Template-Based Automated Service Provisioning – Supporting the Agreement-Driven Service Life-Cycle

Heiko Ludwig[2], Henner Gimpel[1], Asit Dan[2], and Bob Kearney[2]

[1] Universität Fridericiana zu Karlsruhe (TH), Englerstrasse 14,
76131 Karlsruhe, Germany
gimpel@iw.uni-karlsruhe.de
[2] IBM T.J. Watson Research Center, 19, Skyline Drive,
Hawthorne, NY, 10025, USA
{hludwig, asit, firefly}@us.ibm.com

Abstract. Service Level Agreements (SLAs) are a vital instrument in service-oriented architectures to reserve service capacity at a defined service quality level. Provisioning systems enable service managers to automatically configure resources such as servers, storage, and routers based on a configuration specification. Hence, agreement provisioning is a vital step in managing the life-cycle of agreement-driven services. Deriving detailed resource quantities from arbitrary SLA specifications is a difficult task and requires detailed models of algorithmic behavior of service implementations and capacity of a – potentially heterogeneous – resource environment, which are typically not available today. However, if we look at, e.g., data centers today, system administrators often know the quality-of-service properties of known system configurations and modifications thereof and can write the corresponding provisioning specifications. This paper proposes an approach that leverages the knowledge of existing data center configurations, defines templates of provisioning specifications, and rules on how to fill these templates based on a SLA specification. The approach is agnostic to the specific SLA language and provisioning specification format used, if based on XML.

1 Introduction

Agreements, particularly Service Level Agreements (SLAs), play an important role in the binding process of service-oriented architectures. They are used for the reservation of service capacity at defined service levels for a specific customer. Agreements enable a service provider to learn about future demand in advance – as stated in the agreements – and provision the required resources for the agreed service capacity.

The use of agreements to reserve and bind to services is relevant for various types of services. Agreements are used for reserving capacity of software-as-a-service, e.g., Customer Relationship Management services, for scheduling Grid jobs, and also for resource-level services within a complex system such as storage capacity, network bandwidth, computing nodes, and memory. Furthermore, the mechanism of binding to services by agreement is applied within an organization and across organizational boundaries – with changing security requirements, though.

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 283 – 295, 2005.

Traditionally, SLAs have been used primarily between organizations in a, mostly, paper-based process. SLA creation was then followed by a phase of service provisioning that could take once more a significant amount of time, depending on the degree of automation of the design of the resource infrastructure that provisions the service and of the provisioning process.

Recently, however, a number of efforts were undertaken to streamline the creation and monitoring of SLAs for service-oriented architectures by representing SLA contents in a machine-readable format and using electronic (Web services) interactions to negotiate and sign them. WSOL [15] and WSLA [12] are research approaches proposing agreement representations. SNAP is a proposal for an agreement negotiation protocol [4]. WS-Agreement [1] is a specification of the Global Grid Forum that standardizes a top-level agreement structure, a simple negotiation protocol and a compliance monitoring interface. Standardized representations of agreements and negotiation processes enable dynamic service acquisition processes for capacity-aware service clients. However, to be effective in practice, they also demand automation of the provisioning process of service providers, which is the subject of this paper.

Provisioning is the act of deploying, installing and configuring a service [9]. It is an important aspect of management of data centers and networks. Provisioning typically involves the following steps:

1. Identifying the target system state that delivers the service as intended; this involves the derivation of the system's topology, the configuration of firewalls, application servers, database servers and the like, as well as the quantification of those resources, i.e. how many servers of each type.
2. Deriving a process that transitions the system from its current state to the target state, often referred to as change management [10];
3. Executing the process consistently, usually driven by a workflow or script system accessing instrumentation on those resources.

The term provisioning is applied to both, low-level provisioning of servers or other raw resources, i.e. installing and configuring operating systems, as well as to high-level application provisioning, installing, updating and configuring applications on resources that underwent low-level provisioning earlier. Given the complexity of the steps of the provisioning process listed above, particularly steps 1 and 2, a generic solution for automating the end-to-end service provisioning process is a daunting task. While some approaches address partial aspects such as deriving a service topology [6] and deriving and optimizing a provisioning workflow [9], no generic, derivative provisioning solution is available as of now.

