). The ESB is responsible for the service binding. For every service call it queries the global *service registry* to determine a suitable service matching the functional and non-functional requirements of the requestor. If there is no suitable service running, the ESB delegates the processing of the request to the *provisioning manager* (step 10). The provisioning manager is then responsible to provision a suitable service. It first queries the service package repository to determine a suitable service package. Details of this so-called *service package selection* have been already published in [19]. To provision the selected service package, a suitable provisioning engine is needed. The access to the provisioning engines is managed by the remote bootware. The provisioning manager asks the remote bootware for a specific provisioning engine (step 11). If the required provisioning engine is not running, the remote bootware first provisions it into the Cloud (step 5). The provisioning manager then receives the endpoint address of the requested provisioning engine and then calls this provision engine in order to provision the previously selected service package (step 12, step 13). The provisioning manager is extensible through a plugin-system and can therefore interact with a multitude of different provisioning engines.

As soon as the service is provisioned, the endpoint address of the service is known and passed back to the ESB. The ESB then forwards the original service call received from the workflow engine to the just provisioned service (step 14). If the service call has been processed and the service is not needed anymore, the provisioning engine de-provisions it.

C. Shutdown

When the workflow execution is finished, the workflow middleware and all provisioned provisioning engines are not needed anymore. In this case the bootware initiates the de-provisioning process. The remote bootware de-provisions the extended workflow middleware using a suitable provisioning engine; afterwards it de-provisions all running provisioning engines. Finally, the remote bootware itself is de-provisioned by the local bootware. After that, only the global directories are running in the Cloud.

V. Design Decisions

In section IV we presented the on-demand provisioning architecture focusing on the integration of the bootware. In addition, we explained the operation of the bootware by identifying three use cases. In the following, we will present, discuss, and justify the most important design decisions for the architecture of the bootware. We will also show how the design decisions and the resulting architecture fulfill the set of requirements that has been identified in section III.

A. Component Division

The local bootware and the remote bootware are the core components of the bootware system. The remote bootware handles the main part of the provisioning process. It provisions the workflow middleware as well as the required provisioning engine into the Cloud infrastructure selected by the user. The remote bootware is also needed during the execution of a workflow. It is responsible to manage access to the available provisioning engines and to provision additional provisioning engines, if needed. The local bootware however has only the task to provision the remote bootware into the Cloud infrastructure selected by the user.

Dividing the bootware functionality between a local and a remote component together with the decision to realize most of the functionality in the remote bootware has the consequence that the local bootware has to implement only few and simple tasks. By that it is possible to keep the local bootware small and simple, which fulfills requirement R1 (“be lightweight”).

The remote bootware runs in a Cloud infrastructure selected by the user. The workflow middleware and the required provisioning engine are provisioned into the same Cloud infrastructure. As a result, most of the communication and data transfer during the bootstrapping process happens within the same Cloud infrastructure. On the other hand, there is only little communication between the local bootware and the remote bootware. This division therefore contributes to the robustness of the bootware system by reducing the amount of unreliable remote communication. This addresses requirement R3 (“be robust”). An additional advantage of this design is cost reduction. Data transfer within a Cloud infrastructure is in general much cheaper than data transfer crossing the boundaries of a Cloud infrastructure [8]. Similarly, data transmission rates within a Cloud infrastructure are typically

much higher than data transmission rates crossing the boundaries of a Cloud infrastructure [8].

B. Modeling Tool Integration

Many SWfMS support to model, start, and monitor a workflow using one integrated tool [13]. It is typically assumed that the workflow middleware needed to execute a workflow is already available. If a workflow is executed, it is deployed on a known workflow middleware and afterwards instantiated. When following the on-demand provisioning approach presented before, there is no workflow middleware running during modeling time. Therefore, it is necessary to adapt the existing process for executing a workflow. If a workflow is executed, first the needed workflow middleware has to be provisioned before it can be deployed and instantiated. Specifically, this requires to adapt the workflow modeling tool so that the bootware is activated at the right time. In our architecture this is achieved by the bootware adapter.

