On-demand Provisioning of Infrastructure, Middleware and Services for Simulation Workflows

Karolina Vukojevic-Haupt, Dimka Karastoyanova, and Frank Leymann
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

***Abstract* – Service orientation is a mainstream paradigm in business applications and gains even greater acceptance in the very active field of eScience. In SOC service binding strategies have been defined to specify the point in time a service can be discovered and selected for use, namely static binding, dynamic binding at deployment or at run time, and dynamic service deployment. The basic assumption in all these strategies is that the software stack and infrastructure necessary to execute the services are already available. While in service-based business applications this is typically a valid assumption in scientific applications it is often not the case. Therefore, in this work we introduce a new binding strategy for services we call *on-demand provisioning* which entails *provisioning of the software stack* necessary for the service *and* subsequent *dynamic deployment of the service* itself. Towards this goal, we also contribute a middleware architecture that enables the provisioning of the software stack – functionality unavailable in conventional service middlewares. We demonstrate the approach and the capabilities of the middleware and the current state of the implementation of our approach. For this purpose we use an example application from the field of eScience that comprises a scientific workflow management system for simulations.**

Keywords—on-demand provisioning and deprovisioning; dynamic service deployment and provisioning; provisioning engine; bootware; simulation workflows; eScience; SOC; Cloud

I. INTRODUCTION

The field of *eScience* is where "IT meets science" [1] striving to enable ground-breaking scientific research by providing novel IT approaches, techniques and tools to support scientists throughout the life cycle of scientific experiments. The major focus is on shortening the time to new discoveries and revealing knowledge about natural phenomena by providing software systems for different scientific tasks and for many domains. Due to its *interdisciplinary nature*, eScience exhibits a high degree of *complexity*. Apart from the complexity introduced by the large amounts of data to be processed and analyzed, and the computational intensity and distributed characteristics of the IT environment, the field as a whole faces another challenge - the lack of interoperability among existing software. The existing literature reports about initiatives towards identifying software engineering principles for building scientific applications from scratch or from existing applications. In this relation, one major research issue is the integration of existing eScience software and tools for enabling the modeling of more complex scientific experiments and their execution[1].

In our research work in the scope of SimTech (Excellence Cluster Simulation Technology), which deals with fundamental research in eScience and in particular in the field of simulation technology, we have designed and developed a generic and integrated Scientific Workflow Management System – the SimTech SWfMS[2] – allowing for user-friendly modeling of simulation workflows, their execution, monitoring and adaptation so that it supports scientists in their everyday work. We make use of the concept of services and Web services as one technology and thus facilitate the integration of existing simulation software. For the composition and coordination of software we apply the widely accepted workflow technology known from business applications; we will refer to this technology as conventional workflow technology [9], [11], [23]. The system supports the typical trial-and-error fashion of creating and running experiments, which also led to significant improvements in the state of the art in workflow adaptation [10], [11], [23].

Our approach towards scientific workflows and the SimTech SWfMS have been based on the currently existing assumptions of Service-oriented Computing (SOC), namely that a service (i.e. simulation software in our context) is always up and running and available for users. From the point of view of the service provider the services are deployed and provided for use at all times. While in business applications this is in most cases a valid assumption (mainly because of the huge amount of transactions and interactions between software systems), in the field of scientific workflows this is not always the case. Unlike business workflows, simulation workflows are executed less frequently, need a lot of resources and are long-running. As a consequence the needed resources are utilized only for the time of simulation runs, leading to unevenly distributed load. This is also valid for the execution environment needed to run the workflows themselves – during the time the execution environment is not needed it is a waste of resources and additional cost to keep the system running. In case a service (i.e. simulation software) is not available at the moment it is needed by a simulation workflow, the execution of the simulation cannot be completed.

[1] Software Practice Workshop 2012: http://software.ac.uk/maintainable-software-practice-workshop
[2] http://www.iaas.uni-stuttgart.de/forschung/projects/simtech/

Based on these observations, we identify the following *requirements* that still need to be addressed in order to provide a meaningful support for simulation workflows: (a) dynamic allocation of compute and storage resources when they are needed; (b) dynamic release of resources when they are no longer needed; (c) on-demand provision and de-provision the workflow middleware and infrastructure and (d) dynamic deployment and undeployment of simulation services including their software stack.

In order to fulfill these requirements, in this work we *contribute*: (1) A *new service binding strategy* that comprises mechanisms for service software stack provisioning and dynamic service deployment, presented in section II; (2) An *approach* for dynamic provisioning and deprovisioning of workflow execution middleware and services on on-premise or off-premise (Cloud) infrastructures, described in section III; (3) An *architecture* for a SWfMS capable of on-demand provisioning and de-provisioning of simulation execution middleware and simulation services during workflow execution, shown in section IV; and as a part of the overall architecture (4) a system called "*Bootware*" as a piece of basic middleware that is capable of kick-starting the acquisition of infrastructure and provisioning the middleware stack and services for the execution of simulation workflows.

We also discuss challenges and potential improvements (section V) and present our current work towards realization of the presented contributions (section VI). We give an overview of related work and present our conclusions in sections VII and VIII respectively.