The approach presented here tackles exactly this problem, i.e. the automation of end-to-end service-provisioning based on agreement terms. More precisely, we propose a template mechanism that automatically handles service requests: it derives resource types and quantities necessary to guarantee quality requirements, it determines the resource configuration and assembly, and acquires resources from heterogeneous resource managers.

To make provisioning work in practice, service providers capture the experience of their system administrators in provisioning process templates and rules of thumb for capacity planning, an approach that is often pragmatic. The approach proposed in this paper leverages this pragmatic approach for agreement-driven provisioning. It

provides a formal representation for templates of an executable provisioning process and an executable way of defining how to fill in these templates based on content of a formal agreement, which we call the *Agreement Implementation Plan Template* (*IP Template*). These templates are predefined parametric examples of how to provision services of a given basic structure. Thus, the presented mechanism does not go against the administrators' rules of thumb or best practices on a general basis, but embraces them in providing an automatically executable process, and thus more appropriate if time is a crucial factor. The approach enables dynamic acquisition of service capacity in a service-oriented environment in a pragmatic way.

To this end, the remainder of the paper is structured as follows: Section 2 establishes relevant terminology by defining a model of provisioning in the service life-cycle. Section 3 outlines the problem definition in more details, presents a sample scenario where the proposed mechanism is applicable, and points out the major requirements. Section 4 presents the proposed template structure along with the processing of agreement offers to derive resource requirements and trigger provisioning. Finally, Section 5 reviews related work and concludes.

2 Provisioning in the Agreement-Driven Service Life-Cycle

The life-cycle of any electronic service goes through at least four broad stages:

1. a high level service modeling stage for mapping a business problem to a service description,
2. a design and implementation stage where the service technology, usually the application code is developed in support of the above service description,
3. a resource mapping and deployment stage, where a specific set of resources, its topology and quantity, is defined on which the service is deployed, and
4. a runtime monitoring and management stage for managing resources delivering the service.

Supporting performance-related quality of service as part of an agreement requires additional activities throughout these four life-cycle stages. At the modeling stage, in addition to the service definition, associated qualities of service are defined. The service implementation must be implemented in a way that it can be deployed on a variable set of resources. The resource mapping and deployment stage, the service provisioning as defined here, must be able to derive the set of resources required to achieve a given performance level and at runtime, the instrumentation of the service must allow us to assess QoS compliance and adjust resource allocations.

2.1 Provisioning in an Agreement-Driven Service-Oriented Architecture

This life-cycle is managed in the context of an *agreement-driven service-oriented architecture* (*ADSOA*). In this ADSOA, the agreement-related interaction between service provider and customer precedes, and is orthogonal to, the service interaction taking place between service and service client [1]. In an implementation architecture, the *Service Delivery* layer, which exists in any service-oriented architecture, is driven by the *Agreement Management* layer through Agreement Delivery Management as outlined in Figure 1.