The bootware adapter is integrated into the modeling tool so that it is activated before a workflow is deployed. The bootware adapter then calls the local bootware, which in turn initiates the provisioning of the workflow middleware. When the provisioning of the workflow middleware is completed, the bootware adapter receives the endpoint address(es) of the just provisioned workflow middleware as a return value. The bootware adapter stores this address(es) in the modeling tool. Subsequently, the usual process to deploy and instantiate a workflow can be executed.

The application logic needed to connect the bootware with a workflow modeling tool depends on the modeling tool. Therefore, we outsource it to an adapter specific for the modeling tool, the bootware adapter. The core functionality of the bootware is, however, realized by the local bootware and the remote bootware. They are independent of any specific modeling tool and by that fulfill requirement R2 (“be generic”).

C. Plugins

It is a fundamental property of the bootware not to be restricted to a particular Cloud infrastructure. It should be possible for a user to select, depending on his current requirements, an appropriate Cloud infrastructure in which the bootware will then carry out the middleware provisioning process. As a consequence, the bootware has to be able to support different Cloud infrastructures.

Likewise, it should be possible for the user to model a workflow that requires services available in different service package formats. This means that different provisioning engines may be required during the execution of one workflow. These provisioning engines in turn have to be provisioned and managed by the remote bootware.

For both, the support of different Cloud infrastructures as well as the support of different provisioning engines, it is a great advantage when the set of supported systems is easily extensible. By that, the on-demand provisioning system is getting flexible and future-proof. It can easily be adapted to changes in the user’s requirements as well as to changes in the available Cloud infrastructures and services.

In our bootware system extensibility is solved by following a plugin based approach. The local bootware has plugins for different Cloud infrastructures. By that, it is able to provision the remote bootware into different Cloud infrastructures. The remote bootware has plugins for different provisioning engines as well as for different Cloud infrastructures. By that, it is able to provision different provisioning engines in different Cloud infrastructures. By following this solution approach we address requirement R4 (“be extensible”).

D. Provisioning Engine Management

During the bootstrapping process, the remote bootware provisions a suitable provisioning engine which is then used to provision the workflow middleware. During the execution of a workflow, the provisioning manager, which is part of the on-demand extension of the workflow middleware, also requires access to provisioning engines. If a service required by the workflow is not provisioned yet, the provisioning manager is responsible to provision it using a suitable provisioning engine. If the required services are available in different service package formats, multiple provisioning engines are needed during the execution of a workflow [19].

The provisioning engines needed by the provisioning manager are, if they are not provisioned yet, also supposed to be provisioned on demand. This functionality, provisioning a required provisioning engine into a Cloud infrastructure, is basically already realized by the remote bootware. In our on-demand provisioning architecture the remote bootware provides this functionality additionally over an external interface. This allows the provisioning manager to reuse this functionality and therefore avoids redundancy.

VI. Realization

In this section we will present our realization of the bootware system. First, we will give some details about the internal architecture of the local bootware as well as the remote bootware. Second, we present an overview about some technical details of the implementation and then discuss the results of testing the bootware system.

A. Internal Bootware Architecture

The internal architecture of the local bootware is shown in Fig. 3. The bootware offers a *web service interface* to enable external access. The internal processing is realized as *state machine*, which implements the local bootware’s core functionality by interacting with the *instance store*, the *plugin manager*, and the *plugins* shown in the upper part of the figure. The realization of the internal processing as state machine enables to describe it on a formal basis in a clear, comprehensible manner with well-defined execution semantics.

The instance store manages the provisioned components. It knows which components have already been provisioned and in addition stores all data needed to use and de-provision the remote bootware (e.g. the endpoint address of the remote bootware or access credentials to the underlying virtual machine).