II. Dynamic Service Deployment and Provisioning

In SOC the main entities performing a computation or a function are services available via stable interfaces, capable of communicating over a network typically in an asynchronous fashion, and always up and running. Services provide a model for simplified use of existing applications and facilitate application integration in general, however, the complexity of dealing with the integration problem is shifted to the service middleware needed to enable service-oriented interactions. The service middleware, a.k.a. enterprise service bus (ESB) or service bus [24], traditionally enables several strategies for binding to a service. For enabling the strategy *static binding* of services the ESB requires the endpoint (EP) of a service as input and consequently performs an invocation of (or a call to) the identified service. In this case the application using the service provides the concrete endpoint, i.e. service (instance), it wants to use (see Figure 1, A). *Dynamic binding* is the strategy (Figure 1, B) allowing the service user to specify in abstract manner only the needed function without identifying a fixed service EP. In this case the ESB carries out a discovery step in a service registry (not depicted in Figure 1) and selects one service EP of all available (compliant) ones. Then the ESB invokes this service and returns the result to the user. A variation of the dynamic binding strategy is the so-called *dynamic binding with service deployment* (Figure 1, C). For the user the strategy is still dynamic binding, however for the provider the strategy implies also a step of first deploying the service instance on its systems and registering the service EP in the service registry. The rest of the steps are discovery, selection and invocation of the service. In order to address one of the requirements discussed before, i.e. to enable dynamic deployment and undeployment of simulation services, including their software stack and underlying infrastructure, we introduce a new binding strategy (Figure 1, D), which we call *dynamic binding with software stack provisioning*. For the user this strategy manifests again as dynamic binding, however the