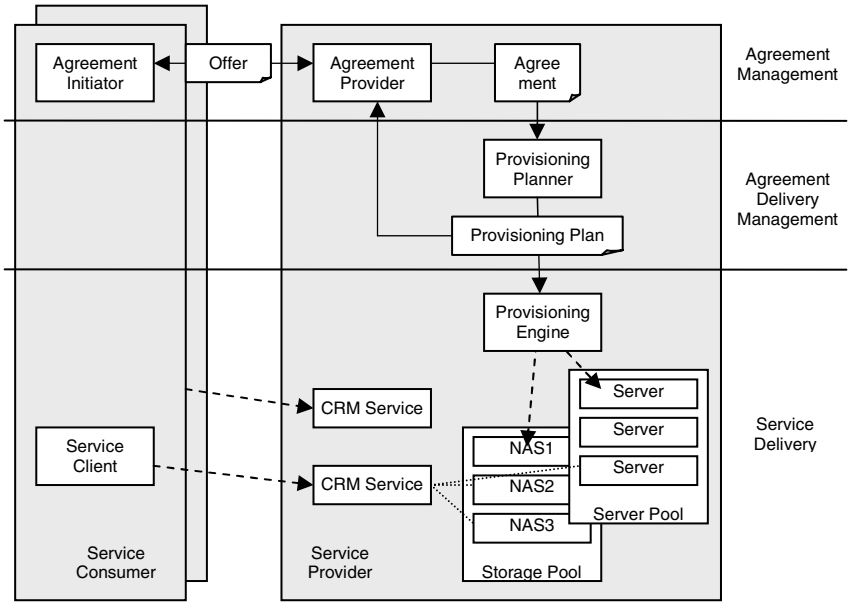

Fig. 1. Provisioning in an Agreement-Driven Service Architecture

The *Service Delivery* layer addresses the implementation of a service on a set of resources. In Figure 1, a CRM service is implemented by resources from two pools, storage and servers (dotted lines). It can be accessed by one or more clients. Resources are configured for a particular service using a provisioning engine, which executes provisioning plans, acquiring resources from resource pools and executing provisioning workflows. Resources can be shared between services or be exclusive. The Service Delivery layer per se is independent of agreements and is present in any non-trivial SOA implementation architecture.

The *Agreement Management* layer manages the portfolio of agreements and enters new agreements by exchanging offers. It also exposes the current state of the agreement, according to the WS-Agreement model.

The *Agreement Delivery Management* layer relates the Service Delivery and Agreement Management. Offers and Agreements are input to a Provisioning Planner, which creates a Provisioning Plan, comprising the set of resources required and the provisioning workflow. This Provisioning Plan can be used by the Agreement Management layer to assess an agreement offer or it can be passed on to the Provisioning Engine of the Service Delivery layer. Mapping the state of the Service Delivery layer to the state of compliance of an agreement is also part of this layer but not the focus of this paper [11].

2.2 Use of Templates

In an ADSOA, service capabilities can be published by an agreement provider as *agreement templates*, potentially in addition to and complementing other forms such as policy-annotated UDDI entries. WS-Agreement defines a template format that contains a partially completed agreement, a definition of named locations where an

agreement initiator can fill in agreement content, i.e. the "fields", and constraints that limit what can be filled in [1]. In the context of a CRM Web service, for example, a field could be the value for the response time of an operation and a constraint could limit the choice to one, two or five seconds. The use of agreement templates, particularly their constraint mechanism, enables service providers to advertise services only at performance levels whose resource implications they have experience with and understand. A service can be advertised using multiple agreement templates.

To capture the provisioning expertise of system administrators, these agreement templates can be associated with *agreement implementation plan templates*. An agreement implementation plan template contains a partially filled provisioning plan, corresponding to the agreement template approach, and a definition how to fill these fields depending on content of agreements. The details of this approach are described in Section 4. Agreement implementation plan templates can be changed independently of agreement templates but must be adapted if agreement templates change. Hence, the joint use of agreement templates and agreement implementation plan templates enables a service designer to anticipate the decision-making of the first three stages of the service life-cycle and automate the execution of provisioning planning for particular agreements in the life-cycle.

3 Problem Definition

The use of agreements in managing service interactions, and hence, agreement-based provisioning is required in all scenarios where service configurations need to be customized based on client requirements. As mentioned earlier, the use of customer-specific SLAs is equally applicable for configuring a business service between enterprises or for managing interactions across resource managers in a complex distributed environment, e.g., having a storage manager, workload manager, cluster manager, etc. The complexity of an agreement driven provisioning process in different scenarios depends on the service and the complexity of customization. For example, in an agreement-based job submission, where the agreement specifies a preference of resources over which the job is to be run, the Provisioning Planner simply invokes the scheduling system with the information on resource preferences. In this case, the agreement implementation plan template specifies the end-point of the scheduling system and how to extract resource preference information to be passed to the scheduling system. Similarly, incremental provisioning for setting up a shared application/web service with a client specific service level objective on average response time simply requires passing the service level objective information to the workload manager managing this service. Again, the associated agreement implementation plan template consists of the end-point of the workload manager and how to extract service level objective information.

3.1 Use Case

A more complex example of agreement-based provisioning may involve multiple steps of deriving information to be passed to one or more provisioning services and/or multiple methods to be invoked. Consider, setting up a CRM software-as-a-service hosted by an application service provider. Also, assume the agreement includes many

details such as how to upload client data, application isolation, firewall and other security requirements, performance and availability requirements, requirement on storage size and data back up, network connectivity requirements, details of metering and billing, etc. Clearly, provisioning such a service requires interaction with many components and deriving resource configuration parameters to be used.