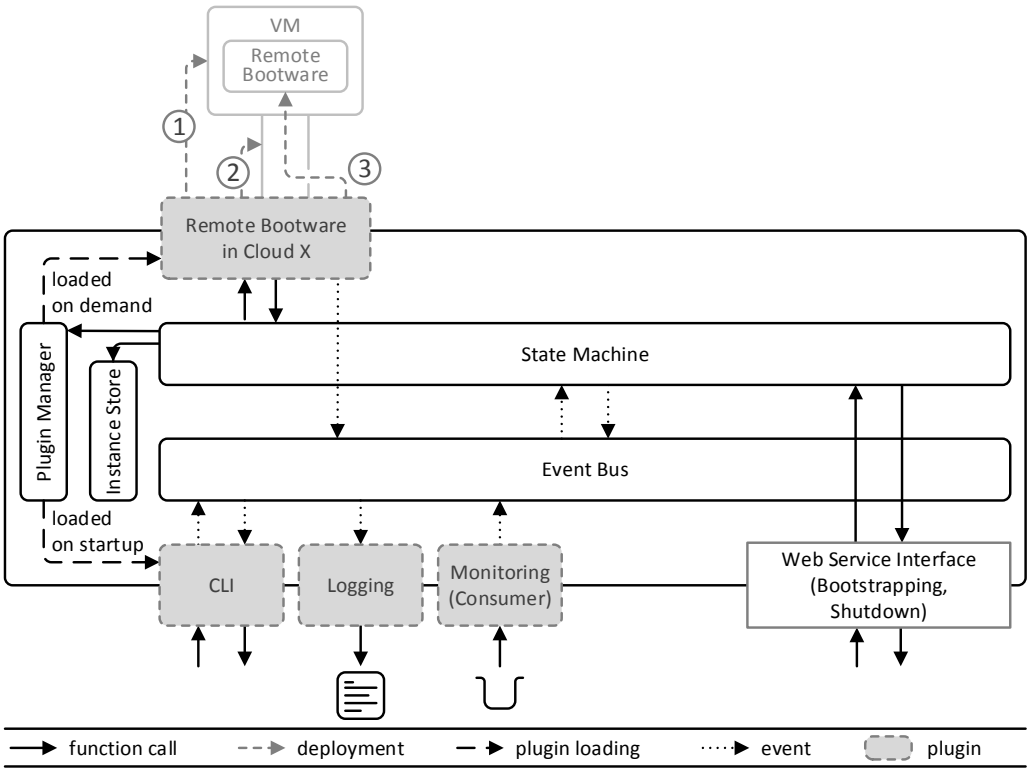

Fig. 3. Internal architecture of the local bootware

The plugin manager is used to load and activate plugins. Some plugins are loaded when starting the local bootware (Fig. 3, lower part). In addition, the plugin manager supports to dynamically load and activate plugins at runtime. The plugin manager also supports to load plugins from a global and remote plugin repository, which enables the central management and sharing of plugins.

The local bootware requires exactly one plugin for the provisioning of the remote bootware. Depending on the desired target Cloud infrastructure defined in the provisioning context, the plugin manager loads a suitable plugin to be used to provision the remote bootware. The provisioning of the remote bootware comprises to first provision a virtual machine (VM) (Fig. 3, upper part, step 1), then open a SSH connection to this VM (step 2) and finally to install and start the remote bootware using this connection (step 3).

The *event bus* is used to integrate plugins realizing secondary functionality (Fig. 3, lower part) loosely coupled and extensible into the bootware. A *CLI plugin* can allow to access the bootware using a command line interface, e.g. for maintenance purposes. The *logging plugin* records the execution of the bootware. The *monitoring plugin* receives execution events from the remote bootware and forwards them to the bootware adapter. This allows using the workflow modeling and monitoring tool to provide feedback about the state of the provisioning process to the user.

The internal architecture of the remote bootware is shown in Fig. 4. Its general structure as well as most of the components are the same as for the local bootware. The main application logic is again implemented as a state machine; however, the application logic itself differs from the local bootware. Another difference compared to the local bootware is the set of plugins that are used by the remote bootware. During the bootstrapping process, the remote bootware requires two plugins in order to provision the workflow middleware: first, a plugin to provision a suitable provisioning engine and second, a plugin that uses this provisioning engine to provision the workflow middleware.

During the runtime of a workflow, the provisioning manager may call the remote bootware to provision additional provisioning engines. Therefore, the remote bootware provides an additional web service interface to be used by the provisioning manager. In addition, it is able to run and manage multiple provisioning engine plugins.