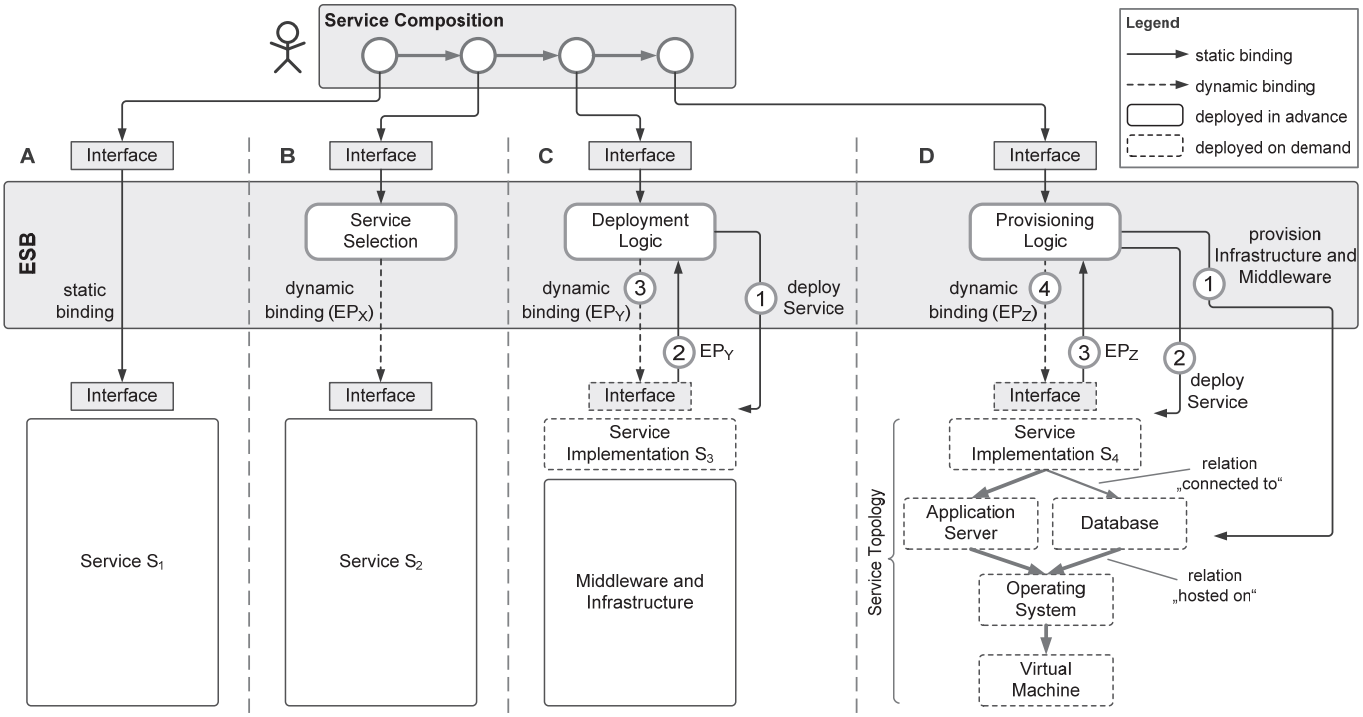

Figure 1 Classification of Service Binding Strategies

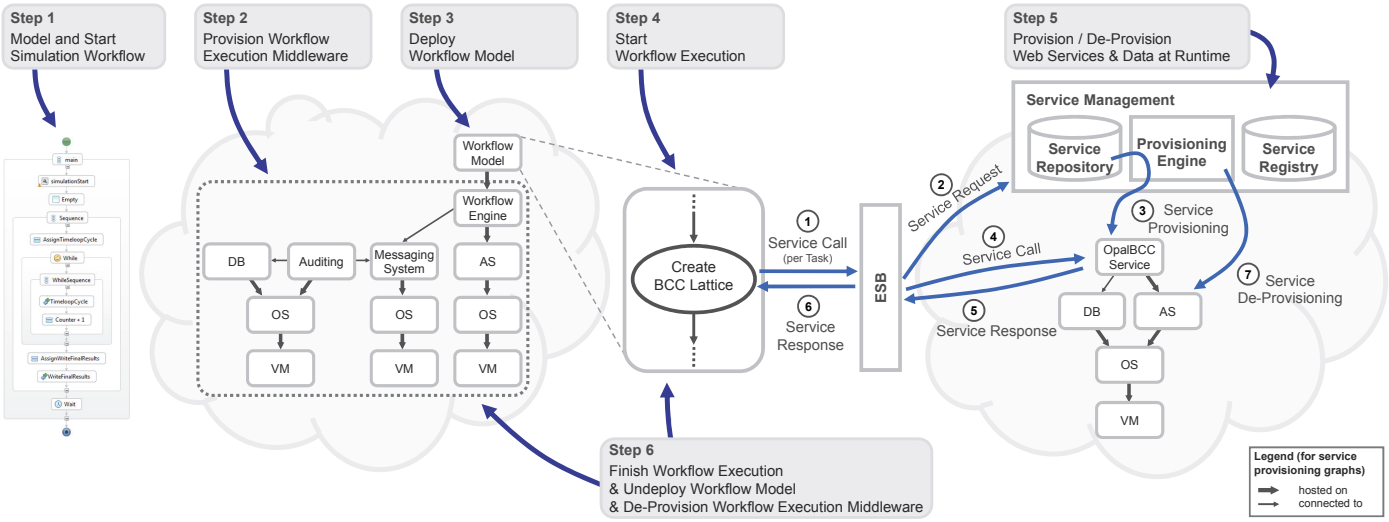

Figure 2 On-demand provisioning and de-provisioning of workflow execution middleware and simulation services

ESB has to perform the following steps. Upon a call from a user, based on the abstract description of the required functionality (typically a description of only interface and operation (in WSDL terms portType and operation)), the ESB starts a (Figure 1, D-1) procedure for provisioning of the infrastructure and the software stack (middleware) needed to execute the application that implements the service. For this it needs the so-called *Service Package* that contains: (i) the service topology, i.e. the horizontal (application components) and vertical decomposition (software stack) of the application, their relationships, as well as the implementation of the components (executables, services, scripts, images, etc.) or references to them; (b) the provisioning logic – the sequence of steps that have to be carried out in order to provision the application and its infrastructure. Examples of existing approaches taking into account the horizontal and vertical decomposition of applications like applications with points of variability and the corresponding deployment processes [13], blueprints [14] and TOSCA topologies and build and management plans [2], [7], stem from the field of Cloud Computing. After the infrastructure and middleware have been provisioned, the service can be deployed (D-2) and its EP is registered in the service registry. Then the ESB uses the EP to invoke the service and returns the result back to the user (D-3 and D-4).

III. On-demand Provisioning of Workflow Execution Middleware and Services

Our approach for on-demand provisioning and de-provisioning of workflow execution middleware and simulation services consists of six basic steps. In Figure 2, we present the approach using an example scenario. The example simulation workflow, i.e. the application, is a Kinetic Monte-Carlo (KMC) simulation modeled as a service composition, which uses several services for simulation of solid bodies implemented by the OPAL application [11]. In the original scenario, this simulation workflow is deployed on a service composition engine and can be run multiple times by executing multiple workflow instances. As an example execution environment, i.e. the software stack, consider the SimTech SWfMS ([9], [10]). The topology of the system consists of a workflow engine, an auditing system, a database, and a messaging system (also shown in Figure 2). The workflow engine is responsible for the execution of workflows. The auditing system stores execution data for analysis and provenance in a database (DB). The messaging system enables the communication between the workflow engine, the auditing system, the monitoring tool and also with other components. The topology shows also the underlying application server (AS), operating systems (OS) and virtual machines (VM), as well as the relations between the components. We can distinguish between two types of relationships. The auditing system is "connected to" the database. It reads and writes data from and to the database. The messaging system is "hosted on" an operating system.

In the *first step* of the approach, the scientist models a simulation workflow using a modeling tool (e.g. available locally on his system) or reuses an existing one. Then the scientist can start the execution of the simulation workflow instances (for example by pressing a start button inside the modeling tool if this functionality is available).

In the *second step*, the middleware, i.e. the SWfMS, and its underlying infrastructure are provisioned to an on-premise or an off-premise Cloud environment, depending on the requirements of the scientist and availability of on-premise resources. For the provisioning of the middleware and the needed infrastructure the topology of the SimTech SWfMS software stack has to be known.

In the *third step*, the workflow model is deployed on the workflow engine. This means that the workflow model is made available for use and scientists can start as many simulation workflow instances as needed for their scientific experiments. The *fourth step* is the workflow (instance) execution. These steps are already supported by existing workflow systems.