3.2 Requirements

The previous discussion leads to a set of requirements to be addressed by an agreement-driven provisioning approach:

- The approach must not be specific for a single service or a class of services, like, for example, a CRM application service. It should generically enable SLAs for a *wide range of services*, from resources to business services.
- The approach must deal with a *variety of provisioning engines*, including schedulers.
- *Agnosticism to the specific SLA language* is desirable as there is no unique way to specify SLAs and this reduces re-implementation and adaptation.
- The approach must provide means for *capturing externalized know-how* of system administrators.
- The overall provisioning process should be *automatically executable*, the main motivation for automatic provisioning.
- The approach has to provide functionality for *deriving resource types and quantities* for a given SLA.
- Furthermore, there has to be a detailed plan of the necessary *resource configuration and assembly* for provisioning these resources.
- Allowing for the *acquisition of resources* from a resource pool is an integral step in provisioning.
- Finally, the mechanism has to be *adaptive to the resource load and availability*, as the system state is non-constant.
- Finally, it might be favorable if the system is able to simultaneously cope with *heterogeneous resource pools* like, for example, different data centers; at its best, this works *across organizational boundaries*.

Manually provisioning an infrastructure that delivers a service as defined in a SLA fulfills all functional requirements outlined above but the automation. However, automation opens up the potential of speeding up the provisioning process. To forestall its properties one can say that it satisfies all above requirements. One more point about the agnosticism to the SLA language is noteworthy: the examples presented in this paper assume SLAs specified according to the WS-Agreement specification and the template itself is exemplified via XML. However, the general components and processes are as well applicable to other languages by adapting the location pointers used in the implementation plan template.

4 Template-Based Agreement Provisioning Framework

Addressing the detailed requirements defined in the previous section, this section introduces the template-based agreement provisioning framework, specifically addressing

provisioning planning of the agreement delivery layer of the ADSOA. This framework comprises two elements, a representation for Agreement Implementation Plan Templates and a Definition of the Provisioning Planning process based on these templates.

4.1 Agreement Implementation Plan Templates

The framework aims at determining the resources required to provision a given SLA which might, for example, be specified as a WS-Agreement for the above-mentioned CRM application service. To this end, the framework's core element is the *agreement implementation plan template* (IP template for short). The IP template contains a description how to create a complete provisioning plan from a given agreement offer. To this end, an IP template comprises four sections:

1. *Agreement parameter identifiers*,
2. *Partial provisioning plan*,
3. *Instance completion description*,
4. *Provisioning engine invocation section*.

The components of an IP template are sketched in Figure 2 and discussed in more detail in the following.

Agreement Parameter Identifiers. The purpose of the agreement parameter identifiers section is to relate the IP template to agreements to which it can be applied. When designing the IP template one has to relate it to a class of potential agreements that follow a similar structure, potentially being created according to an agreement template, as, for example, the WS-Agreement specification draft suggests. However, this is an example and the IP template itself and the process that uses it are agnostic to the semantics of agreements, if based on XML. The section contains an arbitrary number of agreement parameter identifiers – each of which having a unique name and a location pointer. The location pointer points to exactly one location in an agreement, referred to as agreement part. Agreement parts can be any clearly identified substructure of an agreement. The pointer concept is sketched in Figure 2.

In the CRM application service example agreement parts might be performance requirements like response time and throughput, the pricing scheme, and the firewall configuration. If the agreement is specified in an XML data structure, the XPath format can be employed to represent location pointers; the author of such an XPath expression has to make sure that it resolves to one and only one single location in an agreement document. A corresponding agreement parameter identifier using XPath is exemplified below:

```
…
<AgreementParameterIdentifiers>
  <ParameterIdentifier name="AverageResponseTime">
    <LocationPointer>
      //wsag:GuaranteeTerm[@wsag:Name='resTime']/*Value
    </LocationPointer>
  </ParameterIdentifier>
  …
</AgreementParameterIdentifieres>
…
```