All events that occur during the runtime of the remote bootware are published to the internal event bus and processed by the monitoring plugin. The plugin publishes all events to a *message queue*. As mentioned before, the monitoring of the local bootware connects to this queue and therefore receives and processes all monitoring events of the remote bootware.

B. Implementation and Validation

In this section we will show some implementation details and also validate the bootware system. Our main goal was to remove the need to keep complex middleware systems constantly running and also to allow non-technical scientists from the eScience domain to use Cloud resources for running complex systems in the Cloud without having the required technical expertise. Therefore, the main questions a validation of the bootware should answer are "Is it working?" and "Is it working in reasonable time?". In the following we will show, that the answer to both questions is “Yes”.

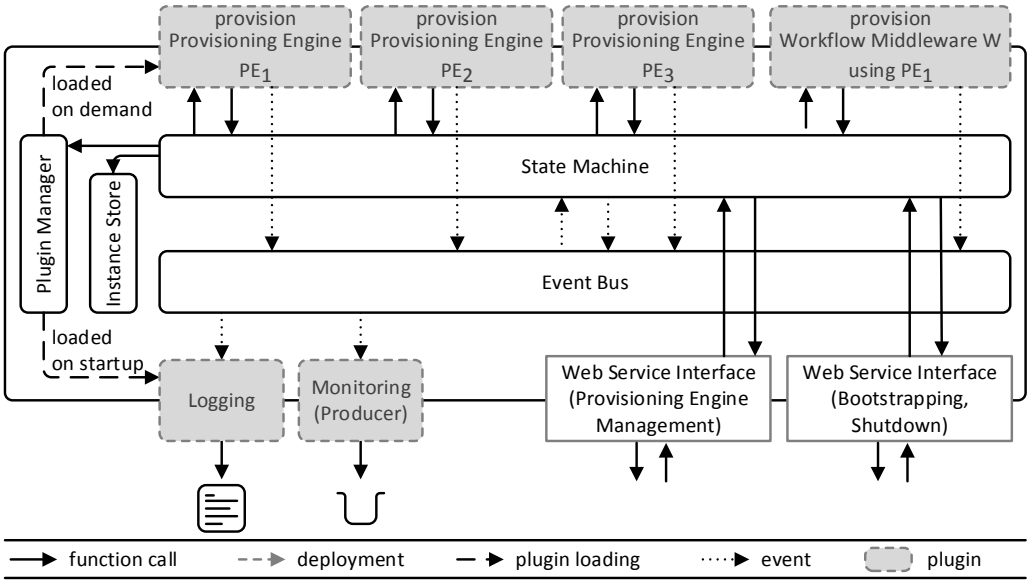

Fig. 4. Internal architecture of the remote bootware

In our work we are using the SimTech SWfMS, comprising an integrated workflow modeling and monitoring tool as well as a corresponding workflow middleware [15], as an exemplary scientific workflow management system. The SimTech workflow middleware has already been extended to support the on-demand provisioning of services [18][19]. For the validation of the bootware, we created a TOSCA based service package for the SimTech workflow middleware, a so called CSAR file (Cloud Service ARchive). In addition, we extended the workflow modeling and monitoring tool of the SimTech SWfMS with a specific bootware adapter. The adapter hooks into the process that is triggered when the user pushes a "run" button in order to execute a workflow.

The local and remote bootware have been implemented based on the OSGi framework Apache Felix[7]. Its native support for creating dynamically extensible software components allows an easy realization of the desired plugin architecture. The internal event bus of the bootware is based on MBassador[8], the state machine has been realized using squirrel-foundation[9].

Regarding the bootware plugins, we provide an initial set to enable testing of the whole system. For the local bootware we implemented a plugin for provisioning the remote bootware into the AWS EC2 Cloud infrastructure. For the remote bootware, we implemented a plugin to provision the OpenTOSCA provisioning engine [1] into the AWS EC2 Cloud infrastructure and a second plugin that allows using OpenTOSCA to provision CSAR service packages.