The *fifth step* of our approach is the on-demand provisioning and de-provisioning of the simulation services during the runtime of the simulation workflow. As presented

earlier, the ESB is responsible for performing a service call to a suitable service on behalf of each task/activity of a simulation workflow instance (see also Figure 1, B). The service processes the request and returns the result to the ESB which in turn passes it back to the workflow [12].

The procedure for provisioning the service is as follows (Figure 2, Step 5): (1) The workflow engine of the SWfMS calls the ESB to perform a service invocation. Since the service call cannot be routed to a matching service, (2) the ESB sends a service request to the service management component, comprising a Service Repository, a Service Registry, and a Provisioning Engine. The Service Repository stores all artifacts needed to provision a service including the topology of its underlying middleware and infrastructure. This set of artifacts forms the Service Package. The Service Registry manages information about all available services. The Provisioning Engine provides the functionality to provision and de-provision a service including its underlying middleware and infrastructure. Upon the call from the ESB to the Service Management Component, (3) the Provisioning Engine retrieves all needed artifacts from the Service Repository and provisions the requested service onto the Cloud environment, thus making it available for use. Then (4) the ESB forwards the service call to the newly provisioned service and (5) later receives the result. In the next step (6) the ESB forwards the result to the invoking workflow activity. Finally, (7) the Provisioning Engine de-provisions the service if it is no longer needed.

When the execution of all workflow instances is finished the workflow model and the workflow execution middleware can be undeployed in the *sixth step* of the approach.

In the description of the second step of our approach we did not give details about how the on-demand provisioning of the workflow execution middleware is realized. We consider the workflow execution middleware also as a service providing the functionality of executing workflows. The Service Package for the workflow execution middleware, capturing the above mentioned topology, is stored in the Service Repository and the workflow execution middleware is provisioned by the Provisioning Engine using the sub-steps of step 5 of our approach. Clearly, the on-demand provisioning of an application and the required software stack for its execution follow the same principle, even though it is applied in two completely different steps in the presented approach.

IV. ARCHITECTURE

In this section we will introduce the architecture of the system supporting on-demand provisioning and de-provisioning of workflow execution middleware and services needed for the execution of (simulation) workflows. We present the architecture in Figure 3. We distinguish between components run locally on the user's machine and the components run on a Cloud. The life cycle phases we describe next are the modeling of simulation workflows, the middleware runtime/execution phase and the service runtime phase.

A. Modeling Phase

The architecture components used during the modeling phase are the Modeling and Monitoring Tool (MT), and the Bootware running locally on the user's machine, and the Service Repository (SRP), the Service Registry (SR) and the User Registry running in a Cloud environment. These components are active during all life cycle phases. The scientist develops his workflow model using the "*Modeling and Monitoring Tool*". During the workflow execution he can monitor the running workflow instances. In contrast to the business workflow domain in the eScience workflow domain there is typically only one role responsible for the modeling, deployment and execution of a workflow. Therefore, just like in many eScience workflow systems, we integrated the modeling, deployment and monitoring functionalities in one tool [23].The Service Registry and the Service Repository provide information about all services that can be used by the workflows.

The *Bootware* is the basic piece of software needed to provision the workflow execution middleware (in a Cloud environment). Instead of provisioning the whole workflow execution middleware in one step, we follow a *two-step* bootstrapping process. In the first step (Figure 3, step 1), upon a call from the Modeling and Monitoring Tool, the Bootware provisions the Provisioning Engine and its underlying middleware and infrastructure on a Cloud environment. This reduces the complexity of the Bootware component by limiting its capabilities to the provisioning of one special component - the Provisioning Engine (PE). The Provisioning Engine itself is a generic component able to provision any kind of service and is a complex system [17]. In the second step (Figure 3, step 2), the Bootware calls the Provisioning Engine and passes a reference to the service package to the workflow execution middleware. The Provisioning Engine uses this reference to get all needed artifacts from the Service Repository and provisions the remaining part of the workflow execution middleware including its underlying infrastructure in a Cloud environment. The Instance Management Component manages the running instances of the workflow execution middleware and makes sure that when the scientist starts the execution of a workflow, his already provisioned workflow middleware is used.

The *Service Repository* contains service packages. The *Service Registry* is a central data store containing information about all available services and enabling their discovery. This includes functional and nonfunctional properties of a service and a reference to the corresponding service package in the Service Repository. The information in the Service Registry is not only about services stored in the Service Repository but also services that are already available (and provided by a third party). Existing Service Registries are, for example, UDDI registries [15] in the traditional SOA domain or myExperiment (http://www.myexperiment.org/) in the eScience domain [16]; however they do not meet all of the above mentioned requirements and require extensions.