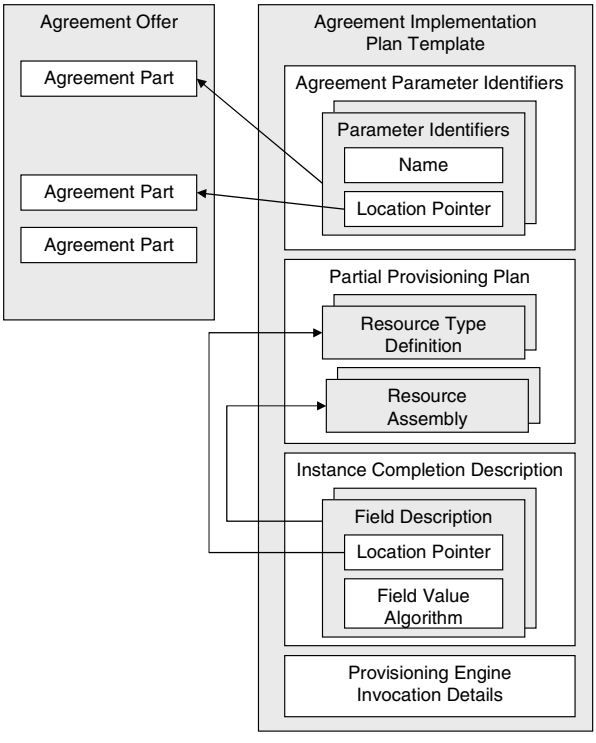

Fig. 2. Pointer structure in agreement implementation plan templates

Partial Provisioning Plan. The partial provisioning plan has a format that is interpreted by a provisioning engine. The plan has open or modifiable fields that will be filled in with values as described in the instance completion description. Most provisioning systems today have their proprietary format but some standards are under development such as CDDLM [2] or IUDD [16]. The presented approach only relies on an XML representation.

We present a simple proprietary example language that we use for provisioning prototypes; in a productive environment, CDDLM or IUDD can easily be used, as the overall mechanism is agnostic to the specific XML language employed. The description comprises the definition of resource types and one or more definitions of resource assembly which are alternatives and among which can be chosen depending on resource availability and cost considerations. Different alternatives might for example be whether to employ a mainframe or a cluster for the CRM application service example. The definition of resource types contains the information to uniquely identify the type of resources to a resource pool, e.g., the cluster management system of a data center, to query the resources availability. The following XML listing exemplifies the definition:

```
…
<PartialProvisioningPlan>
  <ResourceTypeDefinitions>
    <ResourceType name="P-Series">
      <HostType description="pSeries550">
        <HostArchitecture>
          <CPUCount>4</CPUCount>
        </HostArchitecture>
        …
      </HostType>
    </ResourceType>
    …
  </ResourceTypeDefinition>
  …
</PartialProvisioningPlan>
…
```

The definition of resource assembly comprises resource quantity definitions, indicating how many resources for which type are needed for this assembly. Furthermore, the assembly contains a definition how and in which order the resources will be configured and provisioned. This definition might be written in a script language such as Unix shell script or Perl, or in a workflow language such as BPEL4WS.

Instance Completion Description. The instance completion description section of the IP template defines what parts will be filled in and substituted in the partial provisioning plan. For this, the instance completion description comprises a set of field descriptions, each of which explains how to create a value for a specific part of the partial provisioning plan. Such a field description is made up of a location pointer and an algorithm for deducing the field value. The location pointer is analogous to the above-mentioned location pointer. The algorithm is represented in a format that can be automatically interpreted – any such format is possible; the PMAC Expression Language [8], for example, is a suitable representation, as the following code illustrates:

```
…
<InstanceCompletitionDescription>
  <FieldDescription>
    <LocationPointer>
      //ProvisioningProcessDescription/*NumberOfServers
    </LocationPointer>
    <FieldValueAlgorithmDescription>
      <exp:Plus>
        <exp:FloatConstant>
          <Value>02.000</Value>
        </exp:FloatConstant>
        <exp:Divide>
          <exp:FloatConstant>
            <Value>01.000</Value>
          </exp:FloatConstant>
          <exp:Variable name="AverageResponseTime"/>
```

```
        </exp:Divide>
      </exp:Plus>
    </FieldValueAlgorithmDescription>
  </FieldDescription>
  …
</InstanceCompletitionDescription>
…
```

In this example the number of servers to be provisioned increases the shorter the average response time is chosen. The desired response time is extracted from the agreement, i.e. the SLA, via the parameter identifier, as shown in the example above. The deduced number of servers is filled in the partial provisioning plan. Besides the sketched expression language, a field value algorithm can contain a call to an external algorithms, functions, and programs performing more complex estimations for the resource requirements.