All available bootware plugins are provided by a plugin repository. The repository is assumed to be globally available and provides a simple REST API to query and download plugins. It has been developed following the approach proposed in [4]. The local bootware as well as the remote bootware access the plugin repository at runtime to download the required plugins.

TABLE I. EXECUTION TIMES SAMPLE

action	duration
provision workflow middleware	**~17 minutes**
start local bootware	5 seconds
download bootware plugins	15 seconds
local bootware provisions remote bootware:	~102 seconds
provision VM	30 seconds
open SSH connection	20 seconds
upload and start remote bootware	50 seconds
download remote bootware plugins	2 seconds
remote bootware provisions OpenTOSCA provision engine	~7.5 minutes
provision VM	30 seconds
SSH connection	20 seconds
execute installation scripts for OpenTOSCA	6.5 minutes
OpenTOSCA provisions SimTech workflow middleware & ODP extension	~ 7.0 minutes
download remote bootware plugin to interact with OpenTOSCA	1 second
upload service package	2.0 minutes
instantiate service package	3.5 minutes
configure middleware	1.5 minutes
shutdown	**2.5 minutes**
workflow middleware & ODP extension	40 seconds
OpenTOSCA	60 seconds
remote bootware	40 seconds
local bootware	10 seconds

After implementing and testing all components of the bootstrapping enabled architecture, we conducted an overall system test for the on-demand provisioning of the workflow middleware using an exemplary workflow. The bootstrapping process realized by the bootware is independent of the actual workflow that is afterwards deployed and executed on the workflow middleware. Details about the workflow are therefore out of scope of this validation. The execution times of

[7] http://felix.apache.org/

[8] https://github.com/bennidi/mbassador

[9] https://github.com/hekailiang/squirrel

the single steps of the provisioning process as well as the de-provisioning process are shown in Table I. After pushing the "run" button in the workflow modeling tool, it took overall around 17 minutes to provision the SimTech workflow middleware. During the provisioning process altogether three virtual machines are created, one for the remote bootware, one for the OpenTOSCA provisioning engine and one for the SimTech workflow middleware. The de-provisioning process is significantly faster, it took around 2.5 minutes to shut down the complete middleware.

Given the numbers shown in Table I, it takes around 17 minutes to deploy and instantiate a workflow. Although this might seem to be too long at a first glance, the duration of the provisioning process is acceptable when comparing it to the execution time of a typical simulation workflow. Simulation workflows are often very long-running. The OPAL simulation, which has been realized using the SimTech SWfMS, takes weeks up to months to execute [14]. Compared to this, a provisioning time of 17 minutes is quite negligible.

VII. Related Work

The main contribution of this paper is the bootware system. At first glance, there exists no other work that is comparable to what we presented. However, the bootware reuses some concepts that have successfully been applied in different domains and scenarios. In the following, we will give an overview about existing work related to the main concepts the bootware is based on. Afterwards, we consider the bootware as part of our overall on-demand provisioning approach and compare this to other approaches for running scientific workflows in Cloud infrastructures.

The term *bootstrapping* itself is widely used in different domains and therefore overloaded. In statistics, bootstrapping refers to a common resampling method [3]. In linguistics, the term bootstrapping refers to a theory explaining the process of language acquisition of children [12]. The bootstrapping theory describes how children with little linguistic knowledge are able to learn the semantics of new terms and therefore enhance their linguistic knowledge. Bootstrapping is also very often used in the computer science domain. In the context of compiler construction, bootstrapping describes the process of implementing a compiler in the language that it is supposed to compile, a so called self-hosting compiler [5]. Bootstrapping also describes the process of using simple tools in order to develop more sophisticated tools. One common example is to use a simple text editor and an assembler to develop an enhanced IDE (integrated development environment) and a compiler for higher level programming languages. The common idea of bootstrapping, independent of the domain it is applied to, can be summarized as using something simple to obtain or activate something more complex. This idea has also been applied in the design of our bootware. The small and simple local bootware at first activates the remote bootware, which then activates a provisioning engine, which in turn can be used to finally provision complex middleware systems.