We distinguish between two kinds of services. We denote services provided by a service provider, who also manages the service, *provisioned services.* A scientist can use such services, but he has no knowledge about the implementation and the underlying middleware and infrastructure. Provisioned Services follow the always on semantic, they are running and ready to use. As *not provisioned services* we denote services whose artifacts needed for its provisioning are accessible to the

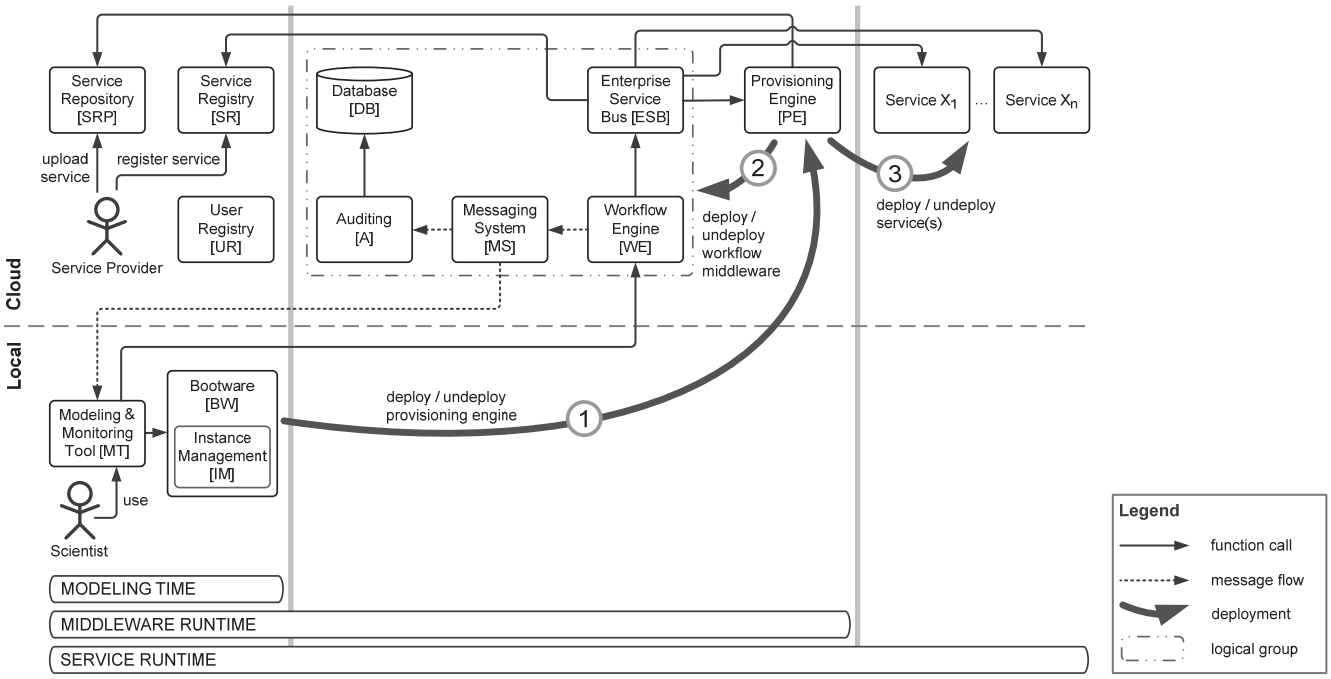

Figure 3 Architecture

scientist. Not provisioned services have to be explicitly provisioned before they can be used. The strategies for static and dynamic service binding (Figure 1, A and B) rely on provisioned services. The service binding approach introduced in this paper is based on not provisioned services (Figure 1 D).

A not provisioned service can be *dedicated* or *shared*. A dedicated service can or may only be used by one user at the same time. Other users can use another instance of the service provisioned with its underlying middleware and infrastructure (more details are presented in section IV.E). An example for such a dedicated service can be a simulation service needing a lot of compute resources without having any elasticity capabilities. If we would share this simulation service, the compute resources are also shared by several computations which increase the response time of each service call. A shared service can in contrast be used by several users at the same time. Although services in general are expected to be shared, in fact there are quite a lot of dedicated services.

The Service Registry stores specific information for each type of service. Independent of the service type a link to the interface description is available. For provisioned services the endpoint is already known and therefore stored, together with the current user. For not provisioned services the Service Registry contains a link to the corresponding service package in the Service Repository and if the service is dedicated or shared. In addition, the number of currently running instances is also stored.

B. *Middleware RuntimePhase*

The workflow middleware runtime phase is supported by the components of the simulation workflow execution middleware provisioned at the end of the modeling phase. In our example these are the SimTech SWfMS, the ESB and the Provisioning Engine. These components interact with the components used already in the modeling phase. The ESB receives service calls from the Workflow Engine to invoke services on behalf of workflow activities (for more details on the interaction of the ESB with the other components see section IV.E). If the service is a not provisioned one, the Provisioning Engine gets all needed artifacts like the implementation of the service via the Service Repository and uses these artifacts to provision the service including its underlying middleware and infrastructure (Step 3 in Figure 3). After the service provisioning is done, the ESB forwards the service call to the newly provisioned service.

C. *Service RuntimePhase*

During the *Service runtime phase* the services are executing the functionality they are implementing. At the beginning of this phase all components of our architecture shown in Figure 3 are provisioned and running. As soon as a service has finished its computation the result is returned to the ESB, which in turn sends it back to the workflow engine.

D. *Deprovisioning of Services and Middleware*

For dedicated services the ESB then calls the Provisioning Engine to de-provision the service. For shared services the ESB first checks if the service is still processing other requests. Only if the service is idle it will be de-provisioned.

After the workflow engine has finished the execution of all running workflows, the Bootware initializes the de-provisioning of the workflow execution middleware. In the first step the Provisioning Engine de-provisions all other middleware components. In the next step the Bootware de-provisions the Provisioning Engine.