Provisioning Engine Invocation. The fourth section of the IP template gives details on the provisioning engine to use. This enables environments with multiple provisioning engines by defining the endpoint reference to which a complete provisioning plan instance is sent. An example in the simplest case is:

```
…
<ProvisioningEngineInvocationDetails>
  <wsa:EndpointReference>
    http://manamgement.ibm.com:8080/provisioning
  </wsa:EndpointReference>
</ProvisioningEngineInvocationDetails>
…
```

4.2 Provisioning Process

The outlined template mechanism (1) identifies the parts of a SLA which are relevant for supplying resources, (2) derives quantities for different resource types, determines and outlines alternative resource assemblies that might be used, and (3) compiles a provisioning plan detailing the required resources, their configuration, and their provisioning. The processing of such an IP template is implemented by the Provisioning Planner; the provisioning itself is carried out be the Provisioning Engine.

Provisioning Planner: Upon receiving an agreement, the Provisioning Planner analyzes it and checks its syntactic correctness. Subsequently, a set of implementation plan templates associated with the agreement offer is retrieved from a template repository and is subsequently used to devise a provisioning plan.

The first applicable template is chosen and processed up to a provisioning plan. The provisioning engine then executes this provisioning plan – it acquires and configures the respective resources and reports the result of this provisioning process back to the agreement provisioning planner. In case of failure, the agreement provisioning planner can devise an alternative provisioning plan from the next IP template retrieved from the repository.

Processing a single IP template involves the following:

1. Verify for each location pointer of each parameter identifier if it points to one and only one location in the received agreement.
2. Retrieve values from the agreement as specified by the parameter identifiers and store them indexed by their respective names.
3. Write a copy of the provisioning process description.
4. For all field descriptions in the instance completion description:
   a. Execute the field value algorithm.
   b. Insert the value returned in the provisioning plan instance at the location given by the field description's location pointer.

With completion of these steps – possibly for several IP templates if there might be failures – the algorithm yields a complete and executable instance of the provisioning plan. Hence, there is no need for the Provisioning Planner to understand the semantics of the provisioning plan as the semantics, e.g., a system administrator's knowledge on how many servers are to utilize to meet a given response time goal, is captured in the IP template itself.

Provisioning Engine. The Provisioning Engine interprets the output of the Provisioning Planner. Although our approach can work with different engines, we illustrate the workings of the provisioning engine along a simple prototypical implementation.

When one of the IP templates results in a complete provisioning plan, the provisioning planner retrieves endpoint reference of the provisioning engine to be used from the provisioning engine invocation details in the IP template. It sends the completed provisioning plan to this provisioning engine which proceeds with the following steps:

1. Select the first resource assembly within the provisioning plan.
2. Check whether the resources can be acquired from the resource pool in the quantity indicated by the resource quantity definition of the respective resource assembly.
   If not, this step is repeated with the next resource assembly, if any. If there is no resource assembly left, a failure notice is returned the provisioning planner and the process is terminated here.
3. Acquire resources in the desired quantity, as step 2 assured that they are available.
4. Execute the assembly provisioning plan to configure the assembly.

If step four completes, the provisioning is complete and the service can be used. The provisioning engine reports the successful resource acquisition back to the provisioning planner which in turn informs the service client that presented the service offer in the first place. The service management then starts the service by making it available to the service client.

Heterogeneous Resource Pools. Up to now, the nature of resource pools and the specific acquisition mechanism was not addressed. The template-based approach is applicable to different resource pools: it can equally be applied for provisioning of resources within a single host, within one data center, across different data centers run

by the same organization, and to inter-organizational resource acquisition. The crucial factor is that the provisioning engine has to be able to communicate with the resources or resource providers respectively. The easiest way is direct access to the scheduler; a more sophisticated acquisition – which might be applied across organizational boundaries and maybe even within a single data center – might be market-based. The provisioning engine could, for example, negotiate with several resource providers as outlined by Czajkowski et al. [4] and Gimpel et al. [7] or it could acquire the resources on a structured marketplace as presented by Buyya et al. [3] and Schnizler et al. [14]. With this, a service provider might become a service broker and distributor, potentially operating without putting forth own resources.