In [16] the term bootstrapping denotes the challenge to discover and connect the nodes of complex distributed systems when provisioning them into Cloud infrastructures. In contrast to existing approaches, which typically rely on centralized components, the authors propose a decentralized and distributed approach. The challenge of discovering the members of a distributed system is relevant for systems, where the set of members is not known in advance and dynamically changed. In context of the bootware or the on-demand provisioning approach in general this assumption does not apply.

The approach presented in [10] focuses on the on-demand provisioning of security services like SSH or VPN for infrastructure Cloud services (IaaS). The bootstrapping process is defined as a set of communication activities that are designed to provide secure access to remote resources. The term *services* refers to rather low level services that are needed to provide access to Cloud infrastructures. In contrast, in our work a service represents some domain specific application logic that can be run on top of Cloud infrastructures.

There are several web portals available that allow for the modeling and (Cloud based) remote execution of scientific workflows. Similarly to our approach, their goal is to support scientists in their work by hiding technical complexity.

The eScience Central platform [21] follows a web-only approach. It provides a web based graphical user interface for the modeling of data-driven workflows as well as for the implementation of the services called by such workflows. For the execution of workflows the eScience Central server is connected to a set of so called *workflow enactment nodes*, each of them running an instance of a workflow engine. In order to execute a workflow, a corresponding request is send to a queue and then processed by an idle workflow enactment node. Both, the eScience Central server as well as the workflow enactment nodes, can be run in a Cloud environment. The main difference to our bootstrapping approach is that the whole system, the eScience Central server as well as the workflow enactment nodes, is provisioned once in advance and then remains continuously running. For execution, a workflow together with all required services and data is simply transferred to an existing workflow enactment node.

The CloudMan system [22], part of the Galaxy project [23], enables the execution of Galaxy workflows on Cloud based Galaxy clusters. Before a workflow can be executed, the CloudMan system has to be deployed and a cluster has to be configured. The creation and management of compute clusters is realized in a user friendly way hiding as many technical details as possible. At runtime, CloudMan supports to scale the compute and storage resources as needed. In contrast to our approach, the provisioning and de-provisioning of the required resources (the cluster) as well as the scaling have to be initiated manually by the user.

Web portals for the modeling and execution of scientific workflows are available for some scientific workflow systems but not for all. Our bootstrapping approach provides a generic solution to run complex workflow systems in Cloud infrastructures independent of any specific workflow system or Cloud infrastructure. In addition, we avoid the need to constantly run a portal server, which may be used only rarely and irregularly while constantly producing costs.

VIII. Summary and Outlook

The main contributions of this paper are the concept as well as the architecture of the bootware system, which comprises three components: the bootware adapter, the local bootware and the remote bootware. The bootware continues existing work for the on-demand provisioning of services in order to enable the additional on-demand provisioning of the workflow middleware. Combining these two approaches enables the fully automated execution of workflows in Cloud infrastructures with minimized resource consumption and costs. All this is realized invisibly in the background; the user simply models a workflow and executes it "with only one click".

In our work, we have first defined a set of key requirements for the bootware and then presented its architecture. We discussed the main design decisions of this architecture and were able to show that it fulfills the requirements identified before. We complemented the paper by giving some details about the realization of the bootware as well as about its successful application as part of the overall on-demand provisioning system.

As part of our future work we plan to optimize the on-demand provisioning of services. Although the current approach optimizes resource consumption, it may be suboptimal regarding costs or latency. Depending on the billing practices of a Cloud provider it may be cheaper to keep a service running until it is needed the next time instead of de-provisioning it and then provisioning it again later. In the end there may be a set of optimization strategies and heuristics that may be applied depending on the specific use case and the user's requirements.

Another aspect we are working on is to perform a thorough evaluation of our overall system for the on-demand provisioning of workflow middleware and services. For that, we plan to apply our system, including the bootware, to real world use cases, for example OPAL [24], a Kinetic Monte Carlo simulation of solid bodies, or a finite element based simulation of structure changes in bones [25]. This will allow us to further demonstrate the usefulness of our approach and to identify potential improvements.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC310/1) at the University of Stuttgart.

References

[1] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner, "OpenTOSCA - A Runtime for TOSCA-based Cloud Applications," Proceedings of ICSOC'13.