E. ESB Functionality

The *ESB* manages and coordinates the on-demand provisioning and de-provisioning of services needed during the execution of a workflow. The actual provisioning and de-provisioning functionality is implemented by the Provisioning Engine. The ESB uses the Service Registry and the Provisioning Engine to ensure that the right service is provisioned at the right time. The logic implemented by the ESB is shown in Figure 4.

If the service call addresses a *provisioned* service, shown in the leftmost path in Figure 4, the service is already available and the endpoint has been fetched from the SR. The ESB forwards the service call to the appropriate service endpoint. After the service has finished processing the call, the ESB receives the result and passes it back to the workflow engine. This case is indeed the standard functionality of an ESB [24] (see section II).

Our approach focuses on and contributes the handling of *not provisioned services*. In this case, the ESB distinguishes between dedicated and shared services. For a *dedicated service*, shown in the middle path in Figure 4, the ESB calls the PE to provision a new instance of the service. With the call to the PE the ESB passes a reference to the SRP. The PE uses this reference to get the service package for the provisioning of this service. In the next step the PE provisions the service including its underlying middleware and infrastructure and passes the endpoint of the service to the ESB. Then the ESB registers the newly provisioned service at the SR and forwards the service call to the service. After processing the service call, the service returns the result to the ESB, which forwards it back to the workflow engine. Then the ESB deletes the service instance from the Service registry and calls the PE to de-provision the service and its underlying middleware and infrastructure.

If the service call addresses a *shared service* (rightmost path, Figure 4), the ESB checks in the SR if there is already an available instance of this service belonging to the same user as the current service call. If no service instance is available, the ESB calls the PE to provision a new instance of the required service. After the provisioning of the service and its underlying middleware and infrastructure is completed, the ESB proceeds as in the previously described case. The ESB and SR manage the number of currently processed service calls and initiate de-provisioning if the service is not used anymore.

V. DISCUSSION

In this section we discuss the challenges our approach faces and points out potential research questions for future work. The challenges we discuss are related to the engineering of components of the overall architecture and/or to the new approaches and mechanisms for dealing with service provisioning.

The features of the *Bootware* are restricted to the provisioning and de-provisioning of the Provisioning Engine, however it should not be restricted to a one Cloud provider - it should support provisioning on multiple Cloud environments and of different provisioning engines. We plan to address this

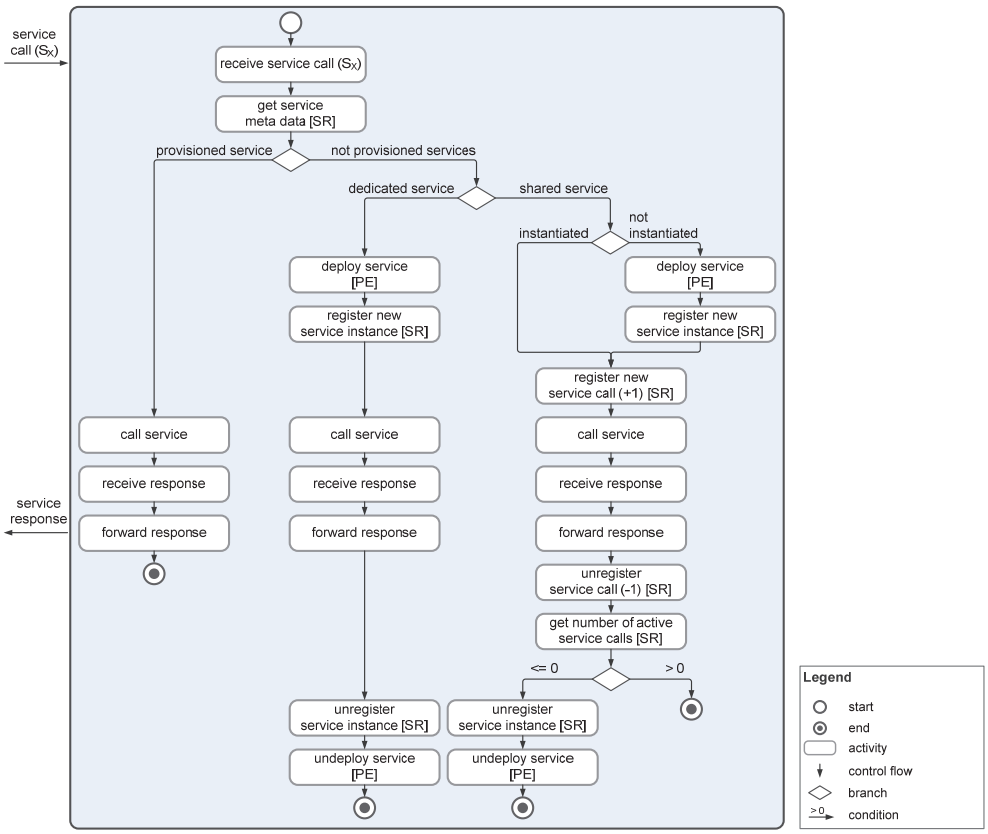

Figure 4 ESB Control Flow

challenge by using software engineering approaches.