5 Summary and Conclusion

In this paper, we proposed a template-based agreement-driven service provisioning process to facilitate automated service provisioning and by that enable an agreement-driven service-oriented architecture providing dynamic service capacity acquisition. In the template-based agreement provisioning framework introduced in this paper, an agreement implementation plan template is associated with an agreement template. It defines a partially filled provisioning plan with a description how to fill the variable, incomplete elements with input from an agreement. A processor for agreement implementation plans is also defined, implementing a template-based provisioning planner. The provisioning planner has been implemented using Java and tested with a set of agreement templates defined using the WS-Agreement standard.

Provisioning planning is very complex and hard to solve with derivative approaches in the general case. However, the proposed approach based on an agreement implementation plan templates associated with agreement templates can capture the experience of system administrators and, hence, solve the provisioning planning problem pragmatically for service delivery environments in which the relationship of typical customer performance requirements and resource capacity is well understood. The proposed approach is also agnostic to the specific agreement language and the language of the provisioning plan, as both can vary depending on the application domain.

In future work, we will investigate how this template-based approach can be combined with derivative approaches for specific application, leveraging the strength of different approaches.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification. Version 1.1, GGF GRAAP working Group Draft 18, May 14, 2004.
2. Bell, D., Kojo, T., Goldsack, P., Loughran, S., Milojicic, D., Schaefer, S., Tatemura, J., Toft, P.: *Configuration Description, Deployment, and Lifecycle Management (CDDLM) Foundation Document*. January 2003, http://forge.gridforum.org/projects/cddlm-wg.
3. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience*, 14(13-15), pp. 1507–1542, 2002.

4. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for Negotiation of Service Level Agreements and Coordinated Resource Management in Distributed Systems. *Job Scheduling Strategies for Parallel Processing: 8th International Workshop (JSSPP 2002)*. Edinburgh, 2002.
5. Dan, A., Dumitrescu, C., Ripeanu, M.: Connecting client objectives with resource capabilities: an essential component for grid service management infrastructures. *Service-Oriented Computing - ICSOC 2004, Second International Conference*, New York, NY, USA, Proceedings, pp. 57-64, ACM 2004.
6. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G.: Reducing the Complexity of Application Deployment in Large Data Centers. Proceedings of the *9th International IFIP/IEEE Symposium on Integrated Management (IM 2005)*, IEEE Press, 2005.
7. Gimpel, H., Ludwig, H., Dan, A., Kearney, B.: PANDA: Specifying Policies for Automated Negotiations of Service Contracts. *Service Oriented Computing – Proceedings of ICSOC 03*, Springer LNCS 2910, pp. 287-302, 2003
8. IBM Corporation: PMAC Expression Language Users Guide. Alphaworks PMAC distribution, www.alphaworks.ibm.com, 2005.
9. Keller, A., Badonnel, R.: Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. *Proceedings of DSOM 2004*, Davis, CA, USA, 2004.
10. Keller, A.: Automating the Change Management Process with Electronic Contracts. Proceedings of the *First IEEE International Workshop on Service oriented Solutions for Cooperative Organizations (SoS4CO '05)*, IEEE Computer Society Press, 2005.
11. Ludwig, H., Dan, A., Kearney, R.: Cremona: an architecture and library for creation and monitoring of WS-Agreements. *Service-Oriented Computing - ICSOC 2004, Second International Conference*, New York, NY, USA, Proceedings, pp. 65-74, ACM 2004.
12. Ludwig, H., Keller, A., Dan, A., King, R.: A Service Level Agreement Language for Dynamic Electronic Services. *Proceedings of WECWIS 2002*, Newport Beach, 2002.
13. Ludwig, H.: A Conceptual Framework for Electronic Contract Automation. *IBM Research Report*, RC 22608. New York, 2002.
14. Schnizler, B., Neumann, D., Weinhardt, C.: Resource Allocation in Computational Grids – A Market Engineering Approach, Proceeding of the WeB 2004, Washington, 2004
15. Tosic, V., Pagurek, B., Patel, K.: WSOL - A Language for the Formal Specification of Classes of Service for Web Services. *Proceedings of ICWS 2003*, pp. 375-381, CSREA Press 2003.
16. Vitaletti, M., Draper, C., George, R., McCarthy, J., Poolman, D., Miller, T., Middlekauff, A., Montero-Luque, C.: *Installable Unit Deployment Descriptor Specification Version 1.0*. W3C Member Submission, 12 July 2004. http://www.w3.org/Submission/2004/SUBM-InstallableUnit-DD-20040712/