[2] E. Deelman, D. Gannon, and M. Shields, "Workflows for e-Science," Springer-Verlag London Limited, 2007.

[3] B. Efron, "Bootstrap methods: another look at the jackknife," The annals of Statistics, 1-26, 1979.

[4] F. Haupt, D. Karastoyanova, F. Leymann, and B. Schroth, "A model-driven approach for REST compliant services," IEEE International Conference on Web Services (ICWS 2014), IEEE, 2014.

[5] M. Lam, R. Sethi, J.D. Ullman, and A. Aho, "Compilers: Principles, Techniques, and Tools," 2006.

[6] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G.C. Fox, "Accessing multiple clouds with cloudmesh," Proceedings of the 2014 ACM international workshop on Software-defined ecosystems, ACM, 2014.

[7] F. Leymann and D. Roller, "Production workflow: concepts and techniques," Prentice Hall PTR, 2000.

[8] A. Li et al., "CloudCmp: Comparing Public Cloud Providers," 10th ACM SIGCOMM conference on Internet measurement, 2010.

[9] T. Hey, S. Tansley, and K. Tolle (Eds.), "The Fourth Paradigm. Data-Intensive Scientific Discovery," Microsoft Research, 2009.

[10] P. Membrey, K.C. Chan, C. Ngo, Y. Demchenko, and C. de Laat, "Trusted virtual infrastructure bootstrapping for on demand services," Seventh International Conference on Availability, Reliability and Security (ARES2012), IEEE, 2012.

[11] M.P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," Proceedings of WISE 2003, 2003.

[12] S. Pinker, "Language learnability and language development, with new commentary by the author," Harvard University Press, 2009.

[13] M. Sonntag and D. Karastoyanova, "Next Generation Interactive Scientific Experimenting Based On The Workflow Technology," Proceedings of MS 2010, 2010.

[14] M. Sonntag, S. Hotta, D. Karastoyanova, D. Molnar, and S. Schmauder, „Using services and service compositions to enable the distributed execution of legacy simulation applications," Towards a Service-Based Internet, Springer Berlin Heidelberg, 2011.

[15] M. Sonntag and D. Karastoyanova, "Ad hoc Iteration and Re-execution of Activities in Workflows," International Journal On Advances in Software. Vol. 5 (1 & 2), Xpert Publishing Services, 2012.

[16] P. Szilágyi, "Decentralized bootstrapping in clouds," IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY 2012), IEEE, 2012.

[17] TOSCA, "Topology and Orchestration Specification for Cloud Applications Version 1.0," 2013. Available from: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[18] K. Vukojevic-Haupt, D. Karastoyanova, and F. Leymann, "On-demand Provisioning of Infrastructure, Middleware and Services for Simulation Workflows," SOCA'13, IEEE, 2013.

[19] K. Vukojevic-Haupt, F. Haupt, D. Karastoyanova, and F. Leymann, "Service Selection for On-demand Provisioned Services," Proceedings of the 18th IEEE International EDOC Conference (EDOC 2014), 2014.

[20] J. Wettinger, V. Andrikopoulos, S. Strauch, and F. Leymann, "Characterizing and Evaluating Different Deployment Approaches for Cloud Applications," IC2E 2014, IEEE, 2014.

[21] H. Hiden, S. Woodman, P. Watson, and J. Cala, "Developing cloud applications using the e-science central platform," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 371(1983), 2013.

[22] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters. BMC bioinformatics", 11(Suppl 12), 2010.

[23] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," Genome Biology 2010, 11:R86, 2010.

[24] P. Binkele and S. Schmauder, "An atomistic Monte Carlo simulation for a precipitaation in a binary system," International Journal for Materials Research, 94, 2003.

[25] R. Krause et al., "Scientific Workflows for Bone Remodelling Simulations," Applied Mathematics and Mechanics, 13(1), 2013.

[26] K. Vukojevic-Haupt, F. Haupt, and F. Leymann, "On-demand Provisioning of Workflow Middleware and Services - An Overview," 9th Symposium and Summer School on SOC, IBM, 2015.

All links were last followed on 24.07.2015