As described in section IV the *Service Repository* is used to store and retrieve Service Packages (section II) in potentially different formats. As the format of the service package itself is a subject of current research and different structures and formats proposed [7], [14], we plan to design a generic Service Repository in future.

The procedure for the on-demand provisioning and de-provisioning of a service (section IV, Figure 4) postulates that as soon as the service request has been processed the service is de-provisioned. While this strategy optimizes the resource consumption, we are aware that the procedure should be adapted towards optimization for response time, cost, and other criteria.

Similarly, knowledge about the execution of a workflow can be used for optimization, e.g. for provisioning of services in advance and thus reduce the execution time of workflow activities. Typically, analysis of historical data from already executed workflows is used, however, novel approaches are needed, due to the uncertainty involved, especially related to the differences in available resources used for the execution of services for previous workflow instances.

VI. Realizations

In the following we will present several aspects of our ongoing work on the realization of the proposed approach and architecture.

Our architecture is designed to be independent of any particular SWfMS. We assume that the SWfMS can be described so that it is possible to automatically provision it to a Cloud environment with a corresponding provisioning engine. In the previous chapter we used a simplified version of the SimTech SWfMS as an example workflow management system and we use this system in the realization of the architecture. The SimTech SWfMS has been developed originally to run on a traditional on-premise infrastructure. The migration of an existing software system to a Cloud Environment is in general a complex task [5]. As a first step, we have shown in [26] that the components of the system can be migrated to a Cloud infrastructure.

We have developed a prototype [29] that implements the control flow logic for the on-demand provisioning and de-provisioning of services shown in Figure 4. The implementation is an extension of the ESB^{MT} system [27], [28] – a multi-tenant service bus we integrated into the SimTech SWfMS. For the creation of service packages we use TOSCA [2]. TOSCA is a standard designed to enable portability and interoperability of Cloud applications [7]. We use the OpenTOSCA[3] container as Provisioning Engine - an open source provisioning engine implementation for TOSCA [30]. The Service Registry and Repository are identified as future work in our research agenda, as well as the Bootware component.

The on-demand provisioning and de-provisioning of workflow execution middleware and simulation services does not only affect applications but also corresponding data. For example when de-provisioning the auditing component of the SimTech SWfMS (see Figure 3), the recorded auditing data may not be lost. A comprehensive discussion about data migration to and from Cloud environments is presented in [6]. Currently we deal with identifying all migration relevant data for the SimTech SWfMS and matching them to the migration scenarios and Cloud data patterns proposed in [6] and have performed first successful application of some of these migration scenarios on the SimTech SWfMS [25].

VII. Related Work

The on-demand deployment of services has extensively been discussed in the domain of grid computing [18], [19], [20]. In contrast to Cloud environments the infrastructure resources, the nodes of the grid, can be selected but not created on demand. As the resources are given, the deployment of services has to take into account the capacity of available resources and match them with the requirements of the services to deploy. The on-demand deployment of services in the domain of grid computing typically represents dynamic deployment of service (Figure 1, C). There is no notion of complex service topologies and no support for the provisioning of resources.

In the FlexiNET project an architecture for dynamic service deployment in Web service based grid environments has been developed [21] by extending the central service discovery component of the Globus Toolkit[4], the so called Monitoring and Discovery System (MDS). For a service request the MDS searches for a matching running service. If no such service is available, the MDS accesses a code repository to get an implementation of the requested service. The MDS also has knowledge about available compute nodes and their resources. After a matchmaking algorithm determined a suitable compute node, the code is installed on this node. After that, the requested service is available and can process the request. In contrast to our approach, MDS relies on the availability of compute nodes capable of hosting and running service implementations. The MDS is not responsible for the provisioning of the corresponding infrastructure and middleware; this is assumed to be available.

In [22] the authors combine workflow execution and on-demand provisioning of services. A load balancer component is placed between a workflow engine and the invoked services. When there is no suitable service available, the load balancer calls an internal provisioning component to provision a new service. The provisioning of services is focused on starting virtual machines. There is no generic architecture to support the provisioning of complex service topologies. The approach does also not consider the on-demand provisioning of the workflow execution middleware.

The concept of horizontal and vertical service composition has been proposed in our previous work [13]. The motivating scenario is the on-demand provisioning of services in order to reduce the resource consumption for the service provider. The vertical composition of a service is determined dynamically at

[3] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

[4] http://www.globus.org/toolkit/

provisioning time with respect to the required quality of service and based on an application model containing variability points. The central part of the proposed enabling architecture is the vertical composition enabled bus. The approach presented here does not consider the dynamic composition of service topologies with respect to quality of service, however can use it in future.

VIII. Summary and Outlook

Based on our previous work on extending conventional workflow technology to support simulation workflows, in this paper we presented an approach for on-demand provisioning and deprovisioning of workflow execution middleware and simulation services. In addition, we contribute architecture of the system supporting this approach, an extended classification of the binding strategies for ESBs which is necessary for realizing the approach. We have also outlined the challenges in terms of concepts, techniques and implementation and provided information about our existing prototype.

The presented approach allows for the development of a distributed system for simulation workflows that comprises two major parts: local for the scientist's computing device and a remote part, which can be provisioned anywhere - on an on-premise or off-premise infrastructure. The novelty and advantage of this system is in the fact that scientist will need to install on their systems only the simulation workflow modeling tool incorporating the Bootware component, which is capable of provisioning and de-provisioning the resources and software necessary for the execution of the simulation workflows on demand.

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 (EXC 310/1) at the University of Stuttgart.

IX. References[5]

[1] T. Hey, S. Tansley, Kristin Tolle (Eds.): *The Fourth Paradigm. Data-Intensive Scientific Discovery*. Microsoft Research. 2009.

[2] *Topology and Orchestration Specification for Cloud Applications* Version 1.0. 18 March 2013. OASIS Committee Specification 01. http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html

[3] *The NIST Definition of Cloud Computing*, NIST Special Publication 800-145, 2011, http://www.nist.gov/itl/cloud/

[4] I. J. Taylor, et al. (Eds.). *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag, New York. 2006.

[5] Andrikopoulos, V.; Binz, T.; Leymann, F.; Strauch, S.: *How to adapt applications for the Cloud environment*. In: Computing, Springer, 2012.

[6] Strauch, S. et al.: *Migrating Application Data to the Cloud Using Cloud Data Patterns*. In: Proceedings of CLOSER'13, 2013.

[7] Binz, T. et al.: *Portable Cloud Services Using TOSCA*. In Internet Computing, IEEE , vol.16, no.3, pp.80,85, May-June 2012.

[8] Deelman, E. et al.: *The cost of doing science on the cloud: the Montage example*. In: Proceedings of SC '08.

[9] Görlach, K. et al.: *Conventional Workflow Technology for Scientific Simulation*. In: Guide to e-Science, Springer-Verlag, 2011.

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

[11] Sonntag, M. et al.: *Using Services and Service Compositions to Enable the Distributed Execution of Legacy Simulation Applications*. In: Proceedings of ServiceWave 2011.

[12] Karastoyanova, D. et al.: *A Reference Architecture for Semantic Business Process Management Systems*. In Proceedings of WI 2008.

[13] Retter, R.; Fehling, C.; Karastoyanova, D.; Leymann, F.; Schleicher, D.: *Combining Horizontal and Vertical Composition of Services*. In: Service Oriented Computing and Applications, Springer, 2012.

[14] Garcia-Gomez, S. et al.: *4CaaSt: Comprehensive Management of Cloud Services through a PaaS*. In: Proceedings of ISPA, 2012.

[15] *UDDI Version 3.0.2*, UDDI Spec Technical Committee Draft, Dated 20041019, http://uddi.org/pubs/uddi_v3.htm

[16] De Roure, D.; Goble, C.; Stevens, R.: *The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows*. In Future Generation Computer Systems 25, 2009.

[17] Lipton, P.: *Escaping Vendor Lock-in with TOSCA, an Emerging Cloud Standard for Portability*. In: CA Technology Exchange 4, 1, 2013.

[18] Kertesz, A. et al.: *An SLA-based resource virtualization approach for on-demand service provision*. In: Proceedings of VTDC '09.

[19] Kecskemeti, et al.: *Automatic Service Deployment Using Virtualisation*. In: Proceedings of PDP, 2008.

[20] Byun, E. et al.: *A dynamic grid services deployment mechanism for on-demand resource provisioning*. In Procedings of CCGrid, 2005.

[21] Chrysoulas, C. et al.: *Applying a Web-Service-Based Model to Dynamic Service-Deployment*. In Proceedings of the Int'l Conf. on Computational Intelligence for Modelling, Control and Automation, 2005.

[22] Dornemann, T.; Juhnke, E.; Freisleben, B.: *On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud*. In: Proceedings of CCGRID '09.

[23] Sonntag, M.; Karastoyanova, D.: *Next Generation Interactive Scientific Experimenting Based On The Workflow Technology*. In Proceedings of MS 2010.

[24] Chappell, D.: *Enterprise Service Bus: Theory in Practice*. 2004.

[25] Passow, S.: *Evaluation of a Methodology for Migration of the Database Layer to the Cloud based on a Research Case Study*. Students Thesis 2409, IAAS, University of Stuttgart, 2013.

[26] Reinfurt, L.: *Migrating the SimTech Simulation Workflow Management System to a Cloud Infrastructure*. Students Thesis 2414, IAAS, University of Stuttgart, 2013.

[27] Strauch, S. et al.: ESB^MT: Enabling Multi-Tenancy in Enterprise Service Buses. In: Proceedings of IEEE CloudCom'12, pp. 456-463, 2012.

[28] Strauch, S. et al.: Enabling Tenant-Aware Administration and Management for JBI Environments. In: Proceedings of IEEE SOCA'12, pp. 206-213, 2012.

[29] Schneider, V.: *Dynamic Provisioning of Web Services for Simulation Workflows*. Diploma Thesis, IAAS, University of Stuttgart, 2013.

[30] Binz, T.; Breitenbücher, U.; Haupt, F.; Kopp, O.; Leymann, F.; Nowak, A.; Wagner, S.: *OpenTOSCA - A Runtime for TOSCA-based Cloud Applications*. In: Proceedings of ICSOC'13.

[5] All links were last followed on 23.10.2013.